Σχεδίαση και Ανάλυση Αλγορίθμων

Ενότητα 1.0

Εισαγωγή στις Βασικές Έννοιες του Μαθήματος

Απαιτήσεις Μαθήματος και Εργαστηρίου

Περιήγηση στις Βασικές Έννοιες του Μαθήματος

Σταύρος Δ. Νικολόπουλος | 2017-18

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

Webpage: www.cs.uoi.gr/~stavros

Διδάσκοντες Μαθήματος

Διαλέξεις

Σταύρος Δ. Νικολόπουλος

Γραφ. Γ-04

Τηλ. 26510 08801

Εργαστήριο

Βασιλική Σταμάτη

Άννα Μπαντή

Ιωσήφ Πολενάκης

Γραφ. Β11, Τηλ. 26510 08872

Γραφ. Γ-05, Τηλ. 26510 8831 / 32

Διαλέξεις-Εργαστήριο Μαθήματος

Διαλέξεις

- **Αίθουσα Ι5**

  Δευτέρα 11:00 – 13:00

  Τετάρτη 12:00 – 14:00

  Πέμπτη 12:00 – 13:00

- **Παρακολούθηση**

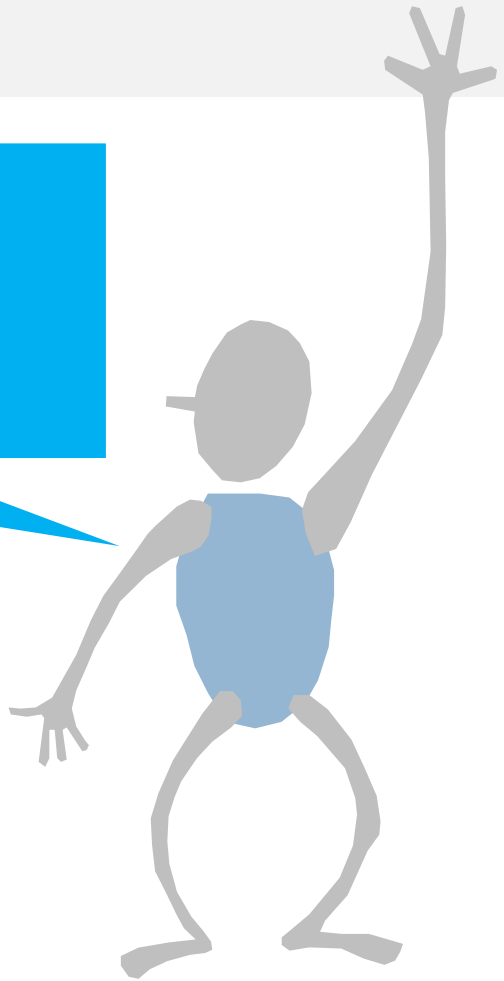
  ...Πάντα!!!

- **Μελέτη**

  ...Συνεχώς!!!

Διαλέξεις

Να έχετε ενδιαφέρον για το μάθημα !
Να κάνετε ερωτήσεις !!
Και να ζητάτε πάντα αποδείξεις !!!



- Παρακολούθηση

  ...Πάντα!!!

- Μελέτη

  ...Συνεχώς!!!

Εργαστήριο

- Εργαστήρια ΠΕΠ-1, ΠΕΠ-2 & ΠΕΛΣ

  Τετάρτη **2-4** μμ και **4-6** μμ

  2 φοιτητές / υπολογιστή

- 

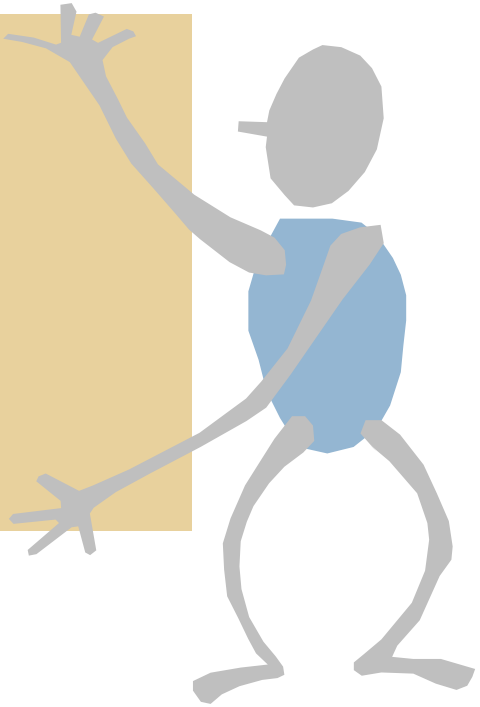
- Δήλωση μέσω e-course www.ecourse.uoi.gr

Εργαστήριο

- Εργαστήρια ΠΕΠ-1, ΠΕΠ-2 & ΠΕΛΣ

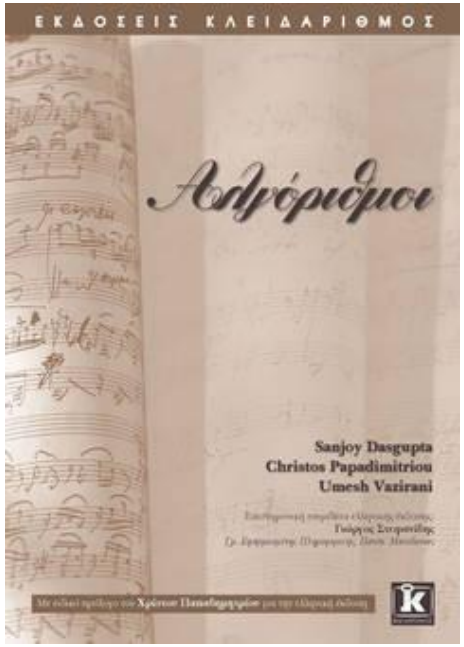
Έναρξη

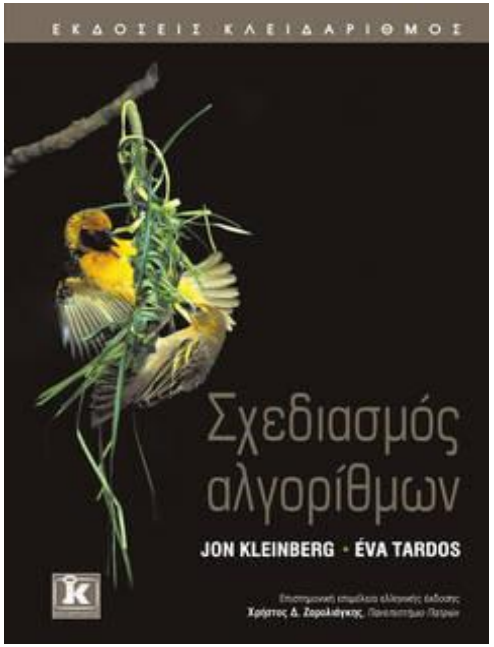
Τετάρτη 28/02/2018



Συγγράμματα Μαθήματος

Εύδοξος

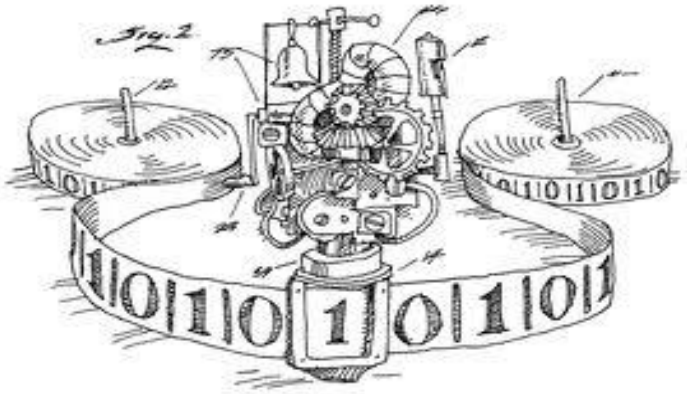


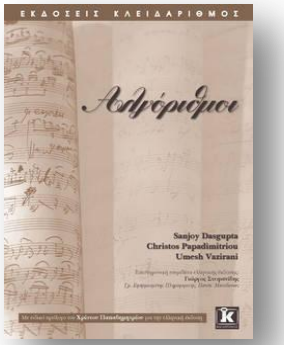


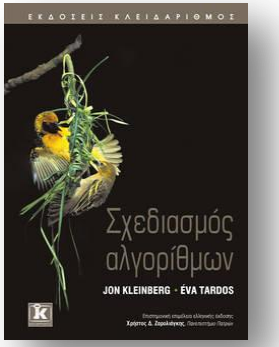
Καλή Μελέτη !!!

Ερωτήσεις... καλού Φοιτητή

Γιατί... Σχεδίαση και Ανάλυση Αλγορίθμων;



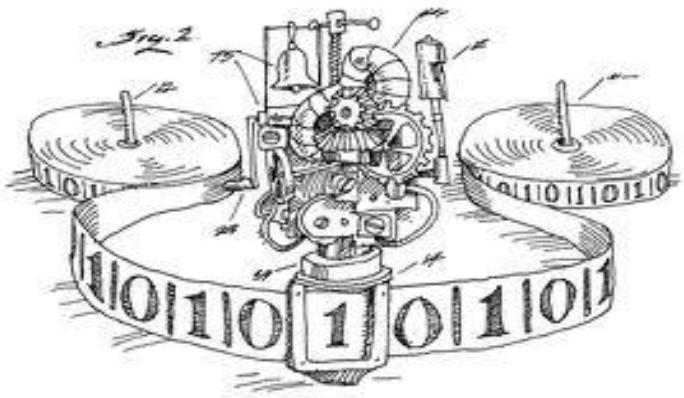




Ποιο... Σύγγραμμα γι' αυτό το Μάθημα;

Ερωτήσεις... καλού Φοιτητή

Γιατί... Σχεδίαση και Ανάλυση Αλγορίθμων;



Ας ξεκινήσουμε την περιήγησή μας...

Μια Πρώτη «Αφελής» Ερώτηση !!!

Θέλεις να είσαι ένας

Καλός Προγραμματιστής ?

ή ένας…

Καλός Επιστήμονας Πληροφορικής ?

Πρωτοπόρος της Επιστήμης σου !!!

Βασικές Έννοιες

Η απάντησή σου !

- Καλός Προγραμματιστής

  Πες μου τι θέλεις να κωδικοποιήσω !

Τι περιμένεις να σου πει ?

Η έλλειψη πρωτοβουλίας δεν εκτιμάται από κανέναν !!!

Βασικές Έννοιες

Πες μου τι να κωδικοποιήσω !

- Πάρε τον παρακάτω Αλγόριθμο
- Και κωδικοποίησέ τον σε C !

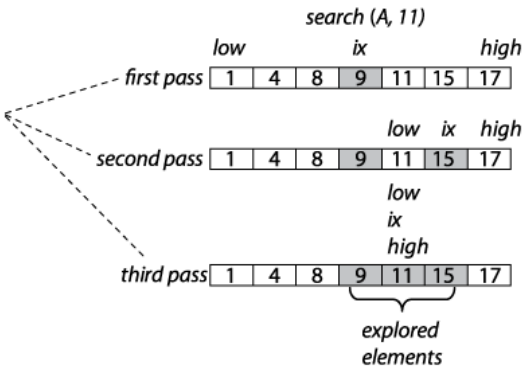
BINARY SEARCH

Best	Average	Worst
O (1)	O (log n)	O (log n)

Array

Divide and Conquer

```
search (A, t)
1.    low = 0
2.    high = n −1
3.    while (low ≤ high) do
4.      ix = (low + high)/2
5.      if (t = A[ix]) then
6.          return true
7.      else if (t < A[ix]) then
8.          high = ix − 1
9.      else low = ix + 1
10.   return false
end
```



Βασικές Έννοιες

Πες μου τι να κωδικοποιήσω !

- Πάρε τον παρακάτω Αλγόριθμο
- Και κωδικοποίησέ τον σε C !

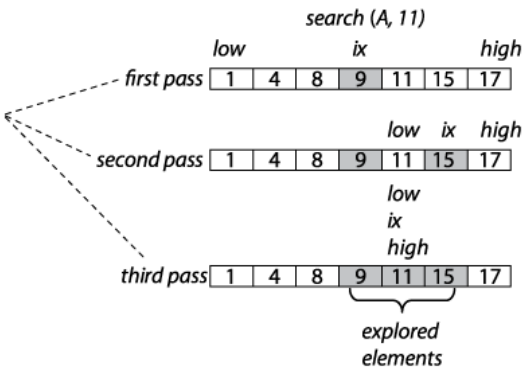
BINARY SEARCH

Best	Average	Worst
O (1)	O (log n)	O (log n)

Array

Divide and Conquer

```
search (A, t)
1.   low = 0
2.   high = n −1
3.   while (low ≤ high) do
4.     ix = (low + high)/2
5.     if (t = A[ix]) then
6.         return true
7.     else if (t < A[ix]) then
8.         high = ix − 1
9.     else low = ix + 1
10.  return false
end
```




```c
#include<stdio.h>
int BinarySearch(int A[],int n,int x)
{
    int low = 0, high = n-1;
    int mid = (low+high)/2;
    if(x == A[mid]) return mid; //Found X, return (exit)
    else if(x < A[mid]) high = mid-1;
    else low = mid+1;
}
int main()
{
}
```

Πες μου τι να κωδικοποιήσω !

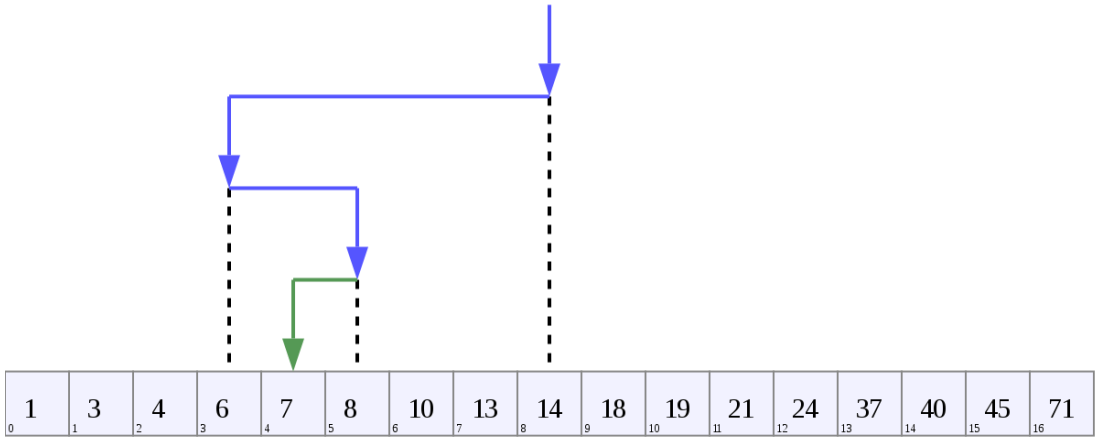
- Πάρε το παρακάτω Πρόβλημα
- Και…

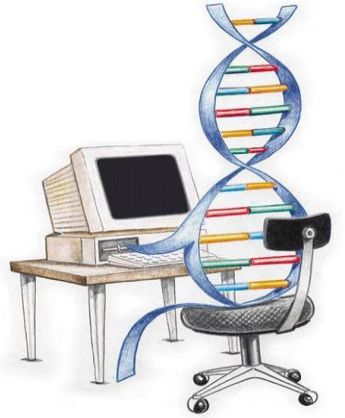
Πρόβλημα
Δυαδική Αναζήτηση

Πες μου τι να κωδικοποιήσω !

- Μια μελέτη έδειξε !!!

  Πολλοί έμπειροι προγραμματιστές κλήθηκαν να κωδικοποιήσουν τη Δυαδική Αναζήτηση !!!



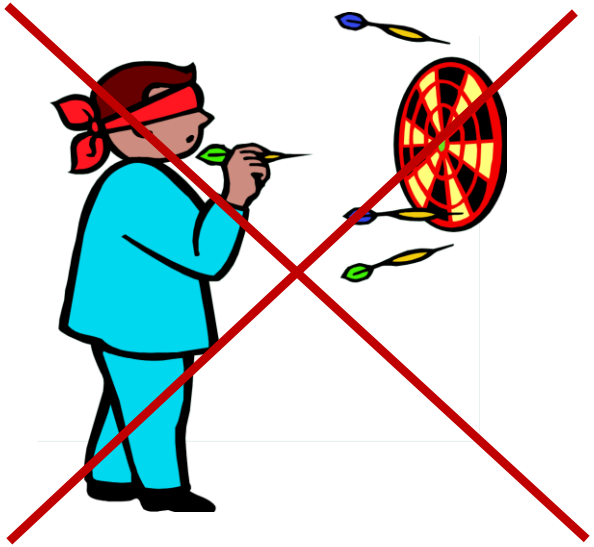


Βασικές Έννοιες

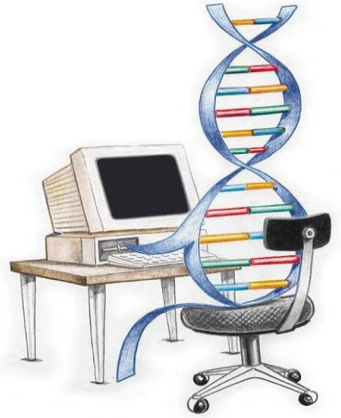
Πες μου τι να κωδικοποιήσω !

- Μια μελέτη έδειξε !!!

  Το 80% από αυτούς έκανε λάθος !!!



Λάθος Στόχος !!!



Μια άλλη απάντηση !

- **Καλός Επιστήμονας Πληροφορικής**

  Γνώστης όλων των Αλγορίθμων της

  Έμαθα αυτό τον υπέροχο αλγόριθμο που θα λειτουργήσει !

Και λοιπόν !!!...
Όλοι οι γνωστοί αλγόριθμοι είναι διαθέσιμοι στις βιβλιοθήκες και στο διαδίκτυο !!!

Βασικές Έννοιες

Έμαθα αυτό τον αλγόριθμο !

- Πάρε έναν αλγόριθμο.
- Μάθε τον κώδικά του !
- Δοκίμασέ τον (trace) μέχρι να είσαι πεπεισμένος ότι αυτός λειτουργεί !!
- Εφάρμοσέ τον !!!

Λάθος Προσέγγιση !!!

ALGORITHM A Recursive Binary Search Algorithm.

procedure *binary search*$(i, j, x: i, j, x$ integers, $1 \le i \le j \le n)$
$m := \lfloor (i + j)/2 \rfloor$
if $x = a_m$ **then**
    return m
else if $(x < a_m$ and $i < m)$ **then**
    return *binary search*$(i, m - 1, x)$
else if $(x > a_m$ and $j > m)$ **then**
    return *binary search*$(m + 1, j, x)$
else return 0

Βασικές Έννοιες

Μια ενδιαφέρουσα απάντησή !

- **Καλός Επιστήμονας Πληροφορικής**
  Πρωτοπόρος της Επιστήμης μου

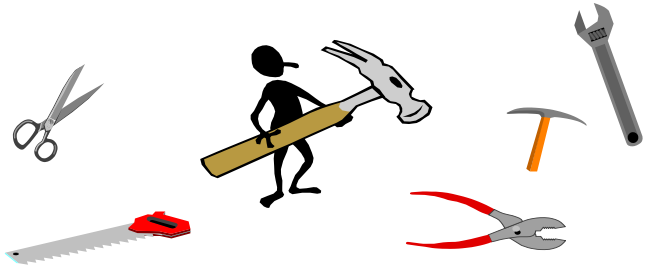
Μπορώ να αναπτύξω ένα Νέο αλγόριθμο
για το πρόβλημα που μου έδωσες !!!

Πάντα θα χρειάζονται
πρωτοπόροι με Νέες και Μεγάλες Ιδέες !!!

Μπορώ να αναπτύξω Νέο αλγόριθμο !

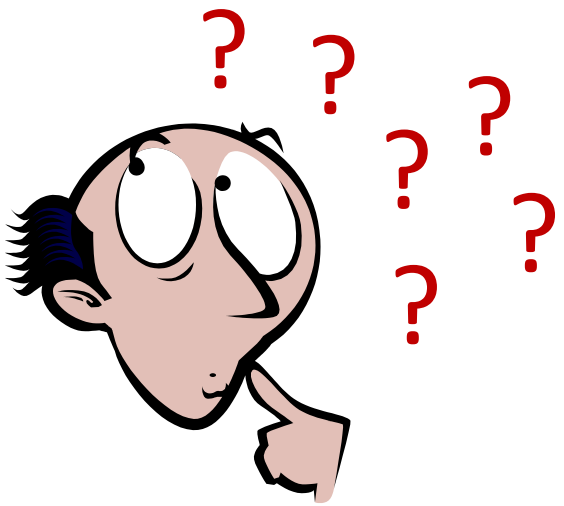
- Έρευνα Αλγοριθμικών Τεχνικών !
- Βελτίωση υπαρχόντων Αλγορίθμων !!
- Ανάπτυξη Νέων Αλγορίθμων για κάθε πρόβλημα που μπορεί να προκύψει !!!

Σωστή Προσέγγιση !!! ✓



Φιλοσοφία Μαθήματος

Τι πρέπει να διαθέτω ;



Προβληματισμοί & Στόχοι

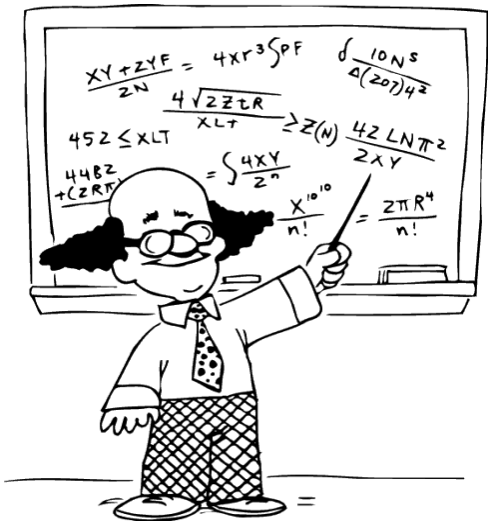
Πρωτότυπη Σκέψη !!!
&
Επιστημονική Γνώση

Πάντοτε ο άνθρωπος
ζητά Λύσεις σε Προβλήματα !!!

Φιλοσοφία Μαθήματος

Δεν πρέπει να μου λείπουν !!!

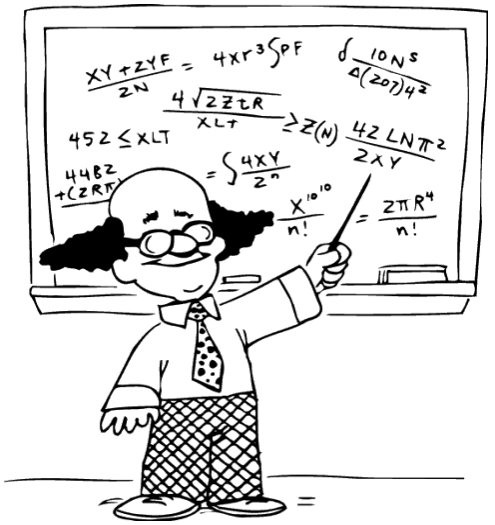
- ✓ Τυπικές μέθοδοι απόδειξης !!!
- ✓ Θεμελιώδη κατανόηση των αλγοριθμικών τεχνικών σχεδιασμού αλγορίθμων !!!
- ✓ Αφαιρετική σκέψη !!!
  (abstract thinking)



Φιλοσοφία Μαθήματος

Η περιγραφή ενός αλγόριθμου μπορεί να αναφέρει:

- ✓ *αφηρημένα αντικείμενα* (ακεραίους, πραγματικούς, συμβολοσειρές, σύνολα, στοίβες, δένδρα, γραφήματα), ή
- ✓ *αφηρημένες πράξεις* («ταξινόμησε τη λίστα», «αφαίρεσε ένα στοιχείο από τη στοίβα», «ακολούθησε μια διαδρομή»), ή
- ✓ *αφηρημένες σχέσεις* ("το μεγαλύτερο από", "ένα πρόθεμα", "ένα υποσύνολο").



✓ **Αφαιρετική σκέψη !!!**
(abstract thinking)

Η ικανότητα να σκέφτεσαι *πέρα* από *συγκεκριμένα αντικείμενα* και *απλές ιδέες* !!!

Θεμελιώδης Διαπίστωση

Το μέλλον ανήκει !!!

Στον επιστήμονα πληροφορικής που διαθέτει:

- ✓ **Περιεχόμενο !!!**

  Ενημέρωση και Κατανόηση των θεμελιωδών προβλημάτων και των λύσεών τους

- ✓ **Μέθοδο !!!**

  Αρχές και Αλγοριθμικές Τεχνικές για την αποτελεσματική επίλυση Νέων Προβλημάτων

Το Μάθημά μας

Το μέλλον ανήκει !!!

Σχεδίαση και Ανάλυση Αλγορίθμων

Σχεδίαση Αλγορίθμων

&

Ανάλυση Αλγορίθμων

Το Μάθημά μας

Το μέλλον ανήκει !!!

Σχεδίαση Αλγορίθμων

Επινόηση Αλγόριθμου & Απόδειξη Ορθότητας

Ανάλυση Αλγορίθμων

Εκτίμηση Πολυπλοκότητας

Στόχος Μαθήματος

Πρέπει να αποκτήσω !!!

Γνώση **Αλγοριθμικών Τεχνικών** και **Αφαιρετική Σκέψη**
έτσι ώστε να μπορώ:

- ✓ να **Σχεδιάζω** νέους **Αλγορίθμους**

  - Νέα Προβλήματα
  - Νέες Επιστημονικές περιοχές
  - Νέες Εφαρμογές

Στόχος Μαθήματος

Πρέπει να αποκτήσω !!!

Γνώση **Αλγοριθμικών Τεχνικών** και **Αφαιρετική Σκέψη**
έτσι ώστε να μπορώ:

- ✓ να Αποδεικνύω την Ορθότητα, και
- ✓ να Υπολογίζω την Πολυπλοκότητα

των **αλγορίθμων** μου!!!

Ερωτήματα – Προβληματισμοί (1)

Πρέπει να Γνωρίζω !

Ποια η θέση των Αλγορίθμων στην Επιστήμη μας

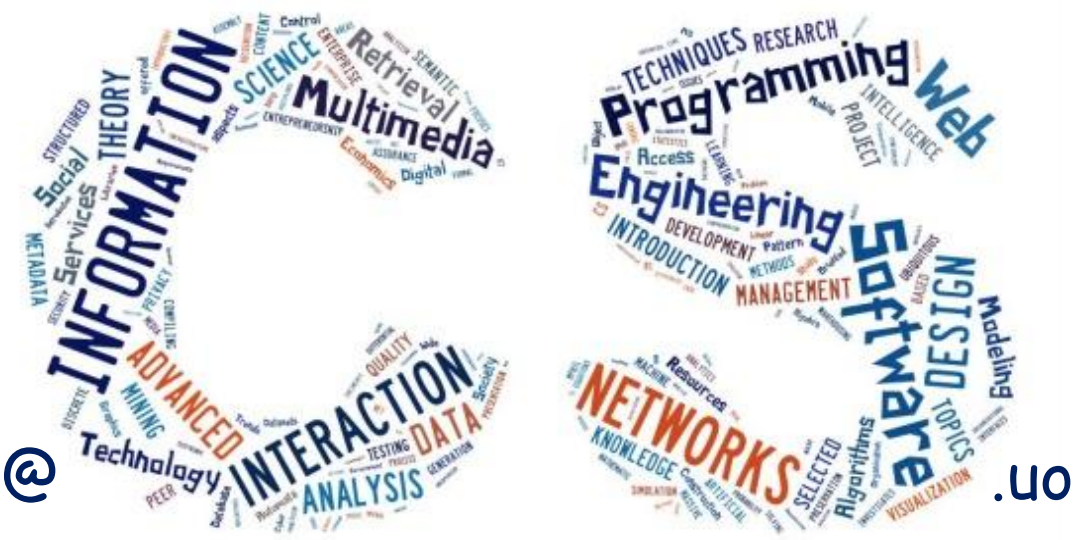
Πόσο σημαντικός τομέας

Τι γνώση υπάρχει

Αποτελέσματα Σταθμούς !

Ερωτήματα – Προβληματισμοί (1)

Τι είναι η Επιστήμη της Πληροφορικής;

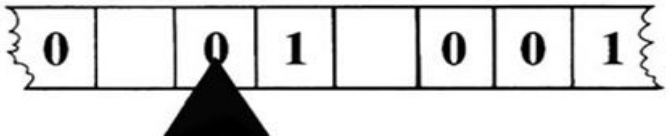


stavros@ .uoi.gr

Ερωτήματα – Προβληματισμοί (1)

Τι είναι η Επιστήμη της Πληροφορικής;

Τι μελετάει;



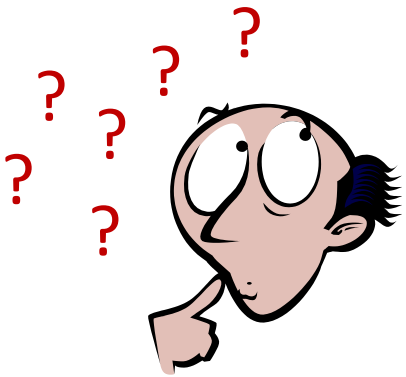
Ερωτήματα – Προβληματισμοί (1)

Τι είναι η Επιστήμη της Πληροφορικής;

Τι μελετάει;

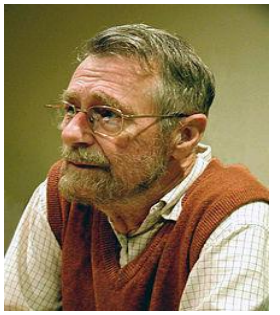
Είναι αλήθεια ότι…

> *είναι η επιστήμη που μελετάει*
> *τους Ηλεκτρονικούς Υπολογιστές (Η/Υ);*



Ερωτήματα – Προβληματισμοί (1)

Edsger W. Dijkstra (1930-2002)



"Computer Science is no more about computers than astronomy is about telescopes."

"Η επιστήμη της Πληροφορικής έχει τόσο σχέση με τους Η/Υ όσο έχει η Αστρονομία με τα Τηλεσκόπια."

Ερωτήματα – Προβληματισμοί (1)

Τι είναι Τελικά;

Θα μπορούσαμε να πούμε ότι είναι
ο επιστημονικός και τεχνολογικός τομέας που:

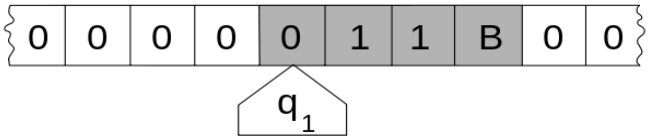
- **μελετάει την**
  - ✓ **αναπαράσταση, αποθήκευση, επεξεργασία, μετάδοση** πληροφοριών μέσω υπολογιστών και δικτύων !
- **και προτείνει**
  - ✓ **τρόπους (αλγόριθμους)**

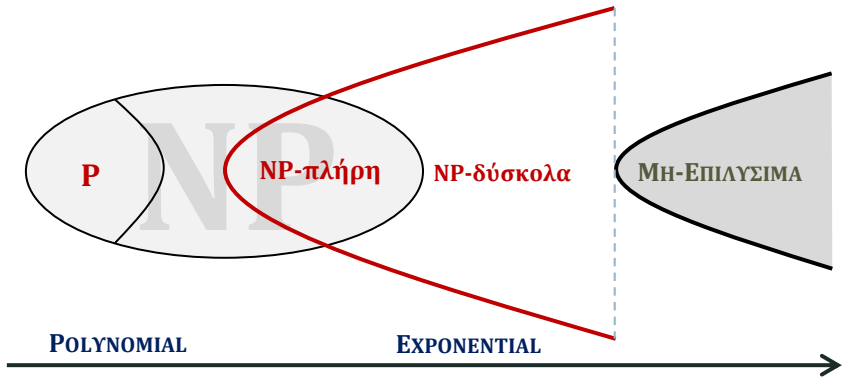
    για την αποδοτική υλοποίηση των παραπάνω εργασιών !!

Ερωτήματα – Προβληματισμοί (1)

Κεντρικά Ερωτήματα της Επιστήμης μας !

- Τι μπορεί να μηχανοποιηθεί και μάλιστα αποδοτικά;
- Ποια προβλήματα μπορούμε να λύσουμε;
- Πόσο καλά μπορούμε να τα λύσουμε;





Ερωτήματα – Προβληματισμοί (1)

Ποια Ερωτήματα μπορούν να Απαντηθούν με Υπολογιστή;

Ερωτήματα – Προβληματισμοί (1)

Ποια Ερωτήματα μπορούν να Απαντηθούν με Υπολογιστή;

- Θα υπάρξει ποτέ Οικονομική Ανάπτυξη στη Χώρα μου;
- Υπάρχει Θεός;
- Η πρόταση «αυτή η πρόταση είναι ψευδής» είναι αληθής;
- Ο αριθμός $2^{43112609} - 1$ είναι πρώτος;

Ερωτήματα – Προβληματισμοί (1)

Ποια Ερωτήματα μπορούν να Απαντηθούν με Υπολογιστή;

- Υπάρχουν ακέραιοι $x, y, z>0$ και $n>2$ τέτοιοι ώστε $x^n + y^n = z^n$;
- Υπάρχει αλγόριθμος που παίρνει ως είσοδο μια πολυωνυμική εξίσωση (π.χ. $x^2 - 2y^2 = 5$ ή $x^3 + y^3 = z^3$) και απαντά εάν έχει λύση ή όχι ;

Το πρώτο πρόβλημα είναι το Τελευταίο Θεώρημα του Fermat που απαντήθηκε (αρνητικά) τον Ιούλιο του 1993 από τον Βρετανό μαθηματικό Andrew Wiles.

Το δεύτερο είναι το 10ο πρόβλημα του David Hilbert που τέθηκε το 1900 και απαντήθηκε (αρνητικά) από τον Yuri Matiyasevich το 1970.

Ερωτήματα – Προβληματισμοί (1)

Ποια Ερωτήματα μπορούν να Απαντηθούν με Υπολογιστή;

- Προϋποθέσεις:
  - ✓ Τα ερωτήματα να καθορίζονται από μια διμελή σχέση εισόδου/εξόδου !
  - ✓ Η είσοδος και η έξοδος να μπορούν να κωδικοποιηθούν με σύμβολα !

Ερωτήματα με τέτοιες προϋποθέσεις είναι τα

Υπολογιστικά Προβλήματα !!!

Υπολογιστικά Προβλήματα

Υπολογιστικά Προβλήματα

Τυπικά περιγράφονται με *διμελείς σχέσεις (στιγμιότυπο-λύση)* μεταξύ συμβολοσειρών!!!

- **Υπολογιστικό πρόβλημα** είναι ένα αντικείμενο που αντιστοιχεί σε ένα σύνολο ερωτήσεων τις οποίες μπορεί να απαντήσει ένας υπολογιστής.
- Ο τομέας των **αλγορίθμων** ερευνά υπολογιστικές μεθόδους επίλυσης των υπολογιστικών προβλημάτων.

Περισσότερα για

Υπολογιστικά Προβλήματα !!!

Ερωτήσεις για Στιγμιότυπα

- Ένα **υπολογιστικό πρόβλημα (ΥΠ)** ορίζει έναν μετασχηματισμό **«δεδομένων εισόδου»** σε **«δεδομένα εξόδου»**.

Δεδομένα Εισόδου ⟺ **Στιγμιότυπο**

Δεδομένα Εξόδου ⟺ **Λύση**

Διαισθητικά, ένα ΥΠ ορίζει **ερωτήσεις** για **στιγμιότυπα**!

Στιγμιότυπο: Οι ακέραιοι 65 και 26

Ερώτηση: Ποιος είναι ο ΜΚΔ των δύο αυτών ακεραίων;

Διμελής Σχέση: (Στιγμιότυπο, Λύση)

- Έτσι, ένα **υπολογιστικό πρόβλημα** μπορεί να θεωρηθεί ως ένα πεπερασμένο σύνολο ***στιγμιοτύπων*** μαζί με μία ***λύση*** για κάθε στιγμιότυπο.

  ✓ Η εύρεση ΜΚΔ (gcd): «Δοθέντος δύο ακεραίων *n και m*, να βρεθεί ο ΜΚΔ αυτών.»
  - R = {((65, 26), 13), ((91, 33), 1), …}

  ✓ Το πρόβλημα της παραγοντοποίησης: «Δοθέντος ενός θετικού ακεραίου *n*, να βρεθεί ένας μη-τετριμμένος πρώτος παράγοντας του *n*.»
  - R = {(4, 2), (6, 2), (6, 3), (8, 2), (9, 3), (10, 2), (10, 5), ...}

  Η *διμελής σχέση* R αποτελείται από όλα τα ζεύγη (n, p), όπου n = input και p = output.

Ιδιότητες Εξόδου

- Σε ένα **υπολογιστικό πρόβλημα** δίδεται μια *είσοδος** και ζητείται ως *έξοδος* μια λύση που ικανοποιεί μια ιδιότητα !

  Έτσι, ένα **υπολογιστικό πρόβλημα** περιγράφεται από την *ιδιότητα* της εξόδου.

- **Παράδειγμα:**
  - ✓ Πρόβλημα Ταξινόμησης: (2, 3, 8, 6, 1, 4) ⇒ (1, 2, 3, 4, 6, 8)
  - ✓ Πρόβλημα Μεγίστου: (1, 3, 5, 8, 1, 7) ⇒ 8

* *Η είσοδος* να μπορεί να κωδικοποιηθεί με σύμβολα (π.χ., από το αλφάβητο Σ={0,1})

Κατηγορίες

- Σε ένα **υπολογιστικό πρόβλημα** δίδεται μια *είσοδος** και ζητείται ως *έξοδος* μια λύση που ικανοποιεί μια ιδιότητα !

  Έτσι, ένα **υπολογιστικό πρόβλημα** περιγράφεται από την *ιδιότητα* της εξόδου.

- Κατηγορίες υπολογιστικών προβλημάτων, όπως:
  - ✓ Απόφασης (decision)
  - ✓ Βελτιστοποίησης (optimization)
  - ✓ Διερεύνησης (search)
  - ✓ Μέτρησης (counting)

* *Η είσοδος* να μπορεί να κωδικοποιηθεί με σύμβολα (π.χ., από το αλφάβητο Σ={0,1})

Προβλήματα Απόφασης & Βελτιστοποίησης

- Δύο σημαντικές κατηγορίες Υπολογιστικών προβλημάτων:

  - **Απόφασης**

    ✓ Διαπιστώνει αν υπάρχει απάντηση στο ερώτημα που ικανοποιεί τα δεδομένα εισόδου και επιστρέφει «Ναι» ή «Όχι».

  - **Βελτιστοποίησης**

    ✓ Επιστρέφει τη λύση που ικανοποιεί κατά τον καλύτερο τρόπο τα δεδομένα εισόδου.

Προβλήματα Απόφασης & Βελτιστοποίησης

- Δύο σημαντικές κατηγορίες Υπολογιστικών προβλημάτων:

• Απόφασης

✓ **Graph Coloring**

Είσοδος: Γράφημα G και $k \in Z_+$

Ερώτηση: Υπάρχει σωστός χρωματισμός του G με k χρώματα?

• Βελτιστοποίησης

✓ **Graph Coloring**

Είσοδος: Γράφημα G

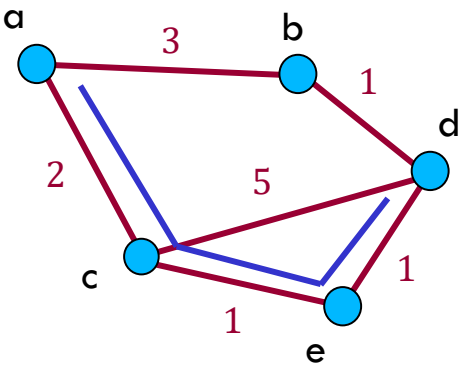
Ερώτηση: Ποιο είναι το ελάχιστο πλήθος χρωμάτων για ένα σωστό χρωματισμός του G?

Μερικά Υπολογιστικά Προβλήματα !

- Το πρόβλημα του 2^{29}
  - Ποιο ψηφίο λείπει από τον αριθμό 2^{29};

    (αποτελείται από 9 διαφορετικά ψηφία: 536870912) ⇒ 4
- Αναγνώριση πρώτων αριθμών
  - $2^{43112609} - 1$ ⇒ «ναι»
- Συντομότερες διαδρομές
  - (({a,b}, 3), ({a,c}, 2), ({b,d}, 1), ({c,d}, 5),

    ({c,e}, 1), ({d,e}, 1), a, d) ⇒ (a,c,e,d) ή (a,b,d)



Μερικά Υπολογιστικά Προβλήματα !

- Αριθμοί Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

$$F_0 = 0, F_1 = 1$$

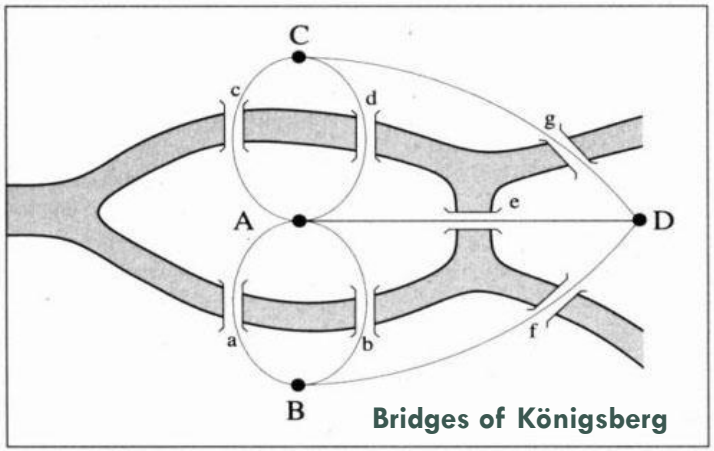
$$F_n = F_{n-1} + F_{n-2}\ , n \geq 2$$

Πρόβλημα:

Δίνεται n. Πόσο γρήγορο μπορεί να υπολογιστεί ο όρος F_n της ακολουθίας Fibonacci;

Μερικά Υπολογιστικά Προβλήματα !

- Κύκλος Euler

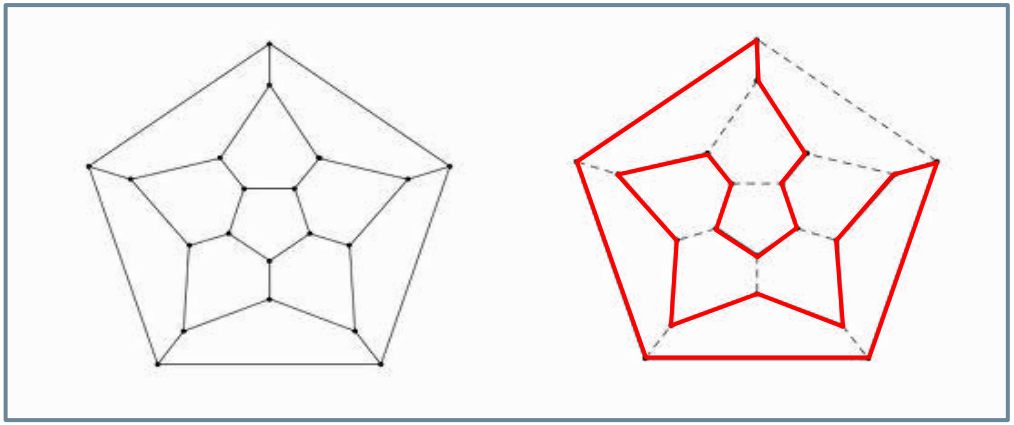


Πρόβλημα:

Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε ακμή *ακριβώς μια φορά*;

Μερικά Υπολογιστικά Προβλήματα !

- Κύκλος Hamilton



Πρόβλημα:

Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε κόμβο *ακριβώς μια φορά*;

Μερικά Υπολογιστικά Προβλήματα !

- Το πρόβλημα του Collatz

Έστω το πρόγραμμα (Half or Triple Plus One):

```
while x!=1 do
   if (x is even) then x=x/2 else x=3*x+1
```

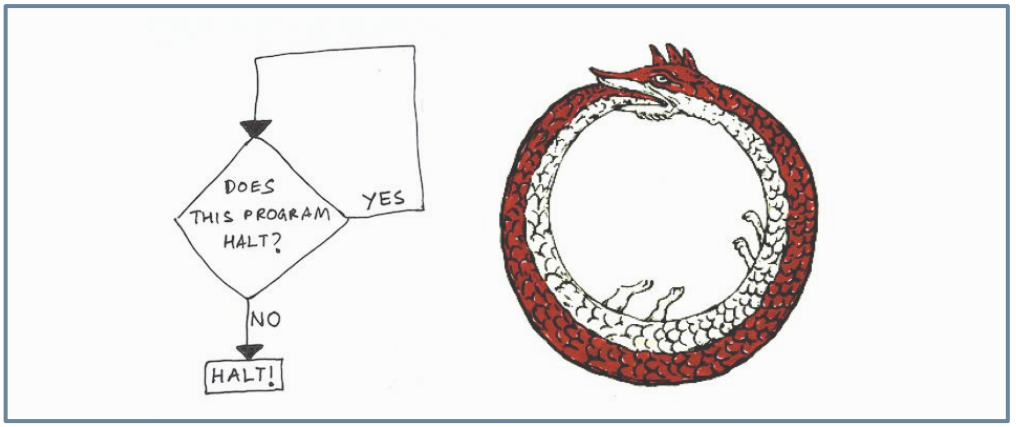
Παράδειγμα **x = 7:** `7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1`

Πρόβλημα:

Δίνεται φυσικός αριθμός **x**. *Τερματίζει* το παραπάνω πρόγραμμα με είσοδο **x**;

Μερικά Υπολογιστικά Προβλήματα !

- Το πρόβλημα του Τερματισμού (Halting Problem)

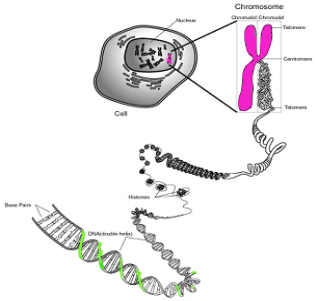


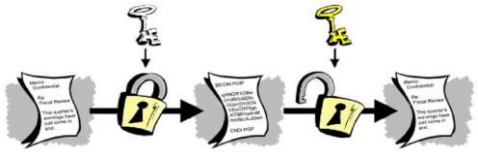
Πρόβλημα:

Δίνεται ***πρόγραμμα Π*** και ***είσοδος x***. *Τερματίζει το πρόγραμμα* ***Π*** *με είσοδο* ***x*** (*ή "τρέχει" επ' άπειρον*);

Εφαρμογές Υπολογιστικών Προβλημάτων

- Εμφανίζονται
  - ✓ Διαδίκτυο
    (δρομολόγηση, θεωρία παιγνίων, ανάθεση πόρων, αναζήτηση)
  - ✓ Βιολογία
    (αναδίπλωση πρωτεϊνών, γονιδίωμα)
  - ✓ Κρυπτογραφία
    (ασφάλεια, μυστικότητα, ηλεκτρονικές υπογραφές, ψηφοφορίες)
  - ✓ και σε πολλά άλλα πεδία !!!



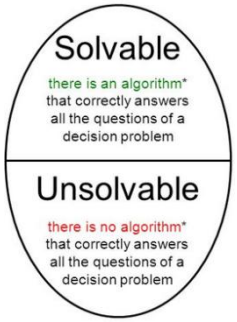


Δύο Σημαντικά Ερωτήματα !!!

- Όλα τα υπολογιστικά προβλήματα έχουν λύση?

  ✓ Όχι

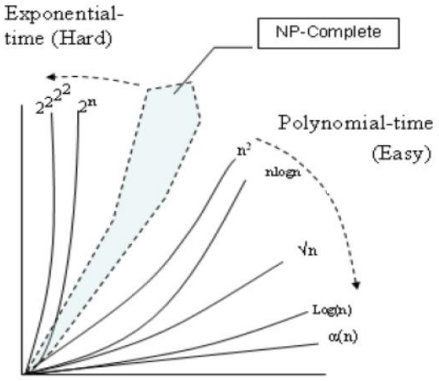
  Υπάρχουν προβλήματα τα οποία είναι μη-επιλύσιμα (πρόβλημα τερματισμού)!



- Όλα τα επιλύσιμα λύνονται το ίδιο εύκολα?

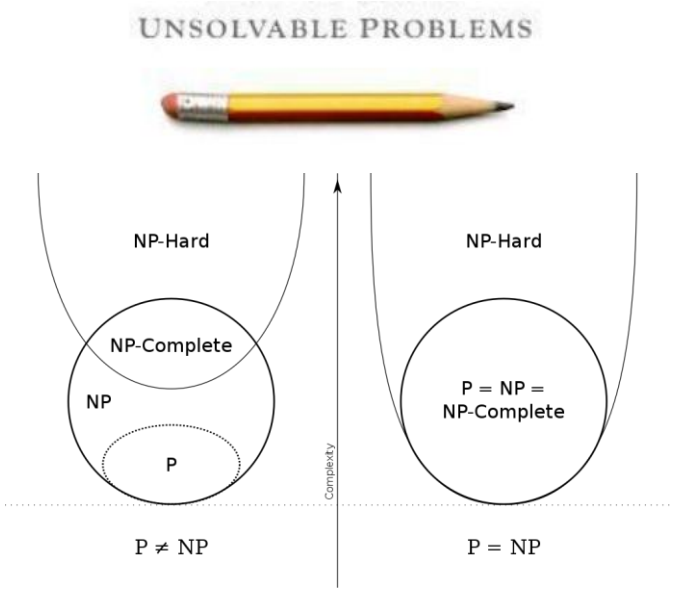
  ✓ Όχι

  Υπάρχουν υπολογιστικά προβλήματα τα οποία λύνονται εύκολα (ακολουθία Fibonacci, ελάχιστη διαδρομή, κύκλος Euler), και άλλα δύσκολα (κύκλος Hamilton)!



Ερωτήματα προς Απάντηση !!!

- ✓ Έχει λύση το πρόβλημα Π ή όχι;
- ✓ Εάν ναι, μπορεί να επιλυθεί γρήγορα;
- ✓ Πόσο γρήγορα μπορεί να επιλυθεί;



Αυτά τα ερωτήματα βρίσκουν απάντηση τις έννοιες

Υπολογισιμότητα – Πολυπλοκότητα !!!

