



# ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

<http://eclass.aueb.gr/courses/INF511/>

## Αλγόριθμοι (ΚΕΦΑΛΑΙΟ 5)

Αλκμήνη Σγουρίτσα

Κοδριγκτώνος 12, 2<sup>ος</sup> όροφος

E-mail: [alkmini@aueb.gr](mailto:alkmini@aueb.gr)

# ΚΕΦΑΛΑΙΟ 5: Αλγόριθμοι

- Αρχικές έννοιες και ορισμοί
- Αναπάρασταση αλγορίθμου με ψευδοκώδικα
- Διαγράμματα ροής και δομές επανάληψης
- Βασικοί Αλγόριθμοι:
  - Αλγόριθμος σειριακής αναζήτησης
  - Αλγόριθμος InsertionSort για αλφαβητική ταξινόμηση
  - Αναδρομικοί αλγόριθμοι
- Εισαγωγή στην αλγοριθμική πολυπλοκότητα
- Αλγοριθμική Θεωρία Παιγνίων

# Αλγόριθμος: Ορισμός (1)

- **Διατεταγμένο (ordered)** σύνολο βημάτων (που να ξεχωρίζουν)
- **Πεπερασμένα (finite)** βήματα
- **Σαφώς ορισμένα (unambiguous)** βήματα
- **Εκτελέσιμα εφικτά** βήματα: π.χ. όχι κάτι σαν

`Make a list of all the positive integers`

- το οποίο ορίζει μία **τερματιζόμενη** διαδικασία
  - Αν και ορίζονται και αλγόριθμοι για ατέρμονες διεργασίες, π.χ. Διαίρεση του 1 με το 3

# Αλγόριθμοι-Παραδείγματα

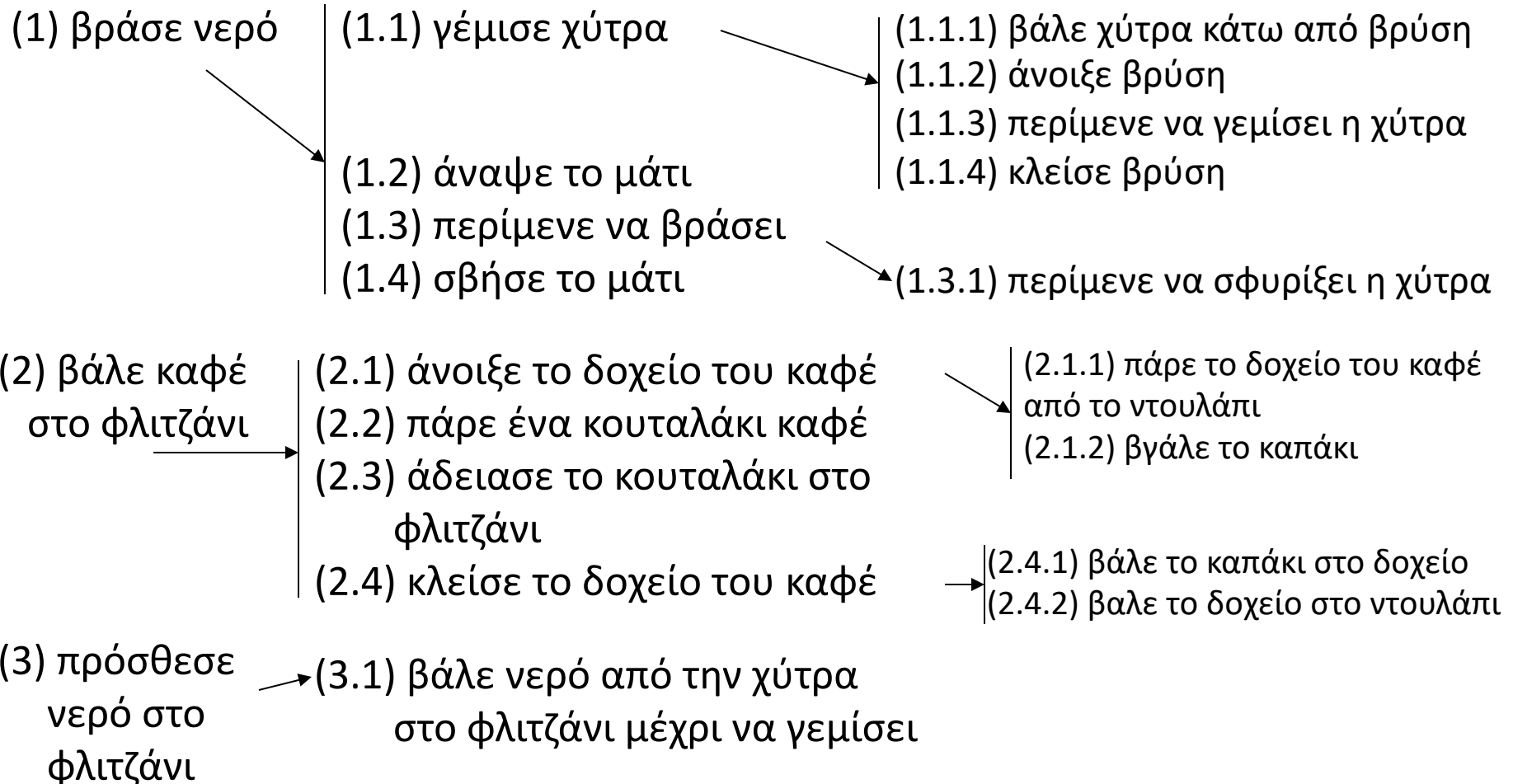
Έχουμε ήδη δει αλγορίθμους για:

- Μετατροπή δεκαδικής αναπάραστασης αριθμού σε δυαδική
- Ανίχνευση και διόρθωση σφαλμάτων
- Χρονοδρομολόγηση διεργασιών από το λειτουργικό σύστημα
- Τον κύκλο μηχανής μιας CPU:

```
As long as the halt instruction has not been executed  
continue to execute the following steps:  
  a. Fetch an instruction.  
  b. Decode the instruction.  
  c. Execute the instruction.
```

# Παράδειγμα Αλγορίθμου

**Αλγόριθμος:** φτιάξε στιγμιαίο καφέ



# Ακολουθία βημάτων για αλγόριθμο

1. βάλε την χύτρα κάτω από βρύση
2. άνοιξε την βρύση → **εάν δεν υπάρχει νερό;**
3. περίμενε να γεμίσει η χύτρα
4. κλείσε την βρύση
5. άναψε το μάτι
6. περίμενε να σφυρίξει η χύτρα → **εάν δεν υπάρχει ρεύμα δεν θα σφυρίξει ποτέ**
7. σβήσε το μάτι
8. πάρε το δοχείο του καφέ από το ντουλάπι
9. βγάλε το καπάκι
10. πάρε ένα κουταλάκι καφέ → **εάν δεν υπάρχει καφές, να δοκιμάσει το επόμενο δοχείο καφέ; εάν είναι όλα άδεια, τι να κάνει;**
11. άδειασε το κουταλάκι στο φλιτζάνι
12. βάλε το καπάκι στο δοχείο
13. βάλε το δοχείο στο ντουλάπι
14. βάλε νερό από την χύτρα στο φλιτζάνι μέχρι να γεμίσει

**Ένας καλός αλγόριθμος πρέπει να λαμβάνει υπόψη όλα τα ενδεχόμενα που μπορεί να προκύψουν κατά την εκτέλεση ενός προγράμματος.**

# Αλγόριθμοι, προγράμματα, Γλώσσες



- Πώς περιγράφουμε έναν αλγόριθμο;
  - Η φυσική γλώσσα (ελληνικά, αγγλικά) μπορεί να είναι **ασαφής**  
«μαζεύετε τριαντάφυλλα όσο σας επιτρέπεται»
- **Πρόγραμμα (program)**: μια αναπαράσταση ενός αλγορίθμου, γραμμένη για εκτέλεση από υπολογιστή
- **Γλώσσα προγραμματισμού**:
  - εύκολη και περιεκτική έκφραση αλγορίθμων
  - άμεσα κατανοητή από υπολογιστές **και** ανθρώπους
- **Αρχέτυπα (primitives)**: Καλά ορισμένα σύνολα από δομικά μπλοκ από τα οποία μπορούν να κατασκευαστούν αλγόριθμοι

# Αρχέτυπα ψευδοκώδικα (1)

- **Ψευδοκώδικας (pseudo-code)**: ένα λιγότερο επίσημο (από την γλώσσα προγραμματισμού) σύστημα από συμβολισμούς το οποίο μπορεί να αναπαραστήσει τις ιδέες που διέπουν έναν αλγόριθμο
- Ανάθεση:

*όνομα*  $\leftarrow$  *έκφραση*

*name*  $\leftarrow$  *expression*

```
RemainingFunds  $\leftarrow$  CheckingBalance + SavingsBalance
```

- Επιλογή συνθήκης:

**αν** (συνθήκη) **τότε** (ενέργεια 1)

**αλλιώς** (ενέργεια 2)

**if** (συνθήκη) **then** (ενέργεια 1)

**else** (ενέργεια 2)

```
if (year is leap year)
  then (daily total  $\leftarrow$  total divided by 366)
  else (daily total  $\leftarrow$  total divided by 365)
```

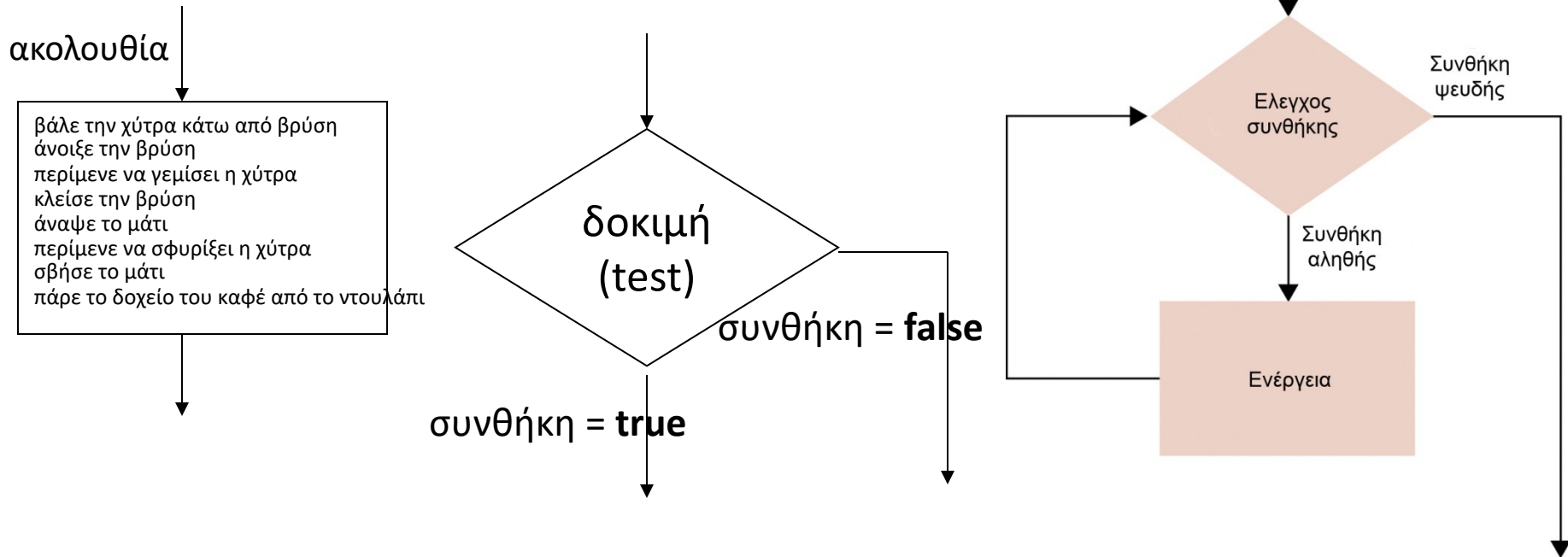


# Αρχέτυπα ψευδοκώδικα (2)

- Επαναλαμβανόμενη εκτέλεση:
  - όσο (συνθήκη) **κάνε** (ενέργεια) [ **while** (συνθήκη) **do** (ενέργεια) ]
  - επανάλαβε** (ενέργεια) **μέχρι** (συνθήκη) [ **repeat** (συνθήκη) **until** (ενέργεια) ]
  - `while (tickets remain to be sold) do (sell a ticket)`
- Διαδικασία (procedure): Μπλοκ από εντολές που εκτελούν μια αυτόνομη λειτουργία
  - Λέγεται και: Υποπρόγραμμα (subprogram) υπορουτίνα (subroutine), συνάρτηση (function)
  - Ορίζεται ως εξής:  
`procedure name ( )`
- Πρέπει να είναι γενική και εφαρμόσιμη σε κάθε περίπτωση
  - Π.χ. η διαδικασία για ταξινόμηση μιας λίστας ονομάτων κατά αλφαβητική σειρά πρέπει να εφαρμόζεται **σε οποιαδήποτε λίστα ονομάτων** και όχι σε μια μόνο
  - Π.χ. **Procedure Sort (List)**

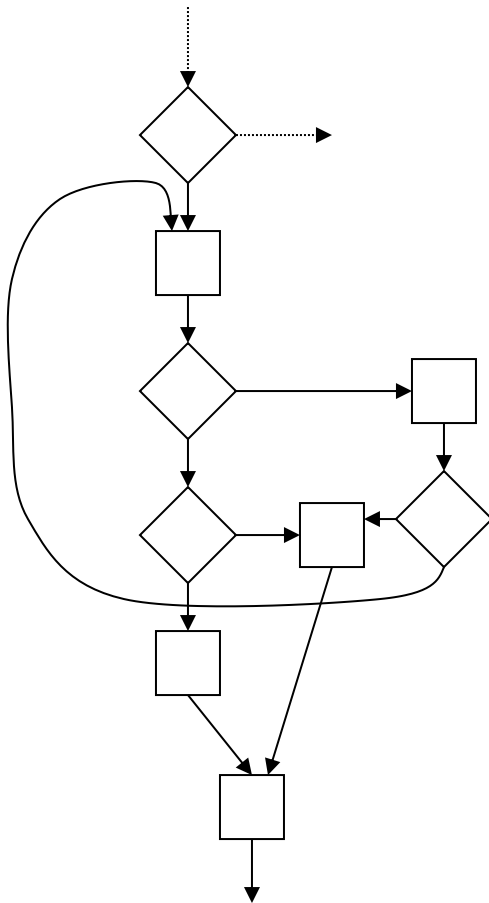
# Διαγράμματα ροής

- Διαγράμματα ροής (flow charts): τρόπος περιγραφής των βημάτων ενός αλγορίθμου
- Βασικά συστατικά διαγραμμάτων ροής:
  - Ενέργεια (ακολουθία)
  - Έλεγχος (δοκιμή) (διάφοροι τύποι)
  - Επανάληψη (διάφοροι τύποι)



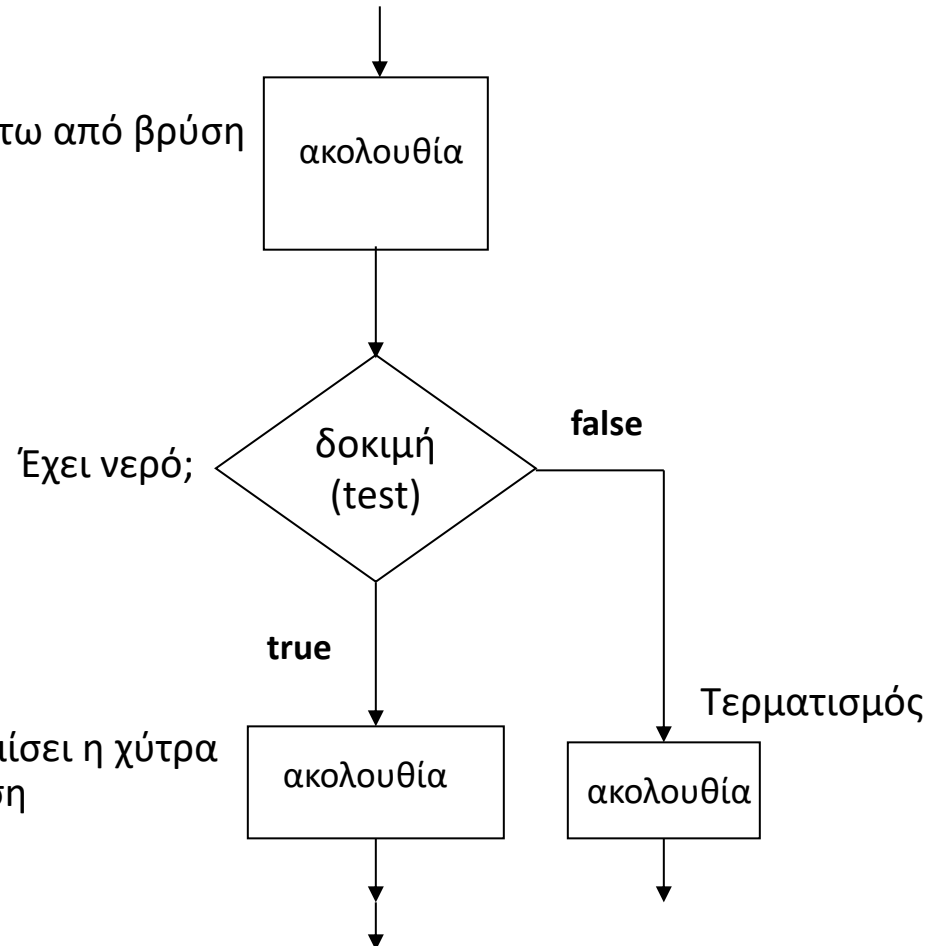
# Σύνθεση βασικών διαγραμμάτων

Αλγόριθμος: σύνθεση βασικών διαγραμμάτων

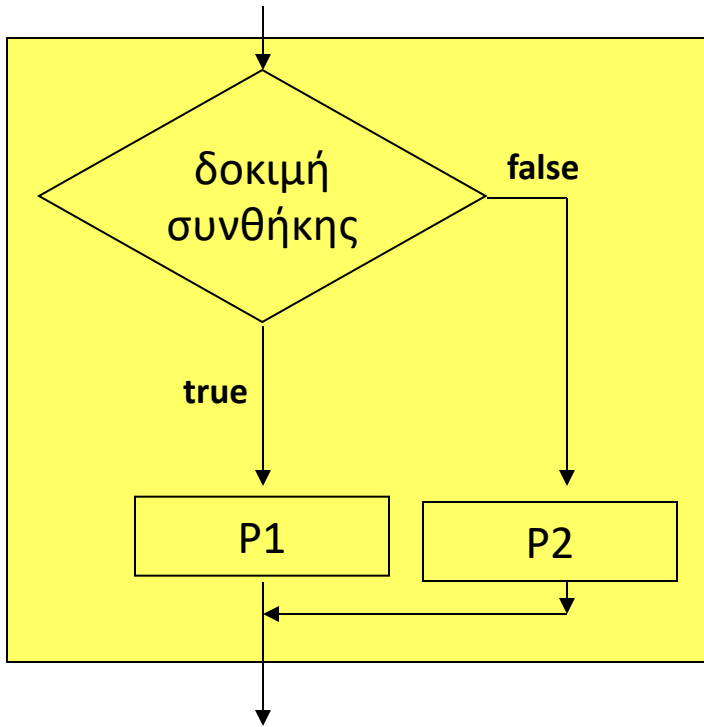


βάλε την χύτρα κάτω από βρύση  
άνοιξε την βρύση

περίμενε να γεμίσει η χύτρα  
κλείσε την βρύση  
...