Υπολογισιμότητα & Πολυπλοκότητα

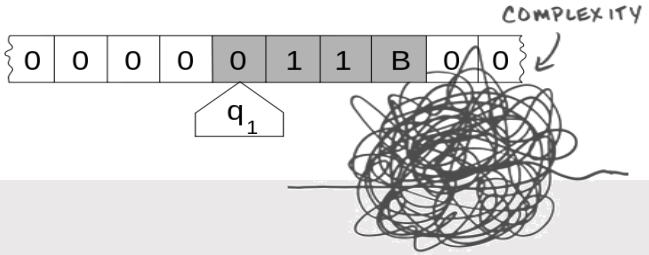
- Η **υπολογισιμότητα** και η **υπολογιστική πολυπλοκότητα** είναι δύο θεμελιώδεις έννοιες της ***θεωρίας υπολογισμού*** η οποία μελετά
  - ✓ **το εάν** μπορεί να λυθεί, και
  - ✓ **το πόσο αποδοτικά** μπορεί να λυθεί

  αλγοριθμικά κάποιο υπολογιστικό πρόβλημα σε ένα υπολογιστικό μοντέλο !!!

Περισσότερα για

Υπολογισιμότητα - Πολυπλοκότητα !!!

Υπολογισιμότητα & Πολυπλοκότητα



Υπολογισιμότητα

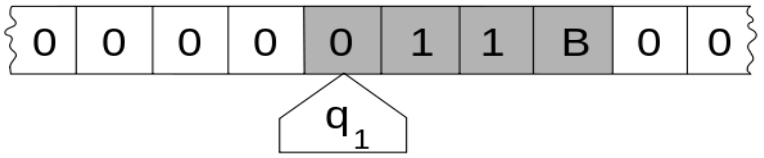
- ***Τι μπορεί να υπολογιστεί*** και τι όχι;

Πολυπλοκότητα

- ***Πόσο γρήγορα*** μπορεί να υπολογιστεί;

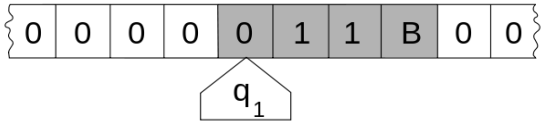
Υπολογισιμότητα (Computability)

- ***Μπορούμε να λύσουμε*** το *υπολογιστικό πρόβλημα* ***Π*** με υπολογιστή;

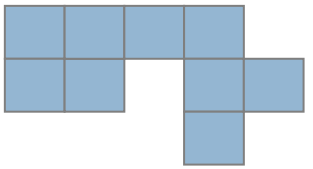


ComputAbility

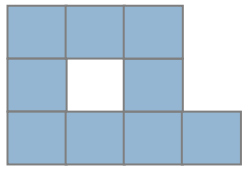
Τι είναι Υπολογισιμότητα;



- Υπάρχει αλγόριθμος που παίρνει ως είσοδο μια πολυωνυμική εξίσωση (π.χ. $x^2 - 2y^2 = 5$) και απαντά εάν έχει λύση ή όχι ;

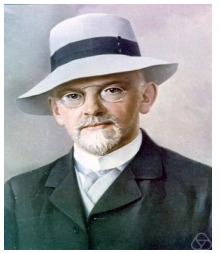


- Είναι το σχήμα που δημιουργείται από τετράγωνα τοπολογικά ισόμορφο με δίσκο; (το πρόβλημα της σφαίρας)



- Δίνεται πρόγραμμα και είσοδος.
  Τερματίζει το πρόγραμμα για αυτή την είσοδο;

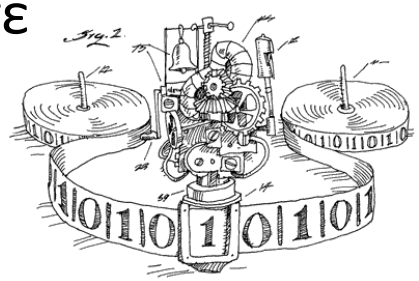
- Υπάρχει αλγόριθμος που να αποκρίνεται (*decides*) *για την* ορθότητα κάθε μαθηματικής πρότασης;



Τι είναι Υπολογισιμότητα Προβλήματος;

Η ***ικανότητα*** της αλγοριθμικής επίλυσης ενός προβλήματος Π

- Η υπολογισιμότητα ενός προβλήματος είναι *στενά συνδεδεμένη* με την *ύπαρξη ενός αλγορίθμου* για την επίλυσή του.
- Η Μηχανή Turing είναι ικανή να υπολογίσει οτιδήποτε είναι δυνατό να υπολογιστεί αλγοριθμικά !
- Τα υπολογιστικά μοντέλα είναι ισοδύναμα !

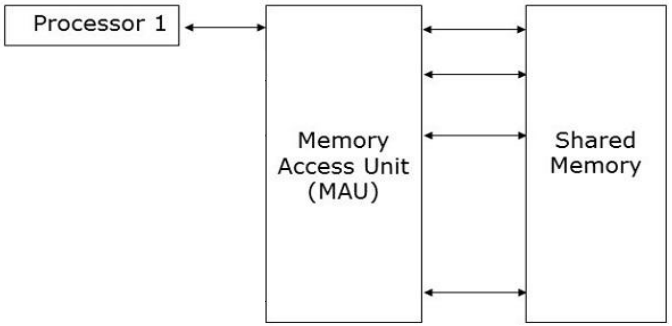


Τι είναι Υπολογιστικό Μοντέλο;

- Ένα **υπολογιστικό μοντέλο** ορίζει το σύνολο των επιτρεπόμενων ***θεμελιωδών εργασιών*** που χρησιμοποιούνται στον υπολογισμό και την αντίστοιχη ***μονάδα κόστους*** τους.

- Στην ανάλυση αλγορίθμων χρησιμοποιούμε συνήθως τη **μηχανή τυχαίας πρόσβασης (RAM):**

John von Neumann
(1903-1957)



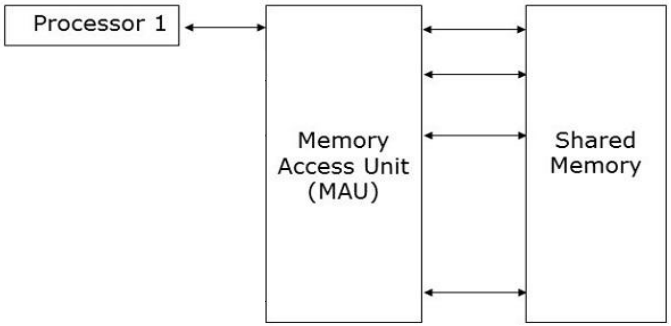
RAM

Τι είναι Υπολογιστικό Μοντέλο;

- Ένα **υπολογιστικό μοντέλο** ορίζει το σύνολο των επιτρεπόμενων ***θεμελιωδών εργασιών*** που χρησιμοποιούνται στον υπολογισμό και την αντίστοιχη ***μονάδα κόστους*** τους.

- Στην ανάλυση αλγορίθμων χρησιμοποιούμε συνήθως τη **μηχανή τυχαίας πρόσβασης (RAM):**

  Κόστος 1 μονάδα για *πρόσβαση ανάγνωσης* και *εγγραφής* στις θέσεις μνήμης του Η/Υ και για *εκτέλεση μιας βασικής πράξης* (πρόσθεση, αφαίρεση, σύγκριση, κλπ).



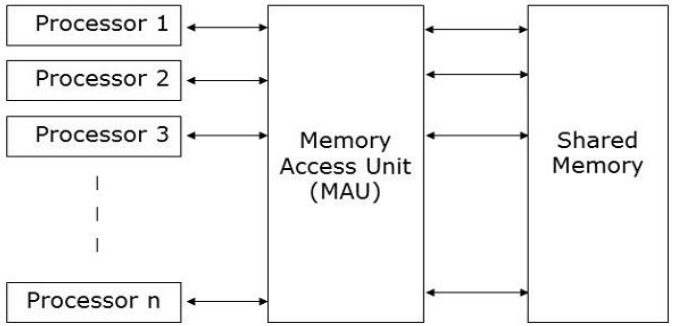
RAM

Τι είναι Υπολογιστικό Μοντέλο;

- Ένα **υπολογιστικό μοντέλο** ορίζει το σύνολο των επιτρεπόμενων ***θεμελιωδών εργασιών*** που χρησιμοποιούνται στον υπολογισμό και την αντίστοιχη ***μονάδα κόστους*** τους.

- Στην ανάλυση αλγορίθμων χρησιμοποιούμε συνήθως τη **μηχανή τυχαίας πρόσβασης (RAM):**

  Κόστος 1 μονάδα για *πρόσβαση ανάγνωσης* και *εγγραφής* στις θέσεις μνήμης του Η/Υ και για *εκτέλεση μιας βασικής πράξης* (πρόσθεση, αφαίρεση, σύγκριση, κλπ).



PRAM

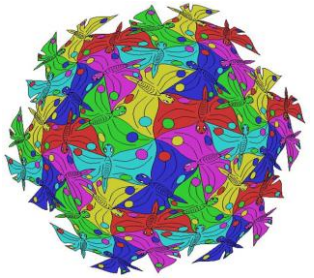
Πολυπλοκότητα (Complexity)

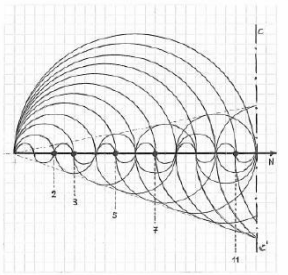
- ***Πόσο γρήγορα*** μπορούμε να λύσουμε το *υπολογιστικό πρόβλημα* ***Π*** με υπολογιστή;

Πρόβλημα

Τι είναι Πολυπλοκότητα;

- Το `101101011101` είναι πιο πολύπλοκο από `000000000000` (Kolmogorov πολυπλοκότητα);
- Τα θηλαστικά είναι πιο πολύπλοκα από τους ιούς;
- Το σκάκι είναι πιο πολύπλοκο από την τρίλιζα;
- Οι επικαλύψεις του Escher είναι πιο πολύπλοκες από τα τετράγωνα πλακάκια του μπάνιου;
- Οι πρώτοι αριθμοί είναι πιο πολύπλοκοι από περιττούς (υπολογιστική πολυπλοκότητα);





Τι είναι Υπολογιστική Πολυπλοκότητα;

- Για τα προβλήματα που επιλύονται (solvable, computable, decidable) μας ενδιαφέρει να υπολογίσουμε το πόσο καλά μπορεί να γίνει αυτό, δηλαδή
  - πόσο γρήγορα, ή
  - με πόση μνήμη, ή
  - με πόσους επεξεργαστές (παραλληλία), ή
  - με πόση κατανάλωση ενέργειας (sensor networks), κ.λπ.

- Αυτό λέγεται Υπολογιστική Πολυπλοκότητα ή, απλά, Πολυπλοκότητα του προβλήματος !!!

Τι είναι Υπολογιστική Πολυπλοκότητα;

Η ***δυσκολία*** της αλγοριθμικής επίλυσης ενός προβλήματος Π

- Η *δυσκολία επίλυσης* του προβλήματος Π είναι *εγγενής* και είναι συνάρτηση του μεγέθους της εισόδου του:

$$\textit{Δυσκολία Επίλυσης Π} = f(\textit{Μέγεθος Εισόδου})$$

- *Μέγεθος Εισόδου* = **n**

$f(\mathbf{n}) = \mathbf{n}$ **Γραμμική** δυσκολία επίλυσης Π

$f(\mathbf{n}) = \mathbf{n^2}$ **Τετραγωνική** >>

$f(\mathbf{n}) = \mathbf{2^n}$ **Εκθετική** >>

Τι είναι Υπολογιστική Πολυπλοκότητα;

Η ***δυσκολία*** της αλγοριθμικής επίλυσης ενός προβλήματος Π

- Η *υπολογιστική πολυπλοκότητα* παρουσιάζει ομοιότητες με την *ανάλυση αλγορίθμων* η οποία μελετά:
  - ✓ *την δυσκολία εκτέλεσης* ενός αλγορίθμου Α που επιλύει το Π

$$\textit{Δυσκολία Εκτέλεσης A} = f(\textit{Μέγεθος Εισόδου})$$

- *Μέγεθος Εισόδου* = **n**

$f(\mathbf{n}) = \mathbf{n}$ **Γραμμική** δυσκολία εκτέλεσης Α

$f(\mathbf{n}) = \mathbf{n^2}$ **Τετραγωνική** >>

$f(\mathbf{n}) = \mathbf{2^n}$ **Εκθετική** >>

Υπολογιστική Πολυπλοκότητα

- ***Δυσκολία Επίλυσης Προβλήματος*** $= f($***Μέγεθος Εισόδου***$)$
- ***Δυσκολία Εκτέλεσης Αλγόριθμου*** $= f($***Μέγεθος Εισόδου***$)$
- ***Δυσκολία = Πολυπλοκότητα*** *(χρόνου, μνήμης, επεξεργαστών,…)*

Σήμερα μας ενδιαφέρει κυρίως ο χρόνος !!!

Υπολογιστική Πολυπλοκότητα

- ***Πολυπλοκότητα Προβλήματος*** $= f(\textit{Μέγεθος Εισόδου})$

- ***Πολυπλοκότητα Αλγόριθμου*** $= f(\textit{Μέγεθος Εισόδου})$

Πρόβλημα Π

Πολυπλοκότητα Αλγόριθμου

↓

Πολυπλοκότητα Προβλήματος

Υπολογιστική Πολυπλοκότητα

- ***Πολυπλοκότητα Προβλήματος*** $= f(\textit{Μέγεθος Εισόδου})$
- ***Πολυπλοκότητα Αλγόριθμου*** $= f(\textit{Μέγεθος Εισόδου})$

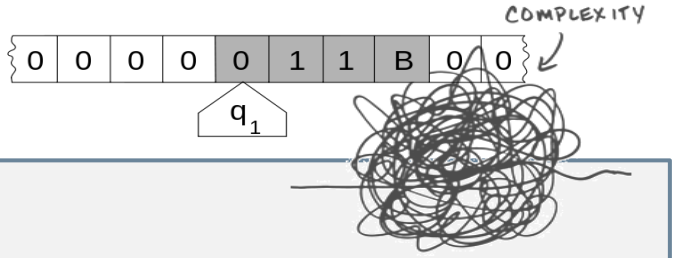
Πολυπλοκότητα Προβλήματος

vs

Πολυπλοκότητα Αλγόριθμου

Υπολογισιμότητα & Πολυπλοκότητα

Τι πρέπει να μας μείνει !!!



- Υπολογισιμότητα
  - ***ποια*** υπολογιστικά προβλήματα μπορούμε να λύσουμε με υπολογιστή και ποια όχι;
- Πολυπλοκότητα
  - ***ποια*** μπορούμε να λύσουμε ***γρήγορα*** και ποια όχι;
  - ***πόσο γρήγορα*** μπορούμε να τα λύσουμε;
- Πολυπλοκότητα Προβλήματος ≠ Πολυπλοκότητα Αλγόριθμου

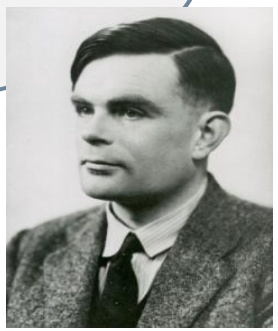
Ερωτήματα – Προβληματισμοί (II)

Αποτελέσματα - Σταθμοί

David Hilbert (1900)

Kurt Gödel (1931)

. . .



Alan Turing (1936)

Ορίζει την έννοια του Υπολογιστή !!!

Αποτελέσματα - Σταθμοί

- **David Hilbert (1900)**:

  Ρωτάει αν μπορούν να αυτοματοποιηθούν τα μαθηματικά;

  δηλαδή, μπορεί να βρει αλγόριθμος που να αποκρίνεται (*decides) για την ορ*θότητα κάθε μαθηματικής πρότασης;

- **Kurt Gödel (1931)**:

  Δείχνει ότι αυτό δεν γίνεται με το περίφημο Θεώρημα της μη-πληρότητας !

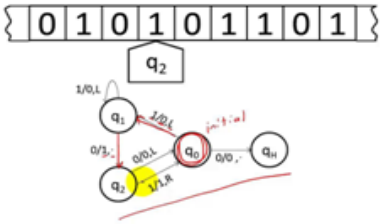
Αποτελέσματα - Σταθμοί

- **Alan Turing (1936)**:

  Ορίζει την έννοια του υπολογιστή και δείχνει ότι:

  δεν μπορούν όλα τα υπολογιστικά προβλήματα να λυθούν αλγοριθμικά !!!

  - Πρόβλημα Τερματισμού
    (Halting Problem)



Αποτελέσματα - Σταθμοί

- **Cook (1971), Karp (1972):**

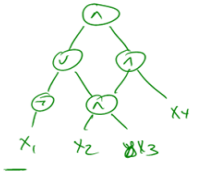
  Από αυτά που επιλύονται, *πολλά δεν μπορούν να λυθούν αποτελεσματικά* !!!

  - C-SAT
  - Traveling Salesman Problem (TSP)
  - Κύκλος Hamilton (HC)

Κατηγοριοποίηση Προβλημάτων

Cook's Theorem

The Circuit Satisfiability Problem is NP- Complete



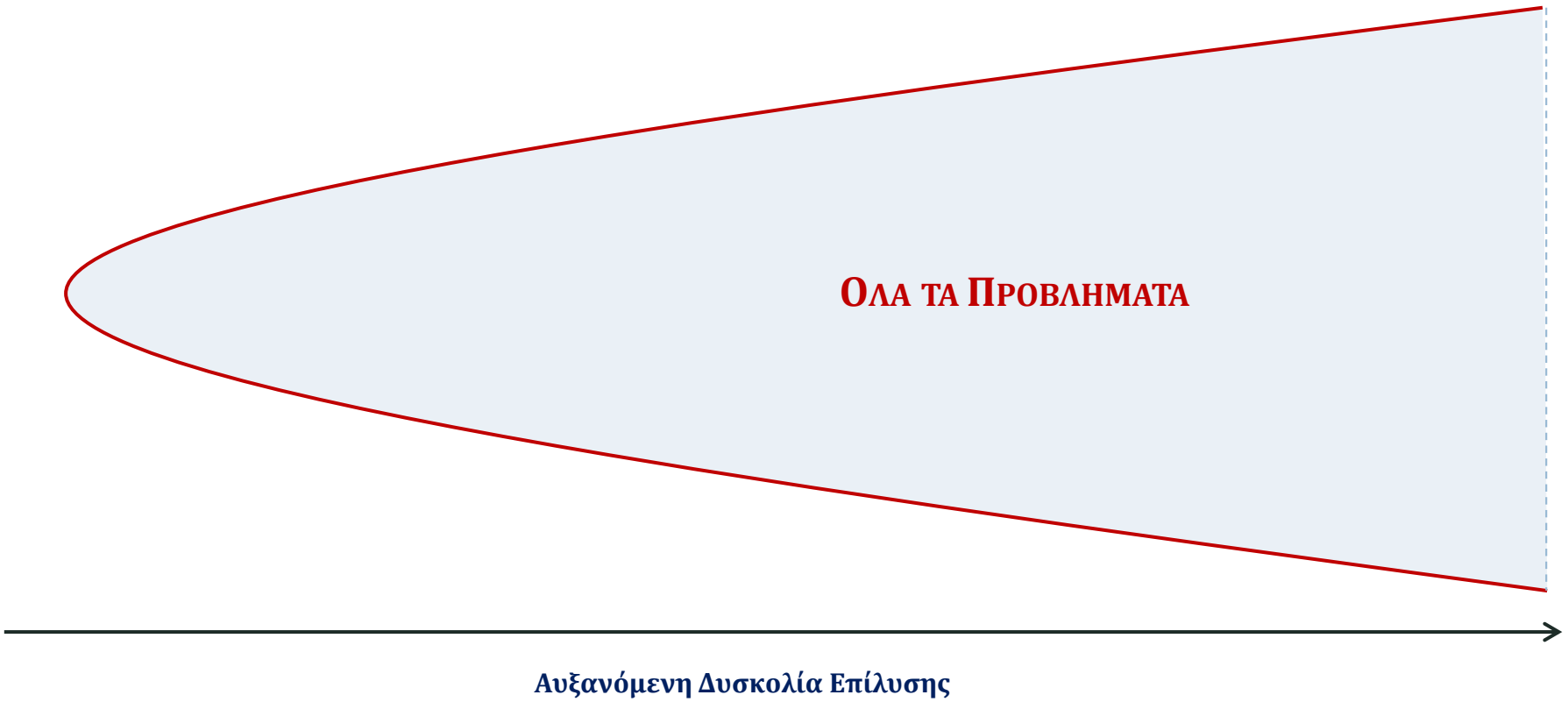
List of 21 Problems that are NP-complete

Richard Karp, 1972

- CLIQUE
- SET PACKING
- **VERTEX COVER**
- **SET COVERING**
- FEEDBACK NODE SET
- FEEDBACK ARC SET
- KNAPSACK
- PARTITION
- MAX-CUT

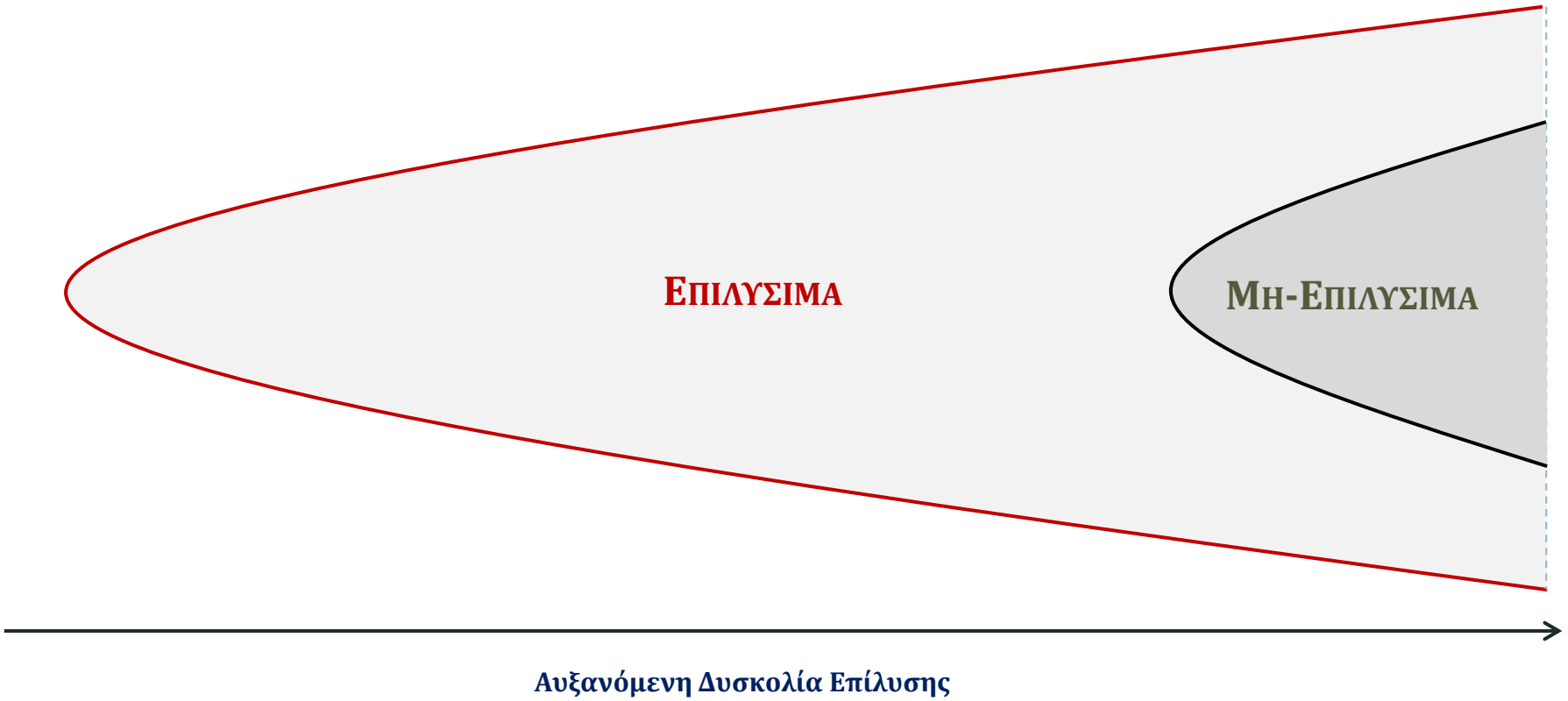
Κατηγοριοποίηση Υπολογιστικών Προβλημάτων

Χρόνος Επίλυσης



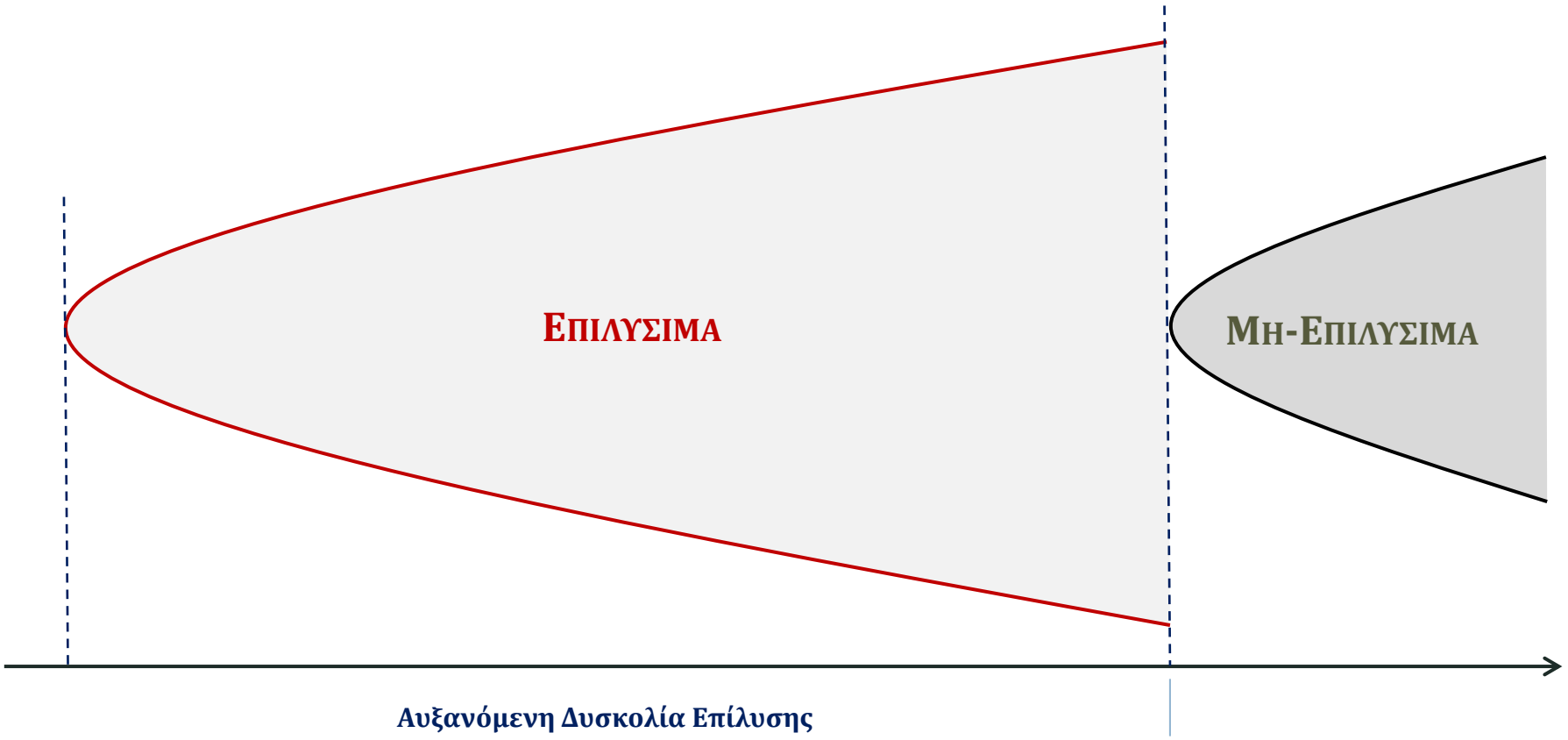
Κατηγοριοποίηση Υπολογιστικών Προβλημάτων

Χρόνος Επίλυσης



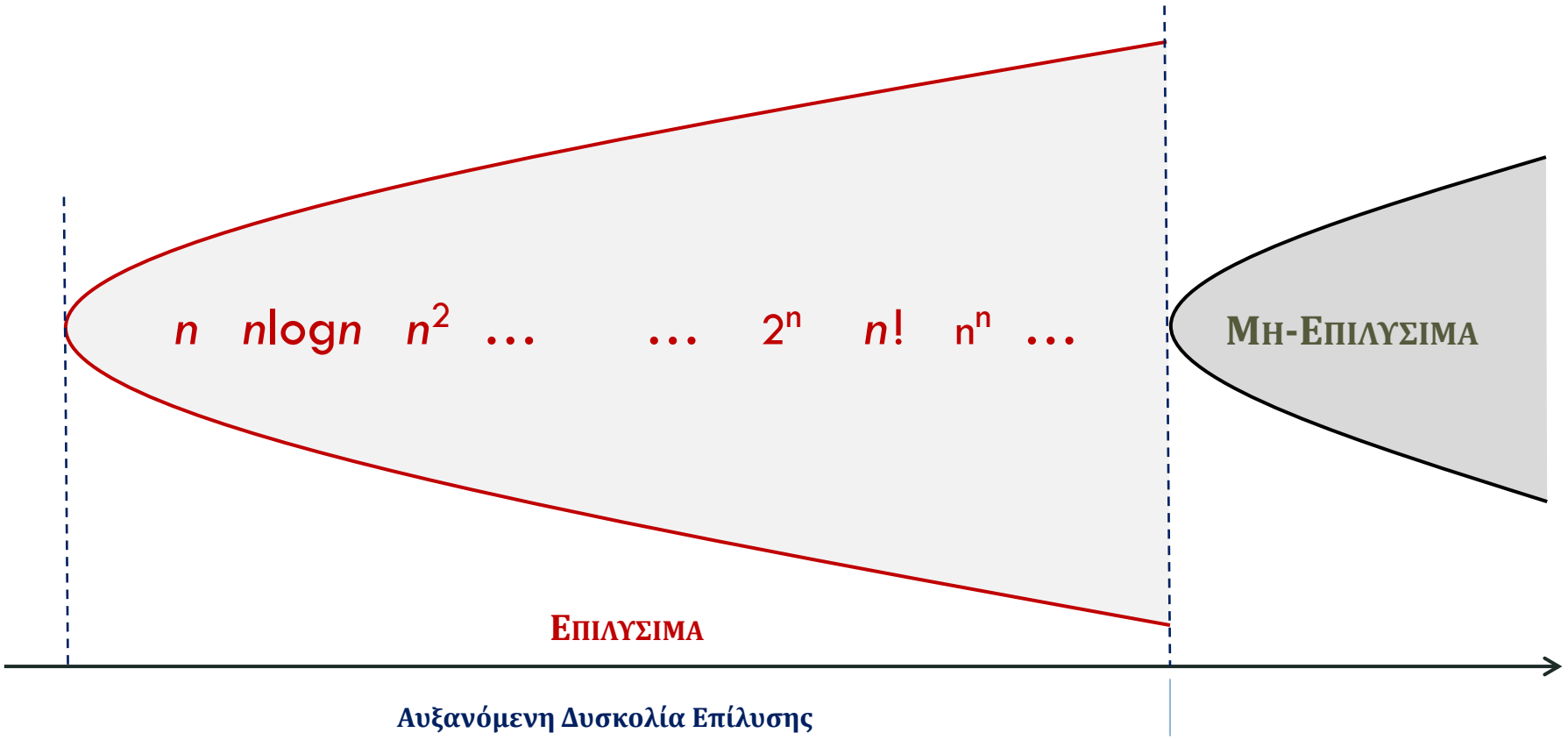
Κατηγοριοποίηση Υπολογιστικών Προβλημάτων

Χρόνος Επίλυσης



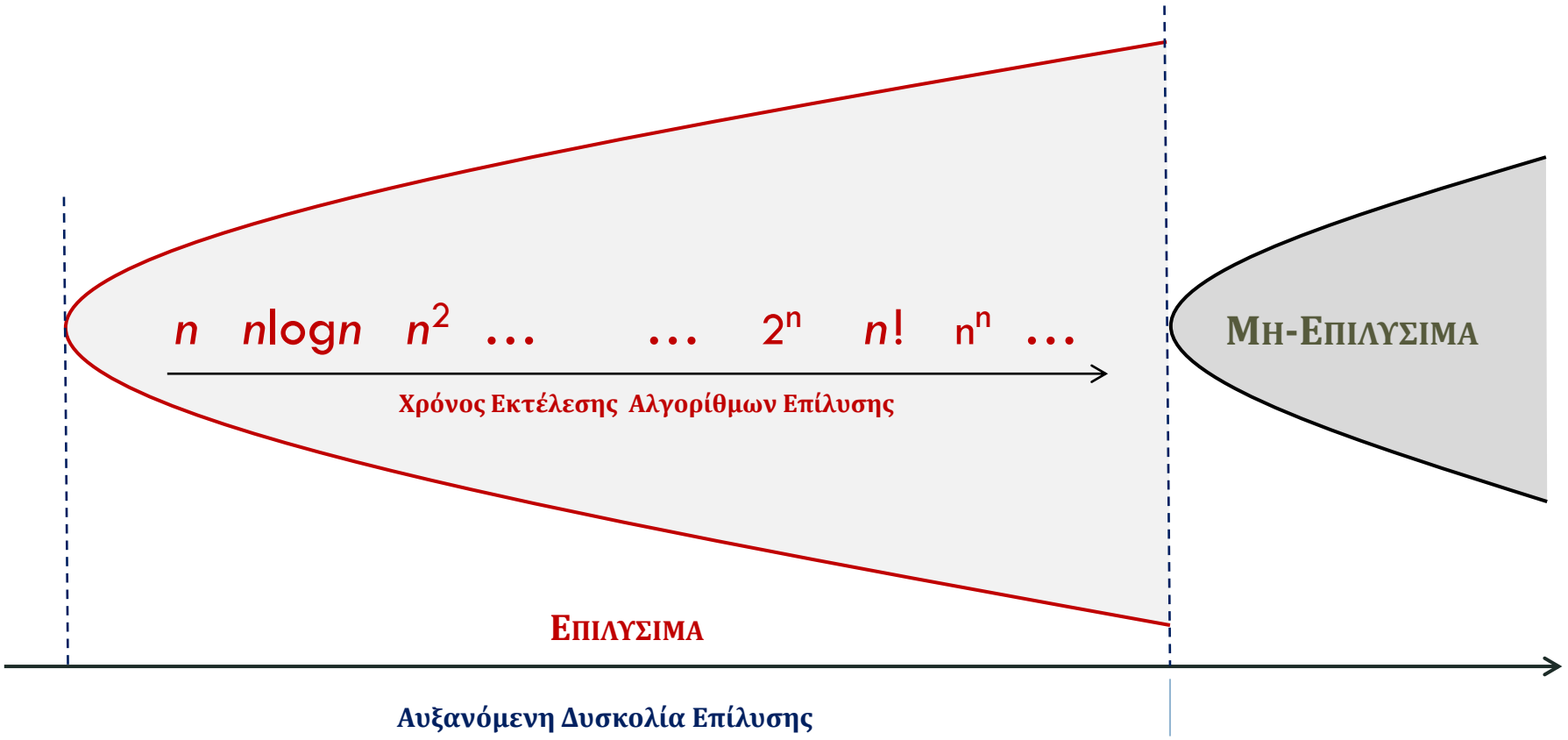
Κατηγοριοποίηση Υπολογιστικών Προβλημάτων

Χρόνος Επίλυσης



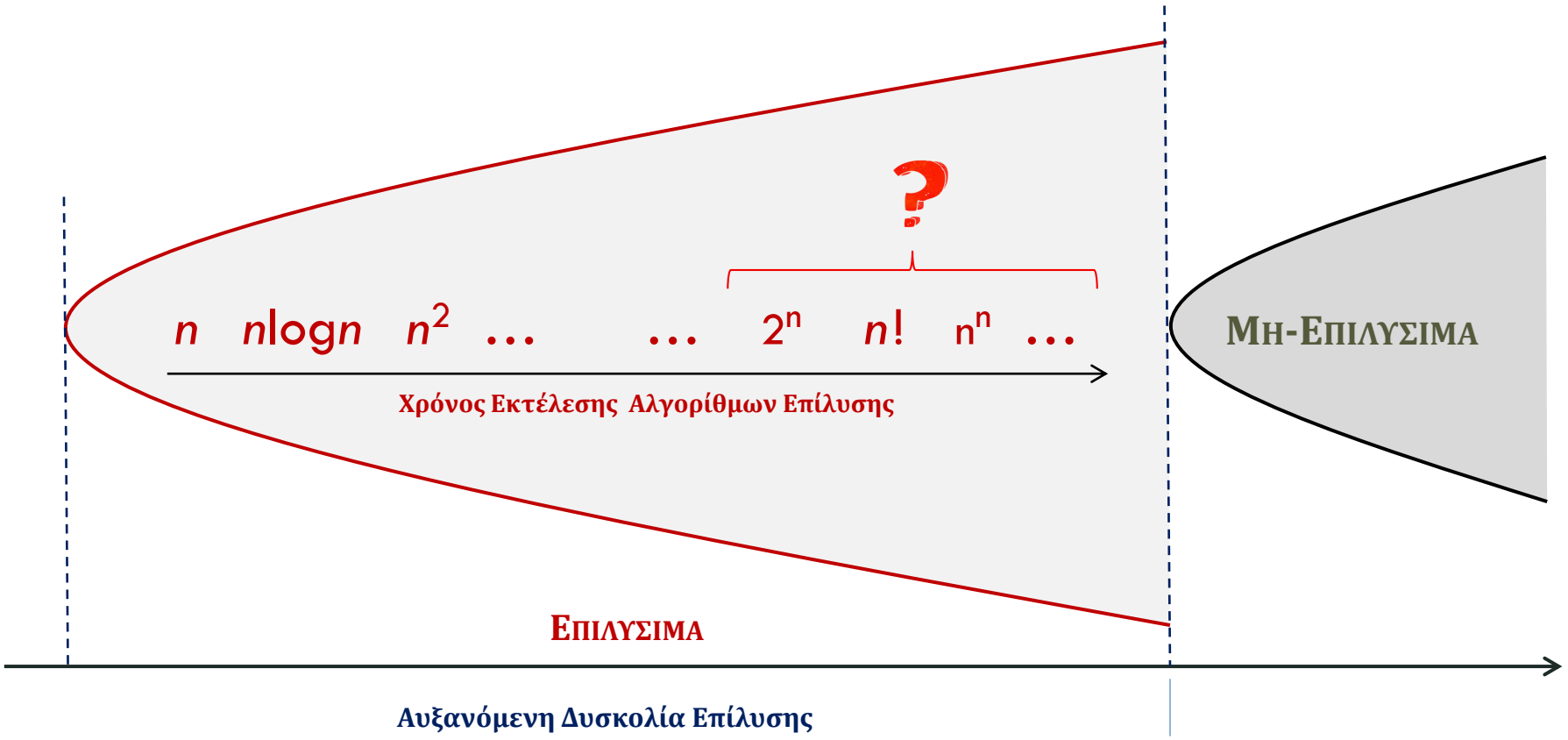
Κατηγοριοποίηση Υπολογιστικών Προβλημάτων

Χρόνος Επίλυσης



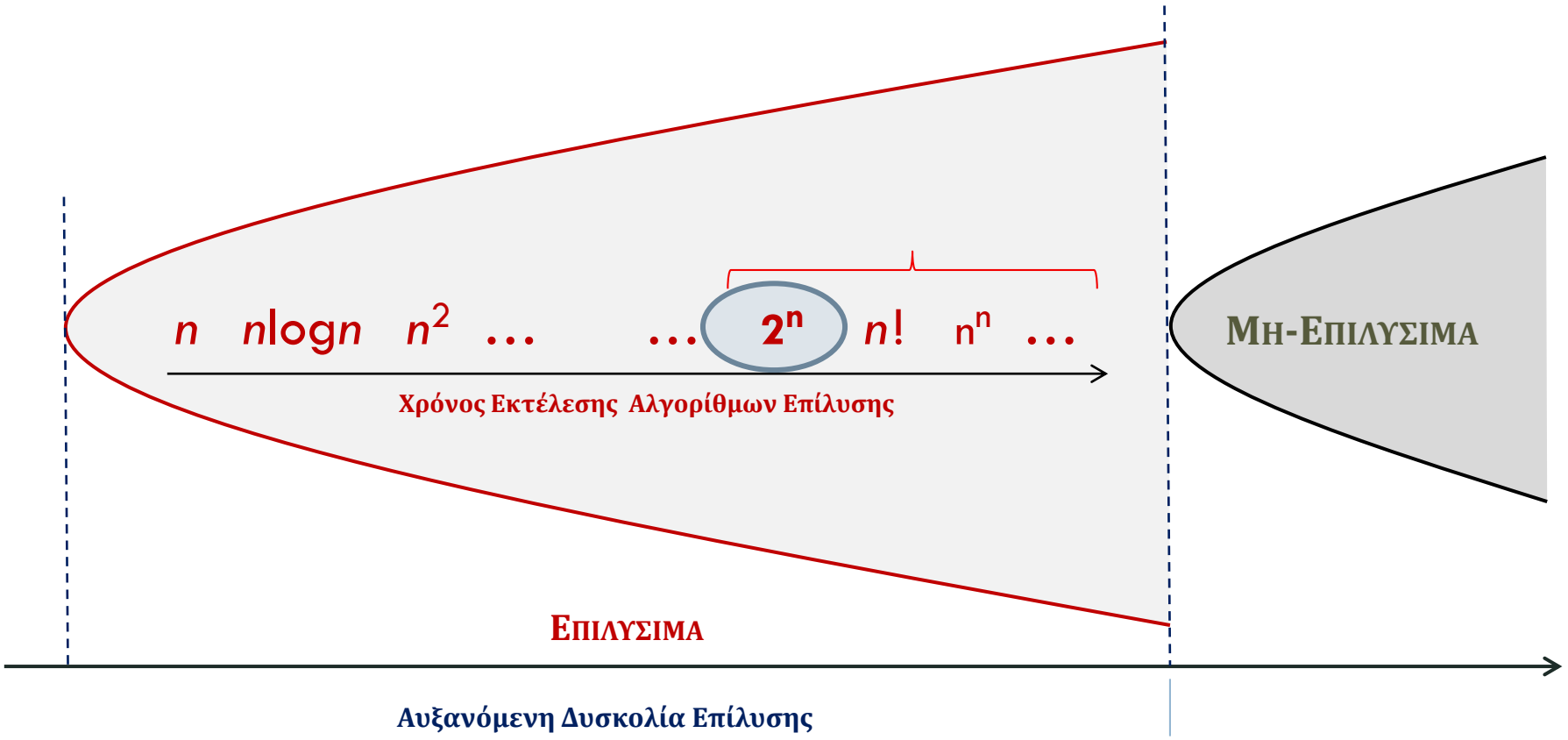
Κατηγοριοποίηση Υπολογιστικών Προβλημάτων

Χρόνος Επίλυσης



Κατηγοριοποίηση Υπολογιστικών Προβλημάτων

Χρόνος Επίλυσης $f(n) = 2^n$?



Ερωτήματα – Προβληματισμοί (III)

Χρόνος Επίλυσης $f(n) = 2^n$

Αλγόριθμος Fibonacci

$$F_n = \begin{cases} 0 & \text{εάν } n = 0 \\ 1 & \text{εάν } n = 1 \\ F_{n-1} + F_{n-2} & \text{εάν } n > 1 \end{cases}$$

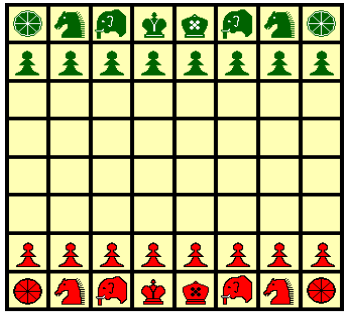
```
Fib(n)
    if n = 0 then return 0
    if n = 1 then return 1
    return Fib(n-1) + Fib(n-2)
```

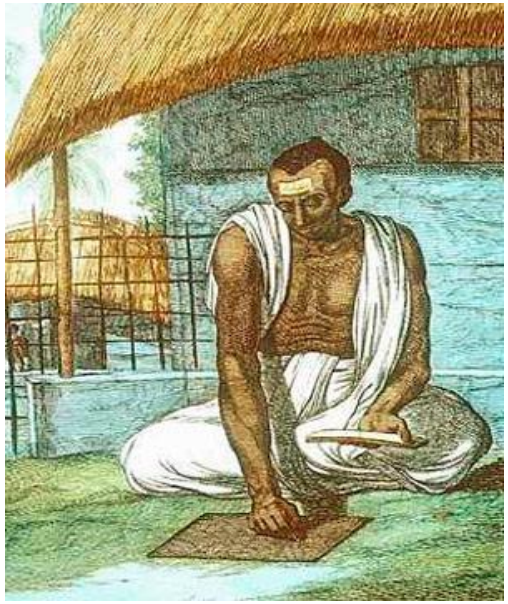
$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$ $\quad T(n) \geq F_n \quad$ *και* $\quad F_n \approx 2^{0.694n}$

Χρόνος Επίλυσης $f(n) = 2^n$

Ιστορία του Sissa

$$2^{64} - 1$$





Χρόνος Επίλυσης $f(n) = 2^n$

Ιστορία του Sissa

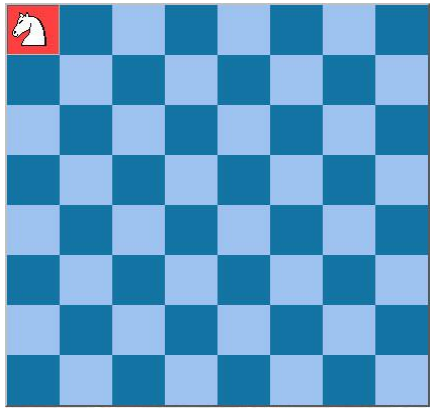
- Σύμφωνα με την παράδοση, κάποτε ο ηγεμόνας της περιοχής που ζούσε ο βραχμάνος Σίσσα, τον κάλεσε να του δείξει το παιγνίδι που είχε εφεύρει.
- Αυτό δεν ήταν άλλο από το Σκάκι !!!
- Τόσο πολύ γοητεύτηκε από το παιχνίδι αυτό ο ηγεμόνας που ρώτησε τον Σίσσα τι θα ήθελε ως ανταμοιβή !...

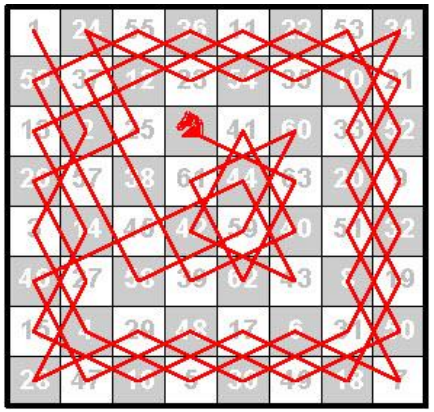
Και τότε ο σοφός Σίσσα ζήτησε...

Χρόνος Επίλυσης $f(n) = 2^n$

Ιστορία του Sissa

- Ζήτησε να του δοθεί ρύζι για την τροφή του, και την ποσότητα του ρυζιού να την καθορίσει μόνος του ο ηγεμόνας, εάν συμφωνεί και αυτός, ως εξής:





Να θέσει ένα κόκκο ρυζιού στο τετράγωνο που βρίσκεται αρχικά ο ίππος και να διπλασιάζει τους κόκκους σε κάθε νέο τετράγωνο που θα μπορεί να βρεθεί ο ίππος !!!

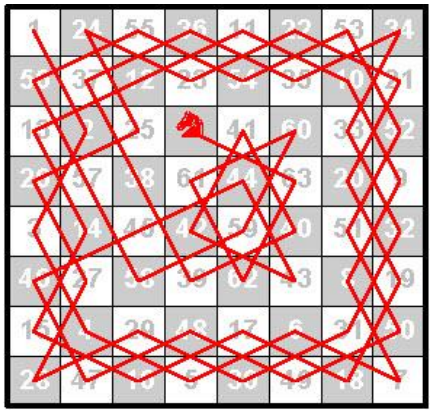
Χρόνος Επίλυσης $f(n) = 2^n$

Ιστορία του Sissa

- Τι συνέβη …

$$2^{64} - 1 = ?$$

$$2^{64} - 1 = 18446744073709551615$$



Οι 18.446.744.073.709.551.615 κόκκοι ρυζιού που θα έπαιρνε ο Σίσσα, έχοντας ότι το βάρος ενός κόκκου ισούται με 0,053 gr, ισοδυναμούσαν στη ποσότητα των **977.677.436.907 τόνων**!

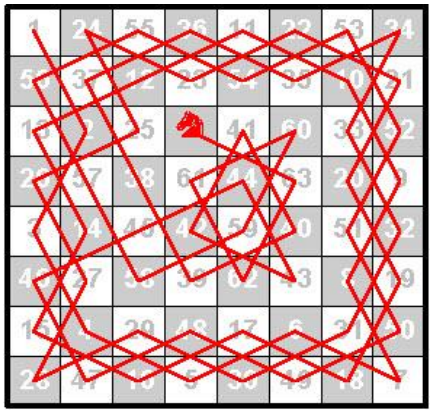
Χρόνος Επίλυσης $f(n) = 2^n$

Ιστορία του Sissa

- Τι συνέβη …

$$2^{64} - 1 = ?$$

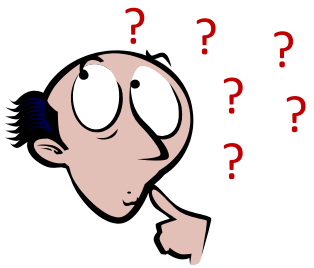
$$2^{64} - 1 = 18446744073709551615$$



Ο Σίσσα πρέπει να γνώριζε για **εκθετικές συναρτήσεις** και για **διαδρομές Hammilton**!!!