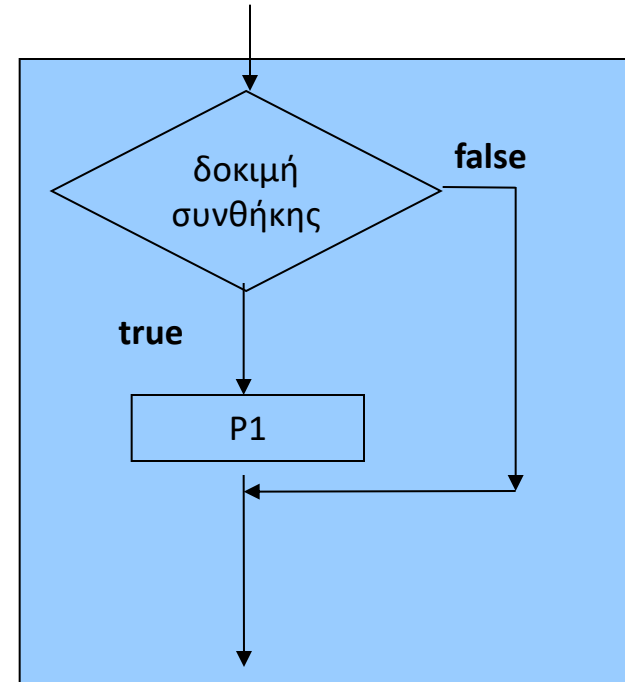


# Η δομή ελέγχου επιλογής (if)



**if condition then P1 else P2**

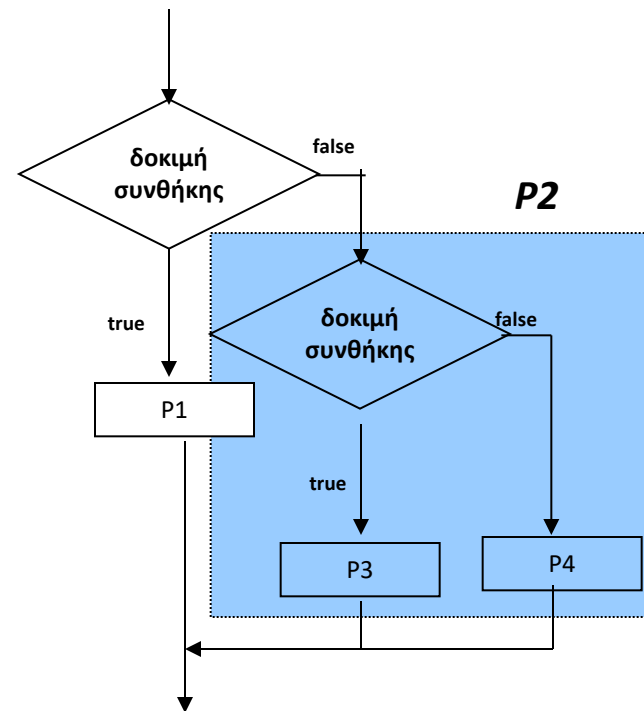
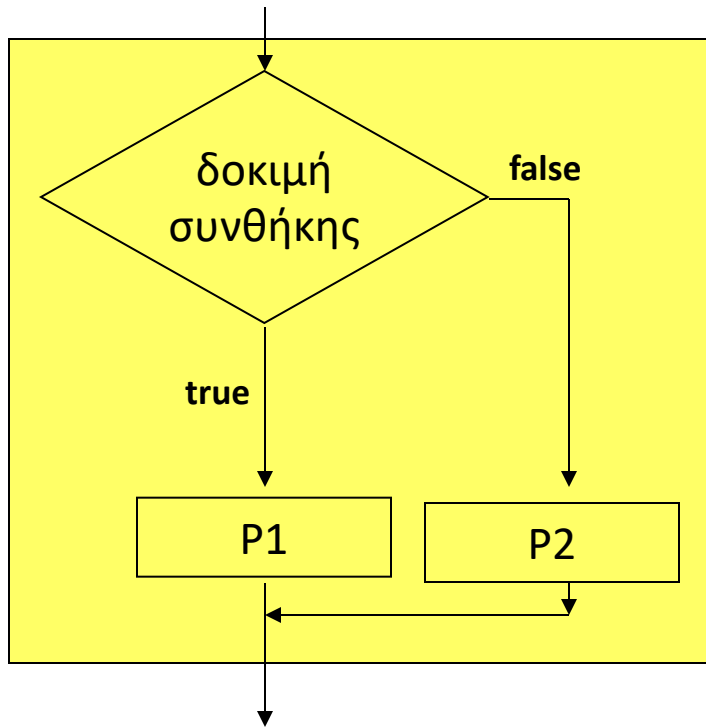
**if  $x > 3$  then  $z := 1$  else  $y := 8$**



**if condition then P1**

**if  $x = 0$  then  $x := 1$**

# Παράδειγμα φωλιασμένων επιλογών



***if condition then P1 else***

***if condition then P3 else P4***

←————— P2 —————→

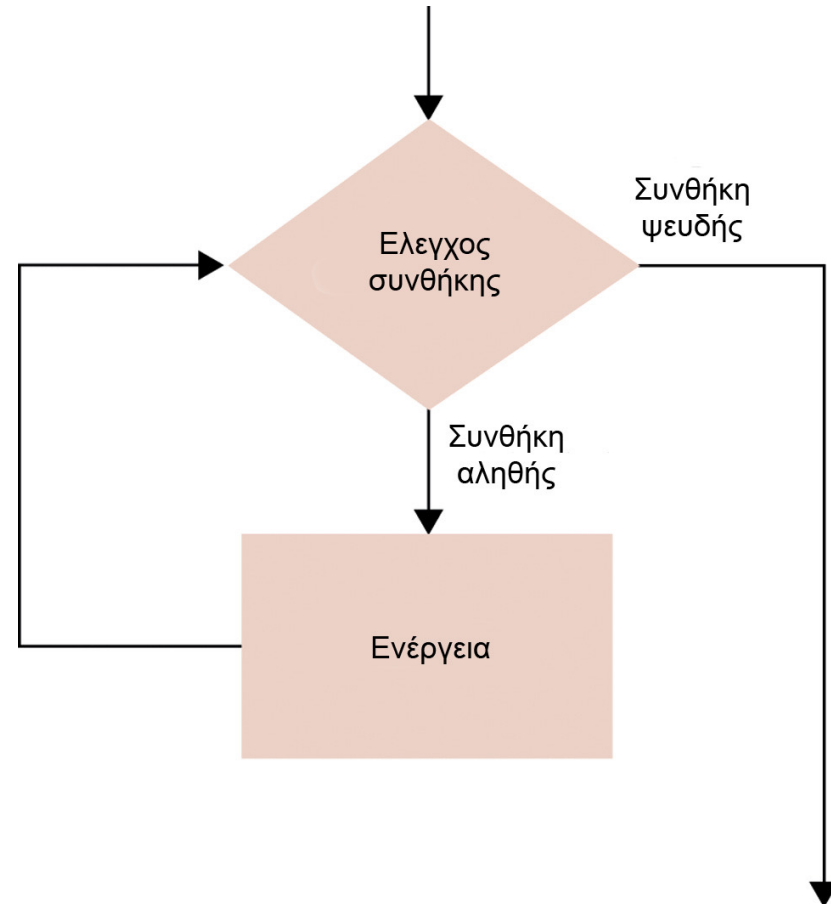
# Δομή Επανάληψης while

Βρόχος **προ-ελέγχου** (pro-test loop):  
έλεγχος συνθήκης πριν τον βρόχο

**όσο** (*συνθήκη*) **κάνε**  
(*σώμα βρόχου*)

**while** (*condition*) **do**  
(*loop body*)

Η δομή βρόχου **ενώ** (**while**)



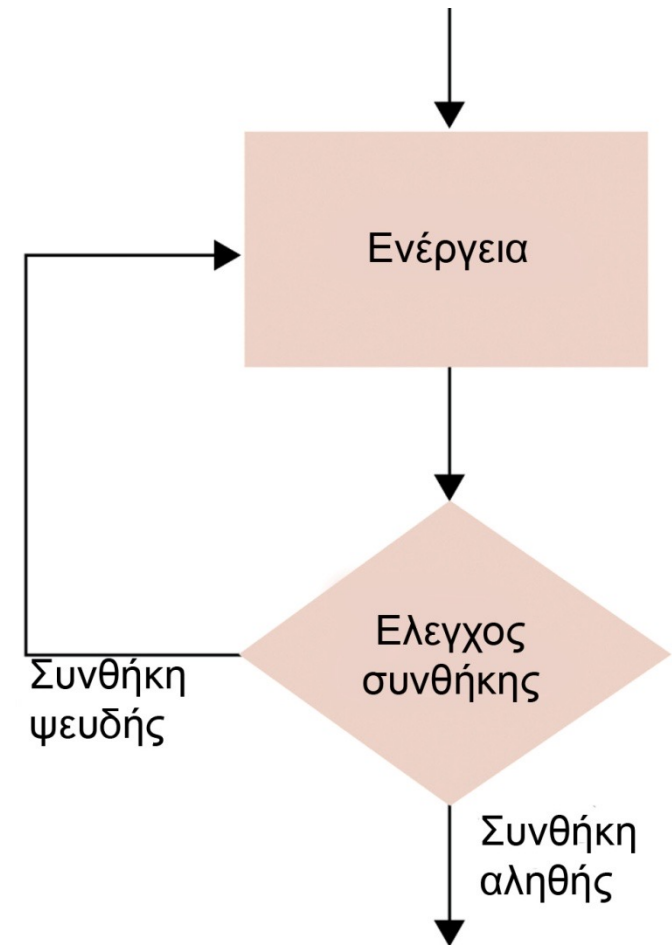
# Δομή Επανάληψης repeat

Βρόχος **μετα-ελέγχου** (post-test loop):  
έλεγχος συνθήκης μετά το βρόχο

Η δομή βρόχου **επανάλαβε (repeat)**

**επανάλαβε** (*σώμα βρόχου*)  
**μέχρι** (*συνθήκη*)

**repeat** (*loop body*)  
**until** (*condition*)

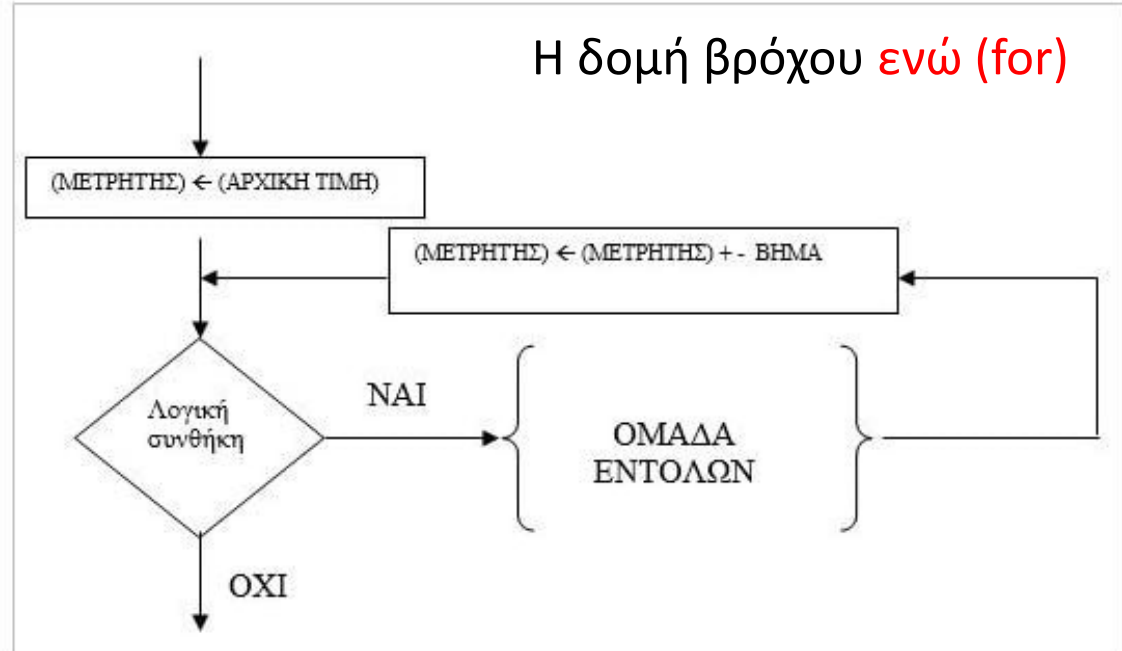


# Δομή Επανάληψης for

Βρόχος επανάληψης με βήματα:

**για** (μετρητής) **από** (αρχική τιμή) **μέχρι** (τελική τιμή)  
**με βήμα** (βήμα)  
(σώμα βρόχου)

**for** (*counter*) **from** (*starting value*) **to** (*final value*)  
**with step** (*step*)  
(*loop body*)





# Σύνοψη: Αρχέτυπα (εντολές) ψευδοκώδικα

- Ανάθεση  
    όνομα  $\leftarrow$  έκφραση
- Επιλογή συνθήκης (if then else)  
    **αν** (συνθήκη) **τότε** (ενέργεια) **αλλιώς** (ενέργεια)
- Επαναλαμβανόμενη εκτέλεση προ-έλεγχος (while do)  
    **όσο** (συνθήκη) **κάνε** (ενέργεια)
- Επαναλαμβανόμενη εκτέλεση μετα-έλεγχος (repeat until)  
    **επανέλαβε** (συνθήκη) **μέχρι** (ενέργεια)
- Επαναλαμβανόμενη εκτέλεση με βήματα:  
    **για** (μετρητής) **από** (αρχική τιμή) **μέχρι** (τελική τιμή) **με βήμα** (βήμα) (ενέργεια)
- Διαδικασία  
    **διαδικασία** όνομα (συγκεκριμένο όνομα της μονάδας)
- **Αναδρομική** (recursive) διαδικασία: καλεί τον εαυτό της

**διαδικασία** Χαιρετισμός()  
Μετρητής  $\leftarrow$  3;  
**όσο** (Μετρητής >0) **κάνε**  
    τύπωσε το μήνυμα “Γεια χαρά”  
    Μετρητής  $\leftarrow$  Μετρητής-1

```
module factorial(N)  
if N=1  
    then answer = 1  
    else answer = N * factorial(N-1)
```

# Προσοχή

Τι συμβαίνει στα παρακάτω;

Μετρητής  $\leftarrow$  3;

**όσο** (Μετρητής  $> 0$ ) **κάνε**

τύπωσε το μήνυμα “Γεια χαρά”

Μετρητής  $\leftarrow$  Μετρητής+1

Μετρητής  $\leftarrow$  1;

**όσο** (Μετρητής  $<> 6$ ) **κάνε**

Μετρητής  $\leftarrow$  Μετρητής+2

# Αλγόριθμος σειριακής αναζήτησης (sequential search)

- **Είσοδος:** ένας πίνακας/λίστα με στοιχεία **ταξινομημένα**
- **Στόχος:** να βρεθεί η ΤιμήΣτόχος (π.χ. ένα όνομα) στον πίνακα/λίστα
- Σειριακή αναζήτηση: αναζήτηση για την ΤιμήΣτόχος στοιχείο προς στοιχείο, από την αρχή προς το τέλος του πίνακα/λίστα

**διαδικασία** Αναζήτηση (Λίστα, ΤιμήΣτόχος)

**αν** (Λίστα άδεια)

**τότε**

(Δήλωσε την αναζήτηση ως ανεπιτυχή)

**αλλιώς**

(Επίλεξε την πρώτη καταχώριση της Λίστας ως ΚαταχώρισηΠροςΈλεγχο.

**όσο** (ΤιμήΣτόχος > ΚαταχώρισηΠροςΈλεγχο και  
υπάρχουν καταχωρίσεις προς σύγκριση)

**κάνε** (επίλεξε την επόμενη καταχώριση της Λίστας ως ΚαταχώρισηΠροςΈλεγχο).

**αν** (ΤιμήΣτόχος = ΚαταχώρισηΠροςΈλεγχο)

**τότε** (Δήλωσε την αναζήτηση ως επιτυχή.)

**αλλιώς** (Δήλωσε την αναζήτηση ως ανεπιτυχή.)

) **τέλος αν**

# Αλγόριθμος σειριακής αναζήτησης (sequential search)

## Παράδειγμα 1

- **Είσοδος:**

-5	-1	0	1	1	2	5	5	8	10	11	15	20	20	21
----	----	---	---	---	---	---	---	---	----	----	----	----	----	----
- **Στόχος:** να βρεθεί η τιμή 5 με σειριακή αναζήτηση
- Πόσες συγκρίσεις χρειάστηκαν; 8

**διαδικασία** Αναζήτηση (Λίστα, ΤιμήΣτόχος)

**αν** (Λίστα άδεια)

**τότε**

(Δήλωσε την αναζήτηση ως ανεπιτυχή)

**αλλιώς**

(Επίλεξε την πρώτη καταχώριση της Λίστας ως ΚαταχώρισηΠροςΈλεγχο.

**όσο** (ΤιμήΣτόχος > ΚαταχώρισηΠροςΈλεγχο και  
υπάρχουν καταχωρίσεις προς σύγκριση)

**κάνε** (επίλεξε την επόμενη καταχώριση της Λίστας ως ΚαταχώρισηΠροςΈλεγχο).

**αν** (ΤιμήΣτόχος = ΚαταχώρισηΠροςΈλεγχο)

**τότε** (Δήλωσε την αναζήτηση ως επιτυχή.)

**αλλιώς** (Δήλωσε την αναζήτηση ως ανεπιτυχή.)

) **τέλος αν**

# Αλγόριθμος σειριακής αναζήτησης (sequential search)

## Παράδειγμα 2

- **Είσοδος:**

-5	-1	0	1	1	2	5	5	8	10	11	15	20	20	21
----	----	---	---	---	---	---	---	---	----	----	----	----	----	----
- **Στόχος:** να βρεθεί η τιμή 6 με σειριακή αναζήτηση
- Πόσες συγκρίσεις χρειάστηκαν; **10**

**διαδικασία** Αναζήτηση (Λίστα, ΤιμήΣτόχος)

**αν** (Λίστα άδεια)

**τότε**

(Δήλωσε την αναζήτηση ως ανεπιτυχή)

**αλλιώς**

(Επίλεξε την πρώτη καταχώριση της Λίστας ως ΚαταχώρισηΠροςΈλεγχο.