Χρόνος Επίλυσης $f(n) = 2^n$

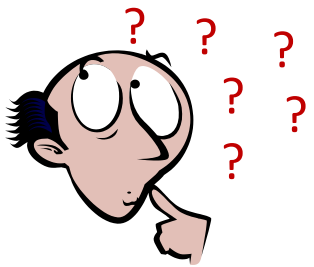
log*n*	*n*	n^2	2^n
3.322	10	100	1024
6.644	100	10000	1267650600228229401496703205376
9.966	1000	1000000	$(1267650600228229401496703205376)^{10}$



Πόσο μεγάλος είναι ο αριθμός 2^{1000}

Χρόνος Επίλυσης $f(n) = 2^n$ και $f(n) = n!$

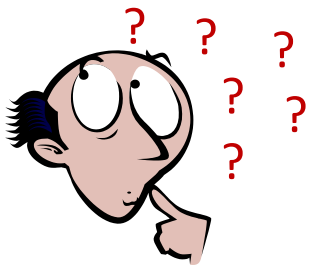
Πρόβλημα Π



Πολυπλοκότητα Χρόνου \ Μέγεθος Εισόδου n	2	8	32	64
$\log n$				
n	2 δευτερα	8 δευτερα	32 δευτερα	1.07 λεπτά
$n \log n$				
n^2				
...				
2^n				
$n!$				

Χρόνος Επίλυσης $f(n) = 2^n$ και $f(n) = n!$

Πρόβλημα Π



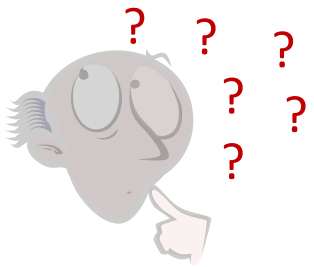
Μέγεθος Εισόδου n

Πολυπλοκότητα Χρόνου

	2	8	32	64
$\log n$	1 δεύτερο	3 δεύτερα	5 δεύτερα	6 δεύτερα
n	2 δεύτερα	8 δεύτερα	32 δεύτερα	1.07 λεπτά
$n \log n$	2 δεύτερα	24 δεύτερα	2.67 λεπτά	6.4 λεπτά
n^2	4 δεύτερα	1.07 λεπτά	17.07 λεπτά	1.14 ώρες
...				
2^n				
$n!$				

Χρόνος Επίλυσης $f(n) = 2^n$ και $f(n) = n!$

Πρόβλημα Π



Μέγεθος Εισόδου n

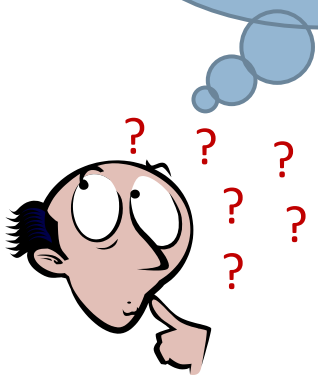
Πολυπλοκότητα Χρόνου

	2	8	32	64
$\log n$	1 δεύτερο	3 δεύτερα	5 δεύτερα	6 δεύτερα
n	2 δεύτερα	8 δεύτερα	32 δεύτερα	1.07 λεπτά
$n \log n$	2 δεύτερα	24 δεύτερα	2.67 λεπτά	6.4 λεπτά
n^2	4 δεύτερα	1.07 λεπτά	17.07 λεπτά	1.14 ώρες
...	...	...	...	...
2^n	4 δεύτερα	4.27 λεπτά	1.36 αιώνες	5.86×10^9 αιώνες
$n!$	2 δεύτερα	4.2 ώρες	8.34×10^{25} αιώνες	4.02×10^{79} αιώνες

Χρόνος Επίλυσης $f(n) = 2^n$ και $f(n) = n!$

Πρόβλημα Π

Ο αριθμός των *ατόμων* στο ορατό *Σύμπαν* υπολογίζεται σε περίπου 10^{80}



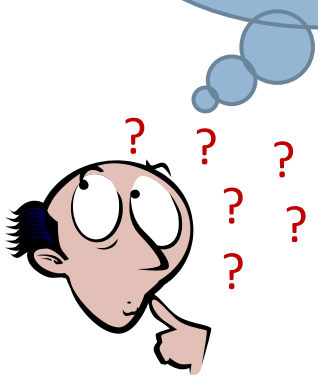
Μέγεθος Εισόδου *n*

Πολυπλοκότ	8	32	64
	3 δεύτερα	**5** δεύτερα	**6** δεύτερα
	8 δεύτερα	**32** δεύτερα	**1.07** λεπτά
	24 δεύτερα	**2.67** λεπτά	**6.4** λεπτά
	4 δεύτερα	**1.07** λεπτά	**17.07** λεπτά	**1.14** ώρες
…	…	…	…	…
2^n	**4** δεύτερα	**4.27** λεπτά	**1.36** αιώνες	**5.86 × 10^9** αιώνες
$n!$	**2** δεύτερα	**4.2** ώρες	**8.34 × 10^{25}** αιώνες	**4.02 × 10^{79}** αιώνες

Χρόνος Επίλυσης $f(n) = 2^n$ και $f(n) = n!$

Πρόβλημα Π

Τώρα Γνωρίζω !!!

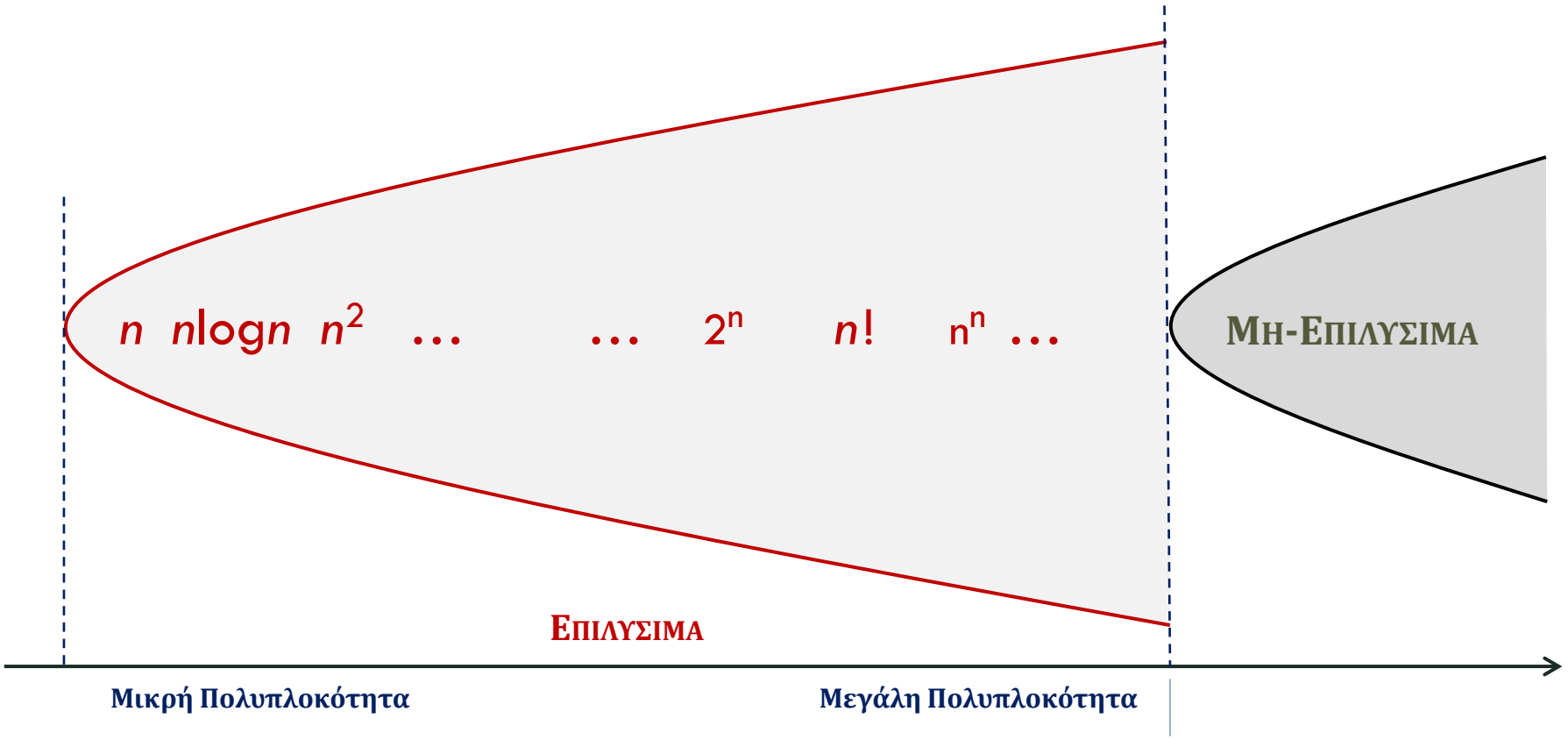


Πολυπλοκότ

	Μέγεθος Εισόδου *n*			
	8	32	64
	3 δεύτερα	**5** δεύτερα	**6** δεύτερα
	8 δεύτερα	**32** δεύτερα	**1.07** λεπτά
	24 δεύτερα	**2.67** λεπτά	**6.4** λεπτά
	4 δεύτερα	**1.07** λεπτά	**17.07** λεπτά	**1.14** ώρες
…	…	…	…	…
2^n	**4** δεύτερα	**4.27** λεπτά	**1.36** αιώνες	**5.86 × 10^9** αιώνες
$n!$	**2** δεύτερα	**4.2** ώρες	**8.34 × 10^{25}** αιώνες	**4.02 × 10^{79}** αιώνες

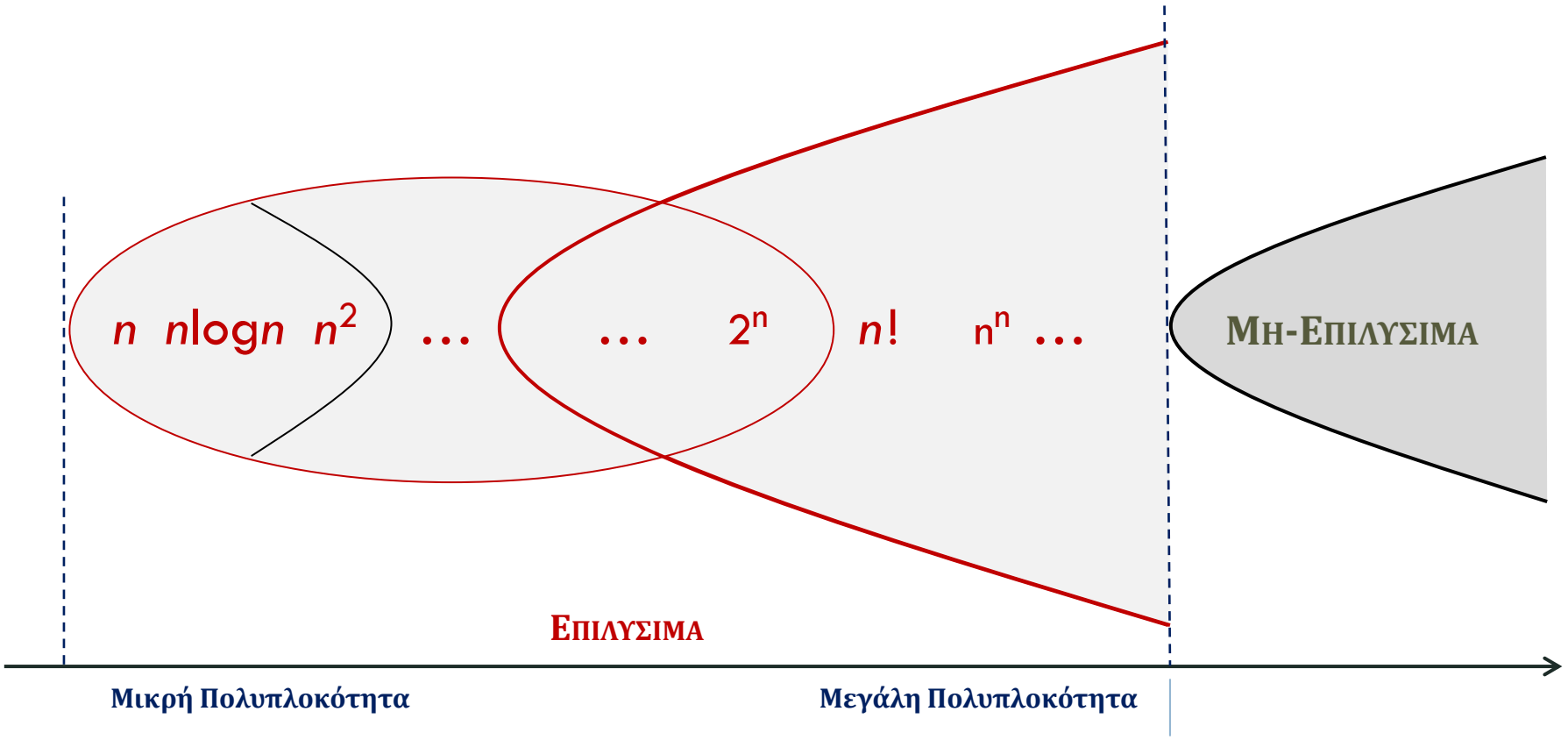
Χρόνος Επίλυσης Προβλημάτων

Πολυπλοκότητα - Κλάσεις Προβλημάτων



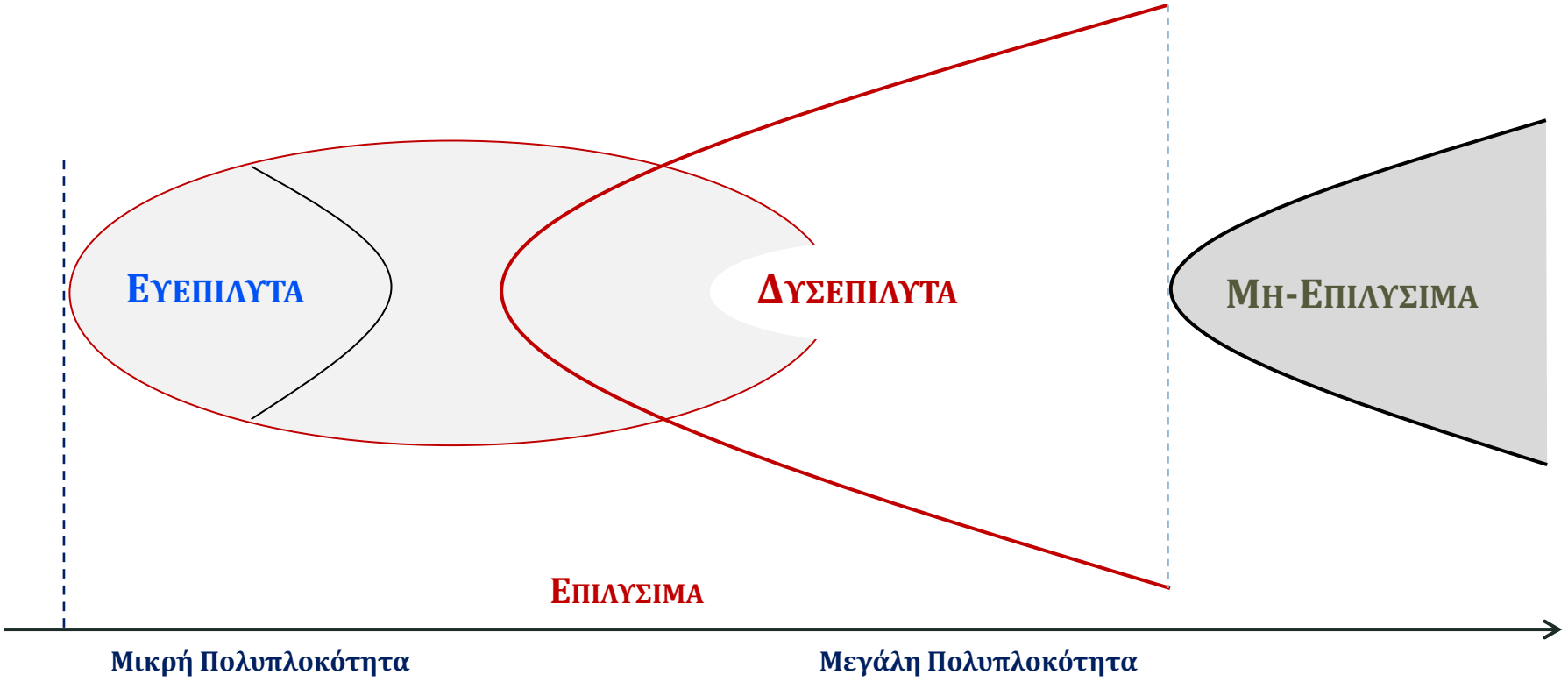
Χρόνος Επίλυσης Προβλημάτων

Πολυπλοκότητα - Κλάσεις Προβλημάτων



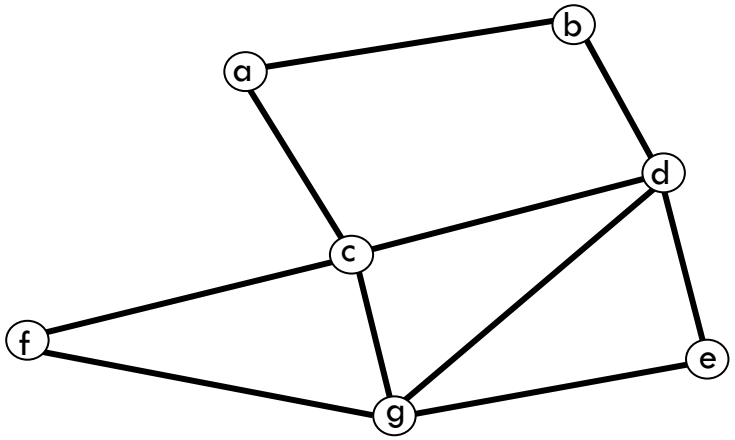
Χρόνος Επίλυσης Προβλημάτων

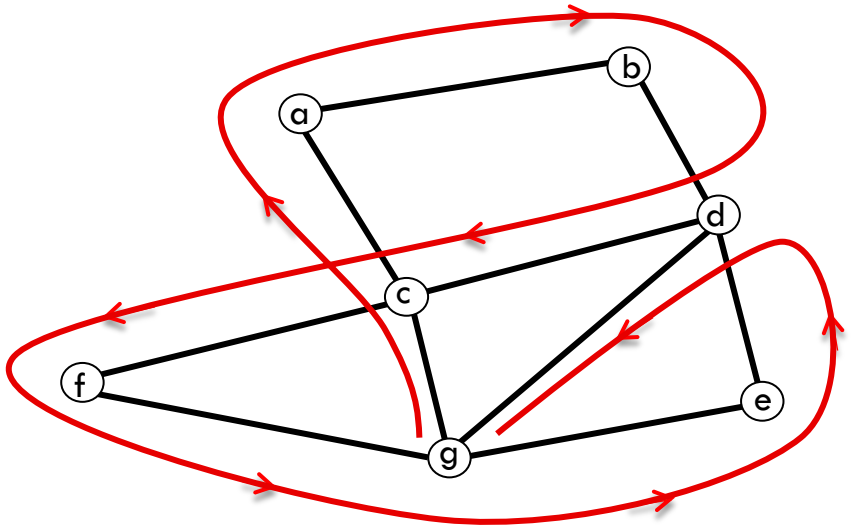
Πολυπλοκότητα - Κλάσεις Προβλημάτων



Πρόβλημα Euler (Leonard Euler 1937)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε ακμή ακριβώς μια φορά;



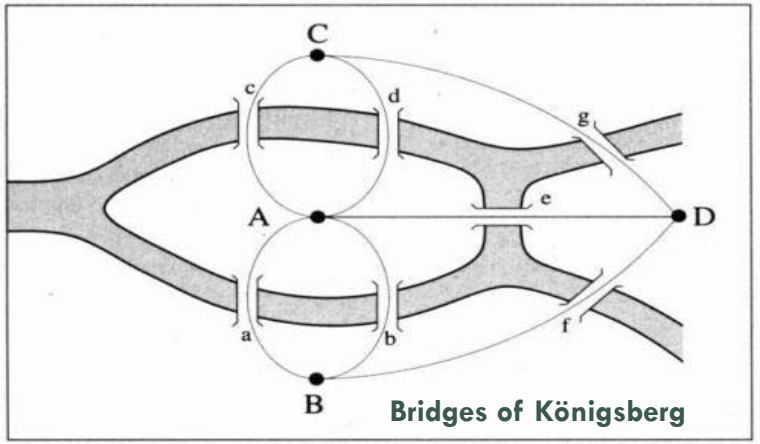


Πρόβλημα Euler (Leonard Euler 1937)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε ακμή ακριβώς μια φορά;

Θεώρημα Euler:

Η απάντηση είναι «Ναι» εάν-ν *κάθε κόμβος v του γραφήματος G έχει* ***άρτιο βαθμό*** *!!!*



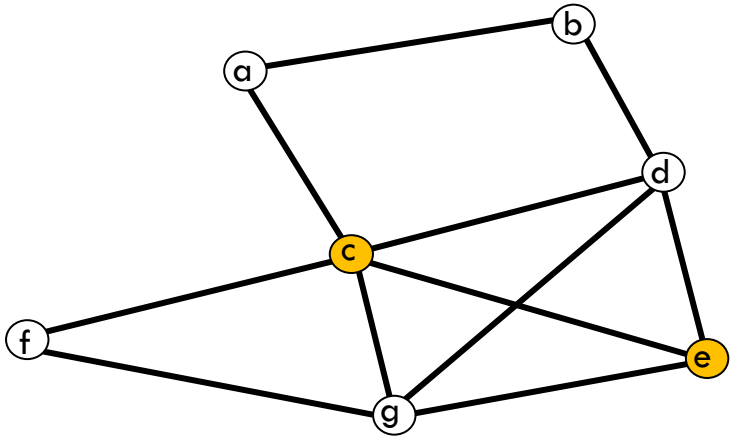
Bridges of Königsberg

Πρόβλημα Euler (Leonard Euler 1937)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε ακμή ακριβώς μια φορά;

Θεώρημα Euler:

Η απάντηση είναι «Ναι» εάν-ν *κάθε κόμβος v του γραφήματος G έχει* ***άρτιο βαθμό*** *!!!*



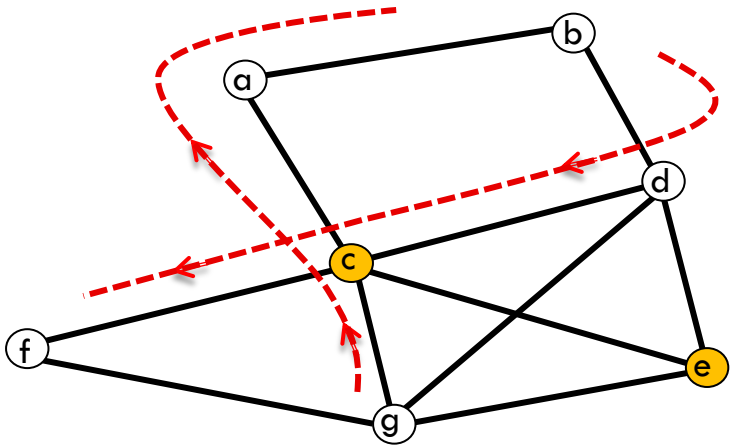
2 κόμβοι έχουν περιττό βαθμό

Πρόβλημα Euler (Leonard Euler 1937)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε ακμή ακριβώς μια φορά;

Θεώρημα Euler:

Η απάντηση είναι «Ναι» εάν-ν *κάθε κόμβος v του γραφήματος G έχει* ***άρτιο βαθμό*** *!!!*



2 κόμβοι έχουν περιττό βαθμό

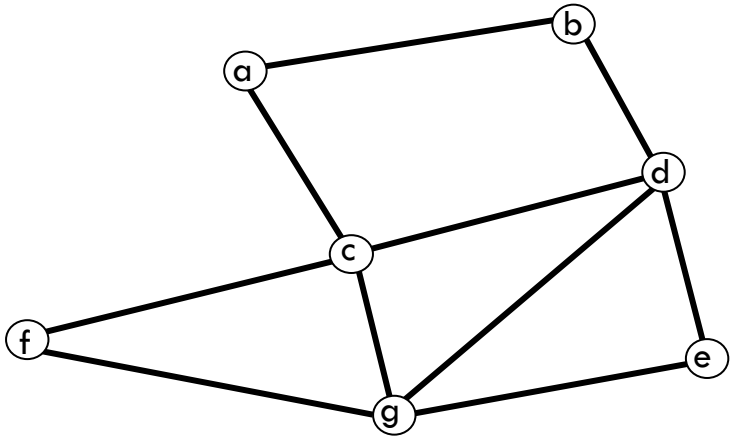
Δεν υπάρχει κύκλος Euler

Πρόβλημα Euler (Leonard Euler 1937)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε ακμή ακριβώς μια φορά;

Θεώρημα Euler:

Η απάντηση είναι «Ναι» εάν-ν *κάθε κόμβος v του γραφήματος G έχει* ***άρτιο βαθμό*** *!!!*



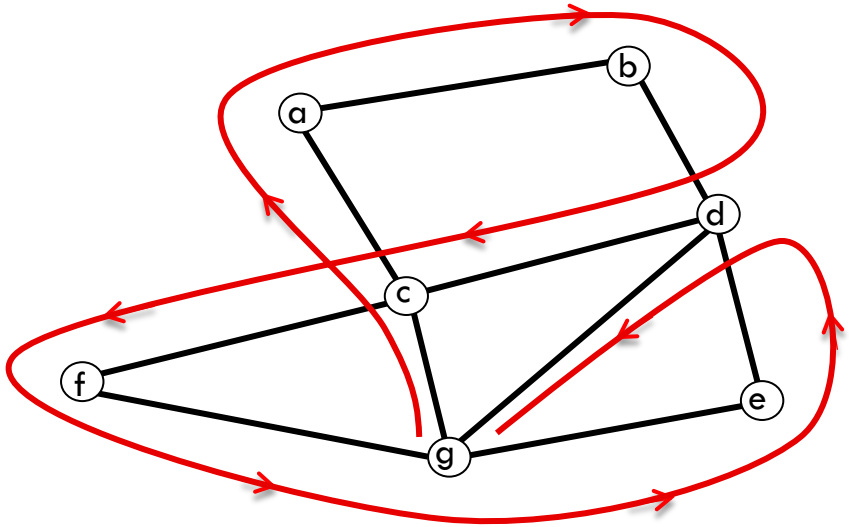
Όλοι οι κόμβοι έχουν άρτιο βαθμό

Πρόβλημα Euler (Leonard Euler 1937)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε ακμή ακριβώς μια φορά;

Θεώρημα Euler:

Η απάντηση είναι «Ναι» εάν-ν *κάθε κόμβος v του γραφήματος G έχει* ***άρτιο βαθμό*** *!!!*



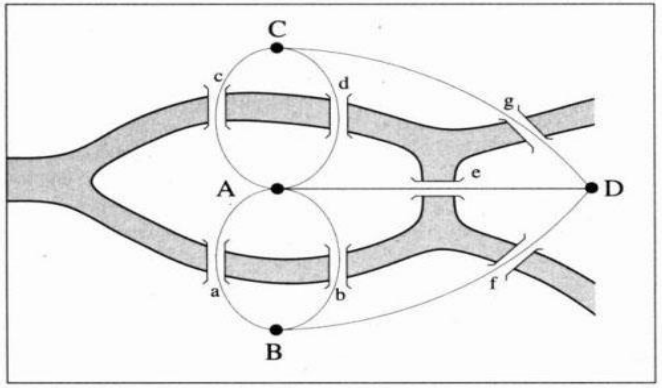
Όλοι οι κόμβοι έχουν άρτιο βαθμό

Υπάρχει κύκλος Euler

Χρόνος Επίλυσης Προβλημάτων

Πρόβλημα Euler (Leonard Euler 1937)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε ακμή ακριβώς μια φορά;



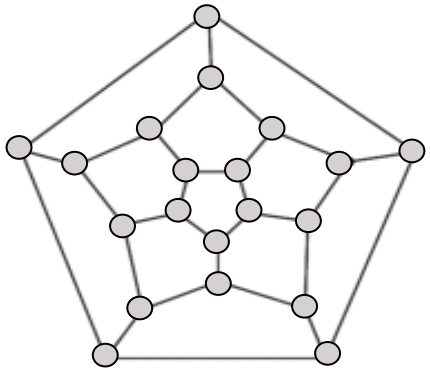
  - Το πρόβλημα είναι **ΕΥΕΠΙΛΥΤΟ** !!!

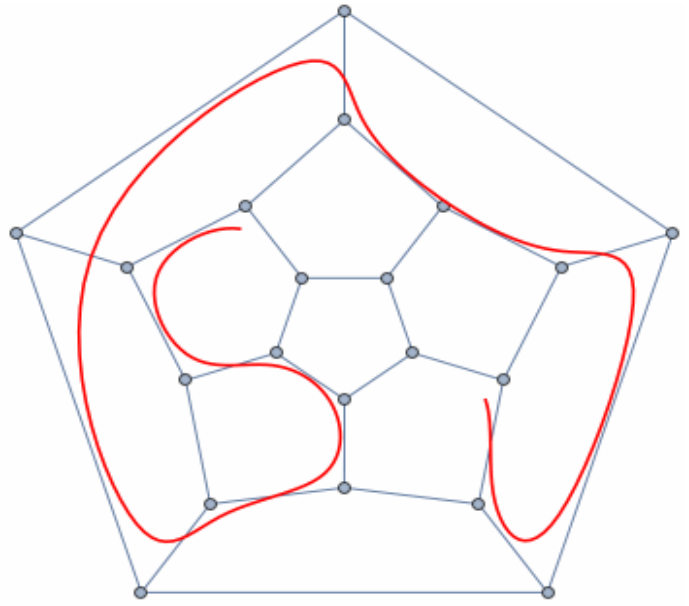
    Για κάθε G τάξης n αρκούν n^2 έλεγχοι:
    χρόνος *πολυωνυμικός* ως προς το μέγεθος της εισόδου.

Τέτοια προβλήματα που η επίλυσή τους χρειάζεται χρόνο $O(n)$, $O(n^2)$, $O(n^3)$ … λέμε ότι ανήκουν στην **κλάση P**

Πρόβλημα Hamilton (William Hamilton)

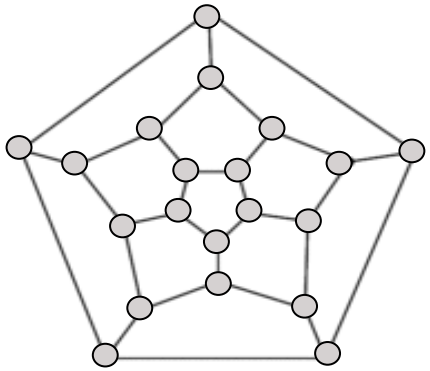
- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε κόμβο ακριβώς μια φορά;

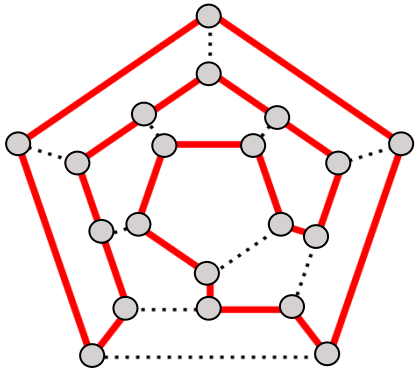




Πρόβλημα Hamilton (William Hamilton)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε κόμβο ακριβώς μια φορά;

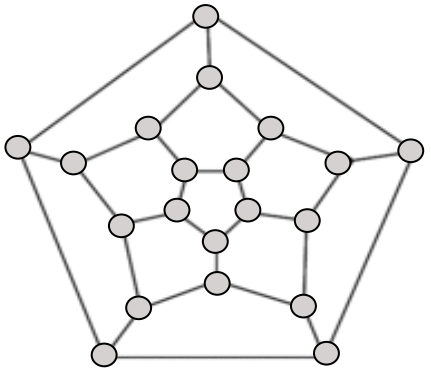




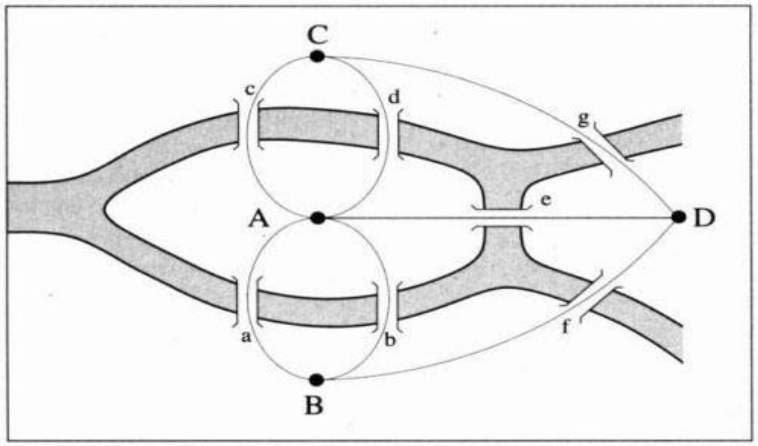
Χρόνος Επίλυσης Προβλημάτων

Πρόβλημα Hamilton (William Hamilton)

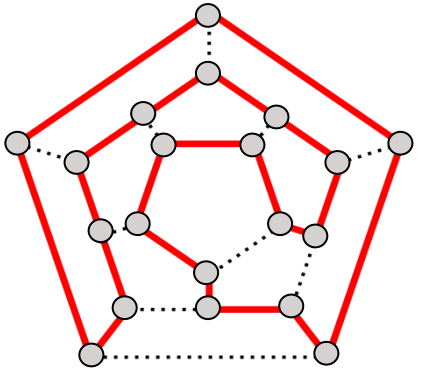
- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε κόμβο ακριβώς μια φορά;



Euler vs Hamilton

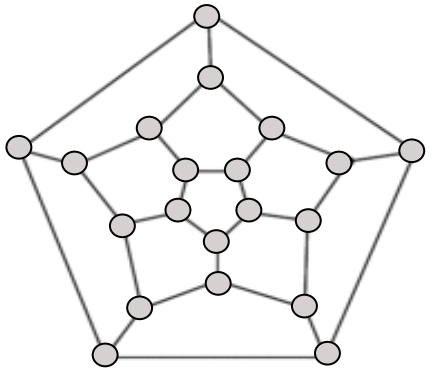


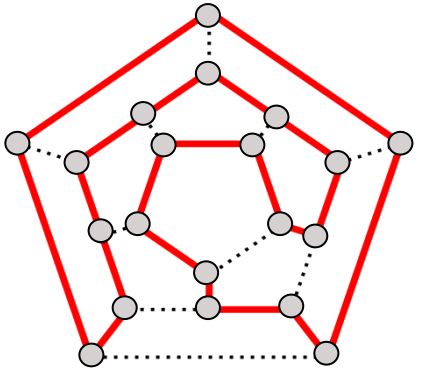
?



Πρόβλημα Hamilton (William Hamilton)

- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε κόμβο ακριβώς μια φορά;
  - Το πρόβλημα του Hamilton είναι πολύ πιο δύσκολο από αυτό του Euler.
  - Δεν γνωρίζουμε κανέναν γρήγορο αλγόριθμο!!
  - Λέμε ότι το πρόβλημα είναι **ΔΥΣΕΠΙΛΥΤΟ** !!!

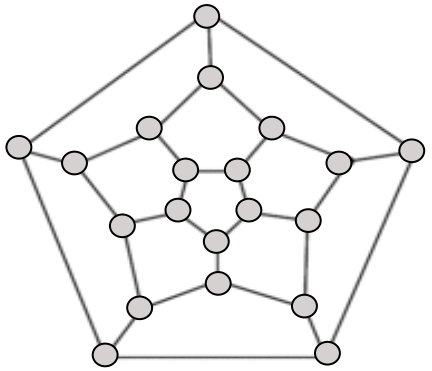




Χρόνος Επίλυσης Προβλημάτων

Πρόβλημα Hamilton (William Hamilton)

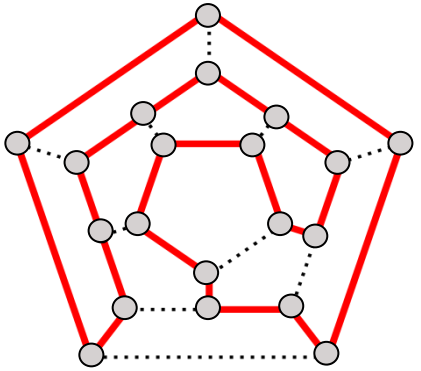
- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε κόμβο ακριβώς μια φορά;



  - Ο καλύτερος αλγόριθμος ελέγχει όλες τις κλειστές διαδρομές $(v_1, v_2, \ldots, v_n)$

Το πλήθος όλων των διαδρομών είναι ***n!***

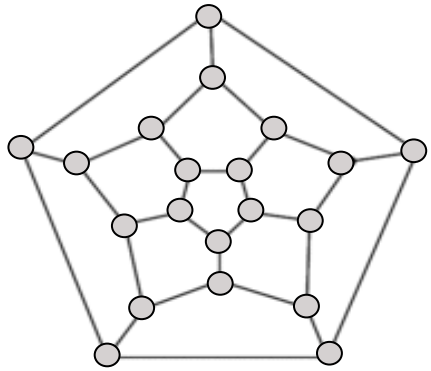
Και επομένως το πλήθος τους είναι ***εκθετικό*** ως προς το μέγεθος *n* της εισόδου !!!



Χρόνος Επίλυσης Προβλημάτων

Πρόβλημα Hamilton (William Hamilton)

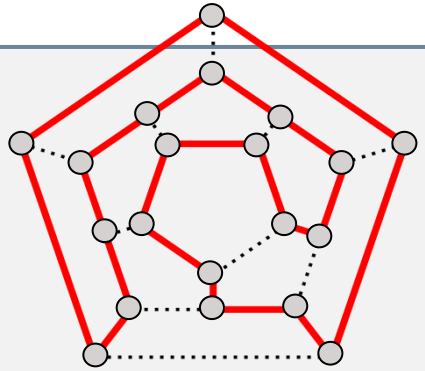
- Δίνεται γράφημα G. Υπάρχει κύκλος που περνάει από κάθε κόμβο ακριβώς μια φορά;



  - Αν όμως μας ***δώσουν μια λύση του***, μπορούμε να την επαληθεύσουμε γρήγορα, σε πολυωνυμικό χρόνο

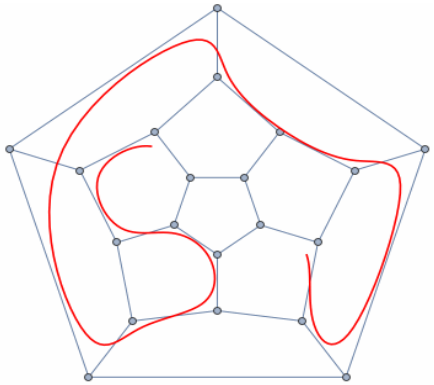
    $O(n^{\kappa})$, $\kappa \geq 1$

Προβλήματα που η **επαλήθευση μιας λύσης τους** χρειάζεται πολυωνυμικό χρόνο, λέμε ότι ανήκουν στην **κλάση NP**



Χρόνος Επίλυσης Προβλημάτων

Πρόβλημα Hamilton (William Hamilton)



- Ίσως να έχει γρήγορο αλγόριθμο επίλυσης !!!

  Κανείς όμως ως τώρα δεν έχει βρει ένα τέτοιο αλγόριθμο, αλλά ούτε και μπόρεσε να αποδείξει ότι δεν έχει.

- Το μόνο που μπορούμε να δείξουμε είναι ότι μια πλειάδα προβλημάτων είναι της ίδιας δυσκολίας με το πρόβλημα **Hamilton**.

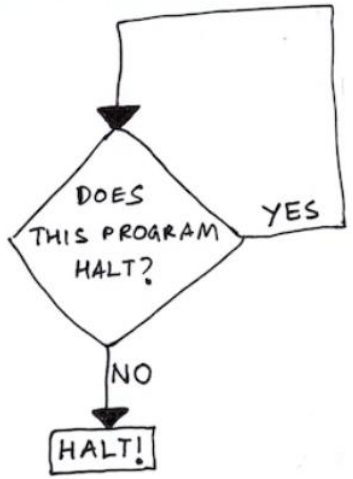
Τα προβλήματα που είναι ***το ίδιο δύσκολα με το πρόβλημα του Hamilton*** τα λέμε **NP-πλήρη** (NP-complete).

Πρόβλημα Τερματισμού (Halting Problem)

- *Δίνεται **πρόγραμμα Π** και **είσοδος x**. Τερματίζει το πρόγραμμα **Π** με είσοδο **x** (ή "τρέχει" επ' άπειρον);*

Μια ισοδύναμη παραλλαγή:

Δίνεται πρόγραμμα Π χωρίς είσοδο. Τερματίζει το πρόγραμμα Π;



Θεώρημα: Το πρόβλημα τερματισμού είναι μη επιλύσιμο, δηλαδή, δεν υπάρχει πρόγραμμα που να απαντάει σε αυτή την ερώτηση.

Πρόβλημα Τερματισμού (Halting Problem)

Απόδειξη: Έστω ότι είναι επιλύσιμο, δηλ., **υπάρχει πρόγραμμα Τ** τέτοιο ώστε: «**Τ(Π,x)** απαντά αν το **Π(x)**, δηλ. το πρόγραμμα **Π** με είσοδο **x**, **τερματίζει ή όχι**».

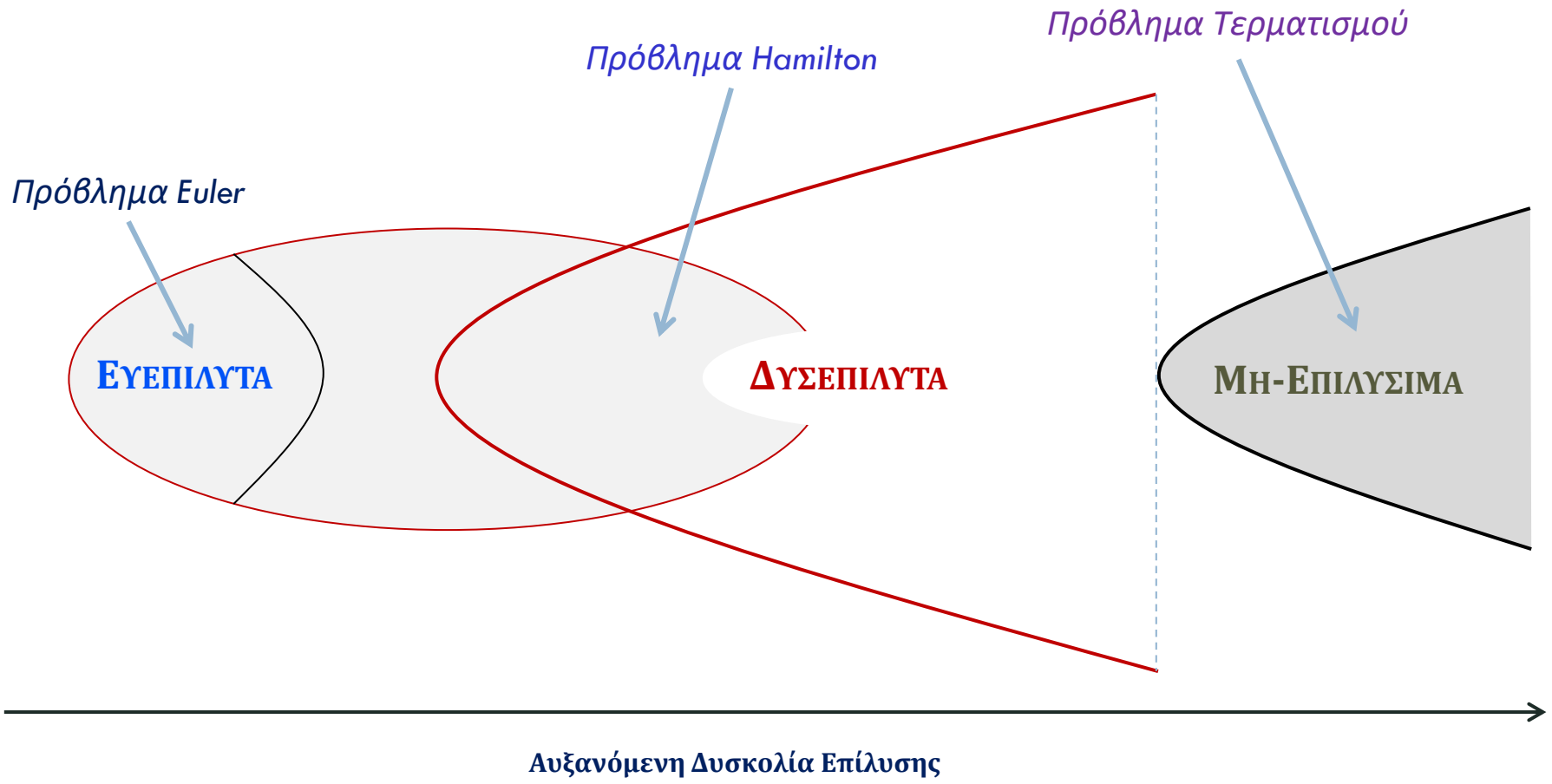
- Μπορούμε τότε να κατασκευάσουμε το πρόγραμμα S(Π):

```
S(Π)
if T(Π, Π) = true then while (true) // loop forever
                  else stop;
```

- S(Π) τερματίζει εάν-ν Π(Π) δεν τερματίζει.
- S(S) τερματίζει εάν-ν S(S) δεν τερματίζει.
- Άτοπο, **άρα το πρόγραμμα Τ δεν μπορεί να υπάρχει**!

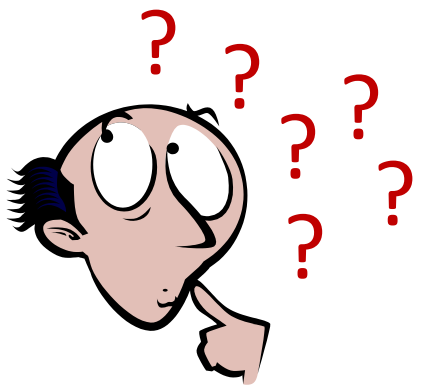
Χρόνος Επίλυσης Προβλημάτων

Κλάσεις Προβλημάτων



Ερωτήματα – Προβληματισμοί (IV)

Δύο Μεγάλα Ερωτήματα…



…τα οποία παραμένουν

Αναπάντητα έως και σήμερα !!!

Ερωτήματα – Προβληματισμοί (IV)

Το μεγάλο Ερώτημα κάποιων…

- Φοιτητών (… μιας άλλης χώρας ☺ !!!)

Ερωτήματα – Προβληματισμοί (IV)

Το μεγάλο Ερώτημα κάποιων…

- Τι είναι πιο δύσκολο; Να βρεις τις λύσεις των ασκήσεων ή να τις αντιγράψεις;

Το μεγάλο Ερώτημα της Επιστήμης μας !!!

- Πόσο πιο δύσκολο είναι να βρεις τη λύση ενός προβλήματος από το να την επαληθεύσεις;

  Είναι το ίδιο δύσκολο ή όχι !!!

million dollar question!

(Clay Institute millennium problems)

Κλάσεις Υπολογιστικών Προβλημάτων

Η Κλάση NP

- Το σύνολο των προβλημάτων που *μπορούμε να **επαληθεύσουμε** τη λύση τους* (αν μας δοθεί) σε πολυωνυμικό χρόνο.

P ⊆ NP

Προφανώς Ισχύει !!!

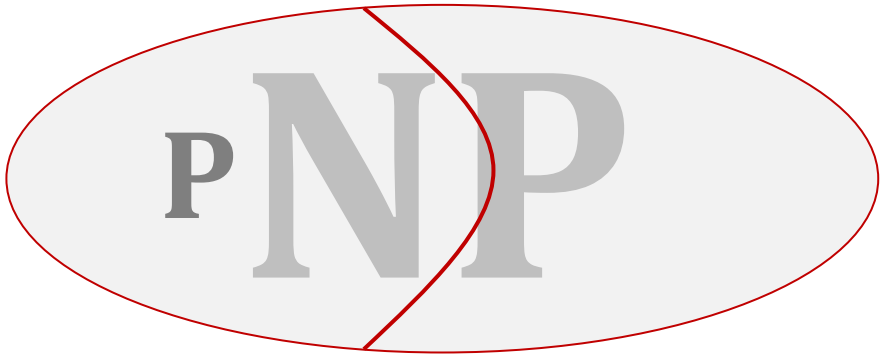
Η Κλάση P

- Το σύνολο των προβλημάτων που *μπορούμε να **βρούμε** τη λύση τους* (να τα λύσουμε) σε πολυωνυμικό χρόνο.

Κλάσεις Υπολογιστικών Προβλημάτων

Η Κλάση NP

- Το σύνολο των προβλημάτων που *μπορούμε να **επαληθεύσουμε** τη λύση τους* (αν μας δοθεί) σε πολυωνυμικό χρόνο.



P ⊆ NP

Προφανώς Ισχύει !!!

Η Κλάση P

- Το σύνολο των προβλημάτων που *μπορούμε να **βρούμε** τη λύση τους* (να τα λύσουμε) σε πολυωνυμικό χρόνο.

Κλάσεις Υπολογιστικών Προβλημάτων

Η Κλάση NP

- Το σύνολο των προβλημάτων που *μπορούμε να **επαληθεύσουμε** τη λύση τους* (αν μας δοθεί) σε πολυωνυμικό χρόνο.

P ⊆ NP

Προφανώς Ισχύει !!!

Η Κλάση P

- Το σύνολο των προβλημάτων που *μπορούμε να **βρούμε** τη λύση τους* (να τα λύσουμε) σε πολυωνυμικό χρόνο.

Κλάσεις Υπολογιστικών Προβλημάτων

Η Κλάση NP

- Το σύνολο των προβλημάτων που *μπορούμε να* ***επαληθεύσουμε*** *τη λύση τους* (αν μας δοθεί) σε πολυωνυμικό χρόνο.

P = NP

Η Κλάση P

- Το σύνολο των προβλημάτων που *μπορούμε να* ***βρούμε*** *τη λύση τους* (να τα λύσουμε) σε πολυωνυμικό χρόνο.