**όσο** (ΤιμήΣτόχος > ΚαταχώρισηΠροςΈλεγχο και  
υπάρχουν καταχωρίσεις προς σύγκριση)

**κάνε** (επίλεξε την επόμενη καταχώριση της Λίστας ως ΚαταχώρισηΠροςΈλεγχο).

**αν** (ΤιμήΣτόχος = ΚαταχώρισηΠροςΈλεγχο)

**τότε** (Δήλωσε την αναζήτηση ως επιτυχή.)

**αλλιώς** (Δήλωσε την αναζήτηση ως ανεπιτυχή.)

) **τέλος αν**

# Αλγόριθμος σειριακής αναζήτησης (sequential search)

- Ο έλεγχος (σύγκριση) συνιστά την πιο «δύσκολη» πράξη σε αυτό το παράδειγμα.
- Ο αριθμός ελέγχων θα καθορίσει το χρόνο εκτέλεσης, εφόσον ελέγξουμε ότι δεν γίνονται πολλές πιο «εύκολες» πράξεις
- Πόσοι έλεγχοι (συγκρίσεις) χρειάζονται στη χειρότερη περίπτωση για σειριακή αναζήτηση ενός ονόματος σε μια λίστα μήκους  $n$ ;
  - Περίπου  $n$  (για την ακρίβεια  $n+1$ )
- Πόσοι έλεγχοι (συγκρίσεις) χρειάζονται στην καλύτερη περίπτωση για σειριακή αναζήτηση ενός ονόματος σε μια λίστα μήκους  $n$ ;
  - 2
- Πόσοι έλεγχοι (συγκρίσεις) χρειάζονται κατά μέσο όρο για σειριακή αναζήτηση ενός ονόματος σε μια λίστα μήκους  $n$  (υποθέστε ότι δεν υπάρχουν επαναλήψεις ονομάτων);
  - Περίπου  $n/2$  (για την ακρίβεια  $(n+1)/2$ )

-5	-1	0	1	2	3	5	7	8	10	11	15	18	20	21
----	----	---	---	---	---	---	---	---	----	----	----	----	----	----

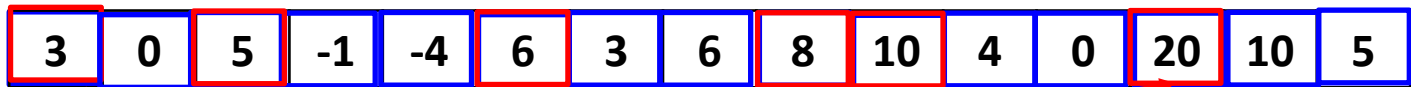
# Αλγόριθμος εύρεσης μεγίστου

- **Είσοδος:** ένας πίνακας/λίστα με στοιχεία **όχι απαραίτητα ταξινομημένα**
- **Στόχος:** να βρεθεί η μέγιστη τιμή στον πίνακα/λίστα

## Παράδειγμα

- **Είσοδος:**

3	0	5	-1	-4	6	3	6	8	10	4	0	20	10	5
---	---	---	----	----	---	---	---	---	----	---	---	----	----	---
- **Στόχος:** να βρεθεί η μέγιστη τιμή
- Πόσα στοιχεία πρέπει να ελέγξω;
  - Πρέπει να ελέγξω **όλο** τον πίνακα, αλλιώς αν δεν ελέγξω κάποιο στοιχείο, εκεί μπορεί να βρίσκεται το μέγιστο.
  - Μπορώ να κοιτάξω τα στοιχεία σειριακά και να **«θυμάμαι»** το μέγιστο που έχω βρει **ως τώρα**.



max = 3 / 5 / 6 / 8 / 10 / 20

max position

# Αλγόριθμος εύρεσης μεγίστου

## Ψευδοκώδικας

Διαδικασία  $\text{max}(A[1..n])$

Είσοδος: Πίνακας στοιχείων  $A[1..n]$

Έξοδος: Η θέση του μέγιστου στοιχείου του  $A$

$\text{max} = A[1]$ ,  $\text{position} = 1$  (αρχικοποίηση)

for  $i = 2$  to  $n$ : (ΣΤΟΝ ψευδοκώδικα το βήμα εννοείται 1)  
    if  $A[i] > \text{max}$ : (αν βρεις μεγαλύτερο)  
         $\text{max} = A[i]$ ,  $\text{position} = i$  (κάνε update)

return  $\text{position}$

- Πόσοι έλεγχοι (συγκρίσεις) χρειάζονται στη χειρότερη περίπτωση;
  - Περίπου  $n$  (για την ακρίβεια  $n-1$ )
- Πόσοι έλεγχοι (συγκρίσεις) χρειάζονται στην καλύτερη περίπτωση;
  - Χρειάζονται περίπου  $n$  (για την ακρίβεια  $n-1$ ) για οποιαδήποτε είσοδο μεγέθους  $n$ .



# Αλγόριθμος SelectionSort για ταξινόμηση σε αύξουσα σειρά ενός πίνακα αριθμών

- **Είσοδος:** ένας πίνακας/λίστα με στοιχεία **όχι απαραίτητα ταξινομημένα**
- **Έξοδος:** Ο πίνακας/λίστα ταξινομημένος
  
- **Θα χρησιμοποιήσουμε τη διαδικασία max**
- Εύρεση του μεγίστου με τη διαδικασία max. **Ανταλλαγή με το τελευταίο στοιχείο.**
- Εύρεση του μεγίστου στον υπόλοιπο πίνακα με τη διαδικασία max. **Ανταλλαγή με το προτελευταίο στοιχείο.**
- Κ.Ο.Κ

# Αλγόριθμος SelectionSort Παράδειγμα (1/2)

3	0	5	-1	-4	6	3	6	8	10	4	0	20	10	5
---	---	---	----	----	---	---	---	---	----	---	---	----	----	---



swap

3	0	5	-1	-4	6	3	6	8	10	4	0	5	10	20
---	---	---	----	----	---	---	---	---	----	---	---	---	----	----



swap

3	0	5	-1	-4	6	3	6	8	10	4	0	5	10	20
---	---	---	----	----	---	---	---	---	----	---	---	---	----	----

3	0	5	-1	-4	6	3	6	8	5	4	0	10	10	20
---	---	---	----	----	---	---	---	---	---	---	---	----	----	----

3	0	5	-1	-4	6	3	6	0	5	4	8	10	10	20
---	---	---	----	----	---	---	---	---	---	---	---	----	----	----

3	0	5	-1	-4	4	3	6	0	5	6	8	10	10	20
---	---	---	----	----	---	---	---	---	---	---	---	----	----	----

3	0	5	-1	-4	4	3	5	0	6	6	8	10	10	20
---	---	---	----	----	---	---	---	---	---	---	---	----	----	----

# Αλγόριθμος SelectionSort Παράδειγμα (2/2)

3	0	0	-1	-4	4	3	5	5	6	6	8	10	10	20
---	---	---	----	----	---	---	---	---	---	---	---	----	----	----

3	0	0	-1	-4	4	3	5	5	6	6	8	10	10	20
---	---	---	----	----	---	---	---	---	---	---	---	----	----	----

3	0	0	-1	-4	3	4	5	5	6	6	8	10	10	20
---	---	---	----	----	---	---	---	---	---	---	---	----	----	----

3	0	0	-1	-4	3	4	5	5	6	6	8	10	10	20
---	---	---	----	----	---	---	---	---	---	---	---	----	----	----

-4	0	0	-1	3	3	4	5	5	6	6	8	10	10	20
----	---	---	----	---	---	---	---	---	---	---	---	----	----	----

-4	-1	0	0	3	3	4	5	5	6	6	8	10	10	20
----	----	---	---	---	---	---	---	---	---	---	---	----	----	----

-4	-1	0	0	3	3	4	5	5	6	6	8	10	10	20
----	----	---	---	---	---	---	---	---	---	---	---	----	----	----

-4	-1	0	0	3	3	4	5	5	6	6	8	10	10	20
----	----	---	---	---	---	---	---	---	---	---	---	----	----	----

# Αλγόριθμος SelectionSort

## Ψευδοκώδικας

Διαδικασία SelectionSort(A[1..n])

Είσοδος: Πίνακας αριθμών A[1..n]

Έξοδος: Ο πίνακας A ταξινομημένος

for  $i=n, n-1, \dots, 3, 2$ :

$p=\max(A[1..i])$  ( $p$ : η θέση του μέγιστου στο A[1..i])

$\text{swap}(A[p], A[i])$  (βάλτε τον μέγιστο του A[1..i]  
στο τέλος αυτού του υποπίνακα)

- Πόσοι έλεγχοι (συγκρίσεις) και swaps χρειάζονται στη χειρότερη περίπτωση;
  - Για κάθε  $i$  γίνονται  $i-1$  συγκρίσεις (στη max) και 1 swap. Άρα  $i$  ενέργειες.
  - Συνολικά 
$$\sum_{i=2}^n i = \frac{(n+2)(n-1)}{2} = \frac{(n+1)n}{2} - 1$$
- Πόσοι έλεγχοι (συγκρίσεις) και swaps χρειάζονται στην καλύτερη περίπτωση;
  - Χρειάζονται  $\frac{(n+1)n}{2} - 1$  για οποιαδήποτε είσοδο μεγέθους  $n$ .

# Αλγόριθμος InsertionSort για ταξινόμηση σε αύξουσα σειρά ενός πίνακα αριθμών (1)

Input:

8	1	4	9	0	6	5	2	7	3
---	---	---	---	---	---	---	---	---	---

Αλγόριθμος  
Ταξινόμησης με  
Εισαγωγή  
(InsertionSort)

Iteration 1

8	1	4	9	0	6	5	2	7	3
---	---	---	---	---	---	---	---	---	---

Iteration 1

8		4	9	0	6	5	2	7	3
---	--	---	---	---	---	---	---	---	---

Key: 1

Iteration 1

↓

8		4	9	0	6	5	2	7	3
---	--	---	---	---	---	---	---	---	---

> key

Key: 1

Iteration 1

	8	4	9	0	6	5	2	7	3
--	---	---	---	---	---	---	---	---	---

Key: 1

Τα 2 πρώτα στοιχεία είναι σε αύξουσα σειρά

# Αλγόριθμος InsertionSort για ταξινόμηση σε αύξουσα σειρά ενός πίνακα αριθμών (2)

Iteration 2



Key:   

Iteration 2

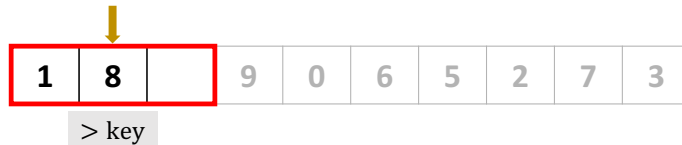


Key: 4

Ξεκινώ **συγκρίνοντας** το 4 με όλα τα στοιχεία **από δεξιά προς τα αριστερά**.

**Σταματώ** είτε **όταν βρω ένα στοιχείο που είναι μικρότερο από το key** (το 4), τότε τοποθετώ το key μετά (δεξιά) από αυτό το στοιχείο, ή **όταν έχω εξετάσει όλα τα στοιχεία αριστερά** (στο κόκκινο πλαίσιο), τότε τοποθετώ το key στην πρώτη θέση.

Iteration 2



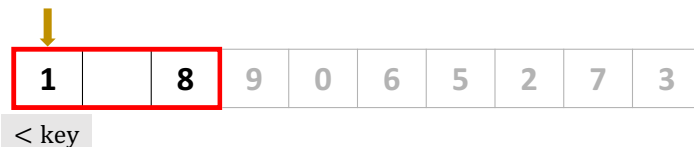
Key: 4

Iteration 2



Key: 4

Iteration 2



Key: 4

Iteration 2



Key:   

Τα 3 πρώτα στοιχεία είναι σε αύξουσα σειρά

# Αλγόριθμος InsertionSort για ταξινόμηση σε αύξουσα σειρά ενός πίνακα αριθμών (3)

Iteration 3



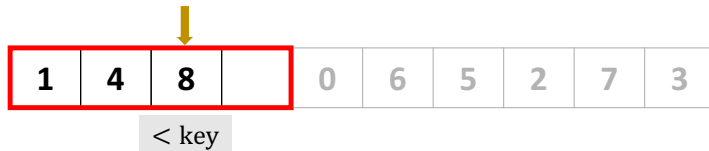
Key:

Iteration 3



Key:

Iteration 3



Key:

Iteration 3



Key:

Iteration 3



Key:

Τα 4 πρώτα στοιχεία είναι σε αύξουσα σειρά

# Ταξινόμηση σε αύξουσα σειρά μιας λίστας αριθμών (4)

Εναλλακτικά: κάνω **swap** το key με το αριστερό στοιχείο όσο το αριστερό στοιχείο (αν υπάρχει) είναι μεγαλύτερο από το key

Iteration 4

1	4	8	9	0	1	4	8	9	0
---	---	---	---	---	---	---	---	---	---

Key: 

0
---

Iteration 4

1	4	8	0	9	1	4	8	9	0
---	---	---	---	---	---	---	---	---	---

Key: 

0
---

Iteration 4

1	4	0	8	9	1	4	8	9	0
---	---	---	---	---	---	---	---	---	---

Key: 

0
---

Iteration 4

1	0	4	8	9	1	4	8	9	0
---	---	---	---	---	---	---	---	---	---

Key: 

0
---

Iteration 4

0	1	4	8	9	1	4	8	9	0
---	---	---	---	---	---	---	---	---	---

Key: 

0
---

Τα 5 πρώτα στοιχεία είναι σε αύξουσα σειρά



# Ψευδοκώδικας InsertionSort

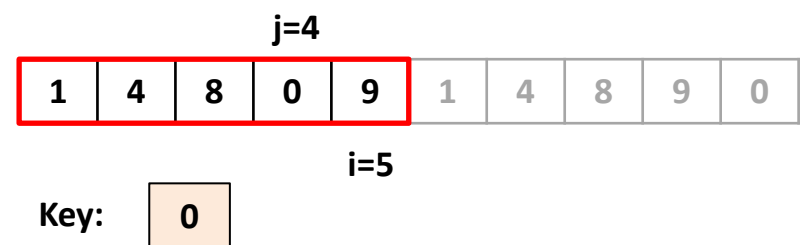
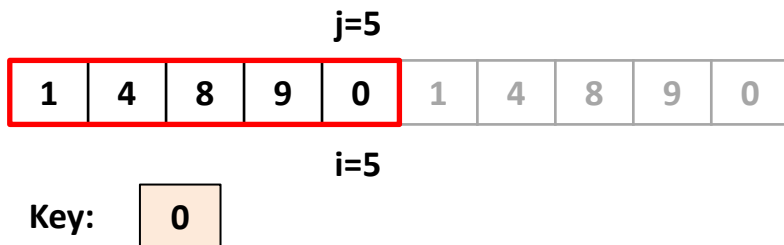
Διαδικασία InsertionSort(A[1..n])

Είσοδος: πίνακας αριθμών A[1..n]

Έξοδος: ο πίνακας A ταξινομημένος

```
for i=2 to n: (i is key's original position)
  j = i       (j is key's current position)
  while (A[j]<A[j-1] and j>1) do
    swap (A[j], A[j-1])
    j = j-1
```

After 1 round of while loop



# Ανάλυση InsertionSort

- Πόσοι **έλεγχοι (συγκρίσεις)** χρειάζονται στη **χειρότερη περίπτωση**;
  - στο 1<sup>ο</sup> βήμα (για το 2<sup>ο</sup> στοιχείο), ο αλγόριθμος θα κάνει 1 σύγκριση
  - στο 2<sup>ο</sup> βήμα (για το 3<sup>ο</sup> στοιχείο), θα κάνει 2 συγκρίσεις
  - κ.ο.κ.
  - στο n-1 βήμα (για το n στοιχείο), θα κάνει n-1 συγκρίσεις
  - Σύνολο συγκρίσεων στη **χειρότερη περίπτωση**:

$$1 + 2 + 3 + \dots + (n - 1) = \frac{1}{2} n(n - 1) = \frac{1}{2} (n^2 - n)$$

- Πόσα **swaps** χρειάζονται στη **χειρότερη περίπτωση**;
  - Ένα σε κάθε σύγκριση. Άρα  $\frac{n(n-1)}{2}$
- Ποια είναι η χειρότερη περίπτωση λίστας;
  - Μία **ανάποδα** ταξινομημένη λίστα
- Πόσοι **έλεγχοι (συγκρίσεις)** και **swaps** χρειάζονται στην **καλύτερη περίπτωση**;
  - Σε **ταξινομημένο** πίνακα: n-1 συγκρίσεις, 0 swaps

# Επιστροφή στη σειριακή αναζήτηση

## Παράδειγμα 1

- **Είσοδος:**

-5	-1	0	1	1	2	5	5	8	10	11	15	20	20	21
----	----	---	---	---	---	---	---	---	----	----	----	----	----	----
- **Στόχος:** να βρεθεί η τιμή **11** με σειριακή αναζήτηση
- **Πόσες συγκρίσεις χρειάστηκαν;** **12**
- **Νέος στόχος:** να βρεθεί η τιμή **11** σε λιγότερα βήματα.
  - Συγκρίνω με το **μεσαίο** στοιχείο. Τι συμπέρασμα βγάζω (ο πίνακας είναι **ταξινομημένος**)?
  - Το 11 βρίσκεται δεξιά του. **Γλίτσω περίπου  $n/2$  συγκρίσεις.**

<del>-5</del>	<del>-1</del>	<del>0</del>	<del>1</del>	<del>1</del>	<del>2</del>	<del>5</del>	<del>5</del>	8	10	11	15	20	20	21
---------------	---------------	--------------	--------------	--------------	--------------	--------------	--------------	---	----	----	----	----	----	----

- Συνεχίζω με τον ίδιο τρόπο στον υποπίνακα που έμεινε

-5	-1	0	1	1	2	5	5	8	10	11	<del>15</del>	<del>20</del>	<del>20</del>	<del>21</del>
----	----	---	---	---	---	---	---	---	----	----	---------------	---------------	---------------	---------------

-5	-1	0	1	1	2	5	5	<del>8</del>	<del>10</del>	<b>11</b>	15	20	20	21
----	----	---	---	---	---	---	---	--------------	---------------	-----------	----	----	----	----

# Ο αλγόριθμος της δυαδικής αναζήτησης - Ψευδοκώδικας (1)

**Διαδικασία** Αναζήτηση (Λίστα, ΤιμήΣτόχος)

**αν** ( Η Λίστα είναι άδεια)

**τότε**

(Ανάφερε ότι η αναζήτηση απέτυχε)

**αλλιώς**

[**Επίλεξε** τη “μεσαία” καταχώριση της Λίστας ως την ΚαταχώρισηΠροςΈλεγχο.