Κλάσεις Υπολογιστικών Προβλημάτων

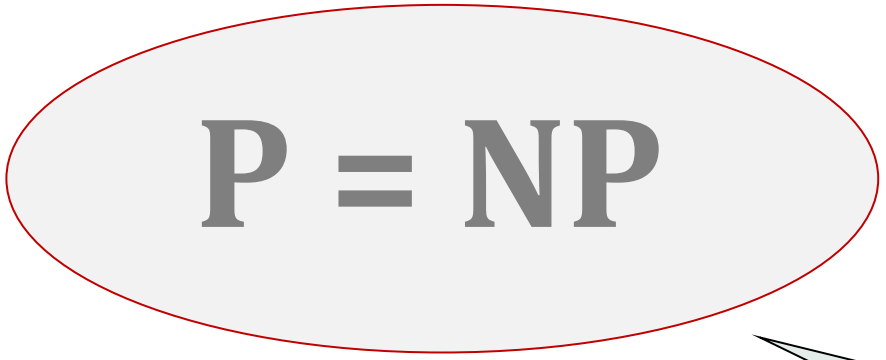
Το μεγάλο Ερώτημα της Επιστήμης μας !!!

P = NP 

million dollar question!

(Clay Institute millennium problems)

Το μεγάλο Ερώτημα της Επιστήμης μας !!!



Εάν **P = NP**

- Τότε, για κάθε δυσεπίλυτο πρόβλημα (όπως, το πρόβλημα του Hamilton) θα υπήρχε πολυωνυμικός αλγόριθμος που θα το επέλυε !!!

million dollar question!

(Clay Institute millennium problems)

Το μεγάλο Ερώτημα της Επιστήμης μας !!!

Εάν **P ≠ NP**

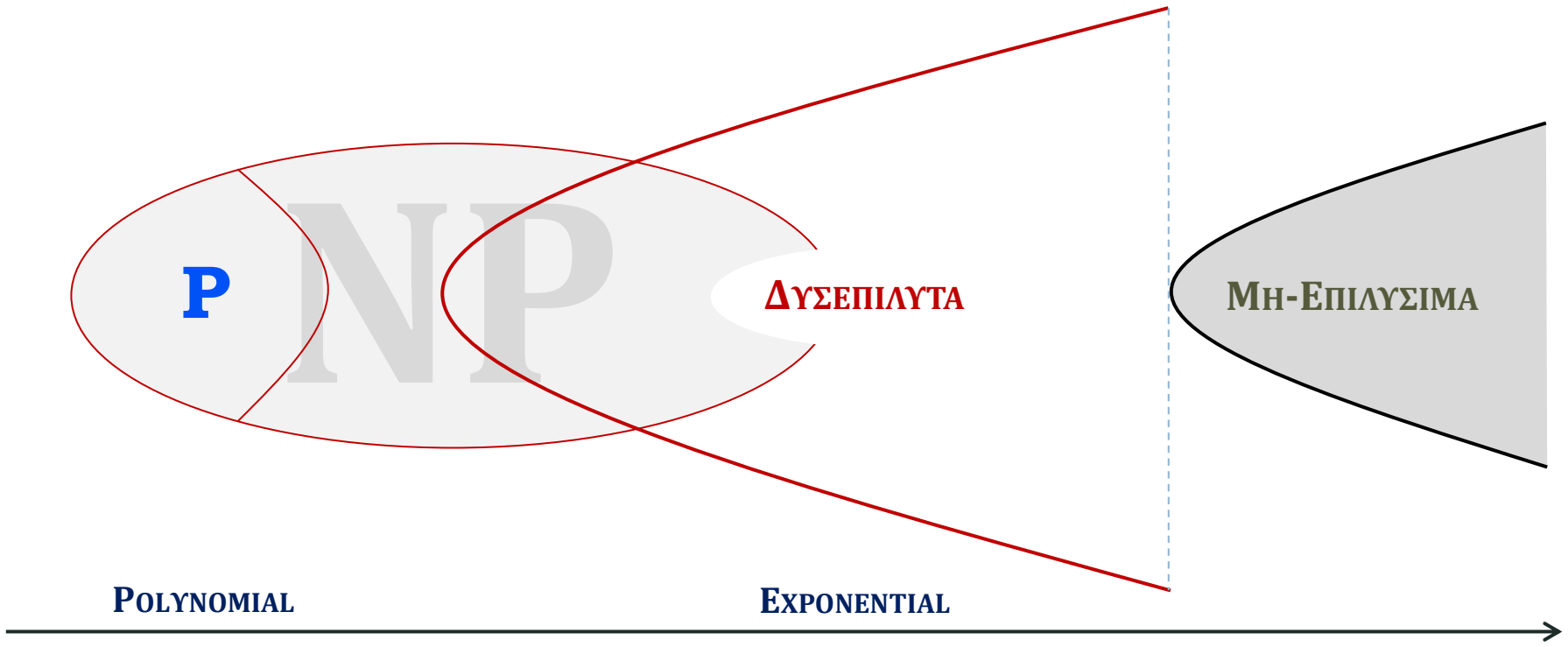
- Τότε, για το πρόβλημα του Hamilton, και για κάθε δυσεπίλυτο, θα είχαμε αποδείξει ότι δεν μπορεί να βρεθεί πολυωνυμικός αλγόριθμος που το επιλύει !!!

million dollar question!

(Clay Institute millennium problems)

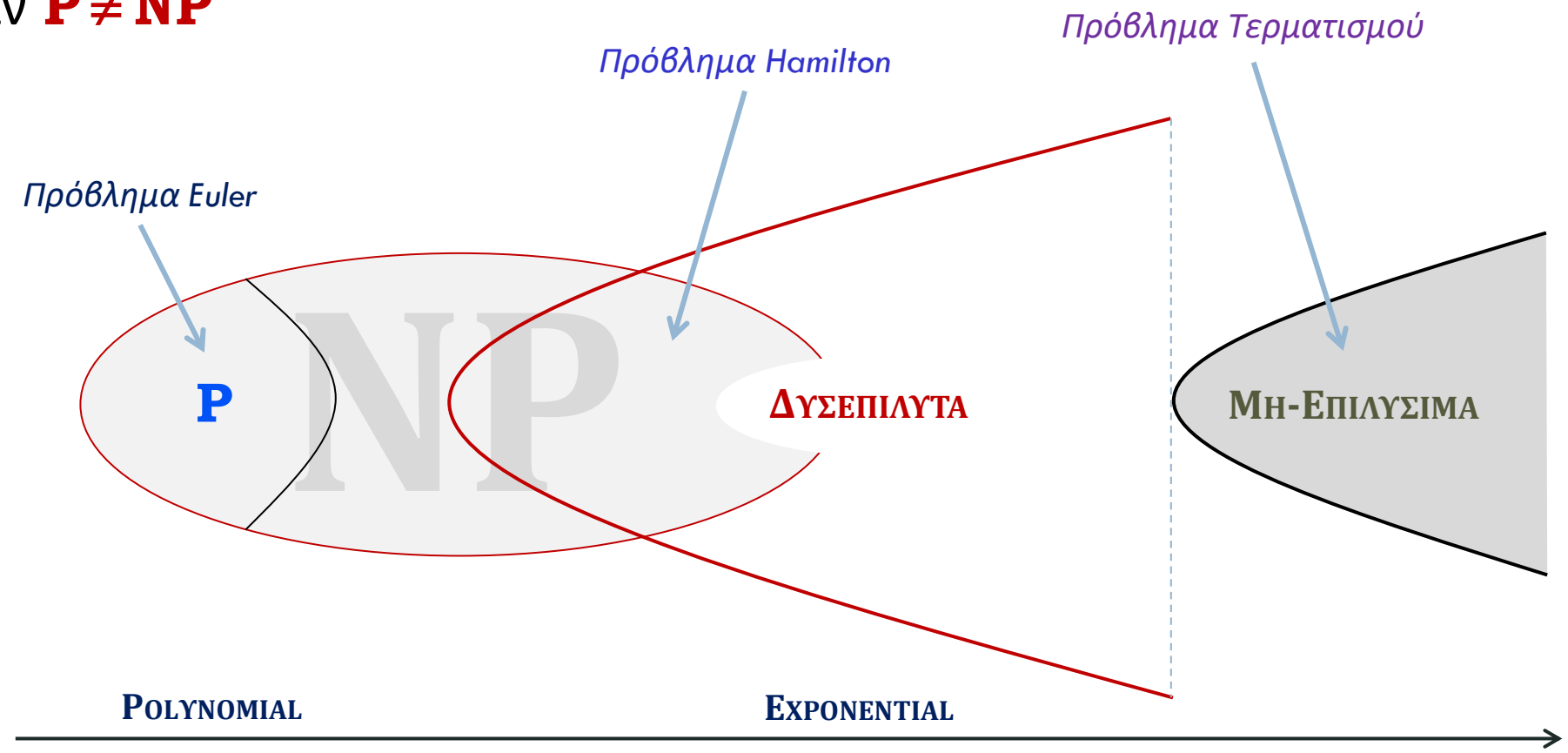
Κλάσεις Υπολογιστικών Προβλημάτων

Εάν **P ≠ NP**



Κλάσεις Υπολογιστικών Προβλημάτων

Εάν **P ≠ NP**



Κλάσεις Υπολογιστικών Προβλημάτων

Εάν **P ≠ NP**

NP Το σύνολο των προβλημάτων που *μπορούμε να **επαληθεύσουμε** τη λύση τους* (αν μας δοθεί) σε πολυωνυμικό χρόνο.

P Τα προβλήματα που *μπορούμε να **λύσουμε*** σε πολυωνυμικό χρόνο (είναι αυτά που λύνονται στην πράξη).

- Το πρόβλημα του Euler ανήκει στο P

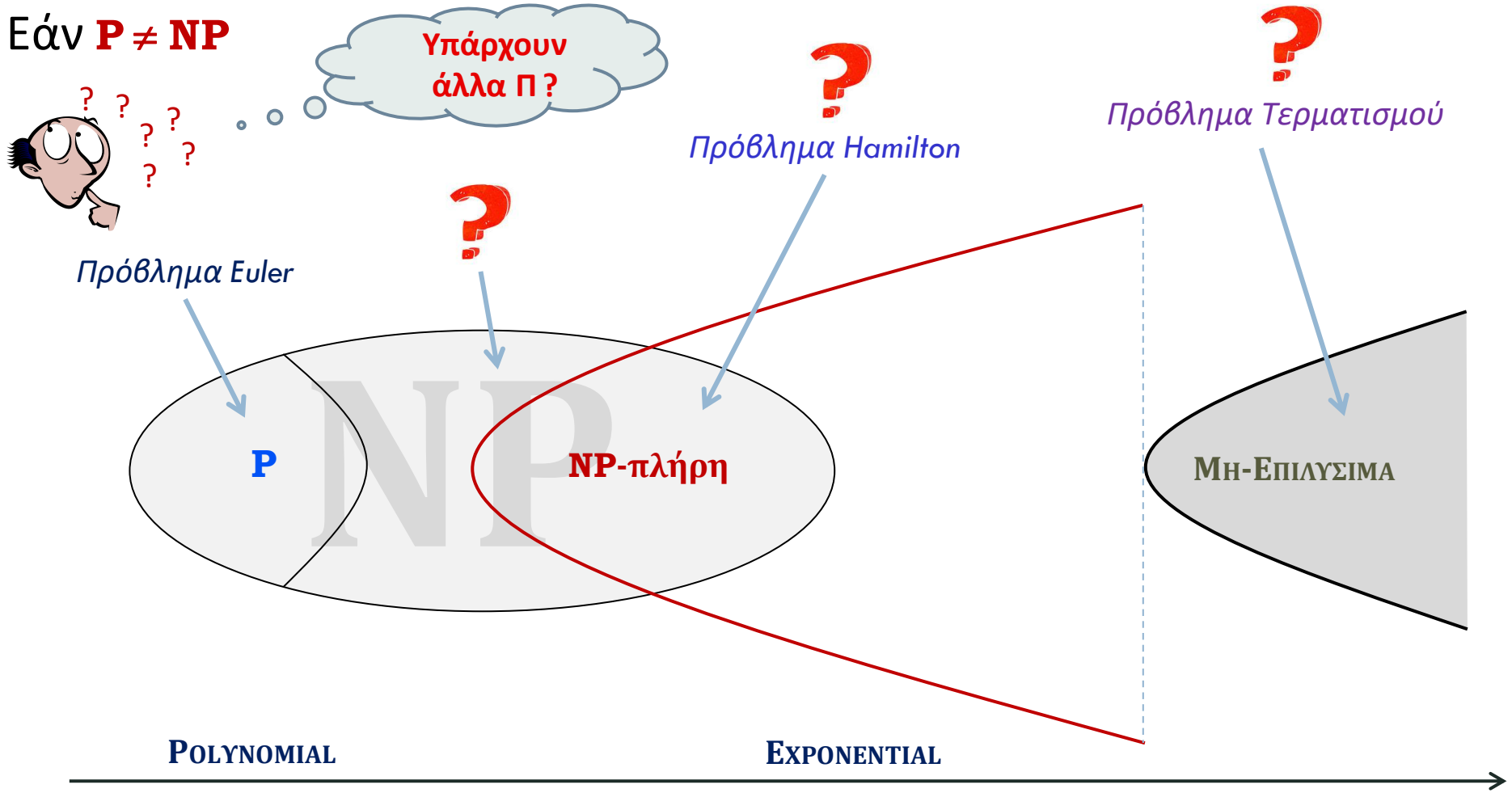
NP-πλήρη Τα ***πιο δύσκολα προβλήματα του* NP**.

Για κανένα δεν έχει βρεθεί πολυωνυμικός αλγόριθμος.

Αν οποιοδήποτε ένα από αυτά βρεθεί, τότε P = NP.

- Το πρόβλημα του Hamilton είναι NP-πλήρες

Ο χάρτης των κλάσεων (μέχρι τώρα)



Άλλα Μη-επιλύσημα Προβλήματα

Πρόβλημα Κάλυψης

Μη-επιλύσημο

- Δίνεται πεπερασμένο πλήθος ειδών (τύπων σχήματος) από πλακάκια, π.χ.

Μπορούμε να καλύψουμε μια δεδομένη επιφάνεια με τέτοια πλακάκια;

- Το πρόβλημα είναι μη-επιλύσιμο, δηλαδή, δεν υπάρχει πρόγραμμα, που να παίρνει για είσοδο τους τύπους πλακιδίων και να απαντά την ερώτηση.

Άλλα Μη-επιλύσημα Προβλήματα

Το Πρόβλημα της Σφαίρας

- Δίνονται τετράγωνα. Είναι το σχήμα τοπολογικά ισόμορφο με δίσκο;

Αν δηλαδή ήταν από πλαστελίνη, μπορούμε να την μετατρέψουμε σε δίσκο, χωρίς να σχίσουμε ή να κολλήσουμε τμήματα της;

- Η απάντηση είναι καταφατική για το πρώτο σχήμα και αρνητική για το δεύτερο.

Άλλα Μη-επιλύσημα Προβλήματα

Το Πρόβλημα της Σφαίρας

Μη-επιλύσημο

- Δίνονται τετράγωνα. Είναι το σχήμα τοπολογικά ισόμορφο με δίσκο;

- Το πρόβλημα αυτό για διαστάσεις υψηλότερες από 3 είναι μη-επιλύσιμο.
- Για τις 3 διαστάσεις προτάθηκε πρόσφατα ένας αλγόριθμος.
- Το πρόβλημα σχετίζετε με το ερώτημα: Τι σχήμα έχει το σύμπαν;

Άλλα Μη-επιλύσημα Προβλήματα

Το 10ο Πρόβλημα του Hilbert

Μη-επιλύσημο

- Δίνεται Διοφαντική εξίσωση, δηλαδή ακέραια πολυωνυμική εξίσωση, π.χ.

$$x^2 - 2y^2 = 5 \quad \text{ή} \quad x^3 + y^3 = z^3.$$

Έχει λύση στους ακέραιους;

- Το πρόβλημα το έθεσε ο Hilbert το 1900.
- Το 1970 ο Yuri Matiyasevich έδειξε ότι είναι μη-επιλύσιμο, δηλαδή, δεν υπάρχει πρόγραμμα, που να παίρνει για είσοδο μια εξίσωση και να απαντά την ερώτηση.

Άλλα NP-πλήρη Προβλήματα

Πρόβλημα Ικανοποιησημότητας (SAT)

NP-πλήρες

- Δίνεται ένας λογικός τύπος (Boolean formula) σε κανονική συζευκτική μορφή (CNF), όπως για παράδειγμα:

$$(x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (x_4)$$

Μπορούμε να βρούμε μια *ικανοποιούσα ανάθεση τιμών αληθείας* (satisfying truth assignment);

- Μια ικανοποιούσα ανάθεση τιμών αληθείας είναι μια ανάθεση τιμών *true* ή *false* σε κάθε μεταβλητή έτσι ώστε ο λογικός τύπος να είναι *true*.

✓ In computational complexity theory, the **Cook's theorem** states that the Boolean satisfiability problem is NP-complete

Άλλα NP-πλήρη Προβλήματα

Πρόβλημα Διαμέρισης (Partition)

NP-πλήρες

- Δίνονται ακέραιοι $a_1, a_2, \ldots, a_n$. Μπορούμε να διαμερίσουμε τους ακεραίους σε δύο σύνολα ίσου αθροίσματος;
  - Παράδειγμα

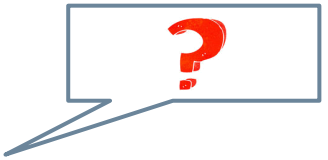
    14175, 15055, 16616, 17495, 18072, 19390, 19731, 22161, 23320, 23717, 26343, 28725, 29127, 32257, 40020, 41867, 43155, 46298, 56734, 57176, 58306, 61848, 65825, 66042, 68634, 69189, 72936, 74287, 74537, 81942, 82027, 82623, 82802, 82988, 90467, 97042, 97507, 99564.

    ✓ Κάθε σύνολο θα πρέπει να έχει άθροισμα 1000000

    ✓ Μπορείτε να βρείτε τη λύση;

Ενδιάμεση Πολυπλοκότητα;

Πρόβλημα Παραγοντοποίησης (Factoring)



- Δίνεται σύνθετος αριθμός N. Βρείτε την παραγοντοποίησή του (τους πρώτους παράγοντές του).

1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413

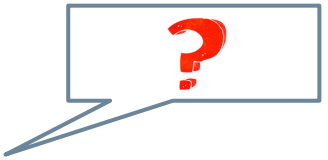
=

3347807169895689878604416984821269081770479498371376856891243138898288379387800228761471165253174308773781446799948936747604366679959042824463379962795263227915816434308764267603228381573966651127923337341714339681027009279873630891 7

x

Ενδιάμεση Πολυπλοκότητα;

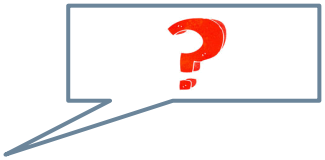
Πρόβλημα Παραγοντοποίησης (Factoring)



- Δίνεται σύνθετος αριθμός N. Βρείτε την παραγοντοποίησή του (τους πρώτους παράγοντές του).
  - Το πρόβλημα Factoring μάλλον δεν ανήκει στην κλάση **P**.
  - Ανήκει στην κλάση **NP** (γιατί;), αλλά μάλλον δεν είναι **NP**-πλήρες.
  - Για κβαντικούς υπολογιστές (που δεν έχουμε ακόμα καταφέρει να κατασκευάσουμε) ανήκει στο **P**.

Ενδιάμεση Πολυπλοκότητα;

Πρόβλημα Ισομορφισμού Γραφημάτων



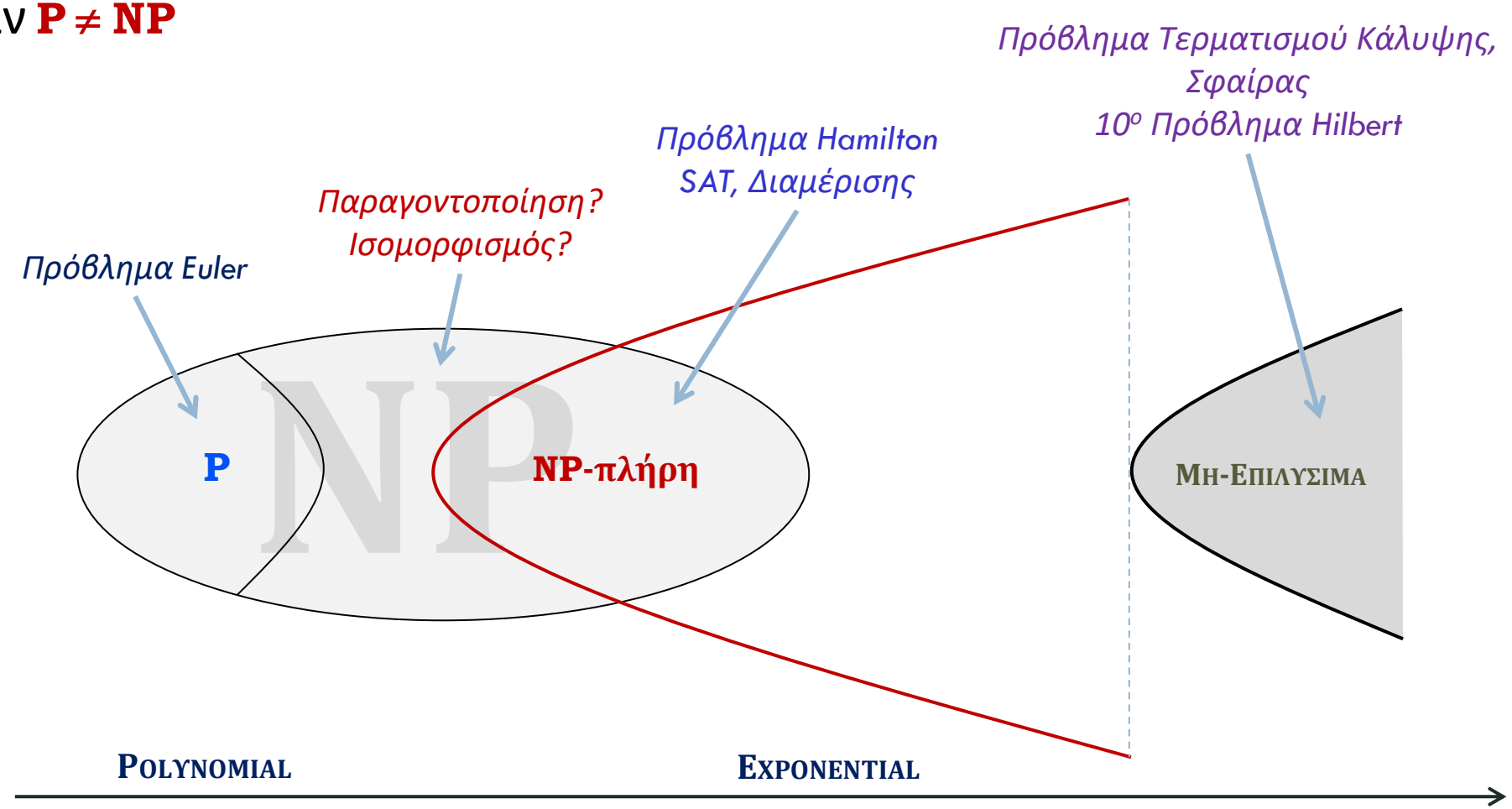
- Δίδονται δύο NxN Boolean πίνακες A_1 και A_2, που είναι οι πίνακες γειτνίασης δύο γραφημάτων G_1 και G_2 τάξης N.

$$A_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

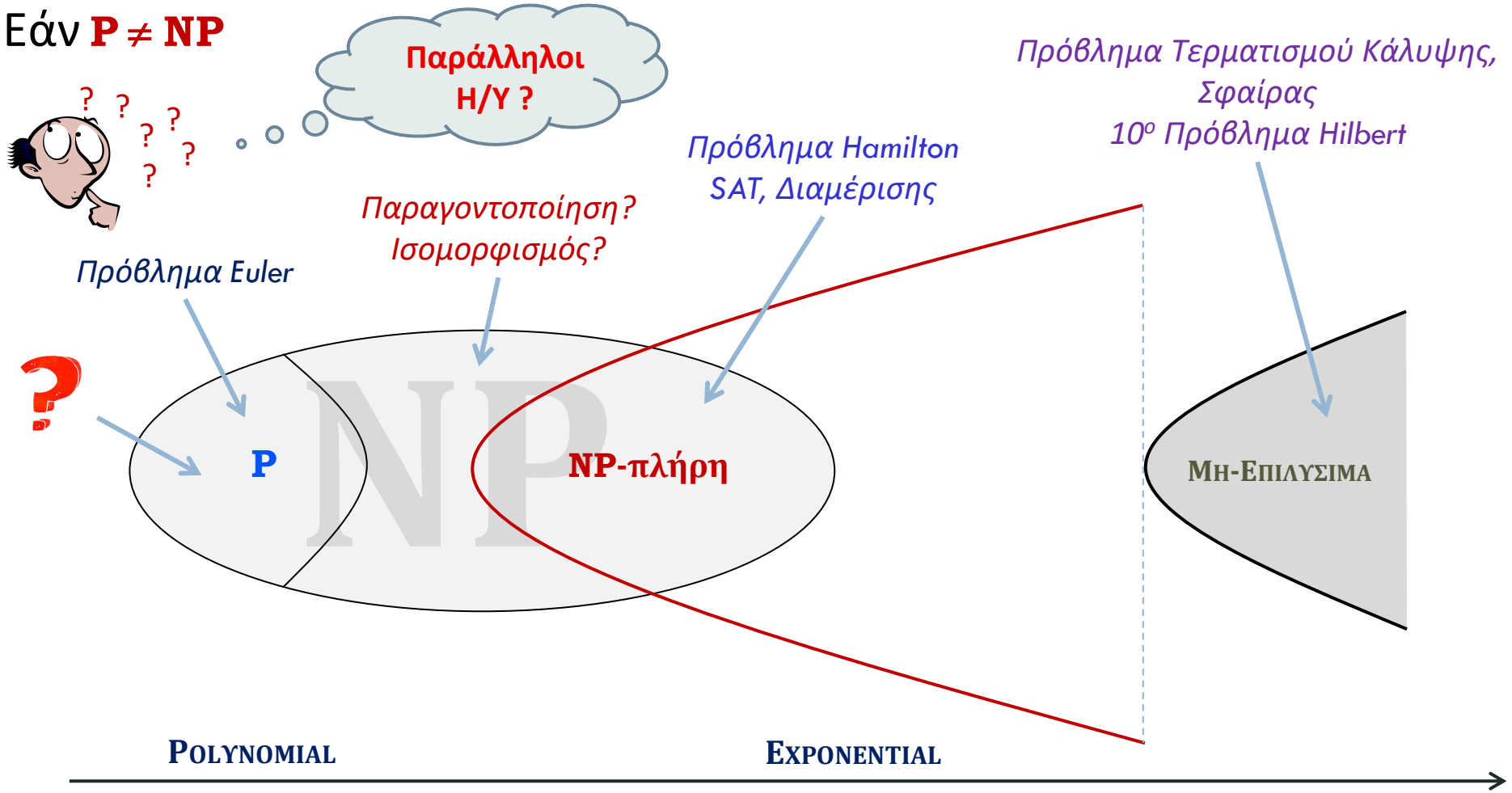
Υπάρχει ακολουθία από ανταλλαγές μεταξύ των γραμμών και στηλών του ενός πίνακα (π.χ., του A_1), έτσι ώστε $A_1 = A_2$;

Ο χάρτης των κλάσεων (μέχρι τώρα)

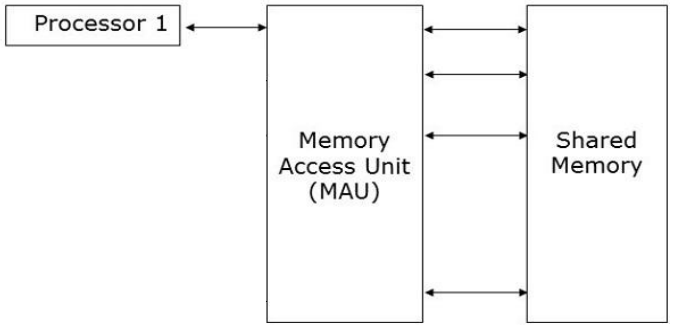
Εάν **P ≠ NP**



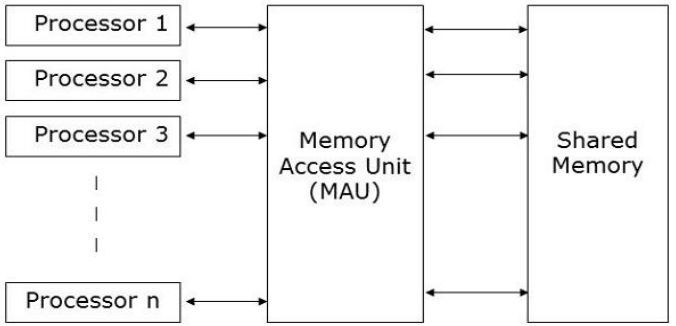
Ο χάρτης των κλάσεων (μέχρι τώρα)



Παραλληλισμός



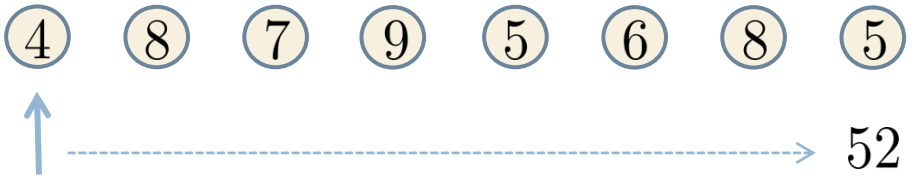
RAM



PRAM

Υπολογισμός Αθροίσματος

- Δίνονται ακέραιοι $a_1, a_2, \ldots, a_n$. Μπορούμε να υπολογίσουμε το άθροισμά τους παράλληλα και γρηγορότερα από O(n);



Πολυπλοκότητα Ακολουθιακής Επίλυσης

T(n) = O(n)

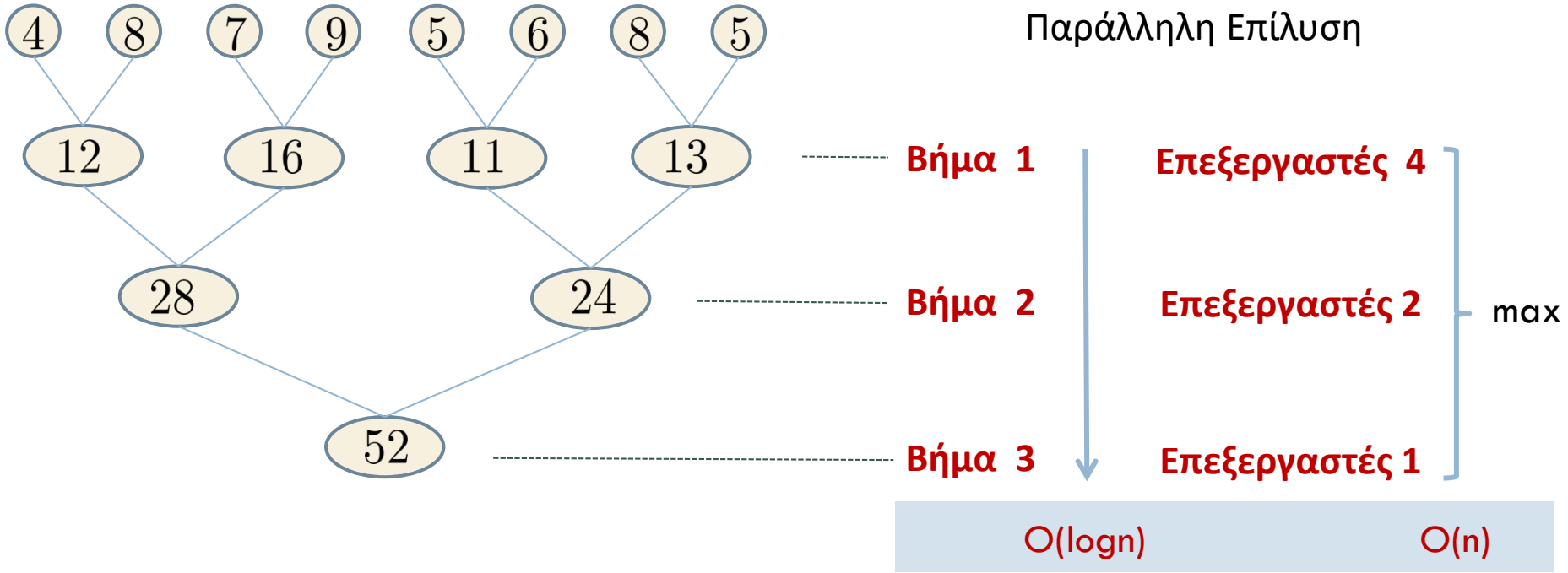
Υπολογισμός Αθροίσματος

- Δίνονται ακέραιοι $a_1, a_2, \ldots, a_n$. Μπορούμε να υπολογίσουμε το άθροισμά τους παράλληλα και γρηγορότερα από O(n);

Εάν διαθέτω μια παράλληλη μηχανή με 4 επεξεργαστές, μπορώ να το λύσω γρηγορότερα;

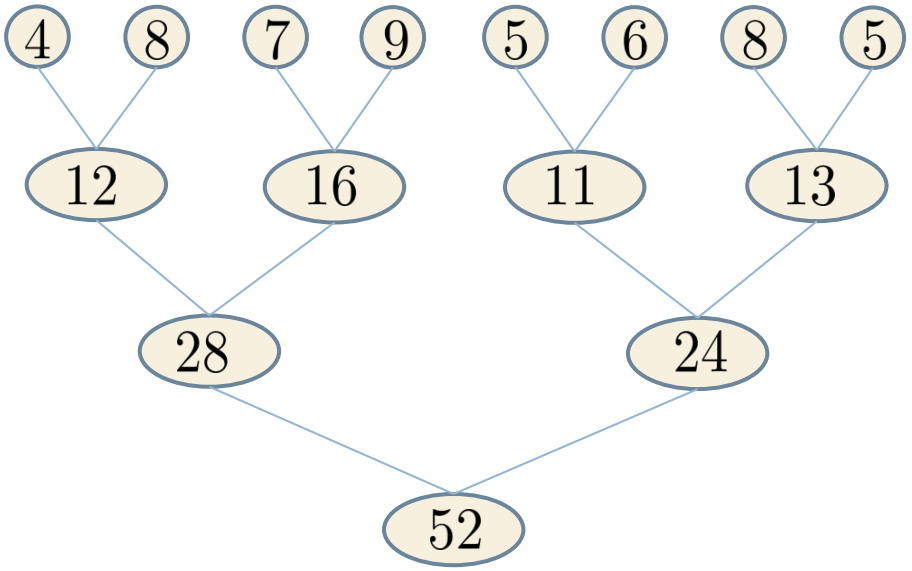
Υπολογισμός Αθροίσματος

- Δίνονται ακέραιοι $a_1, a_2, \ldots, a_n$. Μπορούμε να υπολογίσουμε το άθροισμά τους παράλληλα και γρηγορότερα από O(n);



Υπολογισμός Αθροίσματος

- Δίνονται ακέραιοι $a_1, a_2, \ldots, a_n$. Μπορούμε να υπολογίσουμε το άθροισμά τους παράλληλα και γρηγορότερα από O(n);



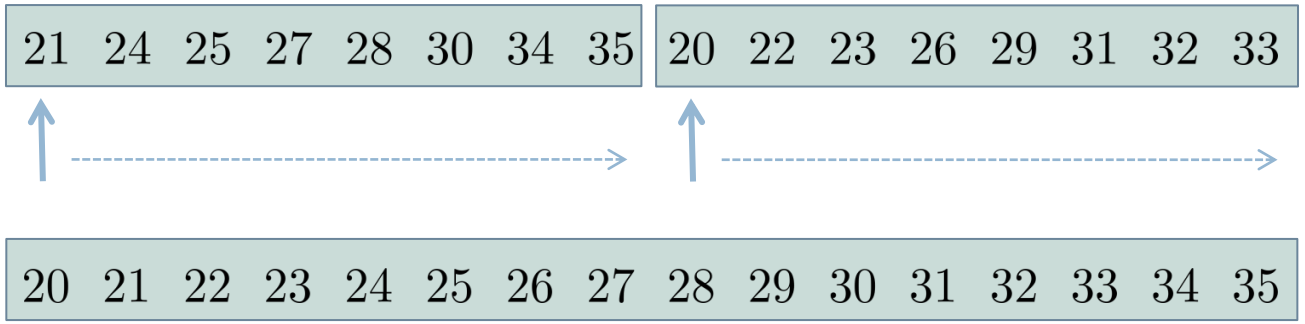
Πολυπλοκότητα
Παράλληλης Επίλυσης

$T(n) = O(\log n)$

$P(n) = O(n)$

Ταξινόμηση με Συγχώνευση

- Δίνονται 2 ταξινομημένες ακολουθίες $(a_1, a_2, \ldots, a_n)$ και $(b_1, b_2, \ldots, b_n)$. Μπορούμε να τις συγχωνεύουμε παράλληλα και γρηγορότερα από $O(n)$;

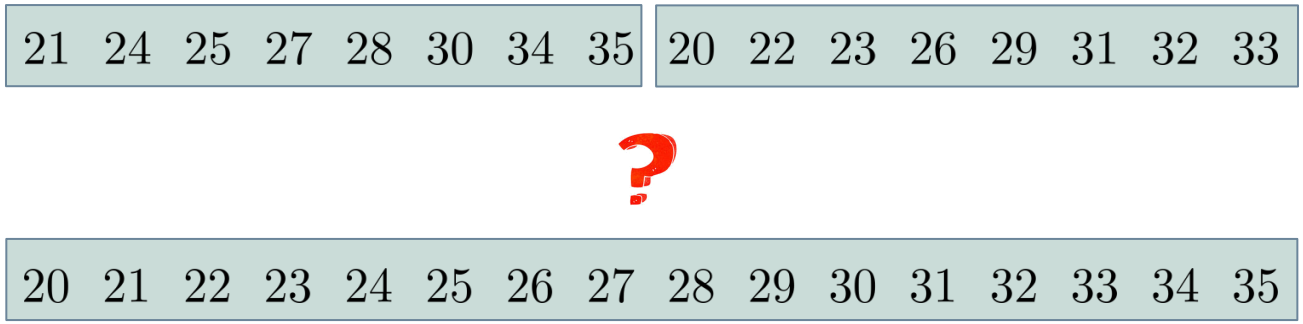


Πολυπλοκότητα Ακολουθιακής Επίλυσης

$T(n) = O(n)$

Ταξινόμηση με Συγχώνευση

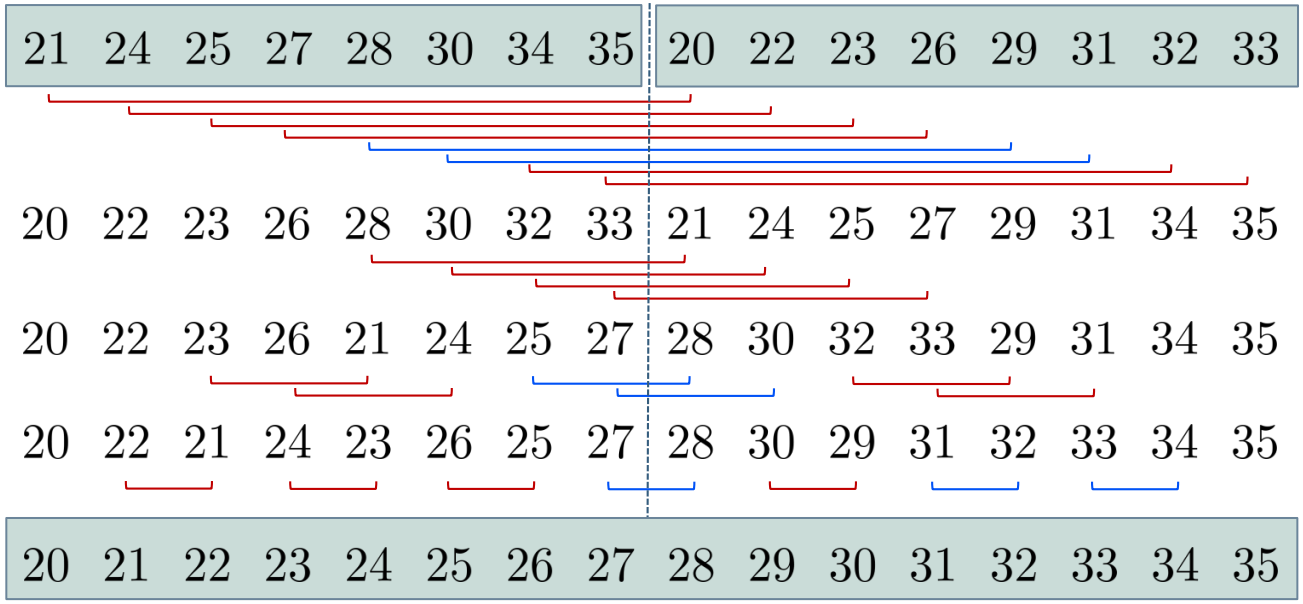
- Δίνονται 2 ταξινομημένες ακολουθίες $(a_1, a_2, \ldots, a_n)$ και $(b_1, b_2, \ldots, b_n)$. Μπορούμε να τις συγχωνεύουμε παράλληλα και γρηγορότερα από $O(n)$;



Σε μια παράλληλη μηχανή με 8 επεξεργαστές, μπορώ να το λύσω γρηγορότερα;

Ταξινόμηση με Συγχώνευση

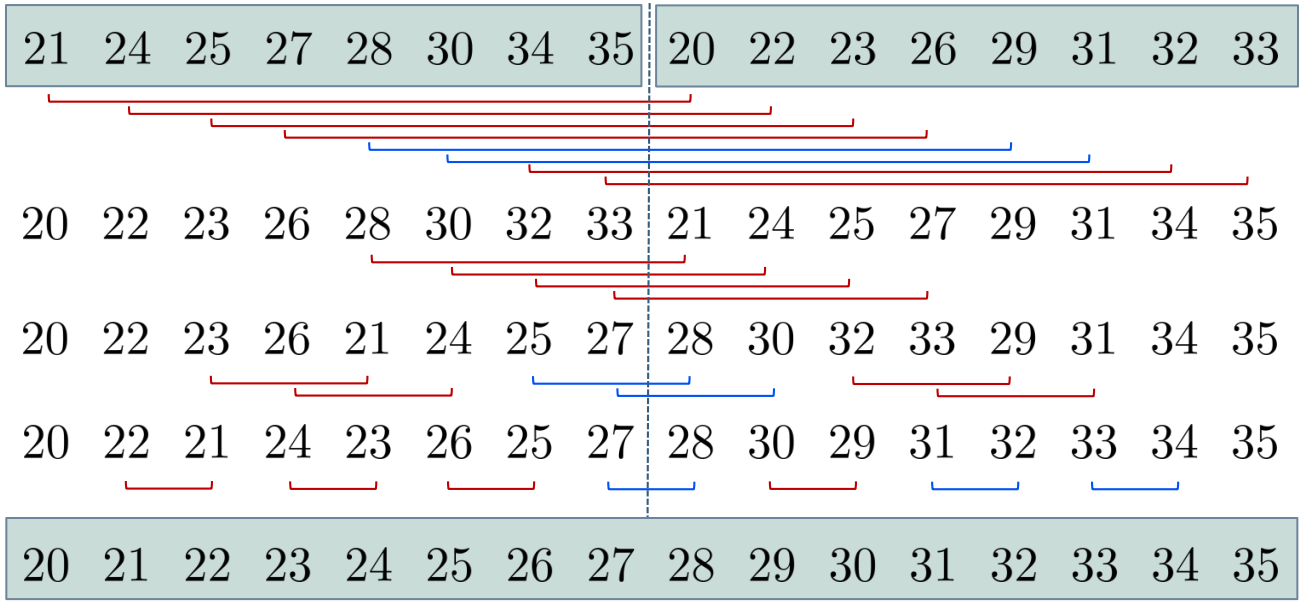
- Δίνονται 2 ταξινομημένες ακολουθίες $(a_1, a_2, \ldots, a_n)$ και $(b_1, b_2, \ldots, b_n)$. Μπορούμε να τις συγχωνεύουμε παράλληλα και γρηγορότερα από $O(n)$;



Παράλληλη Επίλυση

Ταξινόμηση με Συγχώνευση

- Δίνονται 2 ταξινομημένες ακολουθίες $(a_1, a_2, \ldots, a_n)$ και $(b_1, b_2, \ldots, b_n)$. Μπορούμε να τις συγχωνεύουμε παράλληλα και γρηγορότερα από $O(n)$;



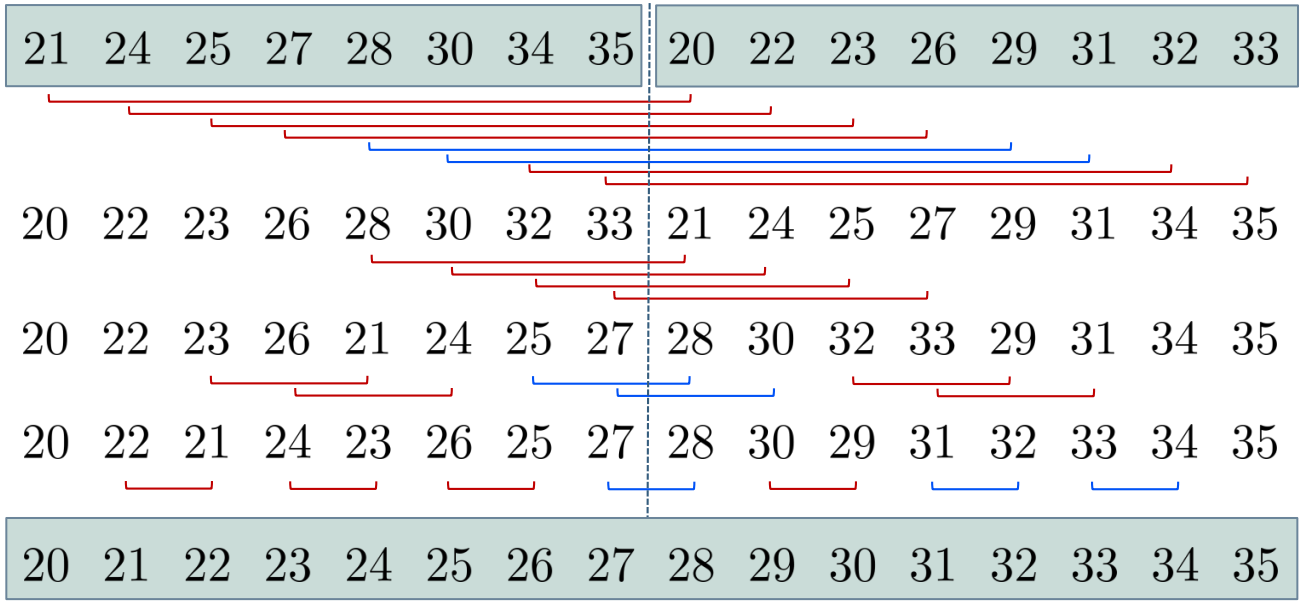
Σε κάθε επανάληψη
Απόσταση και Πλήθος Συγκρίσεων

Απόσταση	Πλήθος
8	8
4	4
2	6
1	7
O(logn)	O(n)

max

Ταξινόμηση με Συγχώνευση

- Δίνονται 2 ταξινομημένες ακολουθίες $(a_1, a_2, \ldots, a_n)$ και $(b_1, b_2, \ldots, b_n)$. Μπορούμε να τις συγχωνεύουμε παράλληλα και γρηγορότερα από $O(n)$;



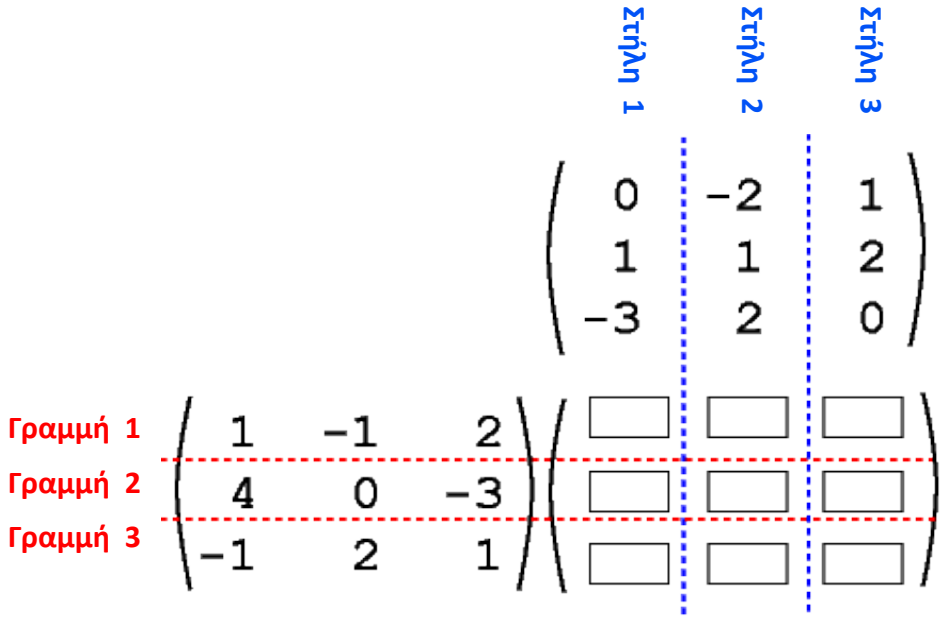
Πολυπλοκότητα Παράλληλης Επίλυσης

$T(n) = O(\log n)$

$P(n) = O(n)$

Πολλαπλασιασμός Πινάκων

- Δίνονται 2 διδιάστατοι πίνακες A(nxn) και B(nxn). Μπορούμε να τους πολλαπλασιάσουμε παράλληλα και γρηγορότερα από O(M(n));



Πολυπλοκότητα
Παράλληλης Επίλυσης

$T(n) = O(\log n)$

$P(n) = O(n^2)$

Η Κλάση NC

- Το σύνολο των προβλημάτων που *μπορούμε να τα **λύσουμε*** σε πολυλογαριθμικό χρόνο σε ένα παράλληλο Η/Υ με πολυωνυμικό πλήθος επεξεργαστών, δηλ.