Εκτέλεσε το παρακάτω μπλοκ εντολών που αντιστοιχεί στην κατάλληλη περίπτωση.

**περίπτωση 1:** ΤιμήΣτόχος=ΚαταχώρισηΠροςΈλεγχο

(Ανέφερε ότι η αναζήτηση είναι επιτυχής.)

**περίπτωση 2:** ΤιμήΣτόχος<ΚαταχώρισηΠροςΈλεγχο

(Εφάρμοσε τη διαδικασία Αναζήτηση για να δεις αν η ΤιμήΣτόχος βρίσκεται στο τμήμα της λίστας που προηγείται της ΚαταχώρισηΠροςΈλεγχο, και ανέφερε το αποτέλεσμα αυτής της αναζήτησης.)

**περίπτωση3:** ΤιμήΣτόχος> ΚαταχώρισηΠροςΈλεγχο

(Εφάρμοσε τη διαδικασία Αναζήτηση για να δεις αν η ΤιμήΣτόχος βρίσκεται στο τμήμα της λίστα μετά την ΚαταχώρισηΠροςΈλεγχο, και ανέφερε το αποτέλεσμα αυτής της αναζήτησης.)

] **τέλος αν**

# Ο αλγόριθμος της δυαδικής αναζήτησης - Ψευδοκώδικας (2)

συνάρτηση **bins**(A[i..j], k)

Είσοδος: Ταξινομημένος πίνακας κλειδιών A[1..n],  
κλειδί k

Έξοδος: Η θέση του k στον A αν υπάρχει,  
διαφορετικά nil

if  $i > j$ : return nil (το στοιχείο δεν βρέθηκε)

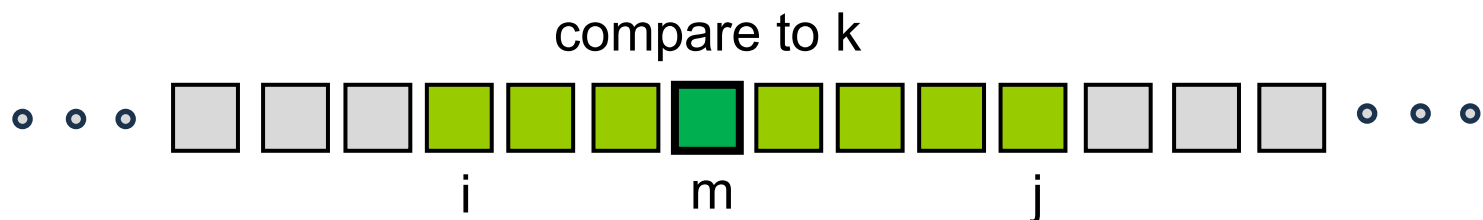
$m = \lfloor (i + j) / 2 \rfloor$  (επέλεξε το μεσαίο στοιχείο του πίνακα)

if  $k = A[m]$ : return m (έλεγξε αν είναι αυτό που ψάχνω)

(αλλιώς διάλεξε ποιον πίνακα θα κρατήσεις)

if  $k > A[m]$ : return **bins**(A[m+1..j], k)

else: return **bins**(A[i, m-1], k)

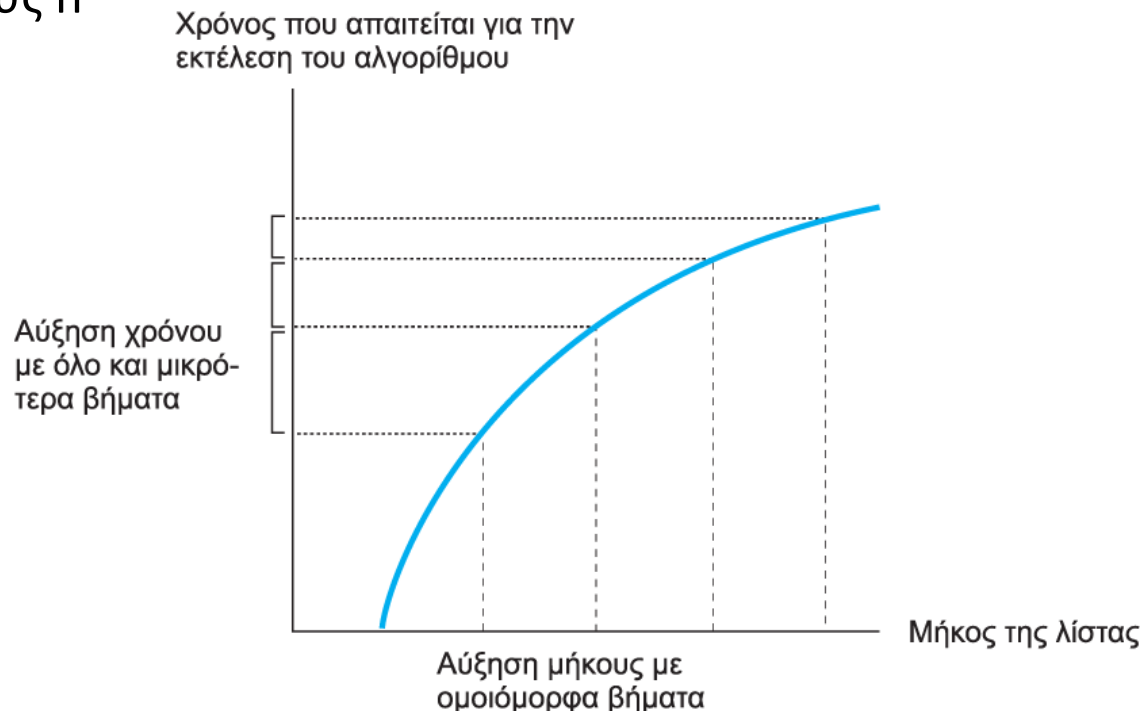


# Ο αλγόριθμος της δυαδικής αναζήτησης – Αποδοτικότητα (Πρακτικό Παράδειγμα)

- Λίστα 30,000 ονομάτων φοιτητών
- Ακολουθιακή (sequential) αναζήτηση: **κατά μέσο όρο** χρειάζεται να περάσει **15,000** ονόματα
  - Έστω **10msec** για να ελέγξει ένα όνομα
  - Μέσος χρόνος για έλεγχο: **150 sec**
- Δυαδική (binary) αναζήτηση: χρειάζεται να ελέγξει **το πολύ 15** ονόματα
  - Όσες φορές (το πολύ) μπορεί να κοπεί μια λίστα 30,000 ονομάτων στην μέση ( $\log_2(30,000)$ )
  - **10msec** για να ελέγξει ένα όνομα
  - Χρόνος για έλεγχο (στη χειρότερη περίπτωση): **0,15 sec**

# Ο αλγόριθμος της δυαδικής αναζήτησης – Αποδοτικότητα

- Ενδιαφερόμαστε για μια λίστα οποιουδήποτε μήκους  $n$
- Χειρότερη περίπτωση (worst case):
  - Για την **ακολουθιακή αναζήτηση**: στη **χειρότερη περίπτωση**, περίπου  $n$  έλεγχοι για λίστα μήκους  $n$
  - **Δυαδική αναζήτηση**: Στη **χειρότερη περίπτωση**, περίπου  $\log_2 n$  έλεγχοι για λίστα μήκους  $n$



# Αλγόριθμος Συγχώνευσης (Merge) δύο ταξινομημένων πινάκων

-5	1	2	5	10	20
----	---	---	---	----	----

-1	0	5	6	8	9
----	---	---	---	---	---

-5	-1	0	1	2	5	5	6	8	9	10	20
----	----	---	---	---	---	---	---	---	---	----	----

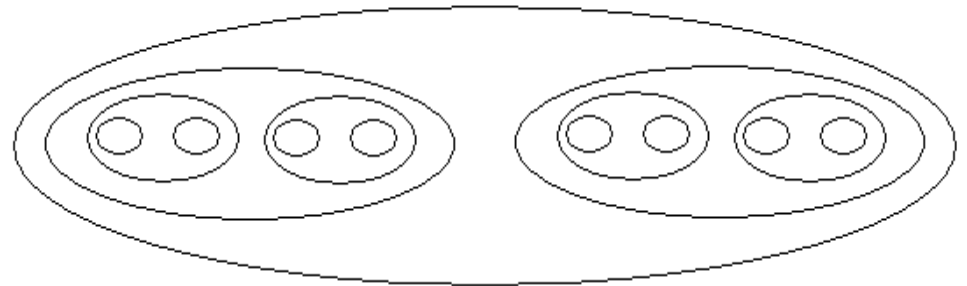
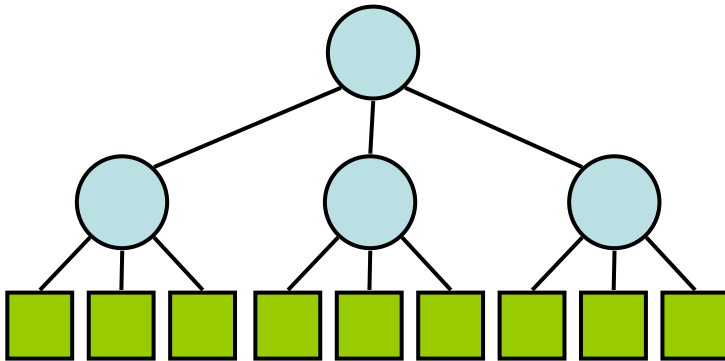
- Πόσοι έλεγχοι (συγκρίσεις) χρειάζονται στη χειρότερη περίπτωση για τη συγχώνευση δύο ταξινομημένων πινάκων μήκους  $n/2$  ο καθένας;
  - Περίπου  $n$  (για την ακρίβεια  $n-1$ )

Στη συνέχεια θα χρησιμοποιήσουμε τη διαδικασία  $\text{merge}(A[1..p], B[1..q])$ , όπου  $A$  και  $B$  δύο πίνακες μεγέθους  $p$  και  $q$ , αντίστοιχα.