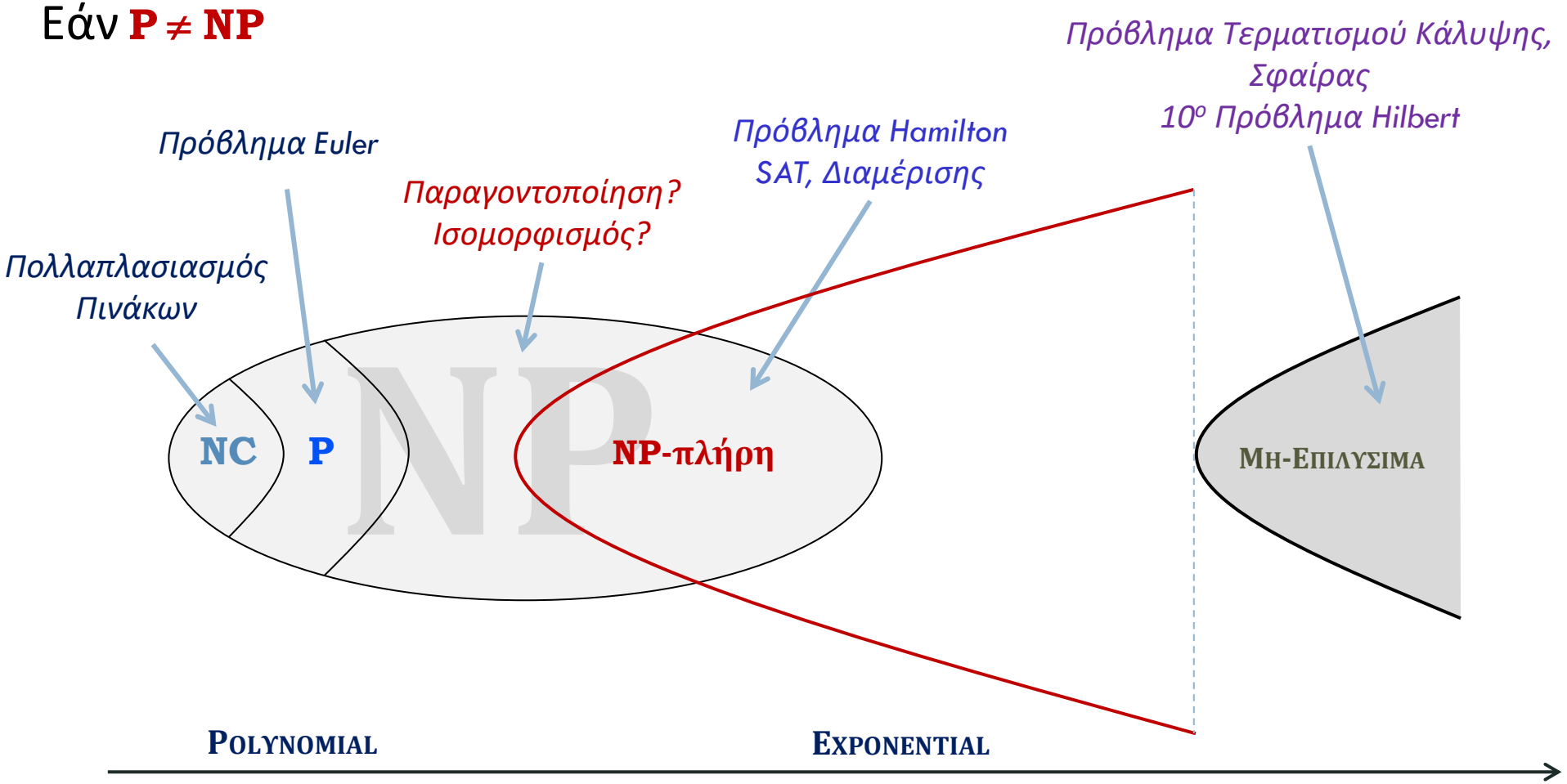
  Λύση του Π ⇔ **$O(\log^c n)$** χρόνο με **$O(n^k)$** επεξεργαστές

  για σταθερές c και k.

- Στην θεωρία πολυπλοκότητας, NC = "Nick's Class".
  - ✓ Ο Stephen Cook επινόησε το όνομα "Nick's class" προς τιμή του Nick Pippenger, για την ερευνητική του συμβολή σε κυκλώματα (circuits) με πολυλογαριθμικό βάθος (polylogarithmic depth) και πολυωνυμικό μέγεθος (polynomial size).

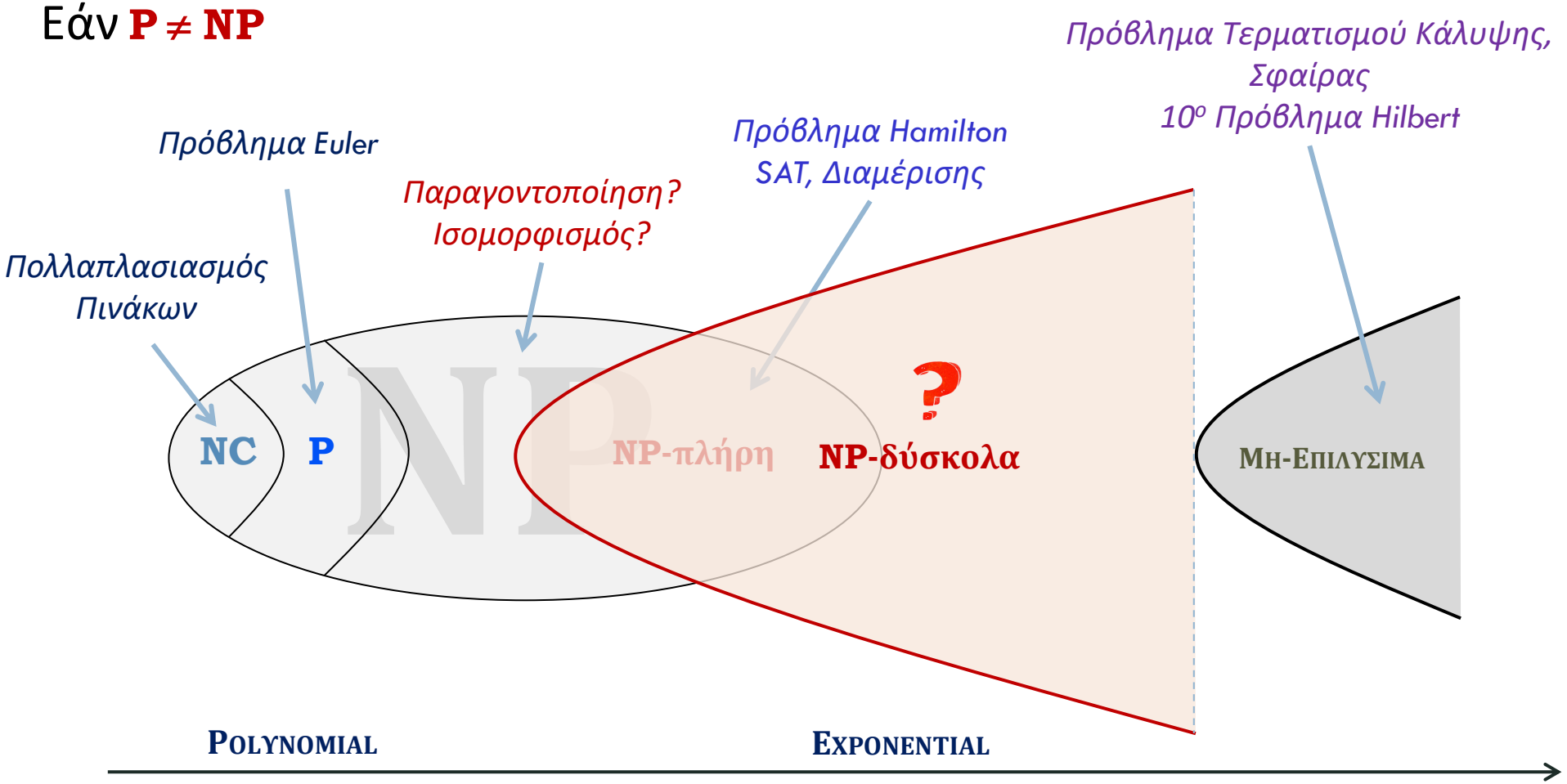
Ο χάρτης των κλάσεων (μέχρι τώρα)

Εάν **P ≠ NP**



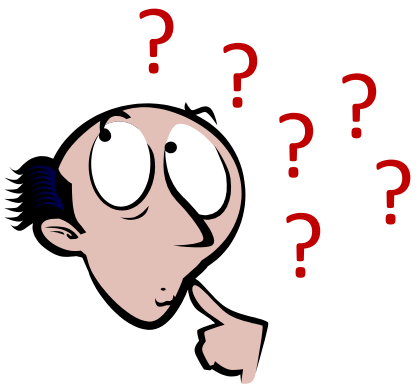
Ο χάρτης των κλάσεων (μέχρι τώρα)

Εάν **P ≠ NP**

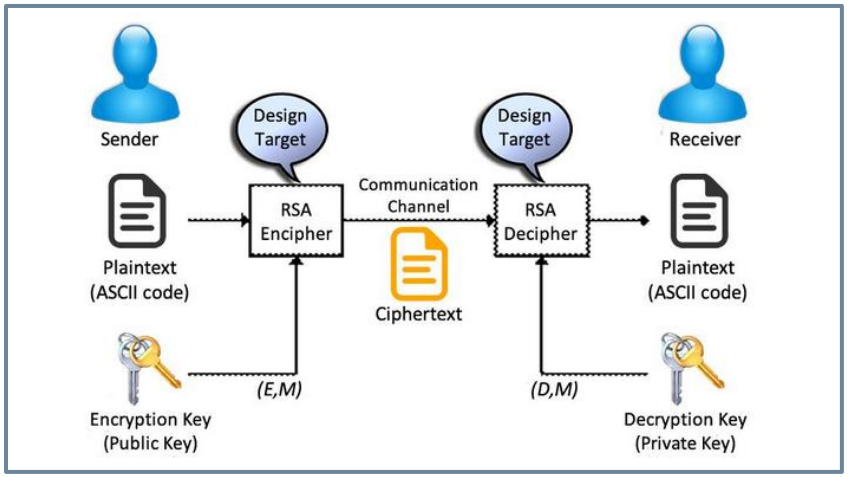


Ερωτήματα – Προβληματισμοί (V)

Η Δυσκολία προς Όφελός μας !!!



RSA



Η Δυσκολία προς Όφελός μας !!!

Το σύστημα RSA

public-key cryptosystem

- Το κρυπτοσύστημα RSA (Rivest-Shamir-Adleman, 1977) και πολλά άλλα συστήματα βασίζονται στη δυσκολία του Factoring.

n
1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413

=

p
3347807169895689878604416984821269081770479498371376856891243138898288379387800228761471165253174308773781446799948936746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917

x

q

Η Δυσκολία προς Όφελός μας !!!

Το σύστημα RSA

- Το κρυπτοσύστημα RSA (Rivest-Shamir-Adleman, 1977) και πολλά άλλα συστήματα βασίζονται στη δυσκολία του Factoring.

  - Η A (Alice) θέλει να στείλει στον B (Bob) ένα μήνυμα m.
  - Ο B διαλέγει 2 μεγάλους πρώτους αριθμούς p και q και ένα ακέραιο e. Υπολογίζει το γινόμενο $n = pq$.
  - Ο B στέλνει στην A τα n και e.
  - Η A στέλνει στον B τον αριθμό $c = m^e \pmod n$.
  - Ο B υπολογίζει το $m = c^d \pmod n$, όπου $d = e^{-1} \pmod{(p-1)(q-1)}$.
  - Παράδειγμα: $p=11$, $q=17$, $n=187$, $e=21$, $d=61$, $m=42$, $c=9$

RSA

Η Δυσκολία προς Όφελός μας !!!

Το σύστημα RSA

- Το κρυπτοσύστημα RSA (Rivest-Shamir-Adleman, 1977) και πολλά άλλα συστήματα βασίζονται στη δυσκολία του Factoring.

Πυλώνες του RSA

- Υπολογιστική **ευκολία** ύψωσης σε δύναμη αριθμού 1000δων ψηφίων.
- Υπολογιστική **ευκολία** εύρεσης πρώτων αριθμών με 1000δες ψηφία.
- Υπολογιστική **ευκολία** υπολογισμού αντιστρόφου modulo n (με το n να έχει 1000δες ψηφία) – μέσω αλγορίθμου Ευκλείδη!
- Υπολογιστική **δυσκολία** παραγοντοποίησης (Factoring) αριθμών με 1000δες ψηφία.

Η Δυσκολία προς Όφελός μας !!!

Υπολογιστική Δυσκολία και Όφελος !!!

- Άλλες Εφαρμογές:
  - Ψηφιακές υπογραφές
  - Ασφάλεια επικοινωνιών
  - Ασφάλεια συναλλαγών
  - Ηλεκτρονικές ψηφοφορίες
  - Bitcoin: νομίσματα και συναλλαγές χωρίς μεσάζοντες!

Η Δυσκολία προς Όφελός μας !!!

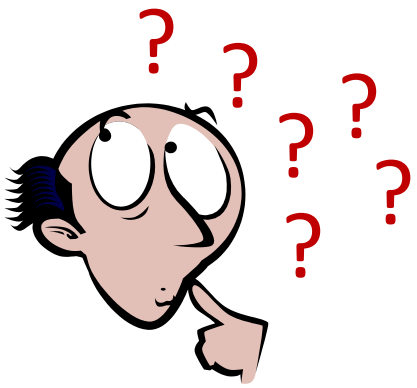
Υπολογιστική Δυσκολία και Όφελος !!!

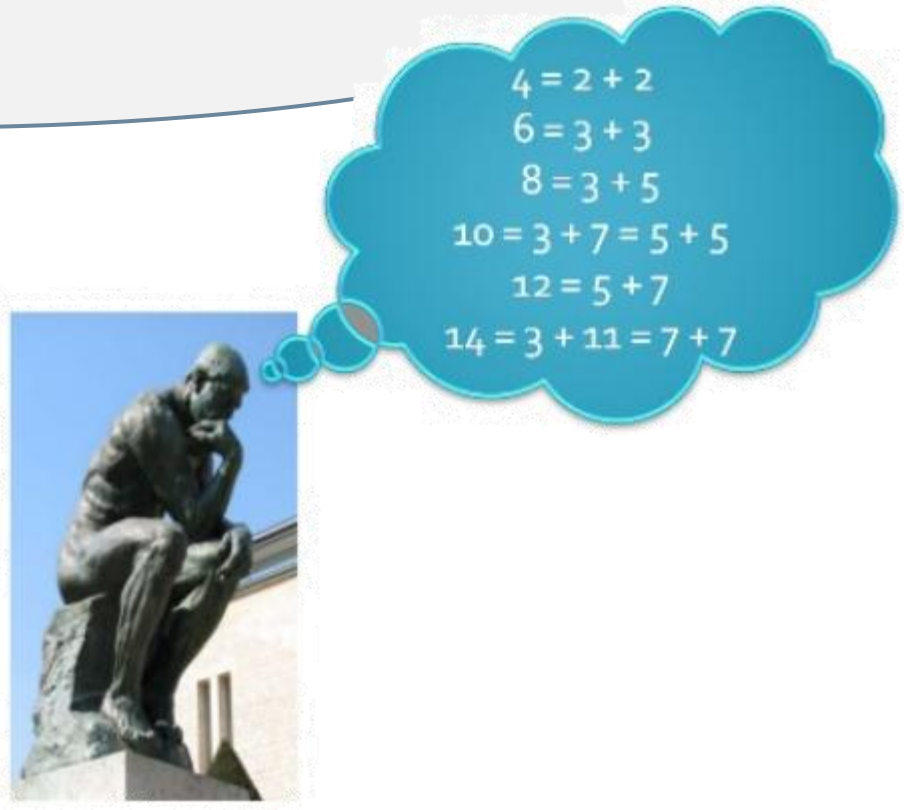
- Τι πρέπει να μας μείνει !!!

  - Πολλές επαναστατικές ιδέες και εφαρμογές της εποχής μας στηρίζονται στον **Υπολογισμό** !
  - Ιδιαίτερα στο **πόσο εύκολα** (γρήγορα) μπορούμε να υπολογίσουμε κάποια πράγματα !!
  - Και στο **πόσο δύσκολο** να υπολογίσουμε κάποια άλλα !!!

Ερωτήματα – Προβληματισμοί (VI)

Εικασίες… Ανοικτά Ερωτήματα !!!





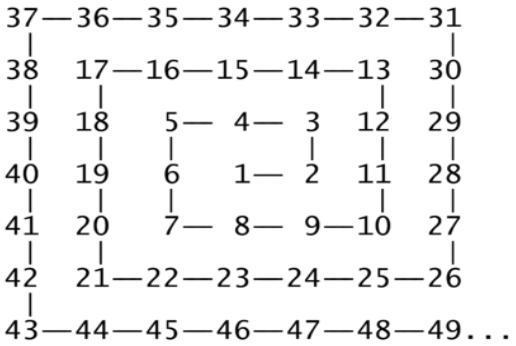
Εικασίες... Ανοικτά Ερωτήματα

Τι είναι Εικασία;

- Μία πρόταση με την οποία, βάσει λογικών σκέψεων, *πιθανολογούμε ό,τι είναι δυνατόν να ισχύει* !!!
- Στα μαθηματικά, μια **εικασία** είναι μία υπόθεση που φαίνεται να είναι σωστή αλλά *δεν μπορούμε να την αποδείξουμε ή να καταρρίψουμε* !!!

Με δύο λέξεις :

Είναι μια *θέση,* όχι **βέβαιη** αλλά **πιθανή !!!**



Εικασία του Γκόλντμπαχ (1690-1764)

- Η εικασία του Γκόλντμπαχ (18ο αιώνας), είναι ένα από τα γνωστότερα *άλυτα προβλήματα* και *παραμένει πεισματικά απρόσιτη* !!!

Εικασία Γκόλντμπαχ: *Κάθε άρτιος θετικός ακέραιος μεγαλύτερος του 2 μπορεί να γραφεί ως άθροισμα δύο πρώτων αριθμών.*

Παράδειγμα:

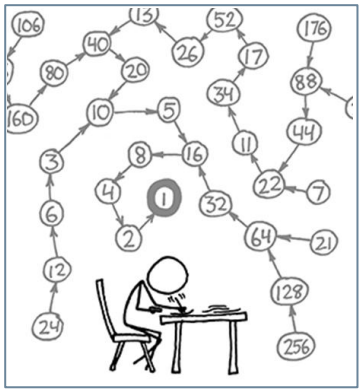
4 = 2 + 2
6 = 3 + 3
8 = 3 + 5
10 = 3 + 7 = 5 + 5
12 = 5 + 7
14 = 3 + 11 = 7 + 7

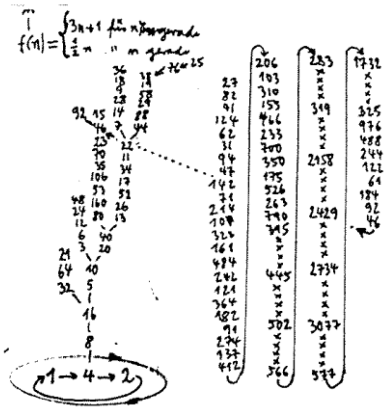
Εικασίες... Ανοικτά Ερωτήματα

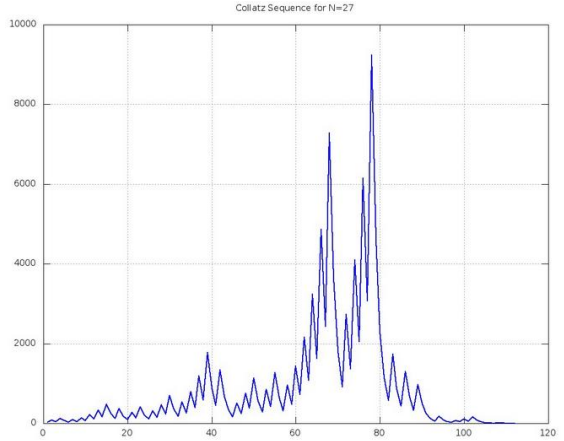
Πρόβλημα Collatz (Lothar Collatz 1937)

- *Δίνεται φυσικός αριθμός x. Τερματίζει το παρακάτω πρόγραμμα Π για αυτή την είσοδο (ή "τρέχει" επ' άπειρον);*

```
while x!=1 do
   if (x is even) then x=x/2 else x=3*x+1
```







Εικασίες... Ανοικτά Ερωτήματα

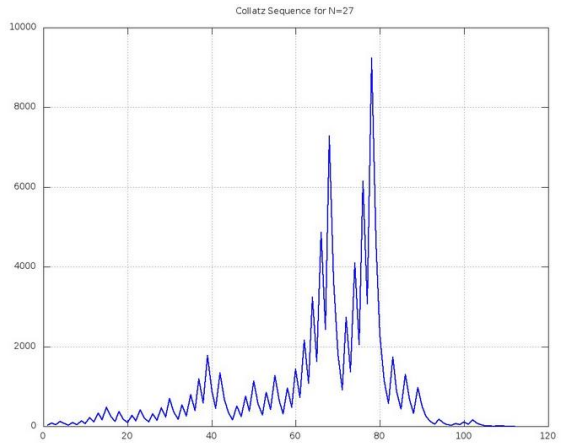
Πρόβλημα Collatz (Lothar Collatz 1937)

- *Δίνεται φυσικός αριθμός x. Τερματίζει το παρακάτω πρόγραμμα Π για αυτή την είσοδο (ή "τρέχει" επ' άπειρον);*

```
while x!=1 do
   if (x is even) then x=x/2 else x=3*x+1
```

Η ακολουθία για **x = 27**:

27, 82, **41**, 124, 62, **31**, 94, **47**, 142, **71**, 214, **107**, 322, **161**, 484, 242, **121**, 364, 182, **91**, 274, **137**, 412, 206, **103**, 310, **155**, 466, **233**, 700, 350, **175**, 526, **263**, 790, **395**, 1186, **593**, 1780, 890, **445**, 1336, 668, 334, **167**, 502, **251**, 754, **377**, 1132, 566, **283**, 850, **425**, 1276, 638, **319**, 958, **479**, 1438, **719**, 2158, **1079**, 3238, **1619**, 4858, **2429**, 7288, 3644, 1822, **911**, 2734, **1367**, 4102, **2051**, 6154, **3077**, 9232, 4616, 2308, 1154, **577**, 1732, 866, **433**, 1300, 650, **325**, 976, 488, 244, 122, **61**, 184, 92, 46, **23**, 70, **35**, 106, **53**, 160, 80, 40, 20, 10, **5**, 16, 8, 4, 2, **1**



Εικασίες... Ανοικτά Ερωτήματα

Πρόβλημα Collatz (Lothar Collatz 1937)

- *Δίνεται φυσικός αριθμός x. Τερματίζει το παρακάτω πρόγραμμα Π για αυτή την είσοδο (ή "τρέχει" επ' άπειρον);*

```
while x!=1 do
   if (x is even) then x=x/2 else x=3*x+1
```

Εικασία Collatz: Το πρόγραμμα Π τερματίζει για κάθε φυσικό αριθμό x.

Η εικασία είναι επίσης γνωστή ως **εικασία 3x+1**

και ως **εικασία Ούλαμ** (*Stanislaw Ulam*)

Εικασίες... Ανοικτά Ερωτήματα

Πρόβλημα Collatz (Lothar Collatz 1937)

- *Δίνεται φυσικός αριθμός x. Τερματίζει το παρακάτω πρόγραμμα Π για αυτή την είσοδο (ή "τρέχει" επ' άπειρον);*

```
while x!=1 do
   if (x is even) then x=x/2 else x=3*x+1
```

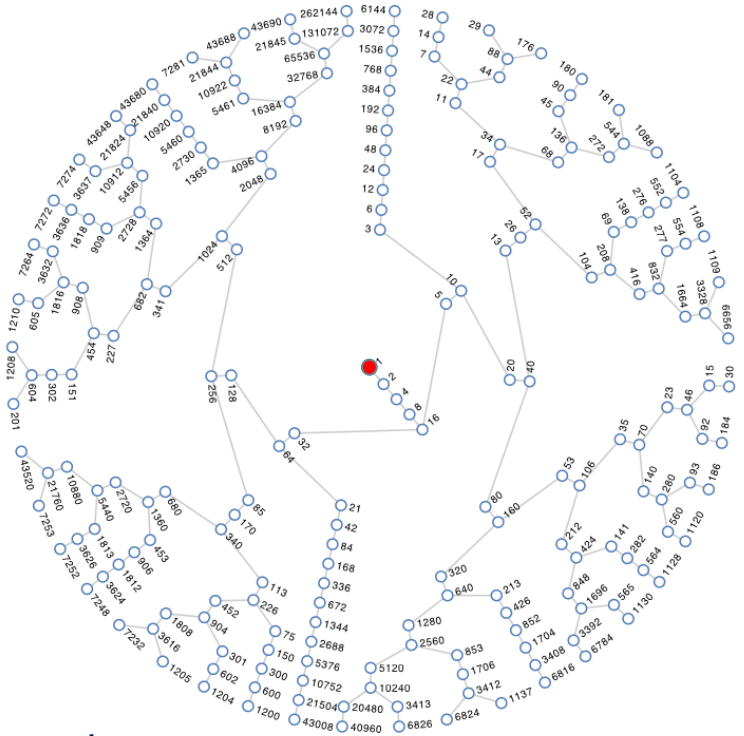
Εικασία Collatz: Το πρόγραμμα Π τερματίζει για κάθε φυσικό αριθμό x.

Δεν γνωρίζουμε αν ισχύει η εικασία (ανοικτό ερώτημα) ούτε αν το πρόβλημα είναι επιλύσιμο από υπολογιστή (αν δηλαδή μπορεί να υπάρξει πρόγραμμα το οποίο για είσοδο x να αποφαίνεται αν τερματίζει ή όχι).

Εικασίες… Ανοικτά Ερωτήματα

Πρόβλημα Collatz (Lothar Collatz 1937)

- **Εικασία Collatz**

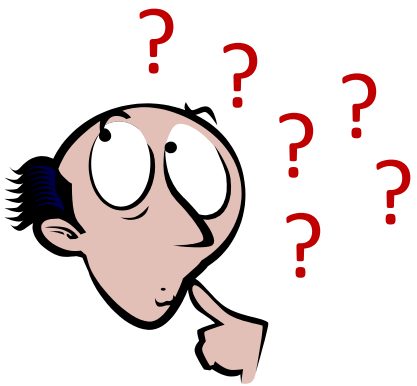


Το 2011, ο **Gerhard Opfer** του Πανεπιστημίου του Αμβούργου, που υπήρξε και μαθητής του Collatz, παρουσίασε μία απόδειξη της εικασίας που είχε όμως σημαντικά κενά, τα οποία επεσήμαναν διάφοροι μαθηματικοί, οπότε και την απέσυρε (τουλάχιστον προσωρινά)!!!

Ο διάσημος Ούγγρος μαθηματικός **Paul Erdős**, όταν ξεκίνησε να ασχολείται με το πρόβλημα, κατέληξε στο συμπέρασμα ότι τα μαθηματικά δεν είναι ακόμη έτοιμα για να προσεγγίσουν τέτοιου είδους προβλήματα…

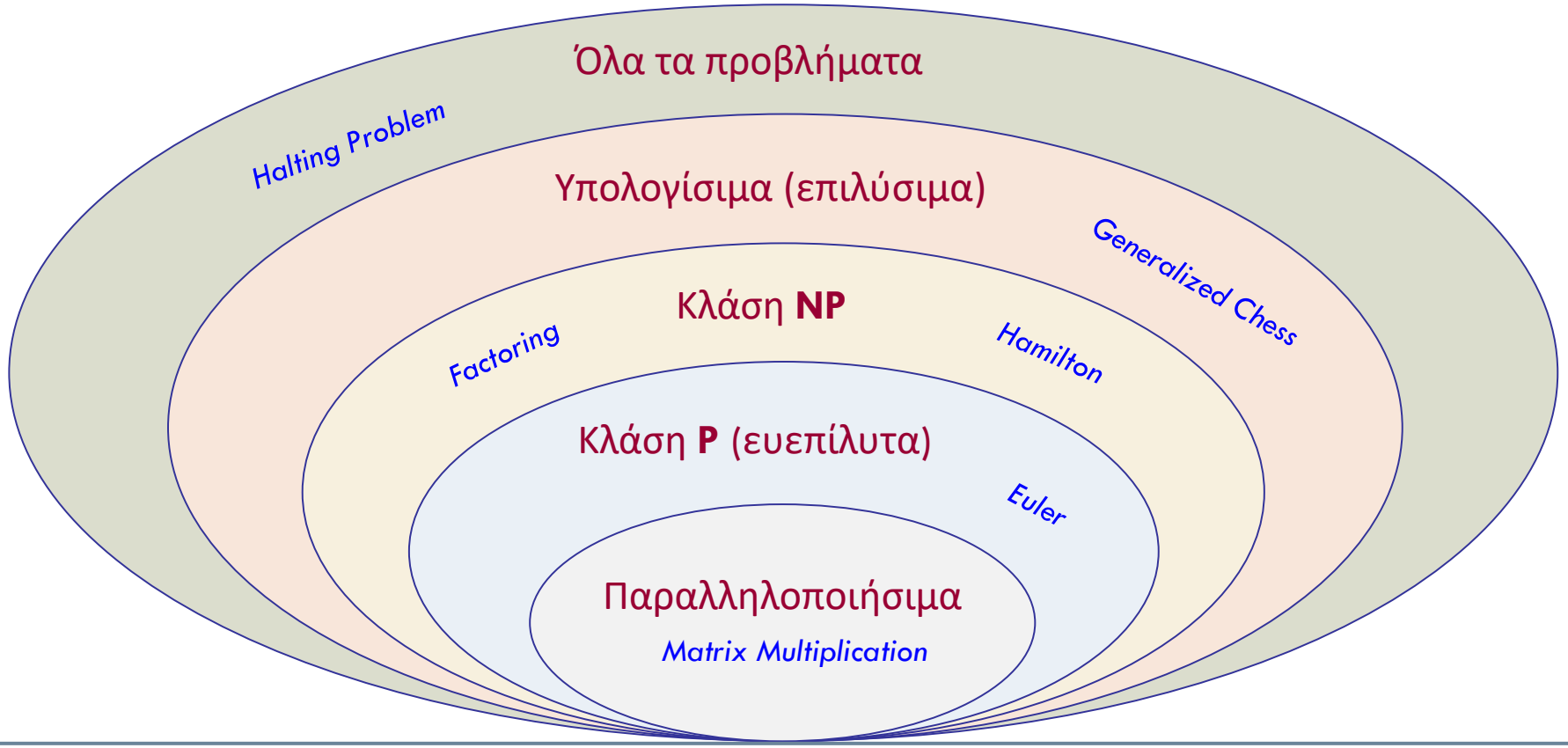
Ερωτήματα – Προβληματισμοί (VII)

5 !!!... I. II. III. IV. V.



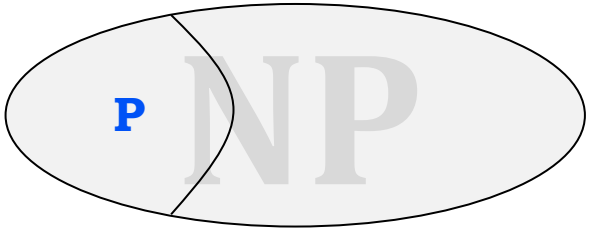
Θέσεις
Ερωτήματα

I. Κατηγοριοποίηση Προβλημάτων



II. Το μεγάλο Ερώτημα!... P =? NP

- Πόσο πιο δύσκολο είναι να βρεις τη λύση ενός προβλήματος από το να την επαληθεύσεις;



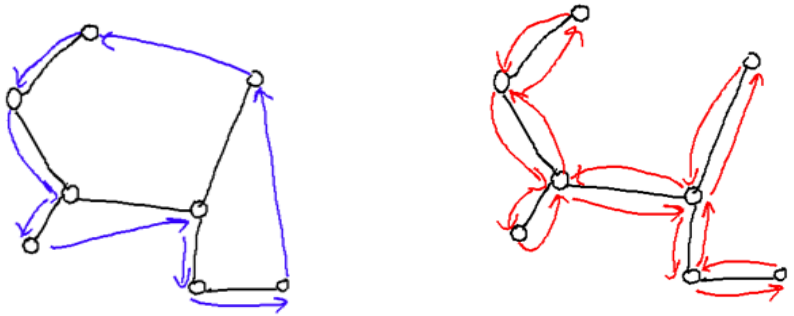
Αυτό είναι ουσιαστικά το **P =? NP** πρόβλημα,
το σημαντικότερο ανοικτό πρόβλημα της Πληροφορικής σήμερα!

Στο http://www.claymath.org προσφέρονται 1εκ. δολάρια για τη λύση του !

III. Γιατί ασχολούμαστε με NP-πληρότητα;

- Γλιτώνουμε την απόλυση (...λέμε τώρα ☺)

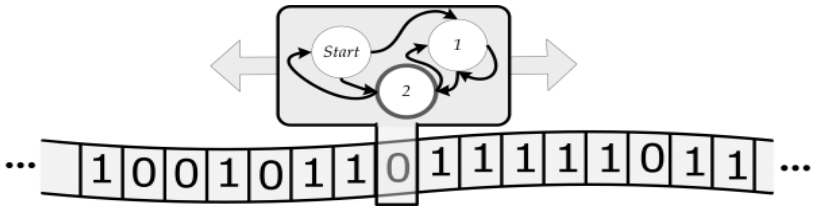
- Στροφή σε πιο ρεαλιστικές λύσεις:
  ειδικές περιπτώσεις, προσεγγιστική επίλυση !!!



- Χρήση δυσκολίας προς όφελός μας !!!
  (π.χ. κρυπτογραφία, ασφάλεια επικοινωνιών, ασφάλεια συναλλαγών, εκλογές)

IV. Μήπως κάνουμε Λάθος;

- Μήπως υπάρχει πιο «έξυπνος» τρόπος υπολογισμού;
- Αυστηρός ορισμός αλγορίθμων με χρήση υπολογιστικών μοντέλων: Alan Turing, Alonso Church, Stephen Kleene, Emil Post, Andrey Markov, κ.ά.
- Το πλέον «φυσικό» μοντέλο:
  Μηχανή Turing.



V. Θέση των Church-Turing

- Μία συνάρτηση, πάνω στους φυσικούς αριθμούς, ***είναι αλγοριθμικά υπολογίσιμη από τον άνθρωπο*** εαν-ν ***είναι υπολογίσιμη από μια μηχανή Turing***!

Ισοδύναμη διατύπωση:

- Κάθε αλγόριθμος μπορεί να περιγραφεί με τη βοήθεια μιας μηχανής Turing !

Ισοδύναμη διατύπωση:

- *Όλα τα γνωστά και άγνωστα υπολογιστικά μοντέλα είναι μηχανιστικά ισοδύναμα* !

V. Θέση των Church-Turing

- Μία συνάρτηση, πάνω στους φυσικούς αριθμούς, ***είναι αλγοριθμικά υπολογίσιμη από τον άνθρωπο*** εαν-ν ***είναι υπολογίσιμη από μια μηχανή Turing***!

Ισοδύναμη διατύπωση:

- Κάθε αλγόριθμος μπορεί να περιγραφεί με τη βοήθεια μιας μηχανής Turing !

Ισοδύναμη διατύπωση:

- Δηλαδή, για κάθε ζεύγος υπολογιστικών μοντέλων, μπορούμε με πρόγραμμα (compiler) να μεταφράζουμε αλγορίθμους από το ένα στο άλλο !!

V. Θέση των Church-Turing

- Μία συνάρτηση, πάνω στους φυσικούς αριθμούς, ***είναι αλγοριθμικά υπολογίσιμη από τον άνθρωπο*** εαν-ν ***είναι υπολογίσιμη από μια μηχανή Turing***!

- Η Θέση των Church-Turing **δεν** μπορεί να αποδειχθεί.
- Δεν υπάρχει σήμερα υπολογιστικό μοντέλο ισχυρότερο από μια μηχανή Turing.
- Θεωρητικά είναι δυνατόν, αλλά πρακτικά απίθανο να διατυπωθεί στο μέλλον ισχυρότερο υπολογιστικό μοντέλο.

Ερωτήματα – Προβληματισμοί (VIII)

Τώρα αρχίζω να Γνωρίζω !

Ποια η θέση των Αλγορίθμων στην Επιστήμη μας !

Πόσο σημαντικός τομέας !

Τι γνώση υπάρχει !

Αποτελέσματα Σταθμούς !

Ερωτήματα – Προβληματισμοί (VIII)

Ας συνεχίσουμε…

Τι είναι… **Αλγόριθμος ?**

Ερωτήματα – Προβληματισμοί (VIII)

- **Αλγόριθμος**

I. Αλγόριθμος είναι μια *καλώς ορισμένη ακολουθία απλών ενεργειών* (αριθμητικών και λογικών) που *επιφέρουν το επιθυμητό αποτέλεσμα*

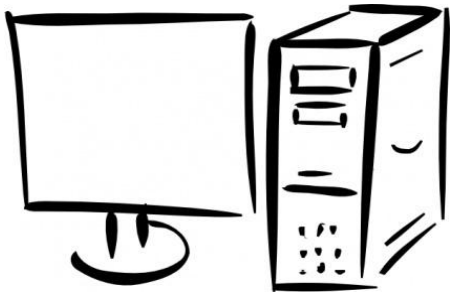
II. Αλγόριθμος είναι μια *υπολογιστική διαδικασία* για την *επίλυση ενός προβλήματος*, βασιζόμενη στην εκτέλεση μιας *καλώς ορισμένης ακολουθίας απλών ενεργειών*

III. Αλγόριθμος είναι μια *υπολογιστική διαδικασία* που *ορίζεται από μια μηχανή Turing*

Ερωτήματα – Προβληματισμοί (VIII)

Η/Υ και Αλγόριθμος

Για να κάνει κάτι ο Η/Υ πρέπει να του περιγράψουμε μια *καλώς ορισμένη ακολουθία απλών ενεργειών* (πρόσθεση, αφαίρεση, καταχώρηση στη μνήμη, ανάκληση από τη μνήμη), οι οποίες εκτελούμενες από τον Η/Υ επιφέρουν το επιθυμητό αποτέλεσμα.



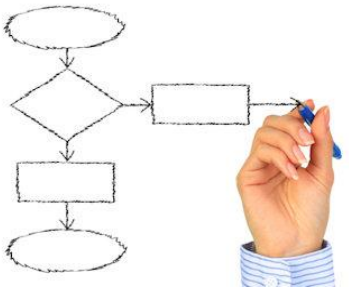
Κάθε αλγόριθμος δέχεται ως είσοδο ένα σύνολο δεδομένων και επιστρέφει ως έξοδο ένα μη-κενό σύνολο αποτελεσμάτων.

Ερωτήματα – Προβληματισμοί (VIII)

- **Πως περιγράφουμε ένα Αλγόριθμο;**

Με πολλούς τρόπους, μεταξύ των οποίων:

- ✓ φυσική γλώσσα
- ✓ διάγραμμα ροής
- ✓ ψευδοκώδικα
- ✓ γλώσσα προγραμματισμού



```
1: procedure EUCLID(a, b)
2:     r ← a mod b
3:     while r ≠ 0 do
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     end while
8:     for <some condition> do
9:         <do stuff>
10:    end for
11:    return b
12: end procedure
```

```
int main( int argc, const char* argv[] ) {

    CvCapture* capture = cvCaptureFromCAM(CV_CAP_ANY);
    IplImage *img = 0;
    IplImage *yuv = 0;

    Configuration * config = new Configuration(true); // setting red and blue detection on.
    config->setDetect(config->RED,true);
    config->setDetect(config->BLUE,true);

    ColoredObjectExtractor * coe = new ColoredObjectExtractor(config); // creating detector

    ColoredObject *coloredObjects,*biggestObject;
    int color,size,i;
```

Ερωτήματα – Προβληματισμοί (VIII)

Ιστορία…

Αλγόριθμος του Ευκλείδη

- Οι αλγόριθμοι προϋπάρχουν των Η/Υ
- Ο αλγόριθμος του Ευκλείδη για υπολογισμό του ΜΚΔ:

$$\gcd(x, y) = \gcd(y, x \bmod y)$$

όπου $x \geq y$ θετικοί ακέραιοι.

Ευκλείδης (300 πΧ)

```
int Euclid(int x, int y)
{
  if y==0 return x;
  return Euclid(y, x%y);
}
```

Παράδειγμα

```
Euclid (128,40) =
  Euclid (40,8) =
   Euclid (8,0) = 8
```

Ερωτήματα – Προβληματισμοί (VIII)

Ιστορία…

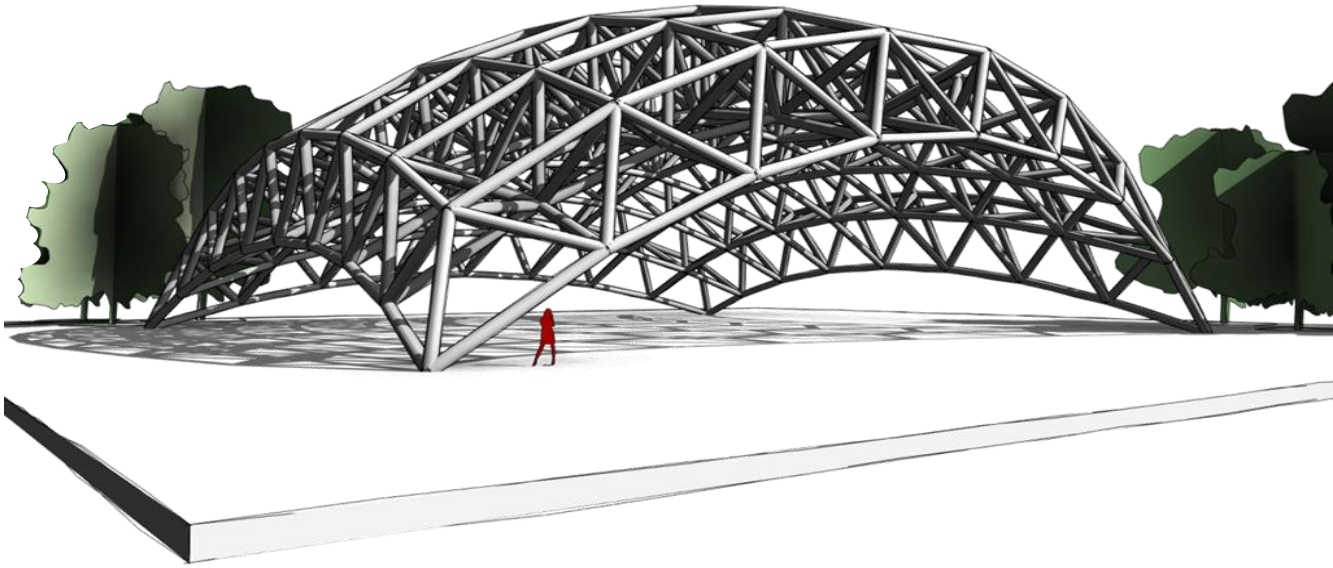
Αλ-Χουαρίζμι

- Η λέξη ***αλγόριθμος*** προέρχεται από το όνομα του Πέρση μαθηματικού, αστρονόμου και γεωγράφου Abu Abdulla Al-Khwarizmi που έζησε τον 9ο αιώνα.
- Το όνομα του Al-Khwarizmi μεταφράστηκε στα Λατινικά ως **Algoritmi** (*Αλγκορίτμι*), από την οποία προέκυψε ο όρος *Αλγόριθμος*.

Αλ-Χουαρίζμι (~781 - 850 μΧ)

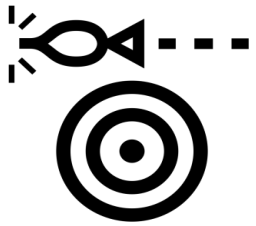
- Το βιβλίο του *Υπολογισμός με Ινδικούς Αριθμούς*, γραμμένο περίπου το 825 μΧ, συνέβαλε στη διάδοση του Ινδικού αριθμητικού συστήματος σε όλη την Μέση Ανατολή και Ευρώπη. Το βιβλίο μεταφράστηκε στα Λατινικά ως *Algoritmi de numero Indorum.*

Σχεδίαση Αλγόριθμου



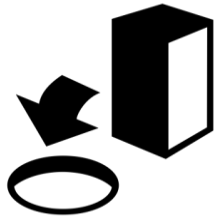
Σχεδίαση, Ορθότητα & Πολυπλοκότητα

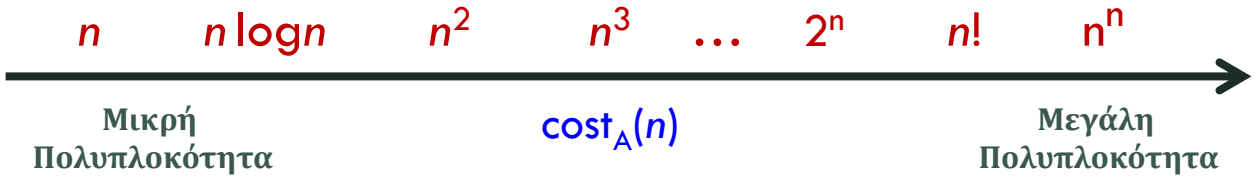
Ποιος είναι ο Στόχος μας !!!



- Σχεδίαση
  Αποτελεσματικών Αλγορίθμων

- Απόδειξη
  Ορθότητας Αλγορίθμων
  Δυσεπιλυσιμότητας ή Μη-επιλυσιμότητας



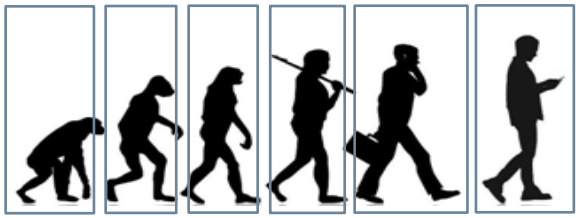


Σχεδιασμός Αποτελεσματικών Αλγορίθμων

- Οι περισσότεροι Αλγόριθμοι περιγράφονται καλύτερα ως:
  - Επαναληπτικοί Αλγόριθμοι
  - Αναδρομικοί Αλγόριθμοι

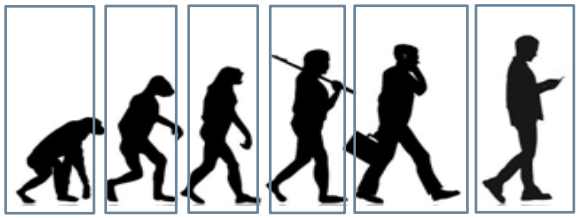
Σχεδιασμός Αποτελεσματικών Αλγορίθμων

- Οι περισσότεροι Αλγόριθμοι περιγράφονται καλύτερα ως:
  - Επαναληπτικοί Αλγόριθμοι
  - Αναδρομικοί Αλγόριθμοι



Σχεδιασμός Αποτελεσματικών Αλγορίθμων

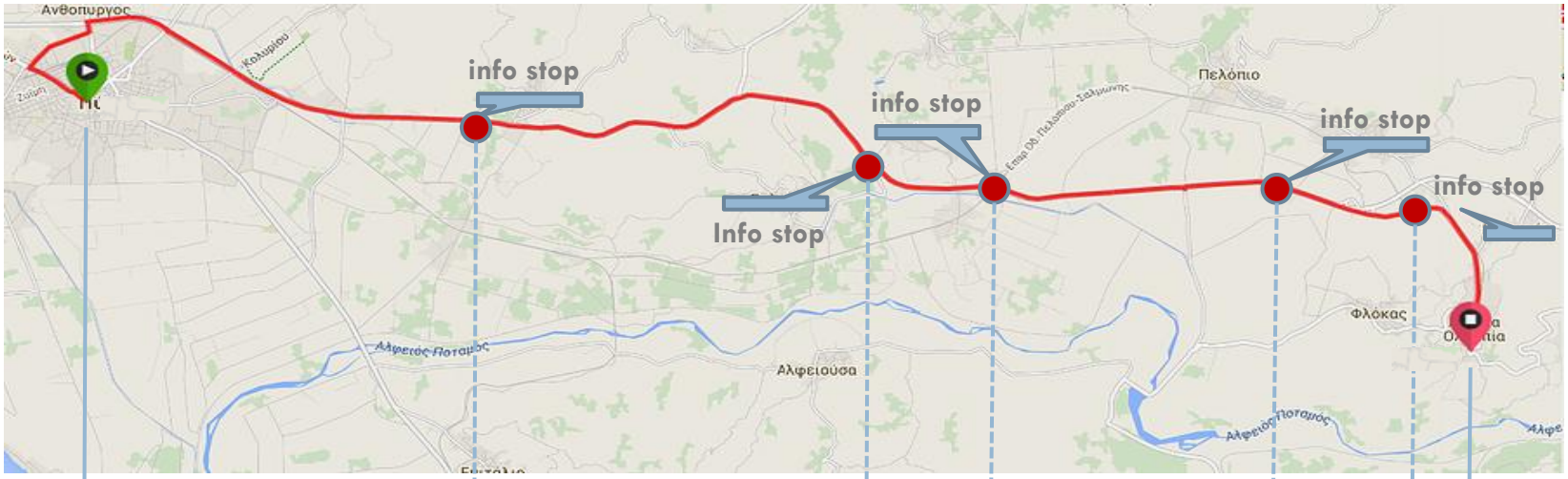
- Οι περισσότεροι Αλγόριθμοι περιγράφονται καλύτερα ως:
  - Επαναληπτικοί Αλγόριθμοι
  - Αναδρομικοί Αλγόριθμοι



Σχεδίαση, Ορθότητα & Πολυπλοκότητα

Επαναληπτικοί Αλγόριθμοι !!!

```
Fib(n)
if n=0 then return 0
f[0]=0 f[1]=1
for i=2 to n do
     f[i] = f[i-1] + f[i-2]
return f[n]
```

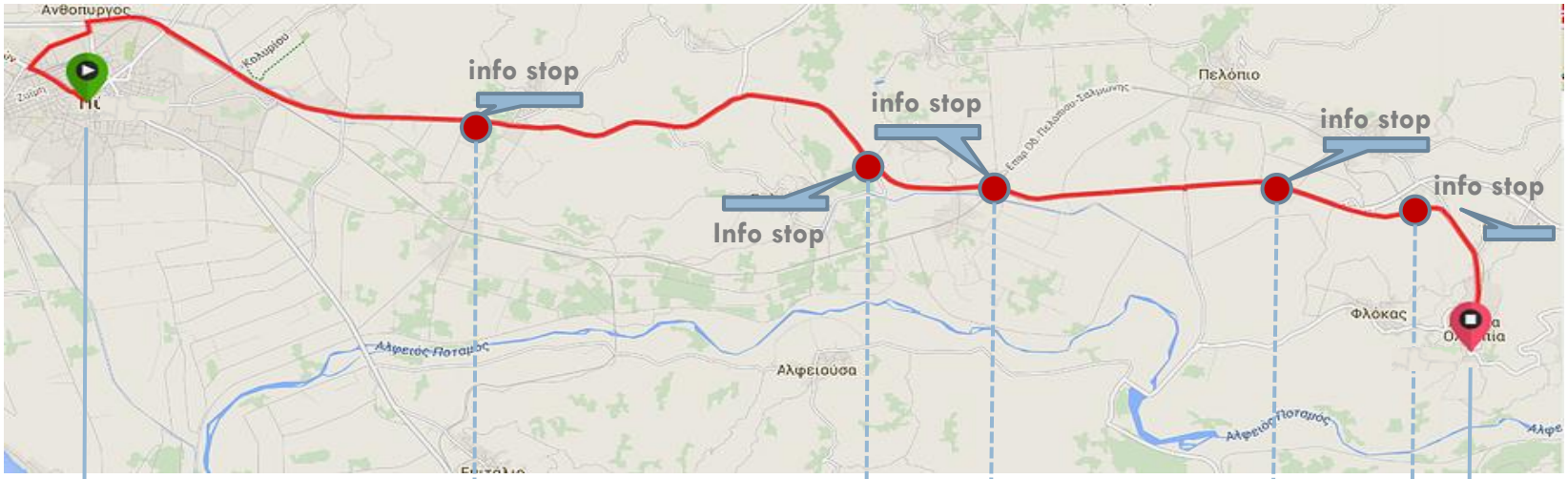


Επαναληπτικοί Αλγόριθμοι !!!

Παράδειγμα: **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …**

↑ **Fib(6) = 8**

```
Fib(n)
if n=0 then return 0
f[0]=0 f[1]=1
for i=2 to n do
     f[i] = f[i-1] + f[i-2]
return f[n]
```



Είσοδος Βήμα_1 Βήμα_2 Βήμα_3 Βήμα_4 Βήμα_5 Έξοδος

n = 4

F(0)=0 **Fib(2)=1** **Fib(3)=2** **Fib(4)=3** **Fib(5)=5** **Fib(6)=8**

F(1)=1

Αναδρομικοί Αλγόριθμοι !!!

```
Fib(n)
    if n=0 then return 0
    if n=1 then return 1
return Fib(n-1) + Fib(n-2)
```

Σενάριο 1

Αναδρομικοί Αλγόριθμοι !!!

Παράδειγμα 1: **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …**

Fib(4) = 3

```
Fib(n)
    if n=0 then return 0
    if n=1 then return 1
return Fib(n-1) + Fib(n-2)
```

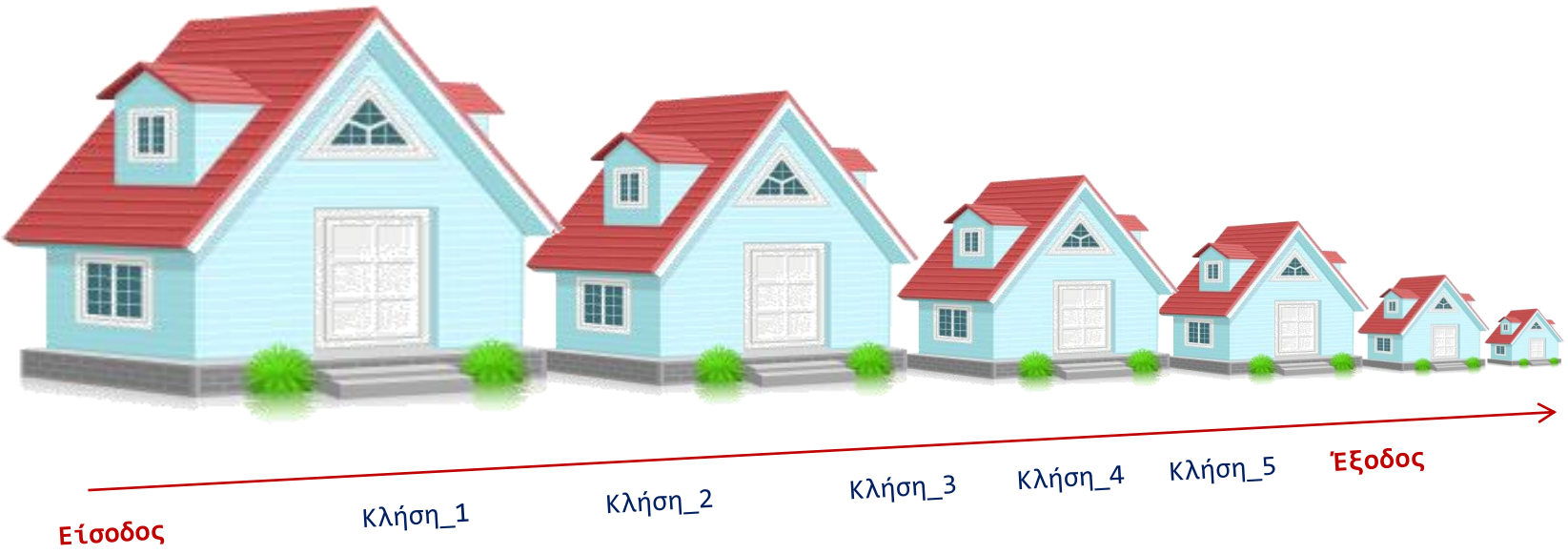
```
Fib(4) = Fib(3)+Fib(2)
Fib(3) = Fib(2)+Fib(1)
Fib(2) = Fib(1)+Fib(0)
Fib(1) = 1
Fib(0) = 0
```

3

Αναδρομικοί Αλγόριθμοι !!!

```
Euclid(x, y)
  if y==0 return x;
  z = x % y
return Euclid(y, z);
```

Σενάριο 2



Αναδρομικοί Αλγόριθμοι !!!

Παράδειγμα 2: **ΜΚΔ (128, 40) = 8**

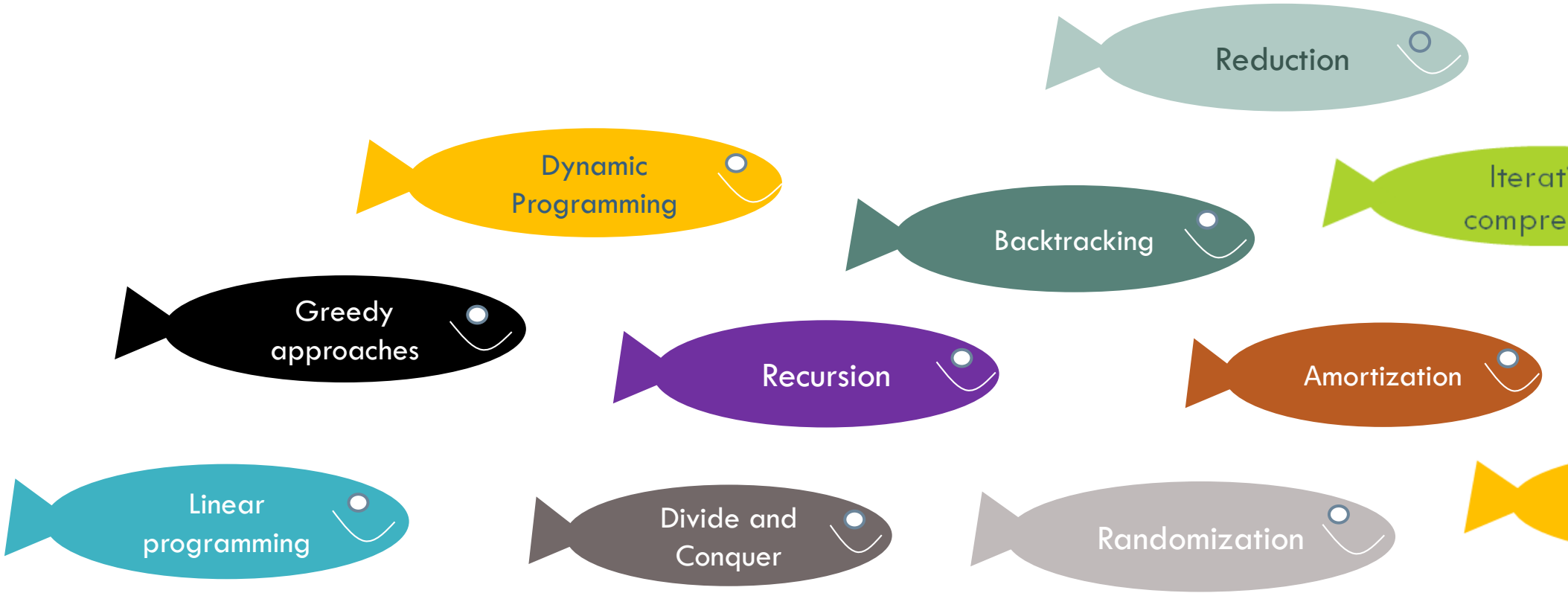
```
Euclid(x, y)
  if y==0 return x;
  z = x % y
return Euclid(y, z);
```

Euclid (128,40) =
Euclid (40,8) =
Euclid (8,0) = 8

Σχεδίαση, Ορθότητα & Πολυπλοκότητα

Αλγοριθμικές Τεχνικές Σχεδίασης !!!

- Εργαλεία Σχεδίασης



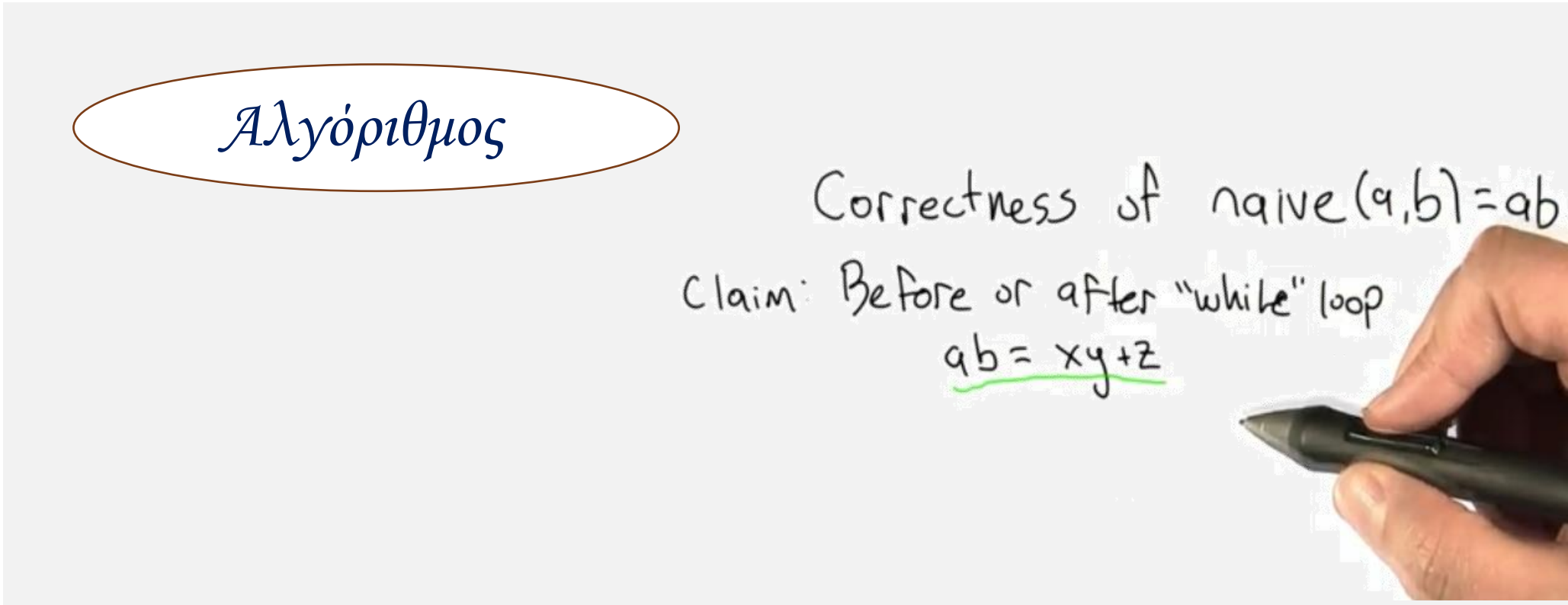
Αλγοριθμικές Τεχνικές Σχεδίασης !!!

- Παίρνοντας το Πτυχίο μας θα πρέπει να γνωρίζουμε:

  - ✓ Αναδρομή (recursion)
  - ✓ Διαίρει και κυρίευε (divide and conquer)
  - ✓ Δυναμικός προγραμματισμός (dynamic programming)
  - ✓ Απληστία (greedy algorithms)
  - ✓ Γραμμικός Προγραμματισμός (linear programming)
  - ✓ Αναγωγές (reduction)

Ορθότητα Αλγόριθμου

Αλγόριθμος



Σχεδίαση, Ορθότητα & Πολυπλοκότητα

Ορθότητα Αλγόριθμου

Αποδεικνύω ότι:

Ο αλγόριθμος A δίδει σωστό αποτέλεσμα για κάθε είσοδο $I_A(n)$ μεγέθους n !!!

Προσοχή !!!

Πρώτα απόδειξη ορθότητα και μετά: C, C++, Java ... και ό,τι άλλο !!!

Σχεδίαση, Ορθότητα & Πολυπλοκότητα

- Στα μαθηματικά και στις άλλες επιστήμες κάνουμε συχνά υποθέσεις. Όταν δείξουμε ότι μια υπόθεση είναι αληθής, τότε την ονομάζουμε θεώρημα ή πρόταση.
- Τα μαθηματικά που διδασκόμαστε στο σχολείο και στο Πανεπιστήμιο, αποτελούνται συνήθως από ορισμούς, θεωρήματα και αποδείξεις που μας δίνονται έτοιμες.
- Η πιο ενδιαφέρουσα πλευρά των μαθηματικών είναι όταν εξερευνούμε τα σύνορα της γνώσης. Εκεί πρέπει να κάνουμε υποθέσεις και μετά να τις αποδείξουμε ή να τις καταρρίψουμε.
- Υπόθεση ⟷ Απόδειξη ⟷ Θεώρημα

Δύο ενδιαφέροντα κείμενα !

Στη πληροφορική συχνά επινοούμε και σχεδιάζουμε αλγορίθμους για την επίλυση προβλημάτων. Όταν δείξουμε ότι ένας αλγόριθμος είναι σωστός τότε καταλήγουμε σε ένα θεώρημα ή πρόταση.

Η πληροφορική που διδασκόμαστε στο σχολείο και στο Πανεπιστήμιο, περιλαμβάνει συνήθως αλγορίθμους και αποδείξεις ορθότητας που μας δίδονται έτοιμες.

Η πιο ενδιαφέρουσα πλευρά της πληροφορικής είναι όταν εξερευνούμε τα σύνορα της γνώσης. Εκεί πρέπει να επινοήσουμε αλγορίθμους και μετά να αποδείξουμε την ορθότητά τους ή να τους απορρίψουμε.

Αλγόριθμος ⟷ Απόδειξη ορθότητας ⟷ Θεώρημα

Σχεδίαση, Ορθότητα & Πολυπλοκότητα

- Στα μαθηματικά και στις άλλες επιστήμες κάνουμε συχνά υποθέσεις. Όταν δείξουμε ότι μια υπόθεση είναι αληθής, τότε την ονομάζουμε θεώρημα ή πρόταση.
- Τα μαθηματικά που διδασκόμαστε στο σχολείο και στο Πανεπιστήμιο, αποτελούνται συνήθως από ορισμούς, θεωρήματα και αποδείξεις που μας δίνονται έτοιμες.
- Η πιο ενδιαφέρουσα πλευρά των μαθηματικών είναι όταν εξερευνούμε τα σύνορα της γνώσης. Εκεί πρέπει να κάνουμε υποθέσεις και μετά να τις αποδείξουμε ή να τις καταρρίψουμε.
- Υπόθεση ⟷ Απόδειξη ⟷ Θεώρημα
- Πολλές φορές, όταν δεν μπορούμε να αποδείξουμε μια υπόθεση, την αναθεωρούμε.

Δύο ενδιαφέροντα κείμενα !

Στη πληροφορική συχνά επινοούμε και σχεδιάζουμε αλγορίθμους για την επίλυση προβλημάτων. Όταν δείξουμε ότι ένας αλγόριθμος είναι σωστός τότε καταλήγουμε σε ένα θεώρημα ή πρόταση.

Η πληροφορική που διδασκόμαστε στο σχολείο και στο Πανεπιστήμιο, περιλαμβάνει συνήθως αλγορίθμους και αποδείξεις ορθότητας που μας δίδονται έτοιμες.

Η πιο ενδιαφέρουσα πλευρά της πληροφορικής είναι όταν εξερευνούμε τα σύνορα της γνώσης. Εκεί πρέπει να επινοήσουμε αλγορίθμους και μετά να αποδείξουμε την ορθότητά τους ή να τους απορρίψουμε.

Αλγόριθμος ⟷ Απόδειξη ορθότητας ⟷ Θεώρημα

Πολλές φορές, όταν δεν μπορούμε να αποδείξουμε την ορθότητα ενός αλγόριθμου, τον αναθεωρούμε .

Σχεδίαση, Ορθότητα & Πολυπλοκότητα

- Στα μαθηματικά και στις άλλες επιστήμες κάνουμε συχνά υποθέσεις. Όταν δείξουμε ότι μια υπόθεση είναι αληθής, τότε την ονομάζουμε θεώρημα ή πρόταση.
- Τα μαθηματικά που διδασκόμαστε στο σχολείο και στο Πανεπιστήμιο, αποτελούνται συνήθως από ορισμούς, θεωρήματα και αποδείξεις που μας δίνονται έτοιμες.
- Η πιο ενδιαφέρουσα πλευρά των μαθηματικών είναι όταν εξερευνούμε τα σύνορα της γνώσης. Εκεί πρέπει να κάνουμε υποθέσεις και μετά να τις αποδείξουμε ή να τις καταρρίψουμε.
- Υπόθεση ⟷ Απόδειξη ⟷ Θεώρημα
- Άλλες φορές, όταν καταφέρνουμε να αποδείξουμε μια υπόθεση, η ίδια η απόδειξη μας βοηθάει να γενικεύσουμε την πρόταση.

Δύο ενδιαφέροντα κείμενα !

Στη πληροφορική συχνά επινοούμε και σχεδιάζουμε αλγορίθμους για την επίλυση προβλημάτων. Όταν δείξουμε ότι ένας αλγόριθμος είναι σωστός τότε καταλήγουμε σε ένα θεώρημα ή πρόταση.

Η πληροφορική που διδασκόμαστε στο σχολείο και στο Πανεπιστήμιο, περιλαμβάνει συνήθως αλγορίθμους και αποδείξεις ορθότητας που μας δίδονται έτοιμες.

Η πιο ενδιαφέρουσα πλευρά της πληροφορικής είναι όταν εξερευνούμε τα σύνορα της γνώσης. Εκεί πρέπει να επινοήσουμε αλγορίθμους και μετά να αποδείξουμε την ορθότητά τους ή να τους απορρίψουμε.

Αλγόριθμος ⟷ Απόδειξη ορθότητας ⟷ Θεώρημα

Άλλες φορές, όταν καταφέρνουμε να αποδείξουμε την ορθότητα ενός αλγόριθμου, η ίδια η απόδειξη μας βοηθάει να γενικεύσουμε το πρόβλημα που επιλύει ο αλγόριθμος.

Πολυπλοκότητα Αλγόριθμου

Αλγόριθμος

Πολυπλοκότητα Αλγόριθμου

Αλγόριθμος

Αποδεικνύω ότι:

Ο αλγόριθμος A για την εκτέλεσή του απαιτεί πόρους $\leq cost_A(n)$ για κάθε είσοδο $I_A(n)$ μεγέθους n

Προσοχή !!!

Πολυπλοκότητα Αλγόριθμου $= cost_A(n) = f(\textit{Μέγεθος Εισόδου})$

Σχεδίαση, Ορθότητα & Πολυπλοκότητα

Πολυπλοκότητα Αλγόριθμου

- Η *πολυπλοκότητα ενός αλγόριθμου Α* εκφράζει το *πόσο καλά μπορεί να εκτελεστεί ο αλγόριθμος, δηλαδή*
  - πόσο γρήγορα, ή
  - με πόση μνήμη, ή
  - με πόσους επεξεργαστές (παραλληλία), ή
  - με πόση κατανάλωση ενέργειας (sensor networks), κ.λπ.

Υπολογιστικοί Πόροι

Έχουν κόστος !!!
Και το κόστος σε πόρους *αυξάνει καθώς αυξάνει το* μεγέθους n της εισόδου $I_A(n)$ του αλγόριθμου

Πολυπλοκότητα Αλγόριθμου

- Το *κόστος ενός αλγόριθμου A* ορίζεται ως η μέγιστη ποσότητα των *υπολογιστικών πόρων* που απαιτεί η εκτέλεσή του σε σχέση με το *μέγεθος της εισόδου του*:

$$\text{cost}_A(n) = \max \{\text{κόστος αλγόριθμου A για είσοδο x}\}$$

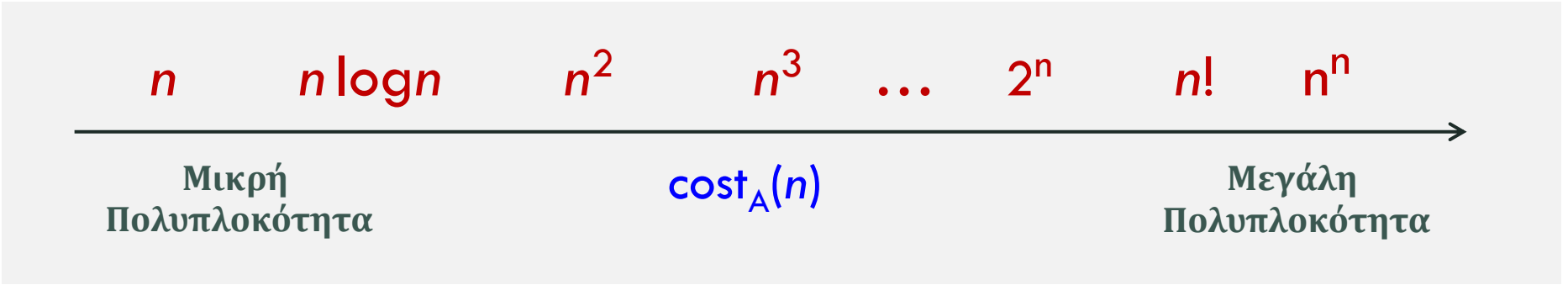
για όλες τις εισόδους
x μήκους ***n***

- *Πολυπλοκότητα του αλγόριθμου A* = $\text{cost}_A(n)$

Πολυπλοκότητα Αλγόριθμου

- *Πολυπλοκότητα του αλγόριθμου A :*

$$\text{cost}_A(n) = \begin{cases} \text{time-cost}_A(n) & T_A(n) \\ \text{space-cost}_A(n) \quad \text{ή} & S_A(n) \\ \text{processor-cost}_A(n) & P_A(n) \end{cases}$$



Πολυπλοκότητα Αλγόριθμου

- *Πολυπλοκότητα του αλγόριθμου A :*

$$\text{cost}_A(n) = \begin{cases} \text{time-cost}_A(n) & T_A(n) \\ \text{space-cost}_A(n) \quad \text{ή} & S_A(n) \\ \text{processor-cost}_A(n) & P_A(n) \end{cases}$$

Παράδειγμα: Για τον αλγόριθμο ταξινόμησης MergeSort (MS) ισχύει:

$$T_{MS}(n) \leq c\, n \log n$$

όπου, c σταθερά

Πολυπλοκότητα Αλγόριθμου

- **Απλοποιήσεις**
  - Συχνά θεωρούμε ως μέγεθος της εισόδου ***μόνο*** *το πλήθος των δεδομένων* (αγνοώντας το μέγεθός τους σε bits).
    - Αυτό δεν δημιουργεί πρόβλημα εφ' όσον ο αλγόριθμος δεν περιέχει πράξεις ή διαδικασίες που να κοστίζουν *εκθετικά* ως προς το μέγεθος των δεδομένων σε bits.
    - Αυτό λέγεται *αριθμητική πολυπλοκότητα* (arithmetic complexity) και είναι συνήθως αρκετά ακριβής μέτρηση.

      Η ανάλυση πολυπλοκότητας σε πλήθος πράξεων ψηφίων λέγεται *bit complexity*.

Πολυπλοκότητα Αλγόριθμου

- **Απλοποιήσεις**
  - Επίσης θεωρούμε ότι κάθε στοιχειώδης πράξη (πρόσθεση, πολλαπλασιασμός, σύγκριση) *έχει κόστος 1*.

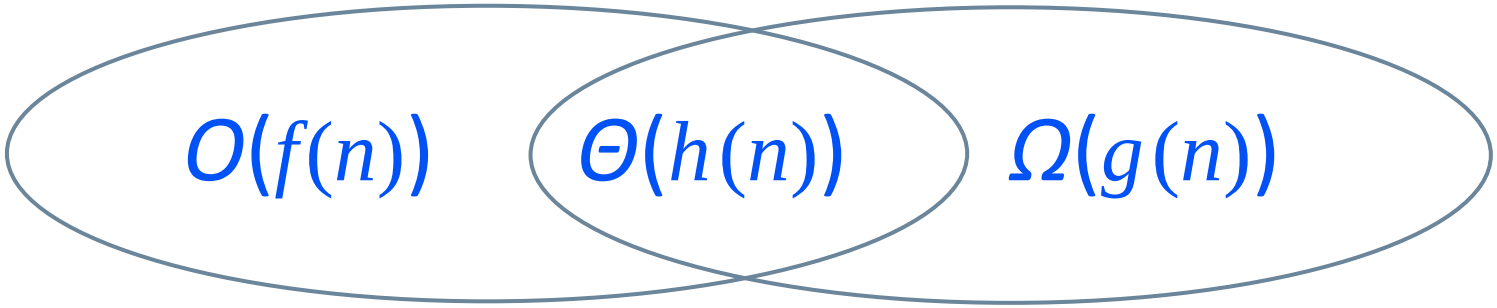
Βασική Πράξη

$$T_A(n) = \#(\text{Βασικών Πράξεων}) = f(\textit{Μέγεθος Εισόδου})$$

Είσοδος αλγορίθμου	$\Leftrightarrow$	Στιγμιότυπο
Μέγεθος Εισόδου Αλγορίθμου	$\Leftrightarrow$	Μέγεθος Στιγμιότυπου

Πολυπλοκότητα Αλγόριθμου

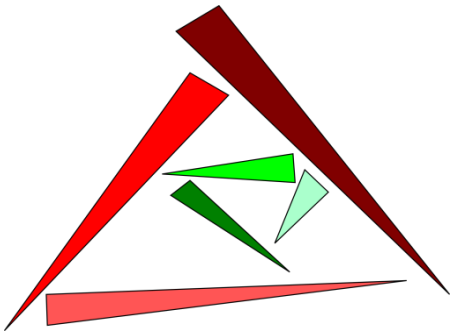
- Απλοποιήσεις
  - Ασυμπωτικοί Συμβολισμοί



  - Παράδειγμα: $T_{MS}(n) \leq c\ n\ \log n = O(n\ \log n)$

Πολυπλοκότητα Αλγόριθμου

- Είδη Πολυπλοκότητας



  - Χειρότερης περίπτωσης (worst case):
    με αυτήν ασχολούμαστε συνήθως.
  - Μέσης περίπτωσης (average case):
    με βάση κατανομή πιθανότητας στιγμιότυπων (instances) του προβλήματος. Συνήθως δύσκολο να οριστεί σωστά.
  - Αντισταθμιστική (amortized):
    εκφράζει την μέση αποδοτικότητα σε μια σειρά επαναλήψεων του αλγορίθμου.

Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

Αλγόριθμοι

Πολυωνυμικοί

$O(n^k),\ k \geq 1$

Πολυωνυμικό Χρόνο Εκτέλεσης

Εκθετικοί

$O(c^n),\ c>1$

Εκθετικό Χρόνο Εκτέλεσης

Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

Αλγόριθμοι

Πολυωνυμικοί

Εκθετικοί

$O(n^k)$, $k \geq 1$

$O(c^n)$, $c>1$

$n, \quad \sqrt{n}, \quad 50n^3, \quad 5000n^4, \quad n^{10}$

$2^n, \quad n!, \quad n^n$

Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

Αλγόριθμοι

Πολυωνυμικοί

Εκθετικοί

$O(n^k)$, *όταν* $k >> 1$?

$O(c^n)$, $c>1$

$n, \quad \sqrt{n}, \quad 50n^3, \quad 5000n^4, \quad n^{10}$

$2^n, \quad n!, \quad n^n$

Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

Αλγόριθμοι

Πολυωνυμικοί

Εκθετικοί

$O(n^k)$, *όταν* $k >> 1$?

$O(c^n), c>1$

$n = 3$

n^{10}

2^n

Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

Αλγόριθμοι

Πολυωνυμικοί

Εκθετικοί

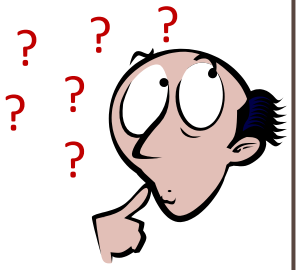
$O(n^k)$, *όταν* $k >> 1$?

$O(c^n), c>1$

$n = 3$

$59049 = n^{10}$

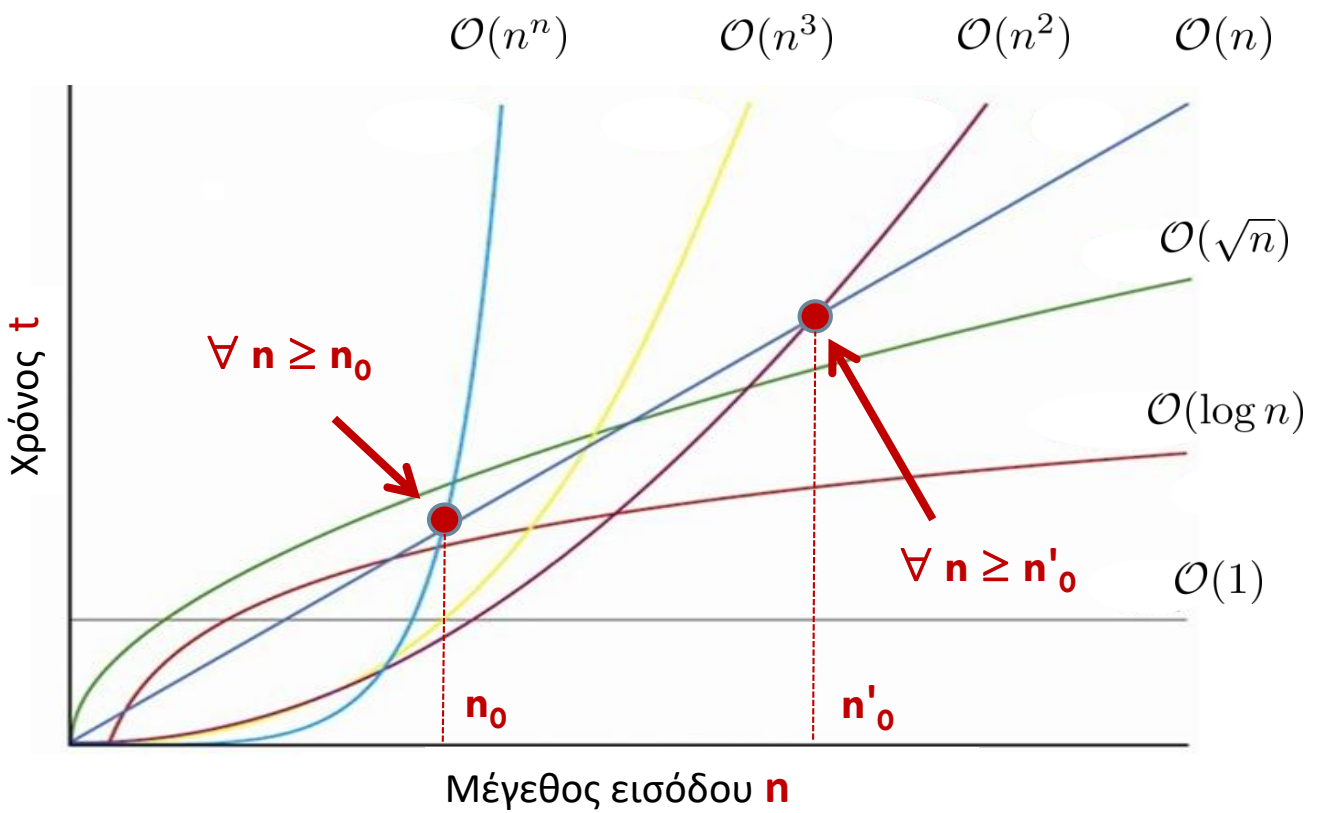
$2^n = 8$



Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

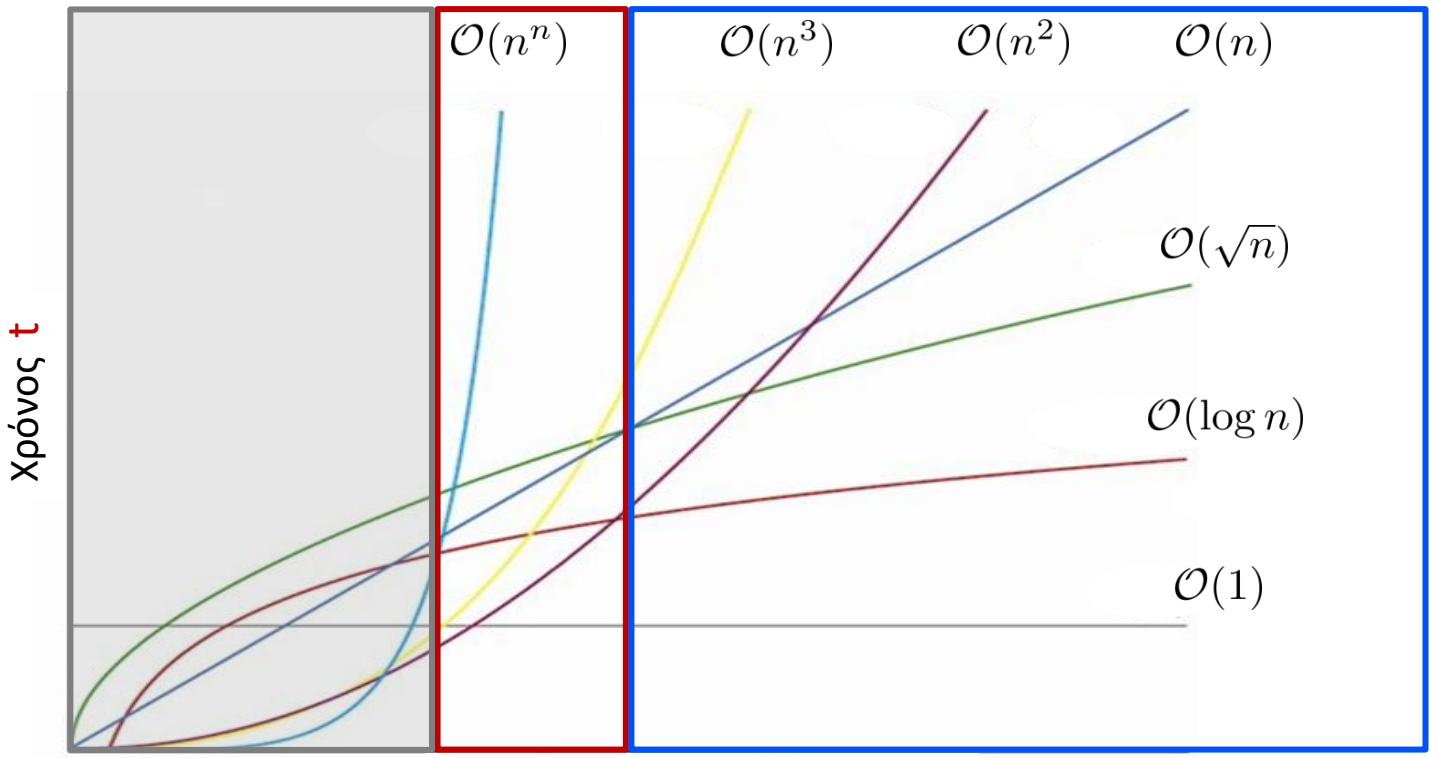
Υπάρχει n_0 τέτοιο ώστε…



Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

Μεγάλες και Μικρές Τιμές Εισόδου



Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

- **Αποτελεσματικός (efficient ή "fast")**

  Εάν η πολυπλοκότητα του αλγόριθμου είναι πολυωνυμική (polynomial) ως προς το μέγεθος της εισόδου (polynomial in the input size, say n):

$$\sqrt{n}, \quad 5n^4, \quad n \log n$$

- **Μη-αποτελεσματικός Αλγόριθμος (inefficient)**

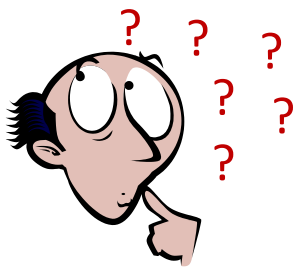
  Εάν η πολυπλοκότητα του αλγόριθμου εκθετική (exponential) ως προς το μέγεθος της εισόδου:

$$2^n, \quad n!, \quad n^n, \quad n^{n-2}$$

Πολυπλοκότητα Αλγόριθμου

Πολυπλοκότητα

Πρόβλημα Π



Πολυπλοκότητα Χρόνου

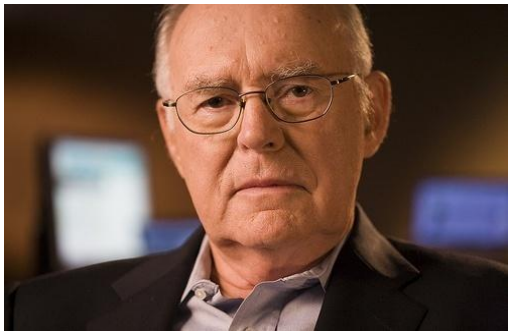
Μέγεθος Εισόδου n

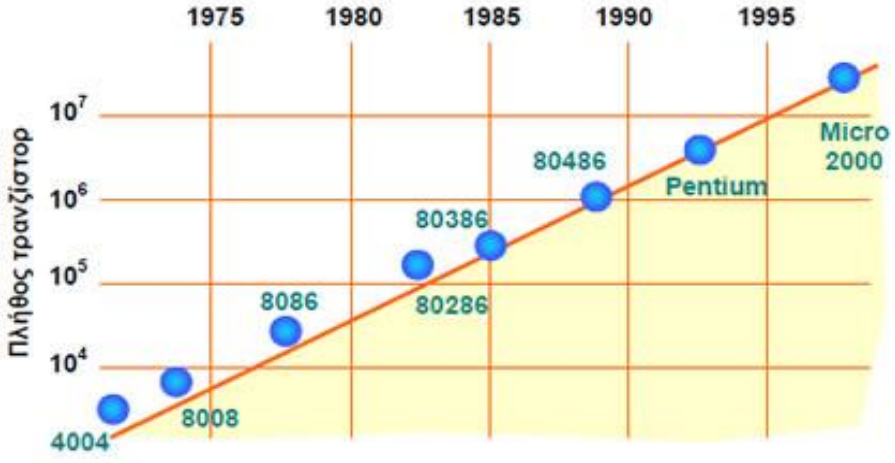
	2	**8**	**32**	**64**
log *n*	1 δεύτερο	3 δεύτερα	5 δεύτερα	6 δεύτερα
n	2 δεύτερα	8 δεύτερα	32 δεύτερα	1.07 λεπτά
n* log *n	2 δεύτερα	24 δεύτερα	2.67 λεπτά	6.4 λεπτά
n^2	4 δεύτερα	1.07 λεπτά	17.07 λεπτά	1.14 ώρες
n^3	8 δεύτερα	8.53 λεπτά	9.1 ώρες	3.03 ημέρες
2^n	4 Δεύτερα	4.27 λεπτά	1.36 αιώνες	5.86×10^9 αιώνες
$n!$	2 Δεύτερα	4.2 ώρες	8.34×10^{25} αιώνες	4.02×10^{79} αιώνες

Ιστορία του Moore

Νόμος του Moore 1965

Ως **«Νόμος του Μουρ»** ονομάζεται η παρατήρηση πως το πλήθος των τρανζίστορ ενός ολοκληρωμένου κυκλώματος **διπλασιάζεται κάθε δύο χρόνια** !!!





Διατηρεί την ισχύ του αξιοσημείωτα καλά επί 40 χρόνια!!!

Ιστορία του Moore

Νόμος του Moore 1965

- ✓ Ο Νόμος του Moore **δίνει αντικίνητρα** για την ανάπτυξη πολυωνυμικών αλγορίθμων **?**

  Δηλαδή, εάν έχουμε έναν εκθετικό αλγόριθμο Α, γιατί να αναπτύξουμε έναν άλλο πολυωνυμικό (εάν μπορούμε) και να μην περιμένουμε μέχρι να κάνει εφικτό τον εκθετικό αλγόριθμο Α ο Νόμος του Moore !!!

- ✓ Συμβαίνει το αντίθετο !!!

  Ο Νόμος του Moore αποτελεί ένα **τεράστιο κίνητρο** για την ανάπτυξη αποδοτικών αλγορίθμων!!!!

Ιστορία του Moore

Νόμος του Moore 1965

Ορίστε γιατί !!!

- ✓ Εάν ένας αλγόριθμος **O(2^n)** για το πρόβλημα SAT είχε στη διάθεσή του **1 ώρα**, θα μπορούσε να λύσει το πρόβλημα:
  - με 25 μεταβλητές το 1975,
  - με 31 μεταβλητές το 1985,
  - με 38 μεταβλητές το 1995, και
  - με περίπου 45 μεταβλητές με τους σημερινού Η/Υ !!!
- ✓ Κάθε μία **επιπλέον μεταβλητή του SAT** απαιτεί **ΔΙΠΛΑΣΙΟ χρόνο αναμονής** του αλγόριθμου για να λύση το πρόβλημα!!!

Ιστορία του Moore

Νόμος του Moore 1965

Αντίθετα…

✓ Το μέγεθος των στιγμιότυπων που επιλύει ένας αλγόριθμος **O(n)** ή **O(n logn)** **100-ΠΛΑΣΙΑΖΕΤΕ κάθε δεκαετία** !!!

Οι **Εκθετικοί Αλγόριθμοι** παρουσιάζουν **πολυωνυμικά αργή πρόοδο**!!!

ενώ

οι **Πολυωνυμικοί Αλγόριθμοι** παρουσιάζουν **εκθετικά γρήγορη πρόοδο**!!!

Πολυπλοκότητα Αλγόριθμου

Αλγόριθμος

Έστω ότι:

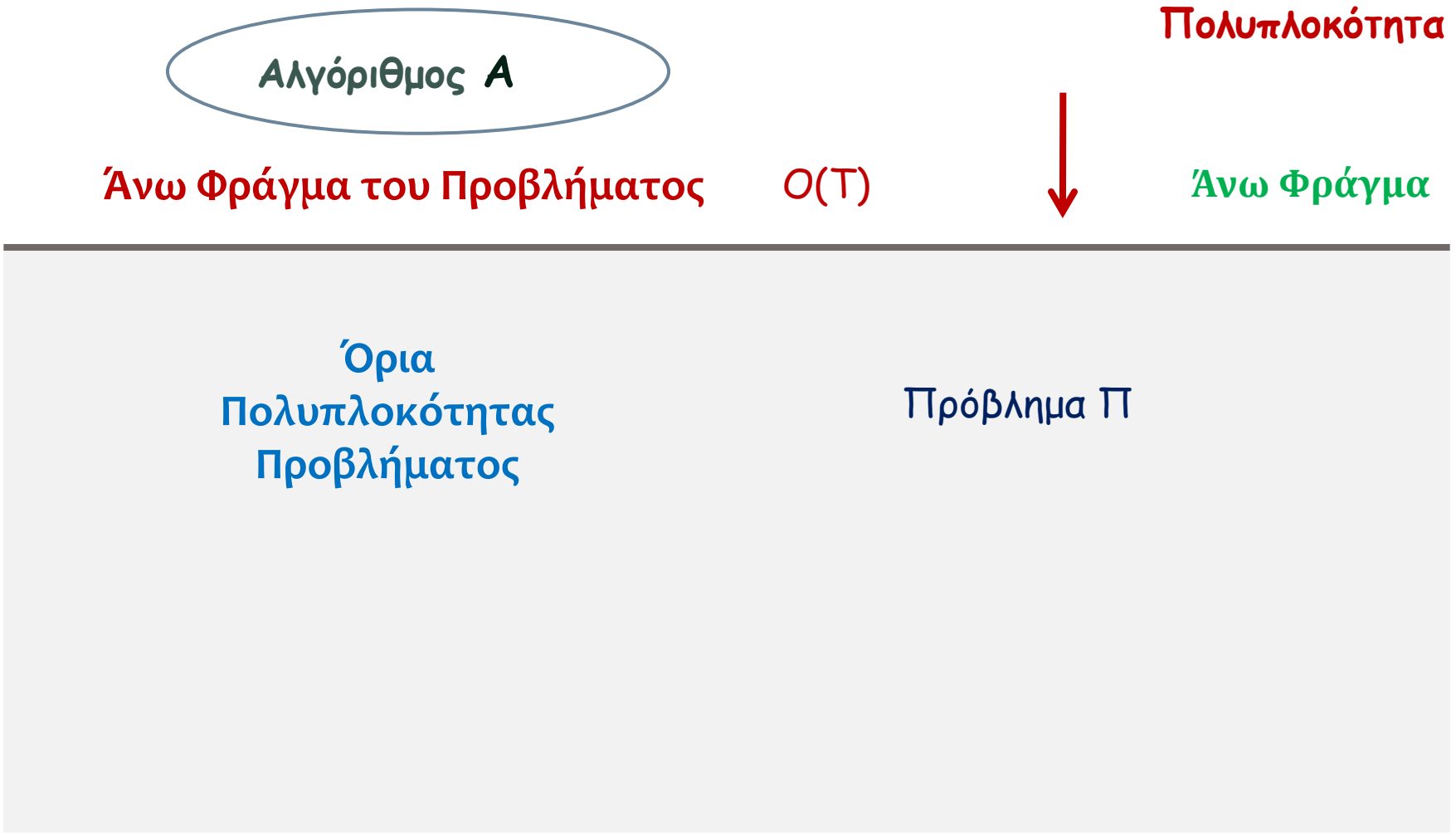
Ο αλγόριθμος Α, που επιλύει το πρόβλημα Π, για την εκτέλεσή του απαιτεί χρόνο $\leq T_A(n) = O(T)$ για κάθε είσοδο $I_A(n)$ μεγέθους n

Ο αλγόριθμος Α ορίζει ένα **άνω φράγμα** του προβλήματος Π που επιλύει !!!

Άνω Φράγμα του Π $\Rightarrow$ $T_A(n) = O(T)$

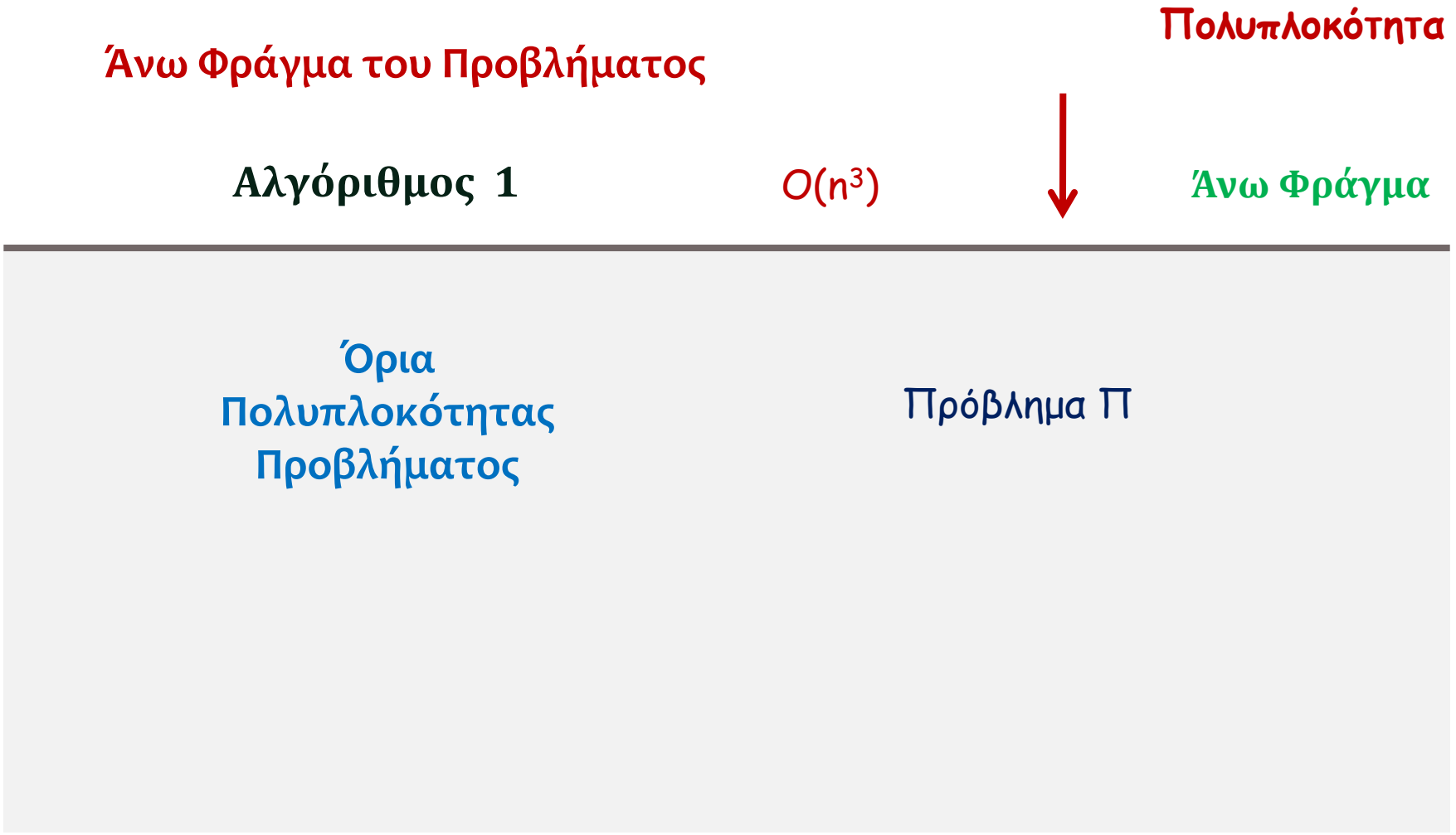
Πολυπλοκότητα Αλγόριθμου και Άνω Φράγματα

Σχηματικά



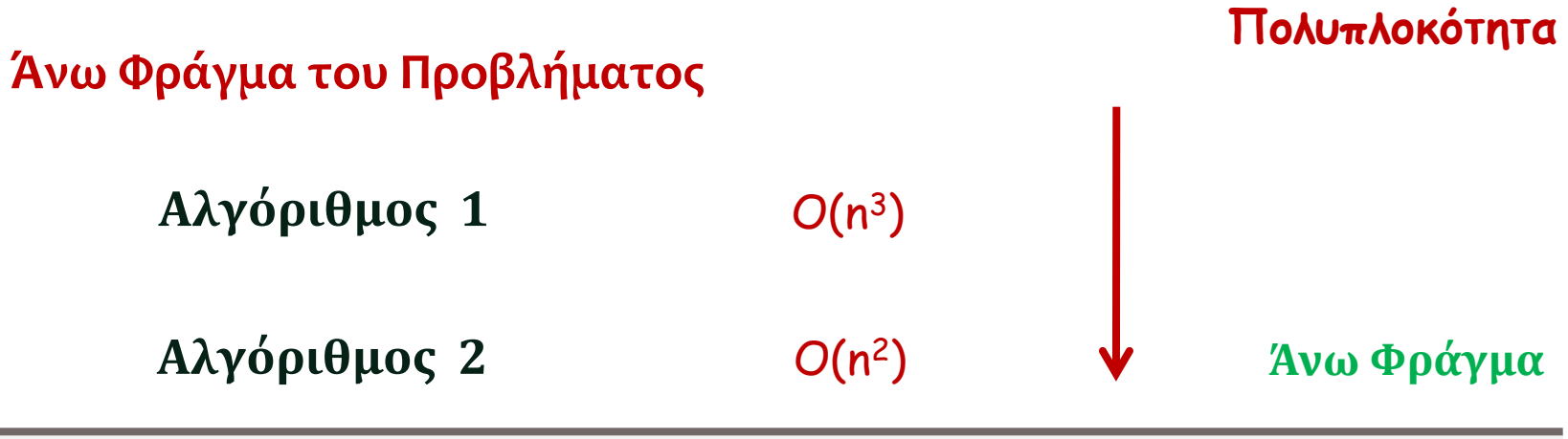
Πολυπλοκότητα Αλγόριθμου και Άνω Φράγματα

Παράδειγμα



Πολυπλοκότητα Αλγόριθμου και Άνω Φράγματα

Παράδειγμα



Όρια Πολυπλοκότητας Προβλήματος

Πρόβλημα Π

Πολυπλοκότητα Αλγόριθμου και Άνω Φράγματα

Παράδειγμα

Άνω Φράγμα του Προβλήματος

Πολυπλοκότητα

Αλγόριθμος 1	$O(n^3)$
Αλγόριθμος 2	$O(n^2)$
Αλγόριθμος 3	$O(n \log n)$

Αλγόριθμος 3
Ταχύτερος
για το πρόβλημα Π

ΣΗΜΕΡΑ

Άνω Φράγμα

Όρια Πολυπλοκότητας Προβλήματος

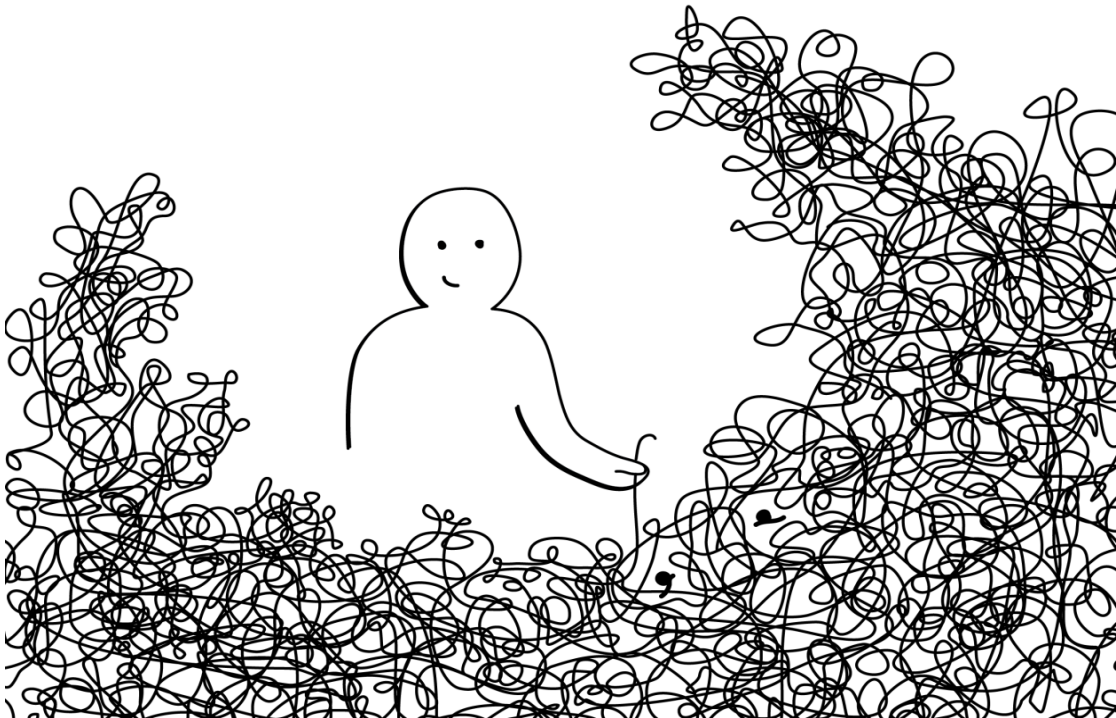
Πρόβλημα Π

Θα επανέλθουμε στο σημείο αυτό !!!

Περισσότερα για Πολυπλοκότητα

Αλγόριθμου vs Προβλήματος

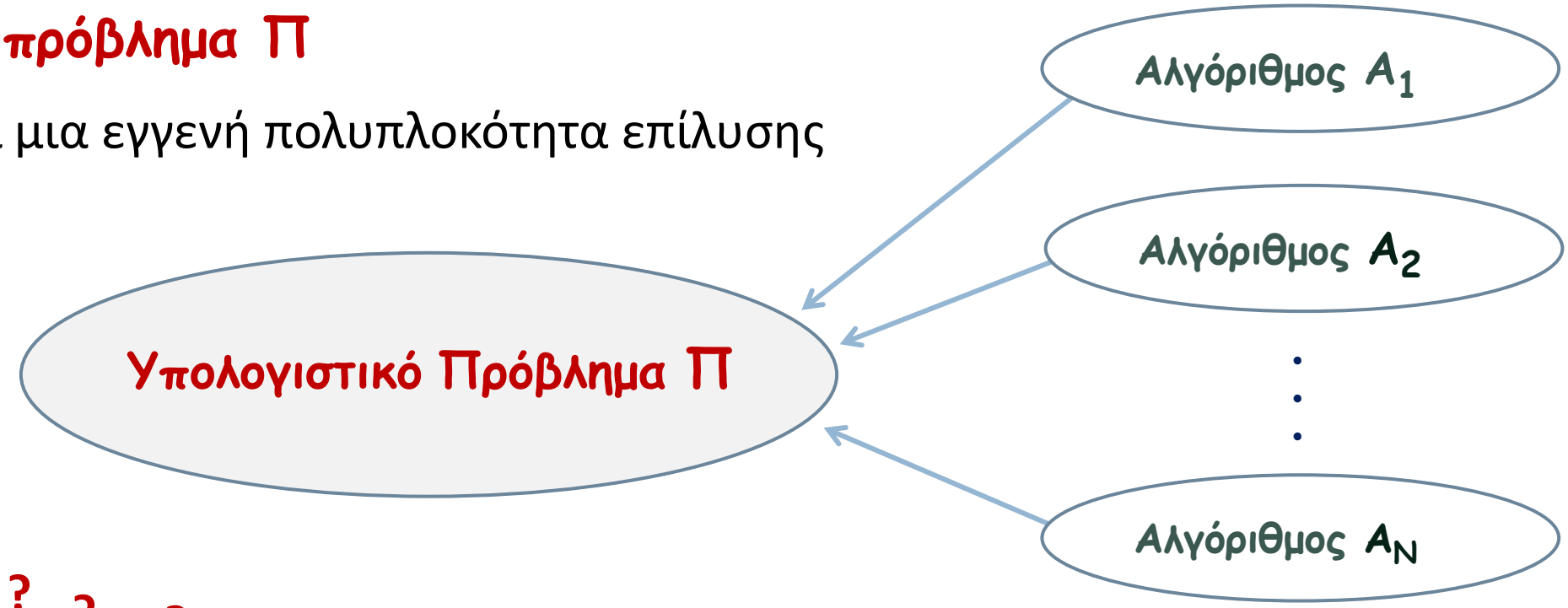
Άνω & Κάτω Φράγματα

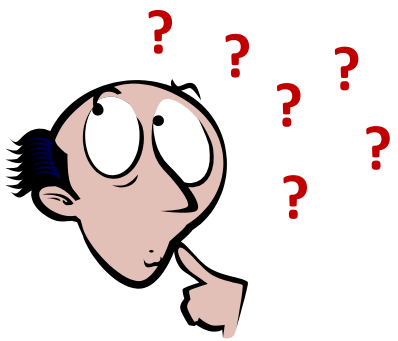


Υπολογιστικά Προβλήματα & Αλγόριθμοι

Το πρόβλημα Π

Έχει μια εγγενή πολυπλοκότητα επίλυσης



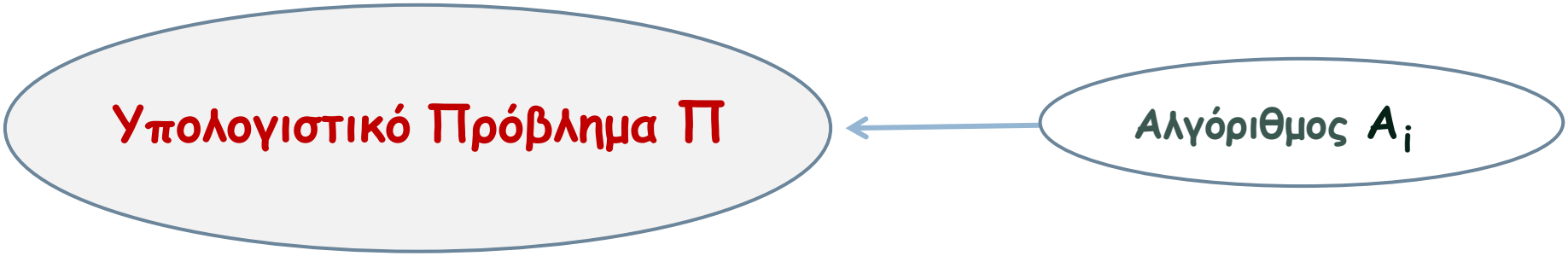


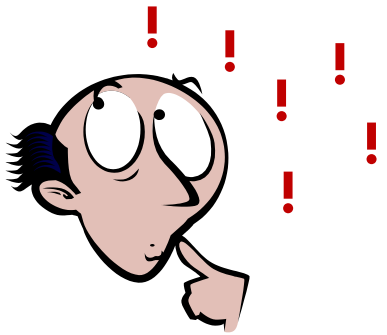
Κάθε αλγόριθμος για το πρόβλημα Π

Έχει τη δική του πολυπλοκότητα εκτέλεσης

Υπολογιστικά Προβλήματα & Αλγόριθμοι

$$\text{Complexity}(\Pi) \leq \text{Complexity}(A_i)$$



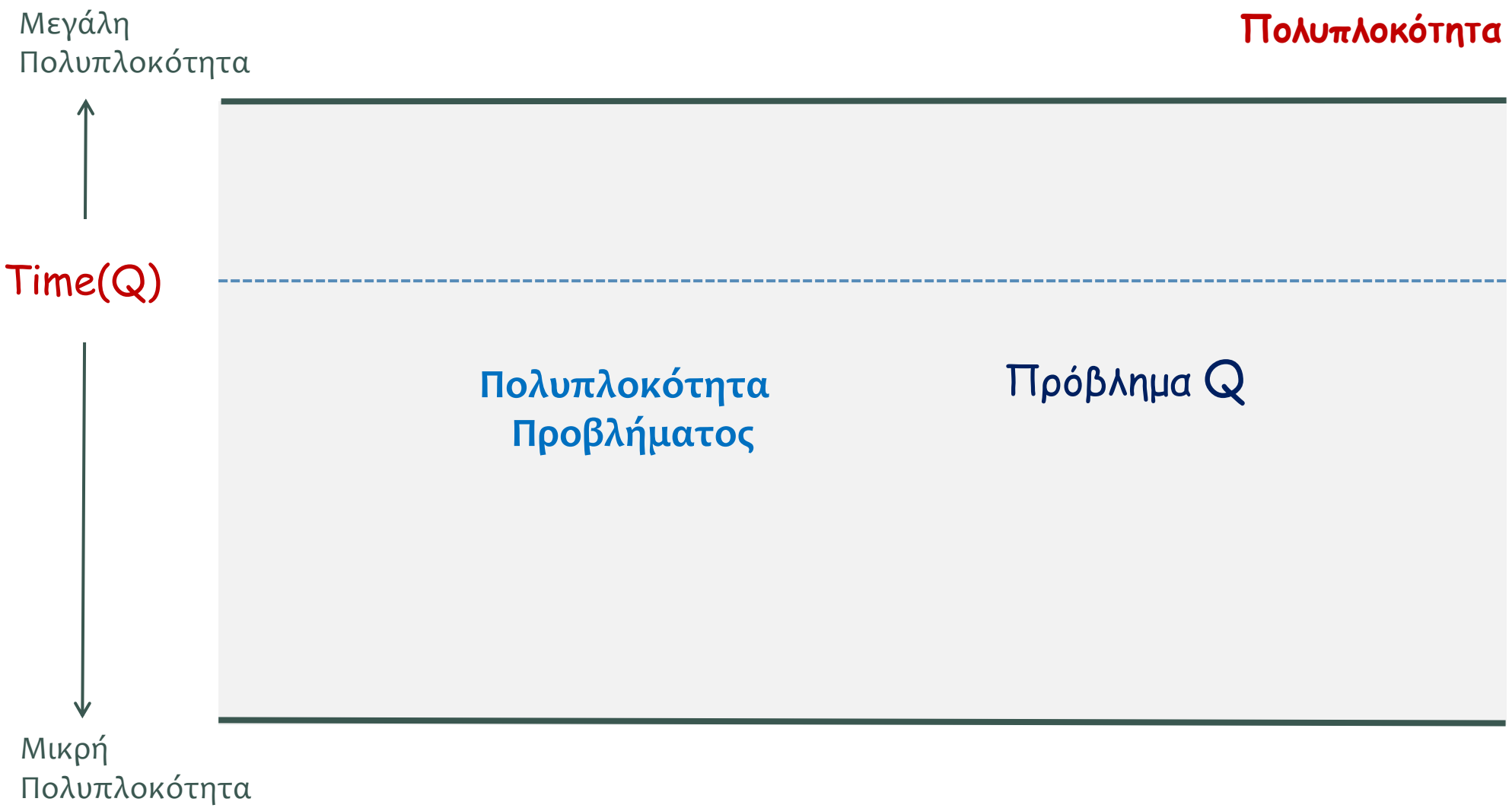


για κάθε Αλγόριθμο A_i $(1 \leq i \leq N)$ που επιλύει το πρόβλημα Π !!!

Complexity = Time !!!...

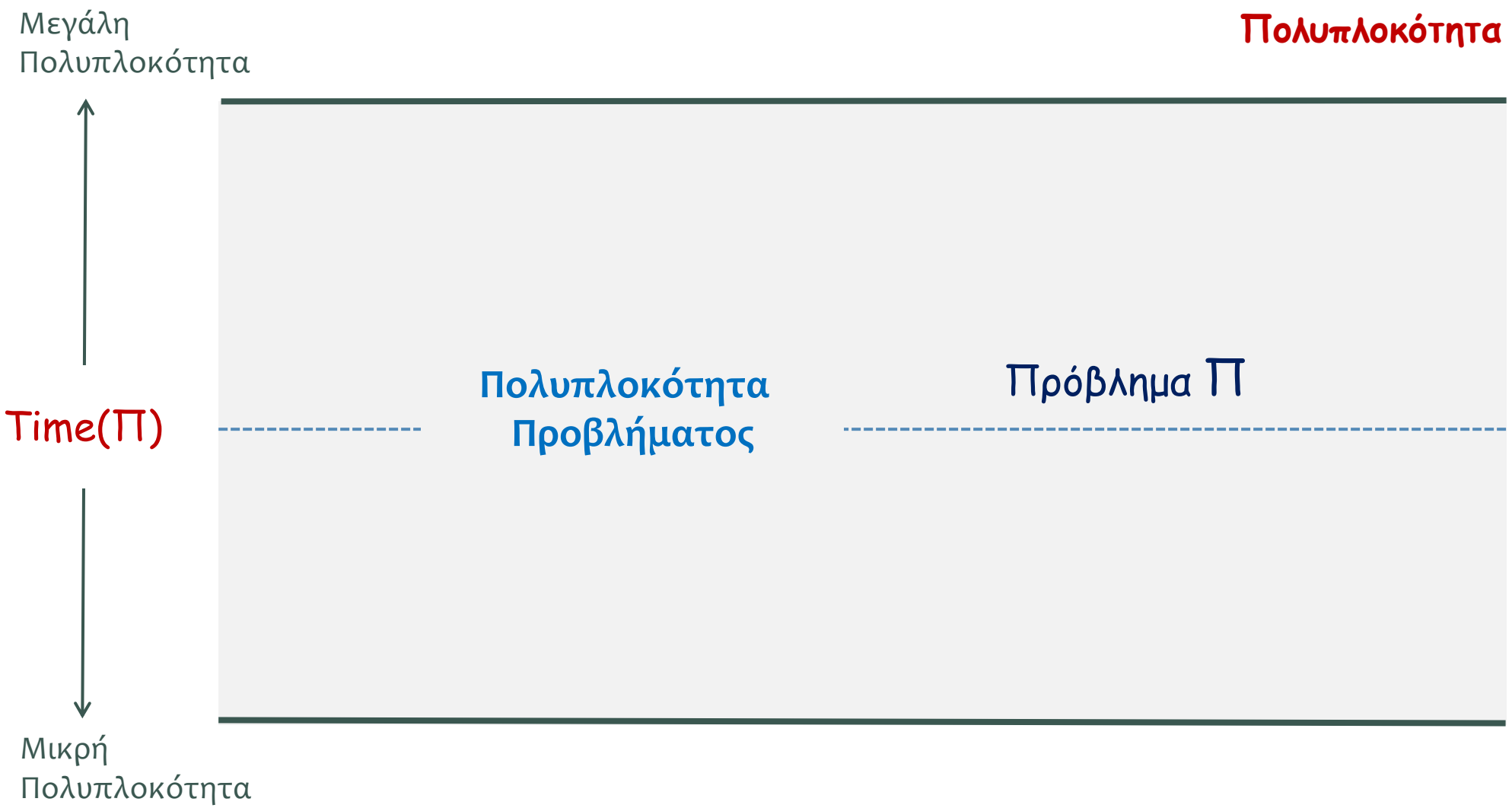
Πολυπλοκότητα Προβλήματος

Πολυπλοκότητα Προβλήματος

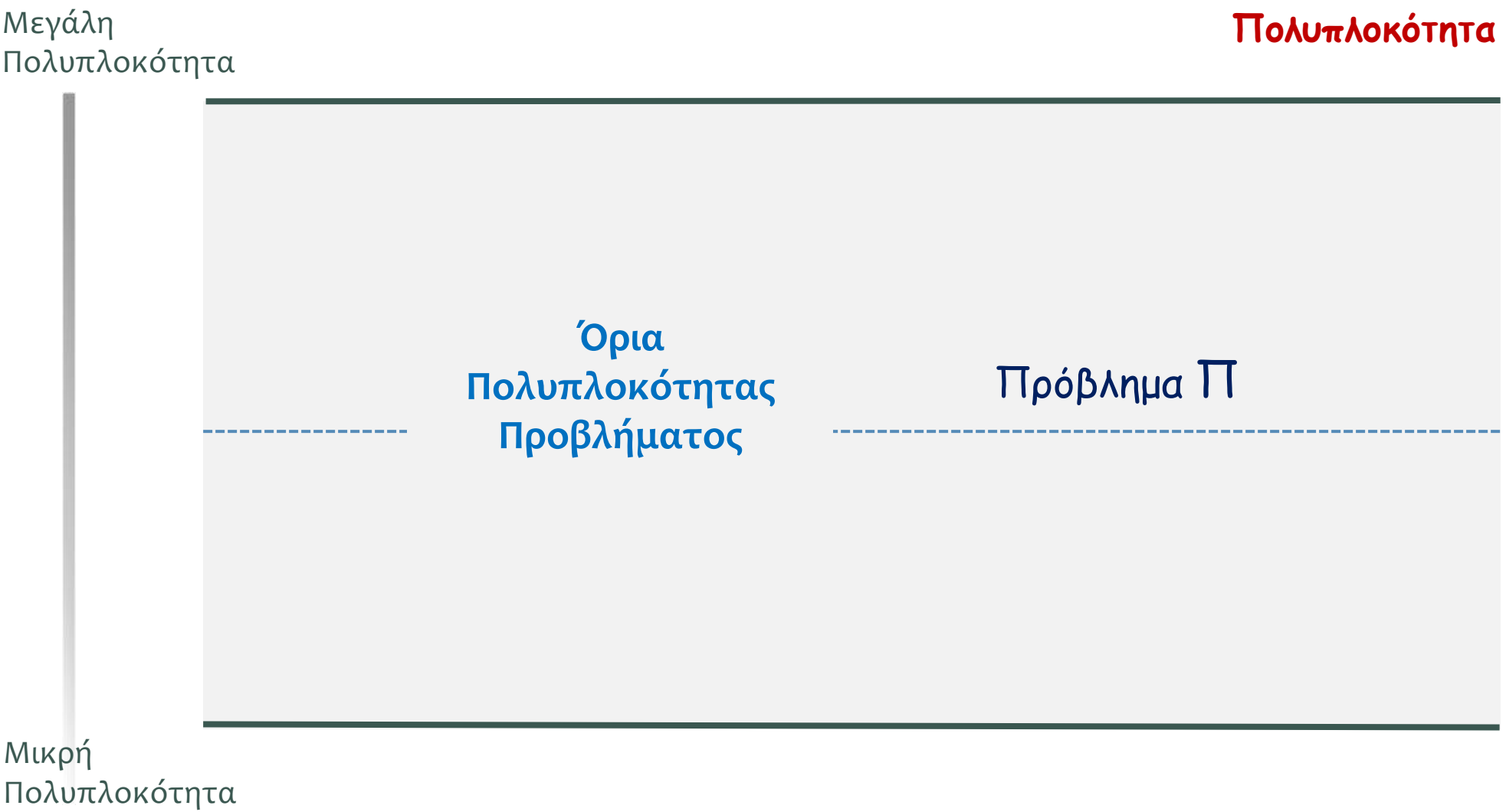


Πολυπλοκότητα Προβλήματος

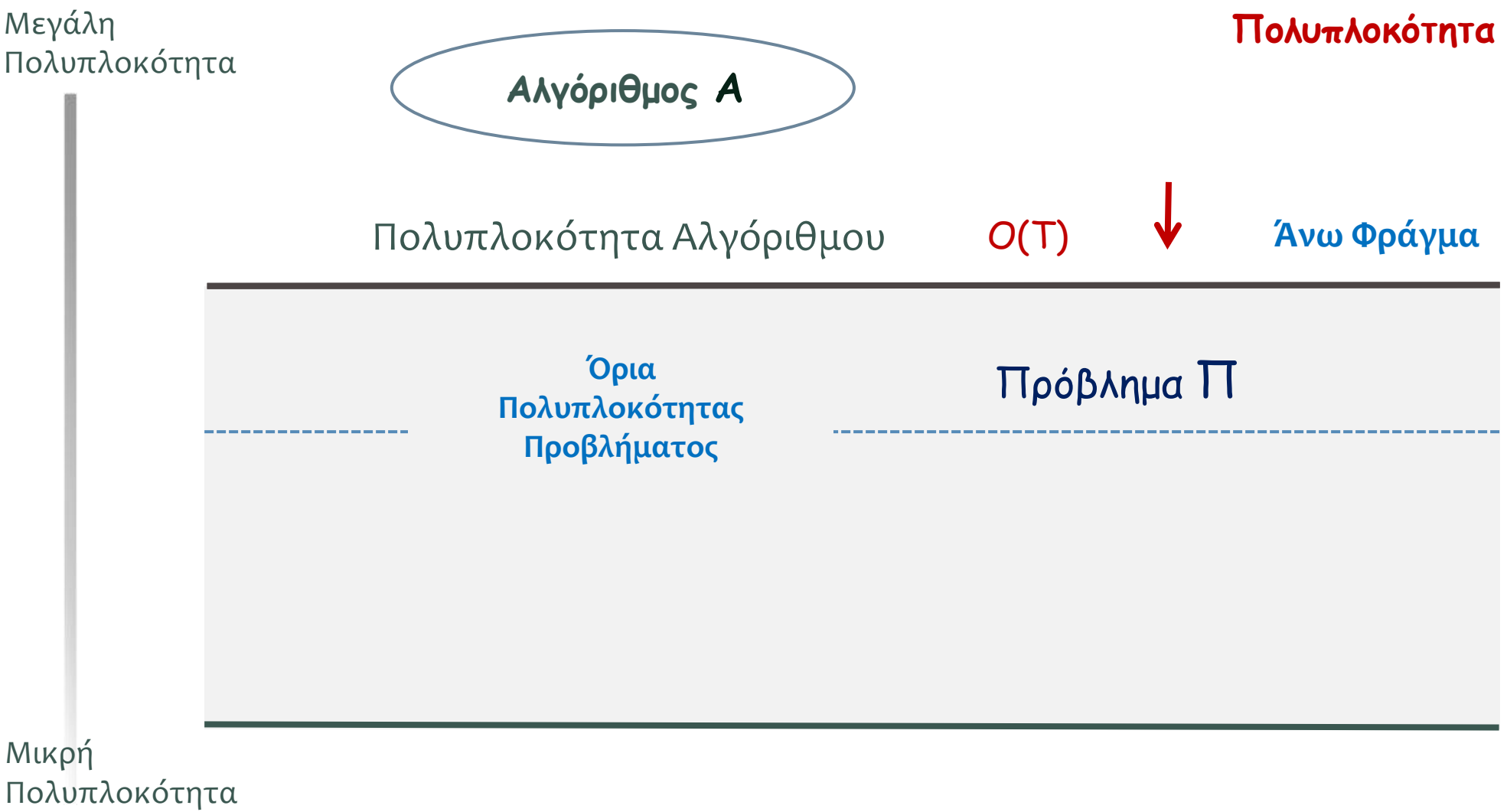
Πολυπλοκότητα Προβλήματος



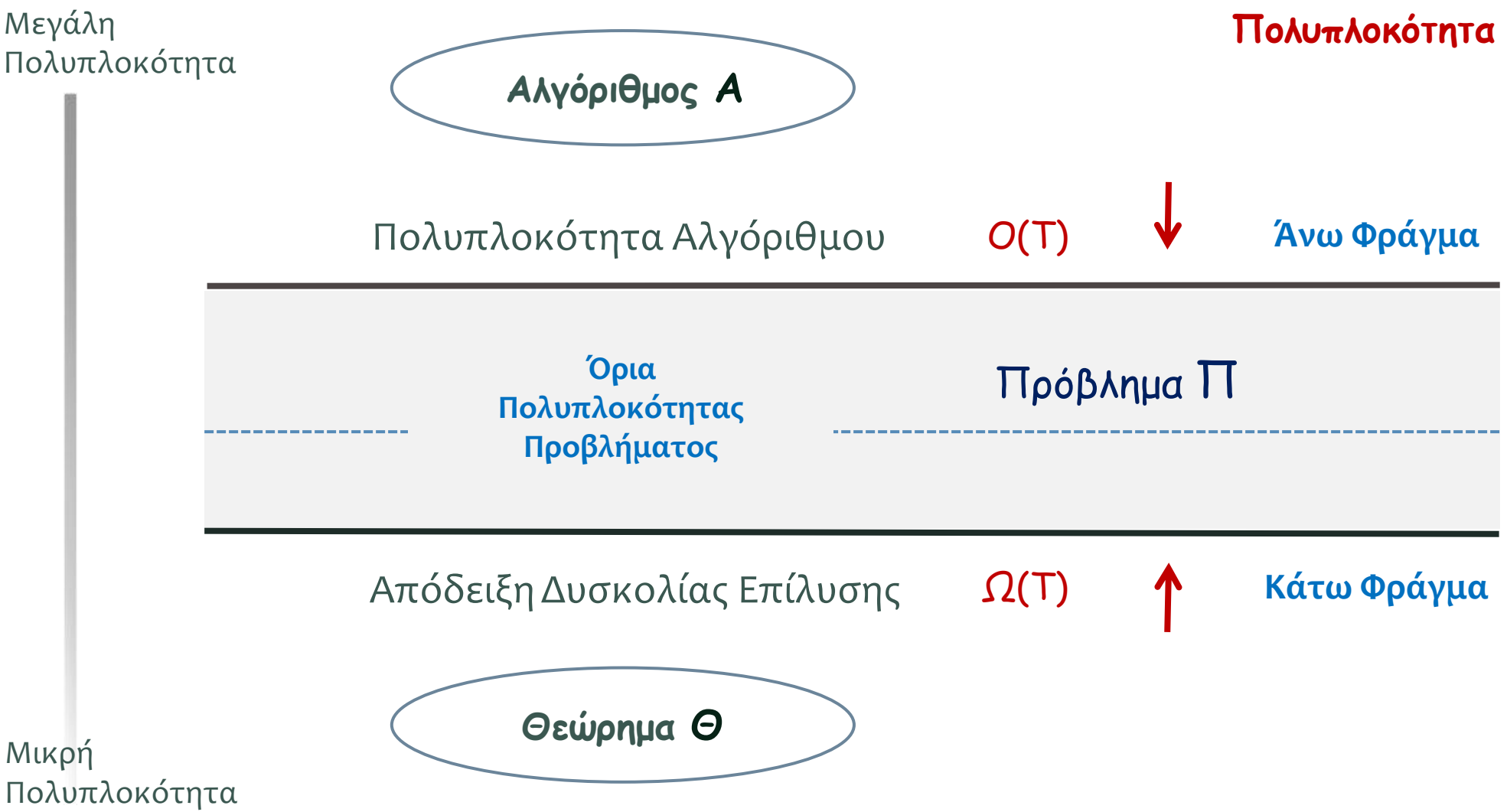
Πολυπλοκότητα Προβλήματος



Πολυπλοκότητα Αλγόριθμου – Προβλήματος

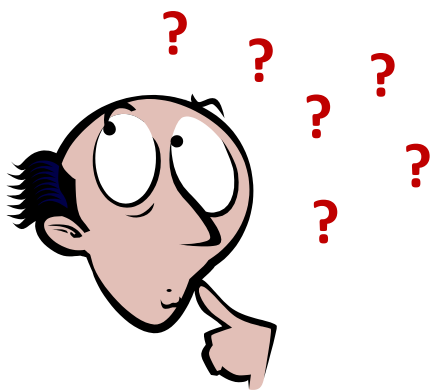


Πολυπλοκότητα Αλγόριθμου – Προβλήματος



Πολυπλοκότητα Αλγόριθμου – Προβλήματος

Τι μπορεί να συμβεί στο μέλλον ?



Πολυπλοκότητα

Αλγόριθμος Α

Πολυπλοκότητα Αλγόριθμου $O(T)$ ↓ **Άνω Φράγμα**

Όρια Πολυπλοκότητας Προβλήματος

Πρόβλημα Π

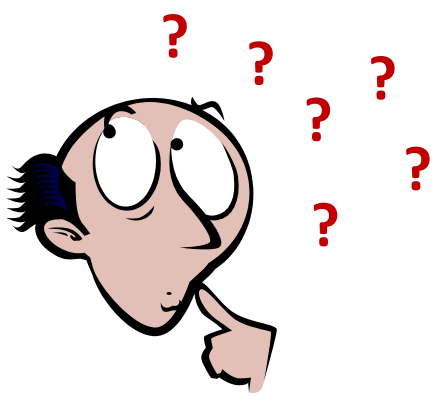
Απόδειξη Δυσκολίας Επίλυσης $\Omega(T)$ ↑ **Κάτω Φράγμα**

Θεώρημα Θ

Πολυπλοκότητα Αλγόριθμου – Προβλήματος

1ον

Να επινοήσουμε ένα καλύτερο Αλγόριθμο !



Αλγόριθμος Α

Πολυπλοκότητα

Πρόβλημα Π

Πολυπλοκότητα Αλγόριθμου $O(T)$ ↓ **Άνω Φράγμα**

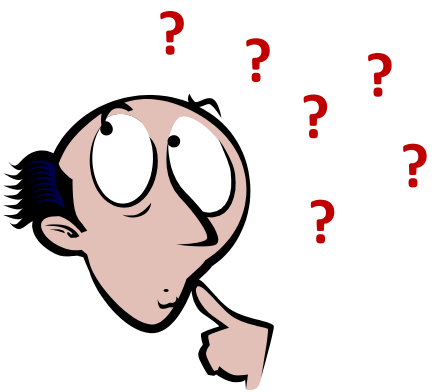
Απόδειξη Δυσκολίας Επίλυσης $\Omega(T)$ ↑ **Κάτω Φράγμα**

Θεώρημα Θ

Πολυπλοκότητα Αλγόριθμου – Προβλήματος

1ον

Να επινοήσουμε ένα καλύτερο Αλγόριθμο !



Αλγόριθμος Α'

Πολυπλοκότητα
Πρόβλημα Π

Πολυπλοκότητα Αλγόριθμου $O(T')$ **Άνω Φράγμα**

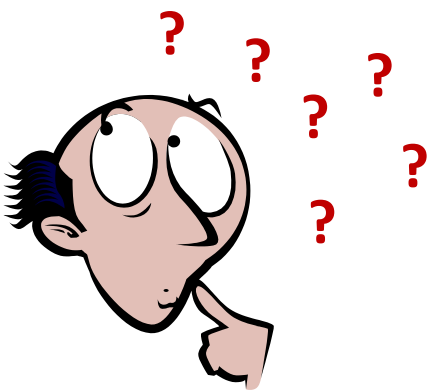
Απόδειξη Δυσκολίας Επίλυσης $\Omega(T)$ **Κάτω Φράγμα**

Θεώρημα Θ

Πολυπλοκότητα Αλγόριθμου – Προβλήματος

2ον

Να αποδείξουμε ένα καλύτερο Κάτω Φράγμα !



Αλγόριθμος Α

Πολυπλοκότητα
Πρόβλημα Π

Πολυπλοκότητα Αλγόριθμου $O(T)$ ↓ **Άνω Φράγμα**

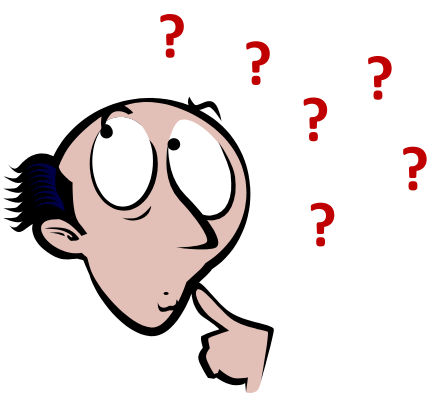
Απόδειξη Δυσκολίας Επίλυσης $\Omega(T)$ ↑ **Κάτω Φράγμα**

Θεώρημα Θ

Πολυπλοκότητα Αλγόριθμου – Προβλήματος

2ον

Να αποδείξουμε ένα καλύτερο Κάτω Φράγμα !



Αλγόριθμος Α

Πολυπλοκότητα
Πρόβλημα Π

Πολυπλοκότητα Αλγόριθμου $O(T)$ ↓ **Άνω Φράγμα**

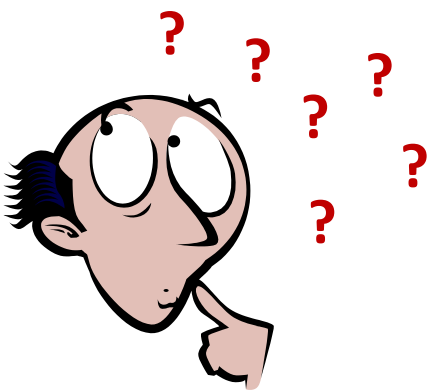
Απόδειξη Δυσκολίας Επίλυσης $\Omega(T')$ ↑ **Κάτω Φράγμα**

Θεώρημα Θ'

Πολυπλοκότητα Αλγόριθμου – Προβλήματος

Στόχος

Να επιτύχουμε και τα Δύο !



Αλγόριθμος Α

Πολυπλοκότητα
Πρόβλημα Π

Πολυπλοκότητα Αλγόριθμου $O(T)$ ↓ **Άνω Φράγμα**

Απόδειξη Δυσκολίας Επίλυσης $\Omega(T)$ ↑ **Κάτω Φράγμα**

Θεώρημα Θ

Πολυπλοκότητα Αλγόριθμου – Προβλήματος

Στόχος

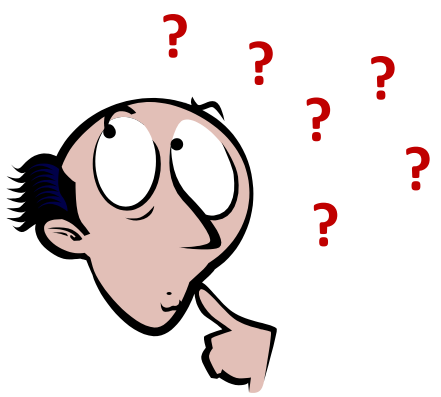
Να επιτύχουμε και τα Δύο !

Αλγόριθμος Α'

Πολυπλοκότητα
Πρόβλημα Π

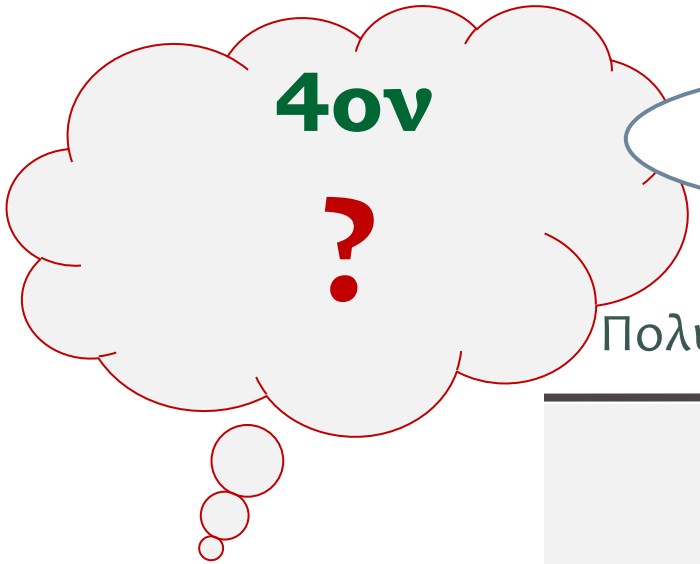
Πολυπλοκότητα Αλγόριθμου $O(T')$ **Άνω Φράγμα**

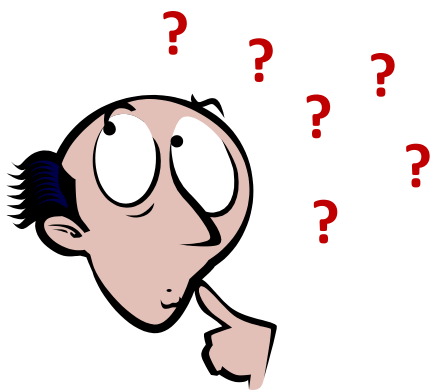
Απόδειξη Δυσκολίας Επίλυσης $\Omega(T')$ **Κάτω Φράγμα**



Θεώρημα Θ'

Πολυπλοκότητα Αλγόριθμου – Προβλήματος





Αλγόριθμος Α

Πολυπλοκότητα
Πρόβλημα Π

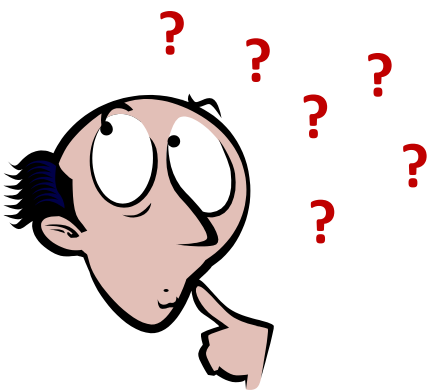
Πολυπλοκότητα Αλγόριθμου $O(T)$ ↓ **Άνω Φράγμα**

Απόδειξη Δυσκολίας Επίλυσης $\Omega(T)$ ↑ **Κάτω Φράγμα**

Θεώρημα Θ

Πολυπλοκότητα Αλγόριθμου – Προβλήματος

4ον
Να μείνουν τα «πράγματα» ως έχουν σήμερα !



Αλγόριθμος Α

Πολυπλοκότητα

Έρευνα
για το πρόβλημα Π
σε δύο κατευθύνσεις

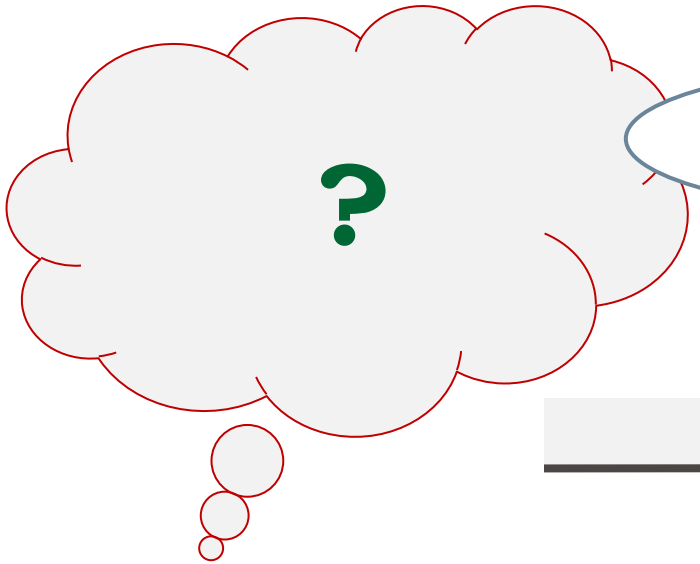
Πολυπλοκότητα Αλγόριθμου $O(T)$ ↓ **Άνω Φράγμα**

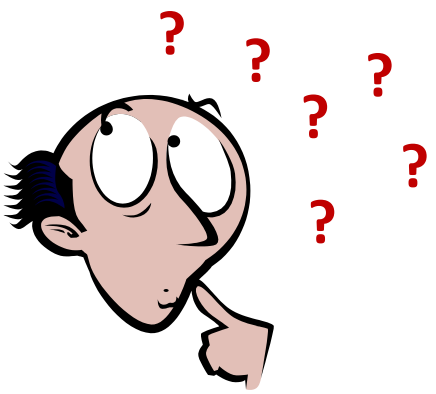
ΚΕΝΟ

Απόδειξη Δυσκολίας Επίλυσης $\Omega(T)$ ↑ **Κάτω Φράγμα**

Θεώρημα Θ

Πολυπλοκότητα Αλγόριθμου – Προβλήματος





Αλγόριθμος A_1

Θεώρημα Θ_2

Πολυπλοκότητα

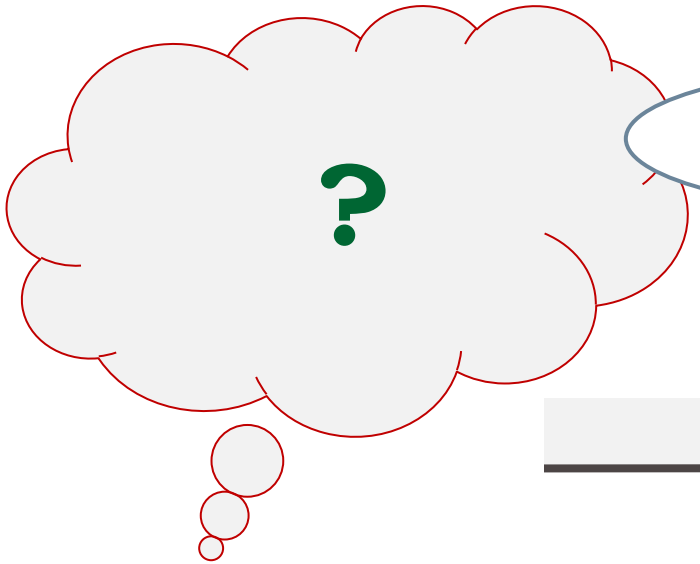
Αλγόριθμος A_1 και Θεώρημα Θ_2 **λάθος**

$\Omega(T_2)$ **Κάτω Φράγμα**

$T_1 < T_2$

$O(T_1)$ **Άνω Φράγμα**

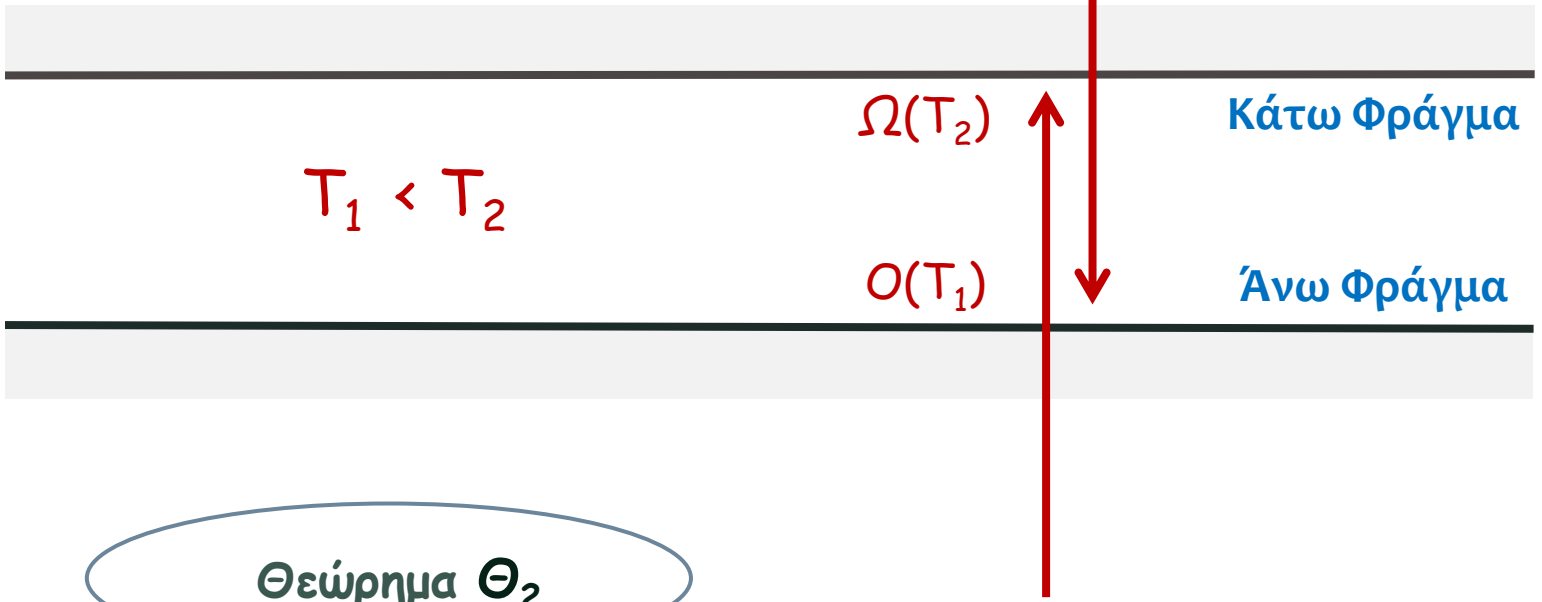
Πολυπλοκότητα Αλγόριθμου – Προβλήματος

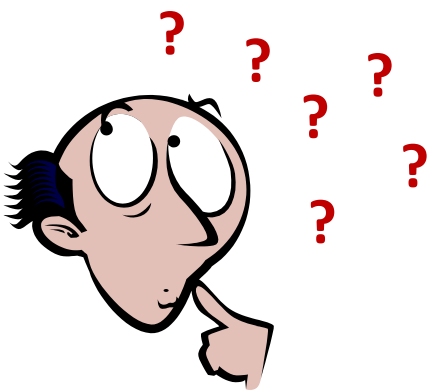


Αλγόριθμος A_1

Πολυπλοκότητα

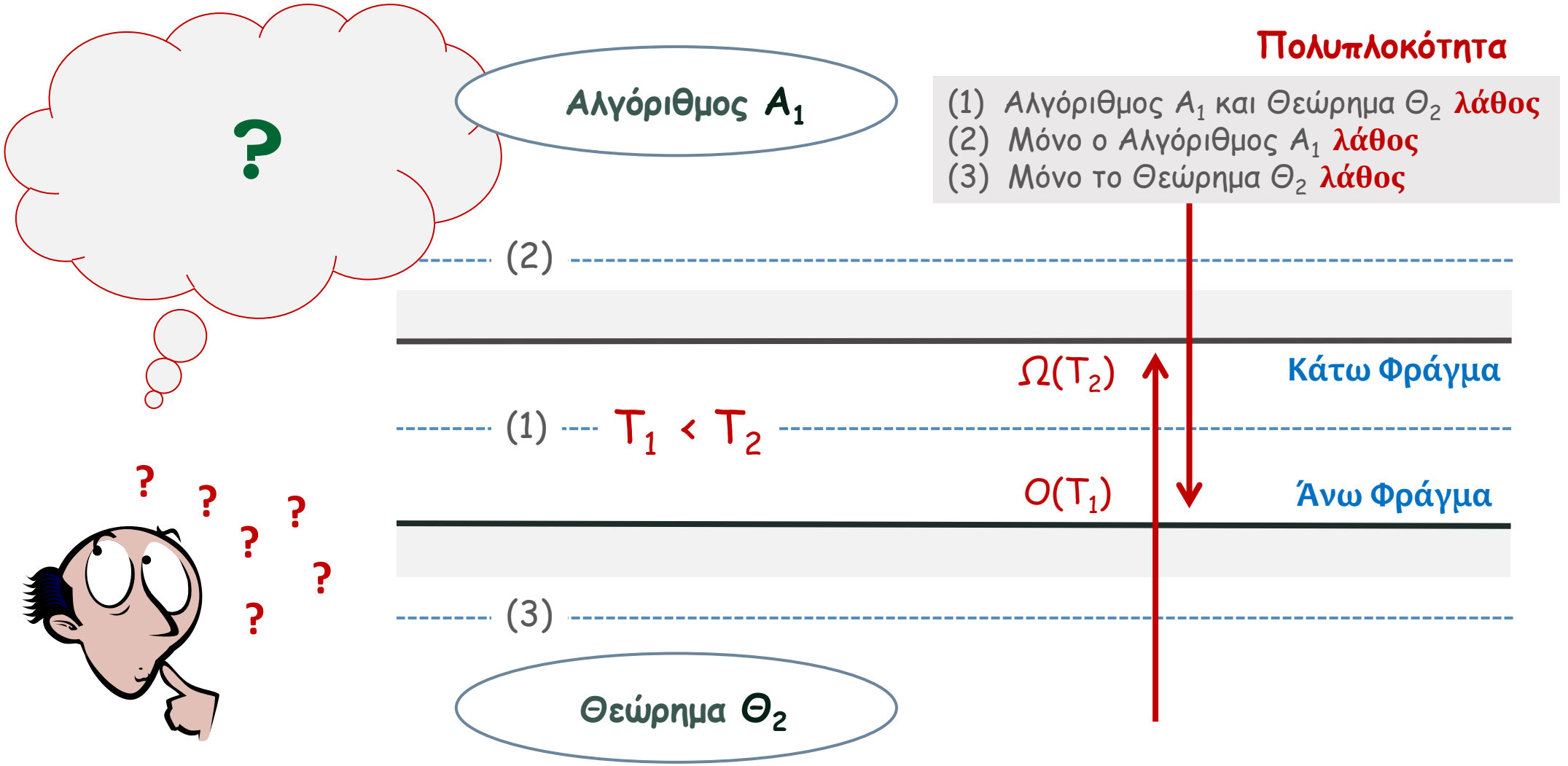
(1) Αλγόριθμος A_1 και Θεώρημα Θ_2 **λάθος**
(2) Μόνο ο Αλγόριθμος A_1 **λάθος**
(3) Μόνο το Θεώρημα Θ_2 **λάθος**





Θεώρημα Θ_2

Πολυπλοκότητα Αλγόριθμου – Προβλήματος



Ταχύτερος & Βέλτιστος Αλγόριθμος

- Ταχύτερος αλγόριθμος
  - *Ο αλγόριθμος με την μικρότερη πολυπλοκότητα σήμερα !!!*
- Βέλτιστος αλγόριθμος
  - *Ο αλγόριθμος με πολυπλοκότητα ίση με το κάτω φράγμα του προβλήματος Π.*

Ένας **Βέλτιστος αλγόριθμος** είναι και ο Ταχύτερος !

Ο **Ταχύτερος αλγόριθμος** δεν είναι πάντα Βέλτιστος !!

Ταχύτερος & Βέλτιστος Αλγόριθμος

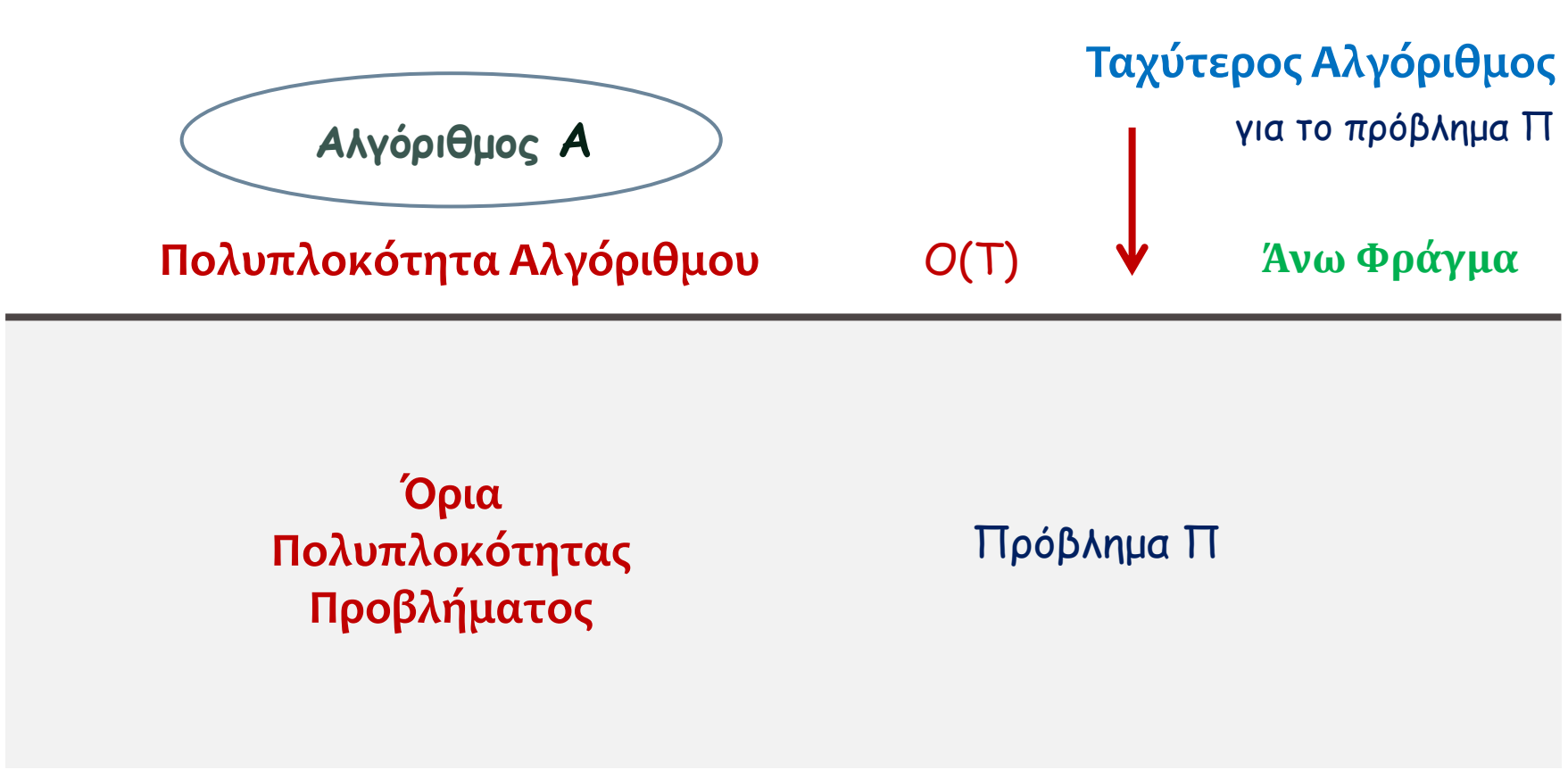
- Ταχύτερος αλγόριθμος
  - *Ο αλγόριθμος με την μικρότερη πολυπλοκότητα σήμερα !!!*
- Βέλτιστος αλγόριθμος
  - *Ο αλγόριθμος με πολυπλοκότητα ίση με το κάτω φράγμα του προβλήματος Π.*

Προσοχή !!!

για να δείξουμε ότι ένας αλγόριθμος είναι ***βέλτιστος*** χρειάζεται η ***απόδειξη*** *αντίστοιχου* ***κάτω φράγματος*** *του προβλήματος* ***Π***.

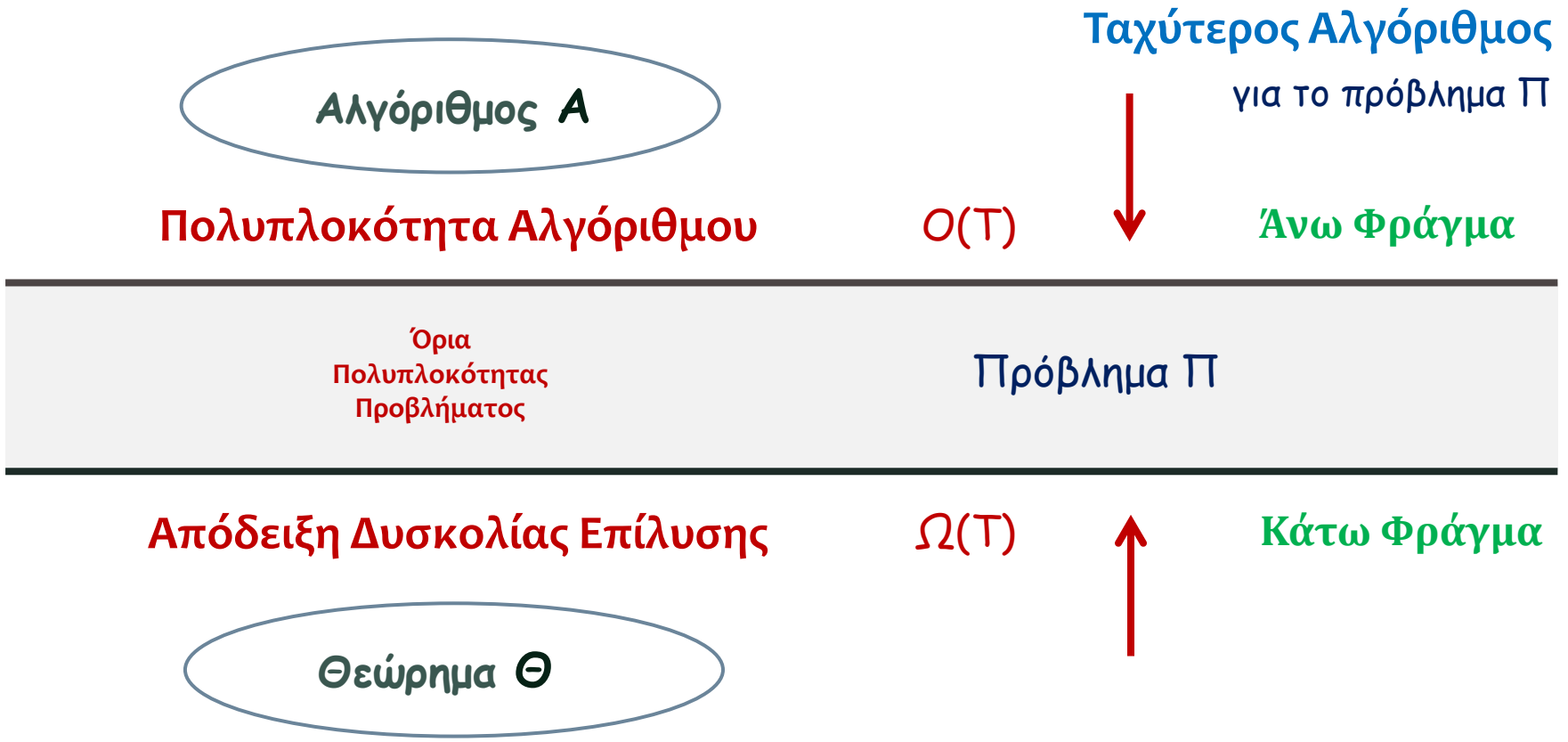
Ταχύτερος & Βέλτιστος Αλγόριθμος

Βελτιστότητα !



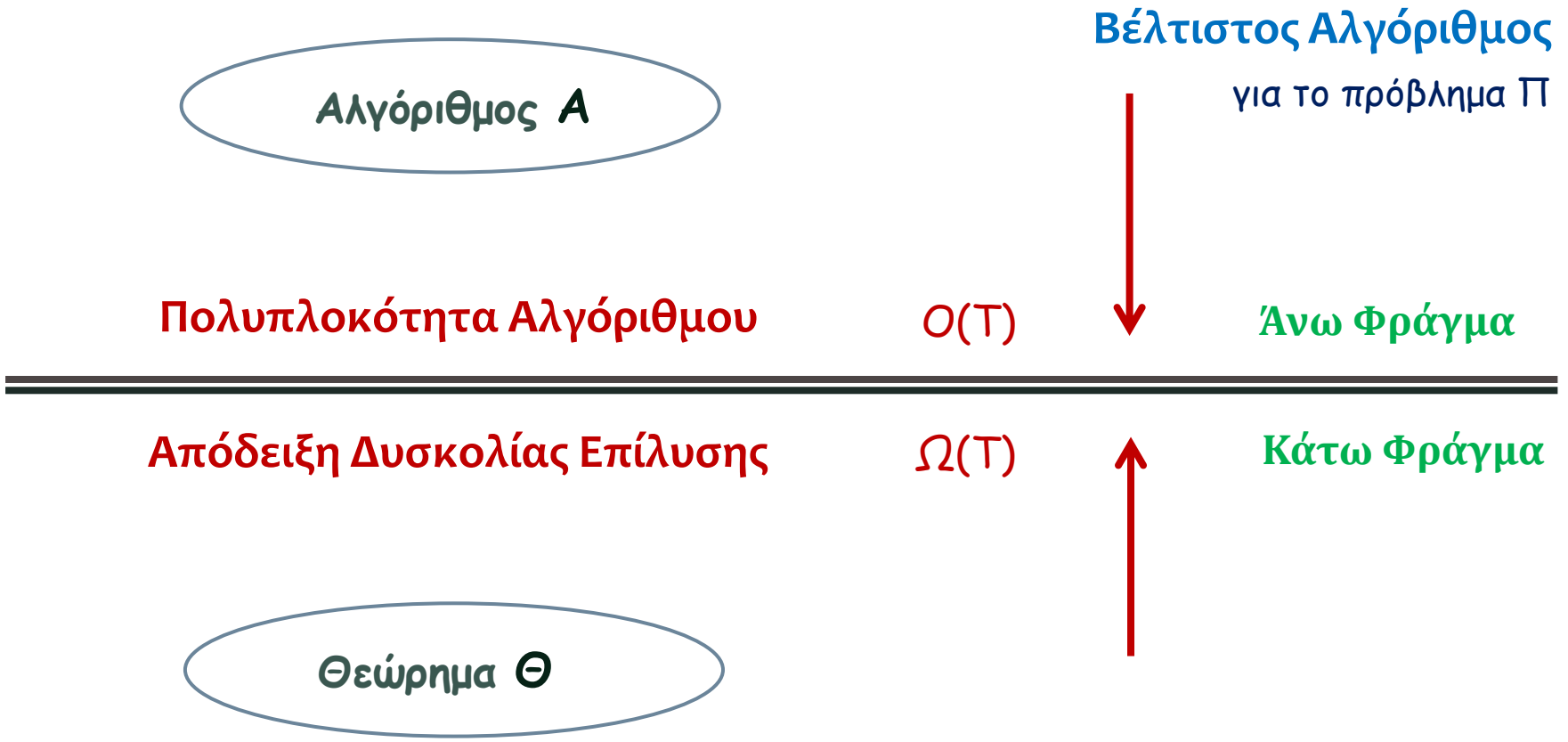
Ταχύτερος & Βέλτιστος Αλγόριθμος

Βελτιστότητα !



Ταχύτερος & Βέλτιστος Αλγόριθμος

Βελτιστότητα !



Πολυπλοκότητα Προβλήματος

- Για τα προβλήματα που επιλύονται (solvable, computable, decidable) μας ενδιαφέρει να υπολογίσουμε το πόσο καλά μπορεί να γίνει αυτό, δηλαδή
  - πόσο γρήγορα, ή
  - με πόση μνήμη, ή
  - με πόσους επεξεργαστές (παραλληλία), ή
  - με πόση κατανάλωση ενέργειας (sensor networks), κ.λπ.

- Αυτό λέγεται (Υπολογιστική) Πολυπλοκότητα !!!

Πολυπλοκότητα Προβλήματος

Η *πολυπλοκότητα του βέλτιστου αλγορίθμου* που λύνει το πρόβλημα Π

Συμβολικά:

$$\text{cost}_{\Pi}(n) = \min \{\text{cost}_{A}(n)\}$$

για όλους τους αλγορίθμους

Α που επιλύουν το **Π**

όπου,

$$\text{cost}_{A}(n) = \max \{\text{κόστος αλγόριθμου Α για είσοδο x}\}$$

για όλες τις εισόδους

x μήκους ***n***

Πολυπλοκότητα Προβλήματος

Η *πολυπλοκότητα του βέλτιστου αλγορίθμου* που λύνει το πρόβλημα Π

Συμβολικά:

$$\text{cost}_{\Pi}(n) = \min \{\text{cost}_{A}(n)\}$$

για όλους τους αλγορίθμους

Α που επιλύουν το **Π**

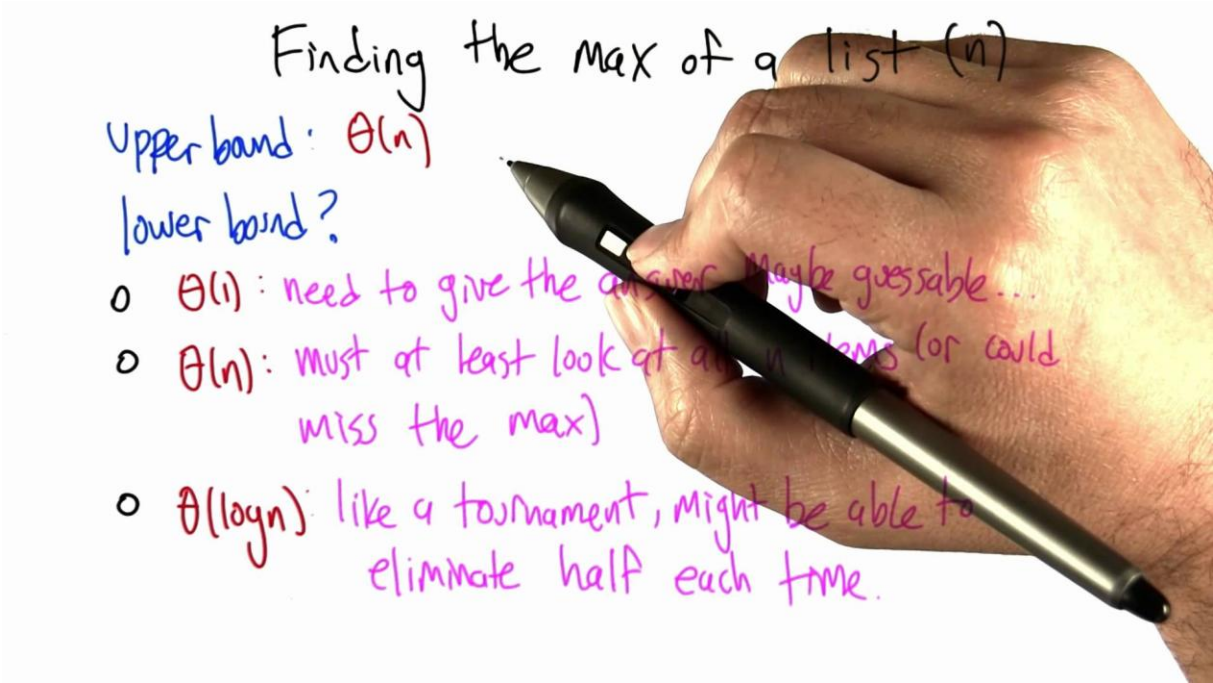
Παράδειγμα: Για το πρόβλημα της ταξινόμησης MergeSort (MS) :

$$\text{cost}_{MS}(n) = O(n \log n)$$

Περισσότερα για Άνω και Κάτω Φράγματα

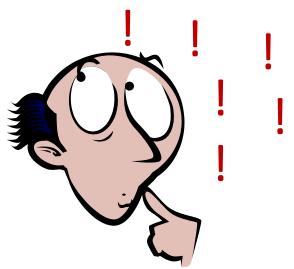
Αλγόριθμος: **Find_Max**

Αλγόριθμος: **Merge_Sort**



Άνω και Κάτω Φράγματα

Σενάριο 1ο



Πολυπλοκότητα

Επίλυση του Προβλήματος

Αλγόριθμος 1 $O(n^3)$ ↓ **Άνω Φράγμα**

Πρόβλημα Π_1

Θεώρημα 1 $\Omega(n)$ ↑ **Κάτω Φράγμα**

Απόδειξη Δυσκολίας Επίλυσης

Άνω και Κάτω Φράγματα

Σενάριο 1ο

Πολυπλοκότητα

Επίλυση του Προβλήματος

Αλγόριθμος 1 $O(n^3)$

Αλγόριθμος 2 $O(n^2)$

Άνω Φράγμα

Πρόβλημα Π_1

Θεώρημα 1 $\Omega(n)$

Κάτω Φράγμα

Απόδειξη Δυσκολίας Επίλυσης

Άνω και Κάτω Φράγματα

Πολυπλοκότητα

Επίλυση του Προβλήματος

Αλγόριθμος 1 $O(n^3)$

Αλγόριθμος 2 $O(n^2)$ — **Άνω Φράγμα**

Πρόβλημα Π_1

Θεώρημα 2 $\Omega(n \log n)$ — **Κάτω Φράγμα**

Θεώρημα 1 $\Omega(n)$

Απόδειξη Δυσκολίας Επίλυσης

Άνω και Κάτω Φράγματα

Σενάριο 1ο

Πολυπλοκότητα

Επίλυση του Προβλήματος

Αλγόριθμος 1	$O(n^3)$
Αλγόριθμος 2	$O(n^2)$
Αλγόριθμος 3	$O(n \log n)$

Αλγόριθμος 3
Βέλτιστος
για το πρόβλημα Π_1

Άνω Φράγμα

Θεώρημα 2	$\Omega(n \log n)$
Θεώρημα 1	$\Omega(n)$

Κάτω Φράγμα

Απόδειξη Δυσκολίας Επίλυσης

Άνω και Κάτω Φράγματα

Σενάριο 2ο

Πολυπλοκότητα

Επίλυση του Προβλήματος

Αλγόριθμος 1 $O(n^3)$

Αλγόριθμος 2 $O(n^{2.768})$

Έρευνα
για το πρόβλημα Π_2
σε δύο κατευθύνσεις

Άνω Φράγμα

ΚΕΝΟ

Πρόβλημα Π_2

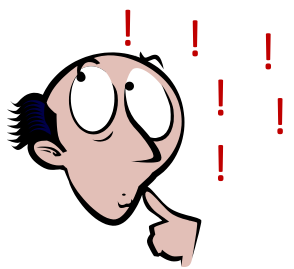
Θεώρημα 1 $\Omega(n^2)$

Κάτω Φράγμα

Απόδειξη Δυσκολίας Επίλυσης

Άνω και Κάτω Φράγματα

Σενάριο 3ο



Πολυπλοκότητα

Επίλυση του Προβλήματος

Πρόβλημα Π_3

?

Αλγόριθμος 1 $O(n^3)$

Θεώρημα 2 $\Omega(n \log n)$ **Κάτω Φράγμα**

Αλγόριθμος 2 $O(n)$ **Άνω Φράγμα**

Θεώρημα 1 $\Omega(n)$ Πρόβλημα Π_3

Απόδειξη Δυσκολίας Επίλυσης

Άνω και Κάτω Φράγματα

Πολυπλοκότητα

Πρόβλημα Π

↓ **Άνω Φράγμα**

Απόδειξε ότι: ∃ αλγόριθμος A (ορθότητα) που επιλύει το Π,
∀ είσοδο I μήκους n, ο αλγόριθμος A απαιτεί το πολύ $W_A(n)$ πράξεις για την λύση του Π

Απόδειξε ότι: ∀ αλγόριθμο A (ίδιας κλάσης) που επιλύει το Π,
∃ μια είσοδος I μήκους n, για την οποία ο A απαιτεί τουλάχιστο $F_A(n)$ πράξεις για την λύση του προβλήματος Π

↑ **Κάτω Φράγμα**

Άνω και Κάτω Φράγματα

Παράδειγμα

Πρόβλημα Εύρεση Μέγιστου

Εύρεση του MAX στοιχείου ενός συνόλου S πληθικότητας |S|=n, με στοιχεία τύπου type(S).

Κλάση Αλγορίθμων: Αλγόριθμοι που συγκρίνουν και αντιγράφουν στοιχεία τύπου type(S).

Βασικές Πράξεις: Σύγκριση ενός στοιχείου του συνόλου S με οποιοδήποτε άλλο στοιχείο τύπου type(S).

Άνω και Κάτω Φράγματα

Παράδειγμα

Άνω Φράγμα

Αλγόριθμος: **Find_Max**

```
begin
    max = S[1]; posmax = 1;
    for i = 2 to n do
        if S[i] > max then
            max = S[i]; posmax = i;
    end;
    return (max, posmax);
end;
```

Το πλήθος των Βασικών Πράξεων (S[i]>max) είναι ακριβώς n-1, και επομένως $W_{Find\_Max}(n) = n-1$.

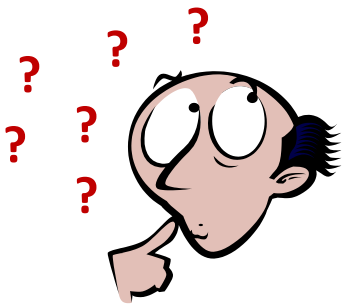
Άνω και Κάτω Φράγματα

Παράδειγμα

Άνω Φράγμα

Αλγόριθμος: **Find_Max**

Υπάρχει Ταχύτερος Αλγόριθμος !!!



Το πλήθος των Βασικών Πράξεων (S[i]>max) είναι ακριβώς n-1, και επομένως $W_{Find\_Max}(n) = n-1$.

Άνω και Κάτω Φράγματα

Παράδειγμα

Κάτω Φράγμα

Υπολόγισε ένα ελάχιστο πλήθος Βασικών Πράξεων που απαιτούνται για την επίλυση του Προβλήματος MAX

- Έστω ένα σύνολο S με n στοιχεία τύπου type(S).
- Υποθέτουμε ότι όλα τα στοιχεία είναι διαφορετικά.

Μπορούμε να κάνουμε αυτήν την υπόθεση γιατί το κάτω φράγμα αποδεικνύεται για την χειρίστη-περίπτωση και επομένως όλες οι είσοδοι είναι αποδεκτές.

Άνω και Κάτω Φράγματα

Παράδειγμα

Κάτω Φράγμα

- Στο σύνολο S υπάρχουν n-1 που δεν είναι max.
- Ένα στοιχείο x του S, ΔΕΝ είναι max μόνο εάν είναι μικρότερο από ένα τουλάχιστο στοιχείο y του S ($x < y$).

  Στη σύγκριση $x < y$, λέμε ότι το x είναι ο "ηττημένος" και y ο "νικητής".
- Άρα, για να υπολογίσουμε το max πρέπει n-1 στοιχεία να "ηττηθούν" συγκρινόμενα…

  Προσοχή !!!… με οποιοδήποτε Αλγόριθμο και αν επινοήσουμε !!!

Άνω και Κάτω Φράγματα

Παράδειγμα

Κάτω Φράγμα

- Κάθε σύγκριση έχει μόνο έναν "ηττημένο" !
- Επομένως, πρέπει να γίνουν τουλάχιστον $n-1$ συγκρίσεις !!!

διότι, εάν κατά τον τερματισμό οποιουδήποτε Αλγόριθμου Α υπάρχουν δύο ή περισσότερα "μη-ηττημένα" στοιχεία, τότε ο Α δεν θα μπορεί να εντοπίσει το max.

- Άρα, $F(n) = n-1$ είναι ένα κάτω-φράγμα για το πλήθος των απαιτούμενων συγκρίσεων !!!

Άνω και Κάτω Φράγματα

Παράδειγμα

Πρόβλημα Εύρεση Μέγιστου

- Η πολυπλοκότητα του Αλγόριθμου **Find_Max** είναι **O(n)**
- Το ελάχιστο πλήθος Βασικών Πράξεων που απαιτούνται για την επίλυση του **Προβλήματος MAX** είναι **Ω(n)**

Άνω Φράγμα **O(n)** ↓

Κάτω Φράγμα **Ω(n)** ↑

Άνω και Κάτω Φράγματα

Παράδειγμα

Πρόβλημα Ταξινόμησης

Ταξινόμηση μιας ακολουθίας $S = (x_1, x_2, \ldots, x_n)$ μήκους $|S|=n$ και με στοιχεία τύπου type(S), σε αύξουσα τάξη.

Κλάση Αλγορίθμων: Αλγόριθμοι που συγκρίνουν και δημιουργούν υποσύνολα στοιχείων τύπου type(S).

Βασικές Πράξεις: Σύγκριση ενός στοιχείου του συνόλου S με οποιοδήποτε άλλο στοιχείο τύπου type(S).

Άνω και Κάτω Φράγματα

Παράδειγμα

Άνω Φράγμα

Αλγόριθμος: **Merge_Sort**

```
begin
    .
    .
   Merge_Sort
    .
    .
    .
end;
```

Το πλήθος των Βασικών Πράξεων (S[i]>S[j]) του Merge_Sort είναι c n logn, και επομένως $W_{Merge\_Sort}(n) = c\ n\ \log n$.

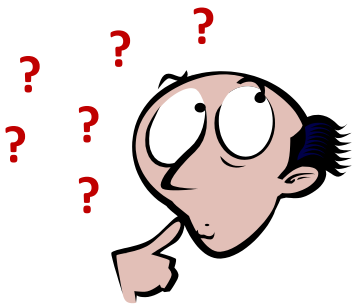
Άνω και Κάτω Φράγματα

Παράδειγμα

Άνω Φράγμα

Αλγόριθμος: **Merge_Sort**

Υπάρχει Ταχύτερος Αλγόριθμος !!!



Το πλήθος των Βασικών Πράξεων (S[i]>S[j]) του Merge_Sort είναι c n logn, και επομένως $W_{Merge\_Sort}(n) = c\ n\ \log n$.

Άνω και Κάτω Φράγματα

Παράδειγμα

Κάτω Φράγμα

Υπολόγισε ένα ελάχιστο πλήθος Βασικών Πράξεων που απαιτούνται για την επίλυση του Προβλήματος SORTING

- Είσοδος $(x_1, x_2, \ldots, x_n)$
- Αρχικά $n!$ περιπτώσεις:

$x_1 < x_2 < x_3 < \ldots < x_n$

$x_2 < x_1 < x_3 < \ldots < x_n$

$x_3 < x_1 < x_2 < \ldots < x_n$

$\ldots$

$x_n < x_{n-1} < \ldots < x_1$

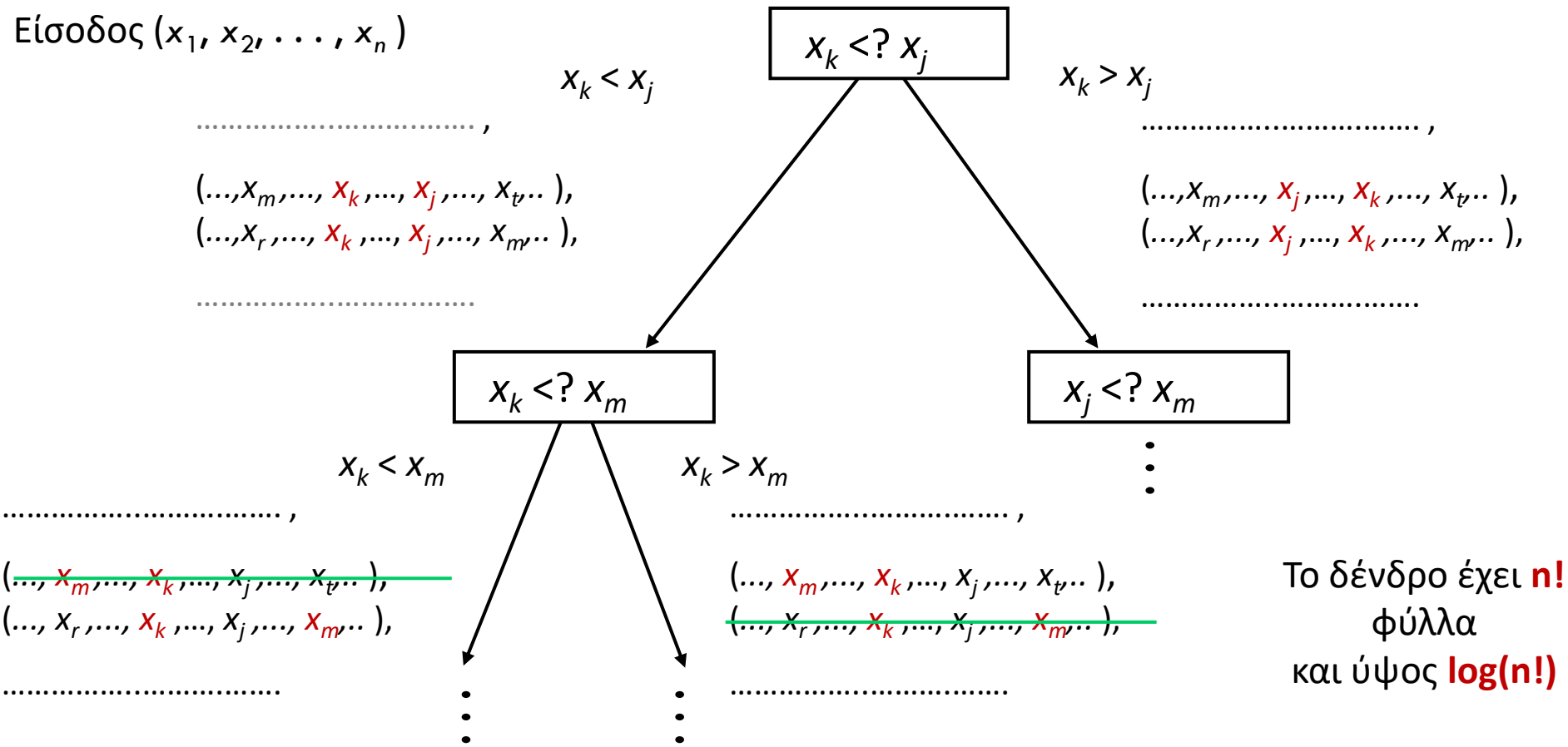
- Σε κάθε σύγκριση το πλήθος των περιπτώσεων υποδιπλασιάζεται (στην καλύτερη περίπτωση)
- Πλήθος συγκρίσεων $\geq \log(n!) \geq (n \log n)/4$

Οποιοσδήποτε αλγόριθμος ταξινόμησης n στοιχείων χρειάζεται $\Omega(n \log n)$ συγκρίσεις

Άνω και Κάτω Φράγματα

Παράδειγμα

Κάτω Φράγμα



Το δένδρο έχει **n!** φύλλα και ύψος **log(n!)**

Άνω και Κάτω Φράγματα

Παράδειγμα

Κάτω Φράγμα

Είσοδος $(x_1, x_2, \ldots, x_n)$

$$\left(n/2\right)^{n/2} \leq \underbrace{n/2 \text{ ακέραιοι με τιμή} \geq n/2}$$

$$n! = \underbrace{1 \times 2 \times 3 \times \ldots \times n/2 \times \ldots \times n}_{n \text{ ακέραιοι με τιμή} \leq N}$$

$$\leq n^n$$

$$n/2 \log(n/2) \leq \log(n!) \leq n \log n$$

$$\log(n!) = \Omega(n \log n)$$

Οποιοσδήποτε αλγόριθμος ταξινόμησης *n* στοιχείων χρειάζεται $\Omega(n \log n)$ συγκρίσεις

Άνω και Κάτω Φράγματα

Παράδειγμα

Πρόβλημα Ταξινόμησης

- Η πολυπλοκότητα του Αλγόριθμου **Merge_Sort** είναι **O(nlogn)**
- Το ελάχιστο πλήθος Βασικών Πράξεων που απαιτούνται για την επίλυση του **Προβλήματος της Ταξινόμησης** είναι **Ω(nlogn)**

Άνω Φράγμα **O(nlogn)** ↓

Κάτω Φράγμα **Ω(nlogn)** ↑

Φράγματα - Κεκτημένη Γνώση !

- **Αλγόριθμοι**: **παρέχουν *Άνω φράγματα*** !
  - ταξινόμηση με συγκρίσεις και ανταλλαγές (Bubble_Sort): $O(n^2)$
  - ταξινόμηση με συγκρίσεις και δημιουργία υποσυνόλων (Merge_Sort): $O(n \log n)$
- **Αποδείξεις δυσκολίας**: **παρέχουν *Κάτω φράγματα*** !!
  - ταξινόμηση με συγκρίσεις και δημιουργία υποσυνόλων: $\Omega(n \log n)$

Φράγματα - Κεκτημένη Γνώση !

- Για τον καθορισμό της πολυπλοκότητας ενός προβλήματος Π απαιτείται αμφίδρομη προσπάθεια !!!... Θέλουμε:

  1. **Το άνω φράγμα** – την *μικρότερη πολυπλοκότητα* απ' όλους του γνωστούς αλγόριθμους που λύνουν το Π.

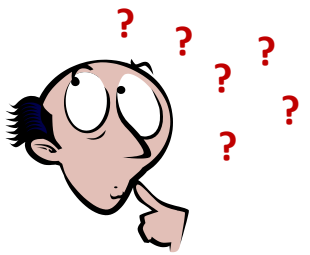
  2. **Το κάτω φράγμα** – την *μεγαλύτερη συνάρτηση* ***f*** για την οποία έχει αποδειχθεί (μαθηματικά) ότι όλοι οι πιθανοί αλγόριθμοι που λύνουν το Π απαιτούν να έχουν πολυπλοκότητα μεγαλύτερη ή ίση από ***f***.

- Εάν υπάρχει **ΚΕΝΟ** μεταξύ (1)-(2) $\Rightarrow$ έρευνα !!!

Πολυπλοκότητα: Ανοικτά Ερωτήματα

Αλγόριθμοι - Προβλήματα – Χρόνος !

- Εκτός από κάποιες ειδικές περιπτώσεις, για κανένα πρόβλημα δεν γνωρίζουμε πόσο γρήγορα μπορεί να λυθεί.



Άνω Φράγμα

Όρια Πολυπλοκότητας Προβλήματος

ΚΕΝΟ

Πρόβλημα Π

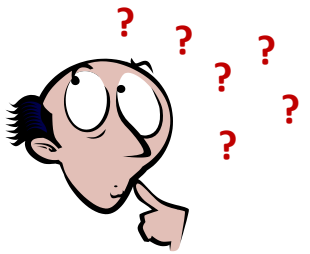
Κάτω Φράγμα

Πολυπλοκότητα: Ανοικτά Ερωτήματα

Αλγόριθμοι - Προβλήματα – Χρόνος !

- Εκτός από κάποιες ειδικές περιπτώσεις, για κανένα πρόβλημα δεν γνωρίζουμε πόσο γρήγορα μπορεί να λυθεί.

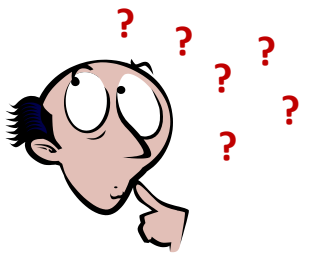
NP-πλήρη προβλήματα 



Αλγόριθμοι - Προβλήματα – Χρόνος !

- Εκτός από κάποιες ειδικές περιπτώσεις, για κανένα πρόβλημα δεν γνωρίζουμε πόσο γρήγορα μπορεί να λυθεί.

NP-πληρότητα: ισχυρή ένδειξη απουσίας αποδοτικού αλγορίθμου !!!



Million Dollar Question!

(Clay Institute millennium problems)

Αλγόριθμοι - Προβλήματα – Χρόνος !

- Ακόμα και για τον πολλαπλασιασμό αριθμών δεν γνωρίζουμε τον βέλτιστο αλγόριθμο!!!
- Ο σχολικός τρόπος πολλαπλασιασμού αριθμών με n ψηφία χρειάζεται $O(n^2)$ βήματα.
- Υπάρχουν αλγόριθμοι που χρειάζονται $O(n \log n)$ βήματα.
- **Ερώτημα** !... Υπάρχει αλγόριθμος που χρειάζεται μόνο $O(n)$ βήματα**;**

Ένα ακόμα ανοικτό ερώτημα στην Επιστήμη μας !!!

Τέλος...

Τώρα Γνωρίζω !

Ποια η θέση των Αλγορίθμων στην Επιστήμη μας !!

Πόσο σημαντικός τομέας !!

Τι γνώση υπάρχει !!

Αποτελέσματα Σταθμούς!

Τέλος…

Δύο ιδέες που άλλαξαν τον Κόσμο !!!

- ✓ **Το 1454**, στη Γερμανική πόλη Mainz, ένας χρυσοχόος ονόματι **Johan Gutenberg** ανακάλυψε έναν τρόπο να τυπώνει βιβλία συνδυάζοντας μετακινούμενα μεταλλικά στοιχεία!

- ✓ **Διαδόθηκε** έτσι η γραφή, η ανάγνωση, η ανθρώπινη διανόηση απελευθερώθηκε, η επιστήμη και η τεχνολογία θριάμβευσαν, πραγματοποιήθηκε η Βιομηχανική Επανάσταση!!!

- ✓ Πολλοί ιστορικοί αναφέρουν ότι όλα αυτά οφείλονται στην **τυπογραφία** !!!

Τέλος…

Δύο ιδέες που άλλαξαν τον Κόσμο !!!

- ✓ **Άλλοι όμως επιμένουν ότι η σημαντική εξέλιξη δεν ήταν η τυπογραφία, αλλά οι *αλγόριθμοι* !!!**

- ✓ Σήμερα έχουμε τόσο πολύ συνηθίσει το δεκαδικό σύστημα ώστε να ξεχάσουμε ότι ο **Gutenberg** θα έγραφε το 1454 ως **`MCDXLVIII`**

- ✓ Πως θα προσθέτατε δύο αριθμούς γραμμένους στο *Λατινικό* σύστημα;

- ✓ *Με τι ισούται το* **`MCDXLVIII + DCCCXII;`**

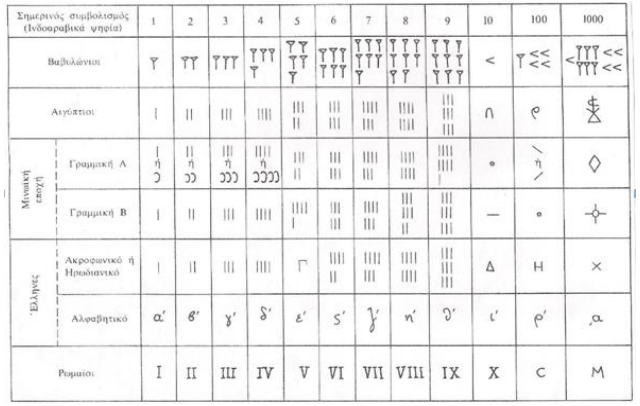
Τέλος…

Δύο ιδέες που άλλαξαν τον Κόσμο !!!

- ✓ Το Δεκαδικό Σύστημα (ανακαλύφθηκε στην Ινδία ~ 600 μΧ) *αποτέλεσε μια επανάσταση* στον **ποσοτικό συλλογισμό**!!!

- ✓ Συμπαγής γραφή τεράστιων αριθμών με μόνο **10 σύμβολα,** και αποδοτική *εκτέλεση αριθμητικών* πράξεων με μερικά **στοιχειώδη βήματα** !!!

- ✓ Το μέσο μετάδοσης με μεγάλη επιρροή ήταν ένα εγχειρίδιο του 9ου αιώνα, γραμμένο στα *Αραβικά*, από έναν άνθρωπο που ζούσε στη Βαγδάτη, τον **Al-Khwarizmi**.



Τέλος…

Δύο ιδέες που άλλαξαν τον Κόσμο !!!

- ✓ Ο **Al-Khwarizmi** διατύπωσε τις **βασικές μεθόδους** για την *πρόσθεση*, τον *πολλαπλασιασμό*, την *διαίρεση*, την εξαγωγή *τετραγωνικών ριζών* και *υπολογισμό των ψηφίων* του π.

- ✓ Οι μέθοδοι ήταν *ακριβείς*, *μη-διφορούμενες*, *μηχανικές*, *αποδοτικές*, *ορθές* – εν συντομία, ήταν ***Αλγόριθμοι*** !!!

- ✓ Το δεκαδικό θεσιακό σύστημα και οι αριθμητικοί του αλγόριθμοι έχουν παίξει τεράστιο ρόλο στον Δυτικό πολιτισμό !!!... Ανέπτυξαν την επιστήμη και την τεχνολογία !!

Και όταν σχεδιάστηκε ο Η/Υ ενσωματώθηκε το θεσιακό σύστημα στα bit, στις λέξεις και στην αριθμητική του μονάδα !!!

Τέλος...

Άνθρωποι που άλλαξαν τον Κόσμο !!!


Muhammad ibn Musa al-...

Euclid


Pythagoras


Leonhard Euler


Leonardo da Vinci


Alan Turing


Archimedes


Pierre de Fermat


Isaac Newton


René Descartes


Blaise Pascal

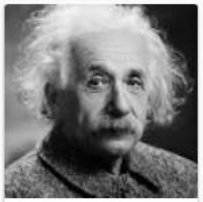

Luca Pacioli


Carl Friedrich Gauss


Gottfried Wilhelm Leib...


Bernhard Riemann

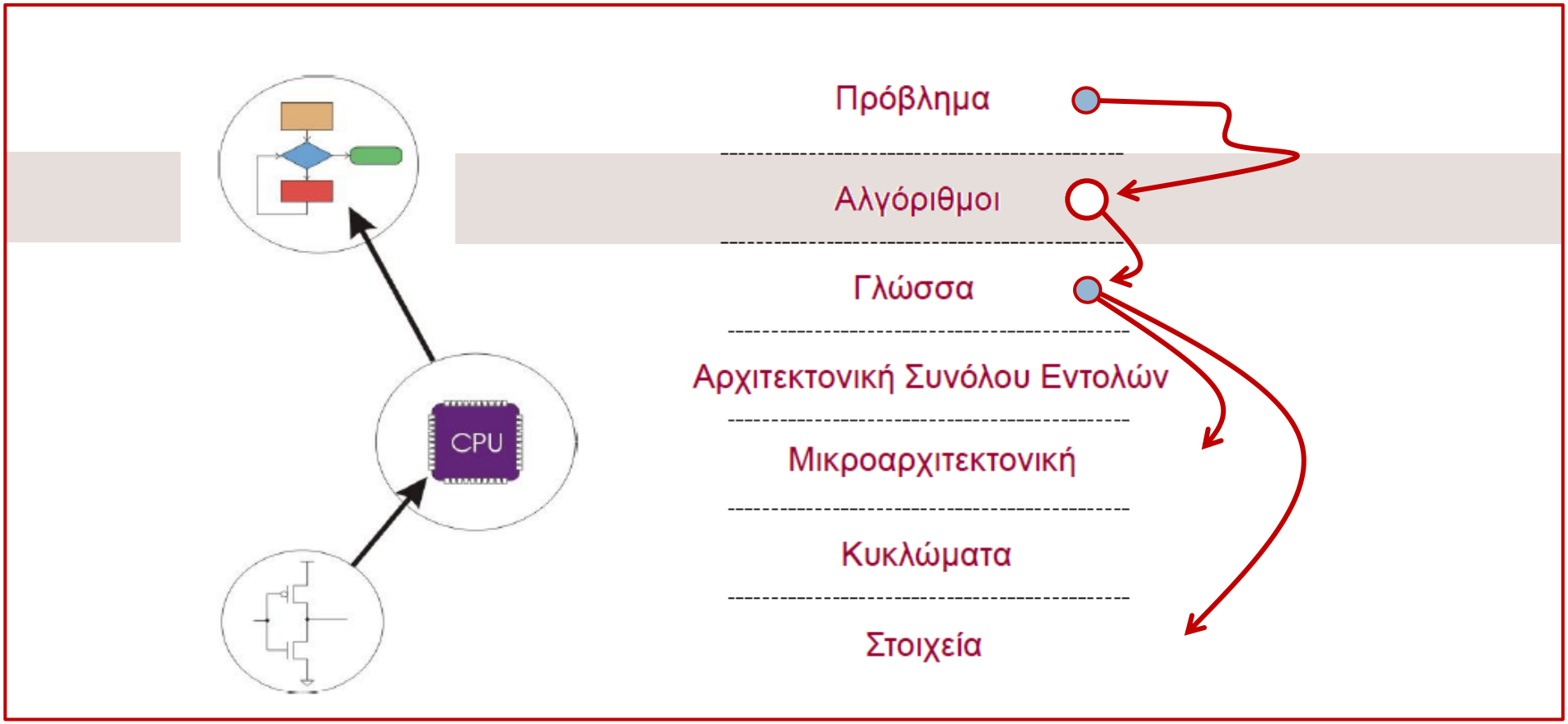

Galileo Galilei


Albert Einstein


Diophantus

Τέλος...

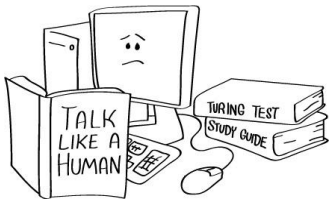
Εν κατακλείδι !!!



Τέλος…

Εν κατακλείδι !!!

- ✓ Είναι σημαντικό να γνωρίζουμε *τί μπορούμε* και *τί δεν μπορούμε να κάνουμε με τους υπολογιστές, και πόσο καλά* !



- ✓ Η γνώση τεχνικών και μεθοδολογιών *σχεδίασης και ανάλυσης αλγορίθμων* είναι απαραίτητη για να κατανοήσουμε την σύγχρονη τεχνολογία … και να συμβάλουμε στην ανάπτυξή της !!

- ✓ Τα *μαθηματικά* είναι πάντα εκεί και αναγκαία !!!

Τέλος...

Καλή Επιτυχία

στο Μάθημά μας!!!

"Teachers open the door, but you must enter by yourself."

Chinese Proverb

Ευχαριστώ για τη Συμμετοχή σας !!!

Σταύρος Δ. Νικολόπουλος

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

Webpage: www.cs.uoi.gr/~stavros