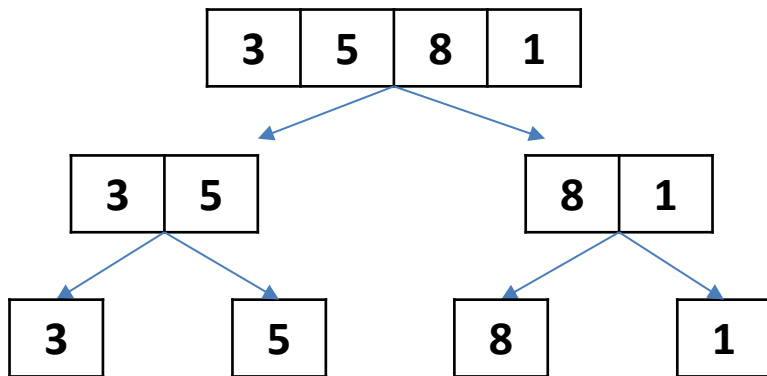
# Διαιρεί και Βασίλευε (Divide and conquer)

- **Διαιρέσε** το πρόβλημα μεγέθους  $n$  σε υπο-προβλήματα μεγέθους  $n/b$
- **Λύσε**  $a$  τέτοια υπο-προβλήματα
- **Συνδύασε** τις λύσεις τους



# Αλγόριθμος Ταξινόμησης Mergesort

- **Είσοδος:** ένας πίνακας/λίστα με στοιχεία **όχι απαραίτητα ταξινομημένα**
- **Έξοδος:** Ο πίνακας/λίστα ταξινομημένος
- **Θα χρησιμοποιήσουμε τη διαδικασία merge**
- Η merge δέχεται σαν είσοδο δύο **ταξινομημένους** πίνακες
- Θα χωρίσουμε τον πίνακα επαναλαμβανόμενα μέχρι να έχουμε **ταξινομημένους** πίνακες. Πότε ξέρουμε ότι είναι ταξινομημένοι;
  - **Όταν έχουν 1 στοιχείο!**
- Μετά τους συγχωνεύουμε με τη merge!



$$\text{merge}(\boxed{3}, \boxed{5}) = \boxed{3 \ 5}$$

$$\text{merge}(\boxed{8}, \boxed{1}) = \boxed{1 \ 8}$$

$$\text{merge}(\boxed{3 \ 5}, \boxed{1 \ 8}) = \boxed{1 \ 3 \ 5 \ 8}$$

# Ψευδοκώδικας Mergesort

Διαδικασία **mergesort**(A[1,...,n])

Είσοδος: Ένας πίνακας αριθμών A[1,...,n]

Έξοδος: Ο πίνακας ταξινομημένος

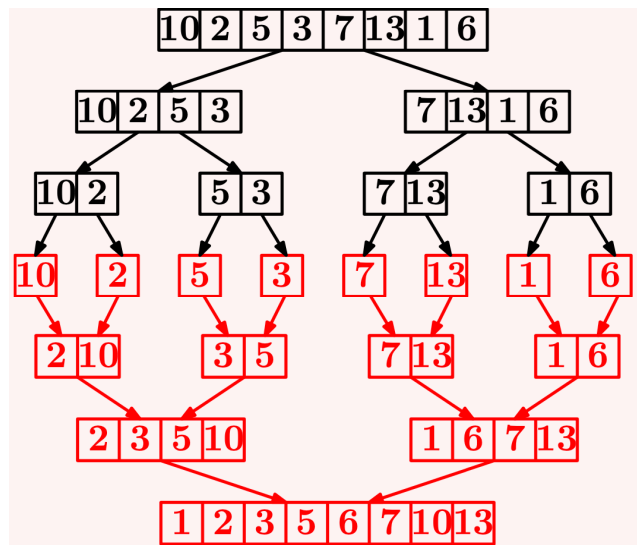
if  $n > 1$ :

return **merge**(**mergesort**(A[1,...,[n/2]]), **mergesort**(A[[n/2]+1,...,n]))

else:

return A

Παράδειγμα: mergesort (10,2,5,3,7,13,1,6)

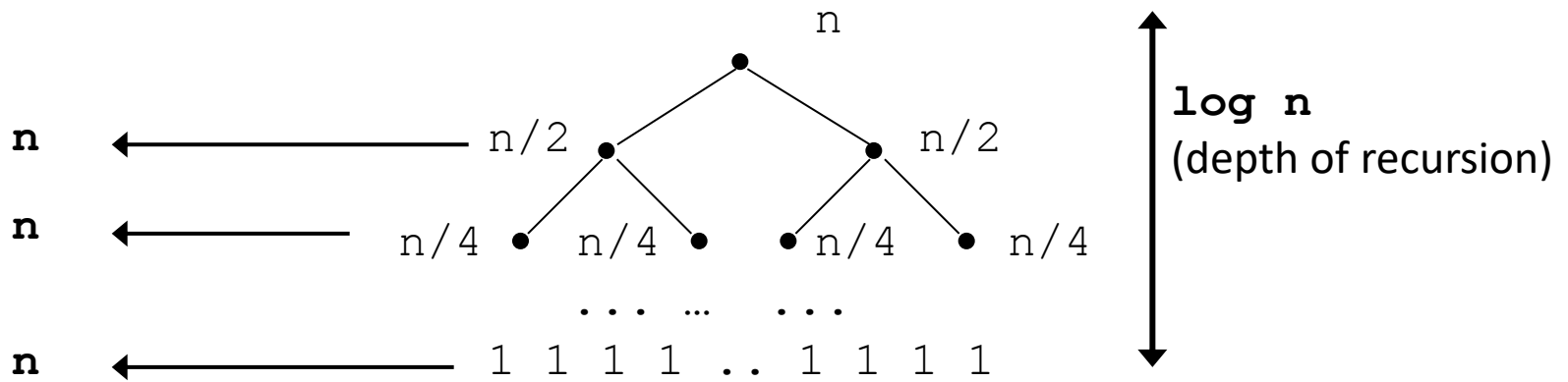


Divide

Conquer

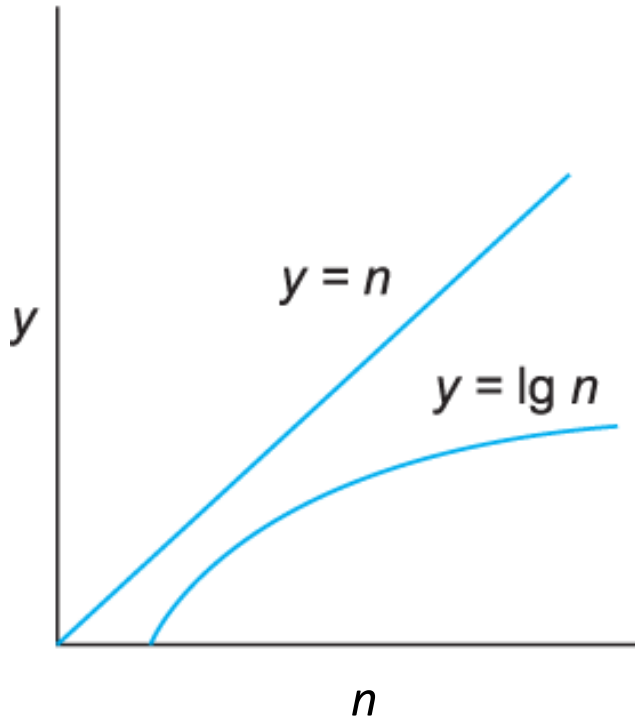
Combine(merge)

# Αποδοτικότητα Mergesort

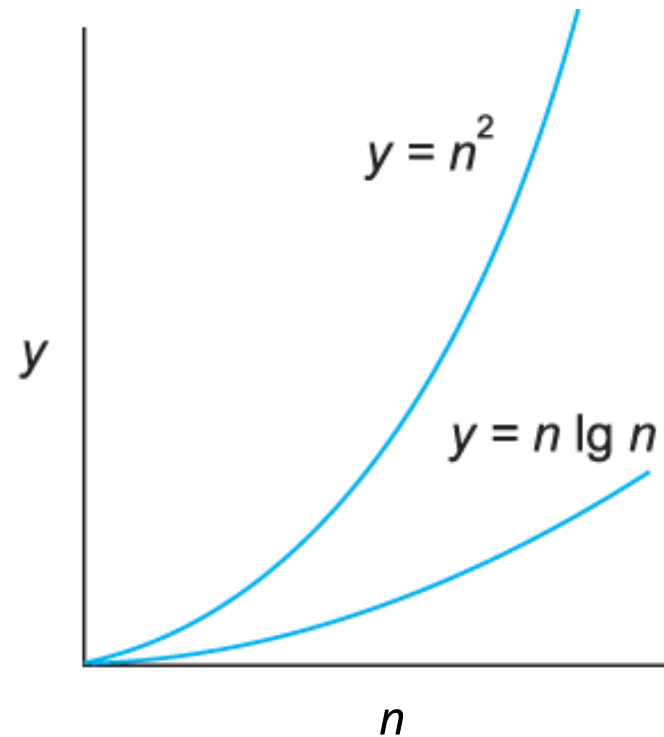


- Πόσες συγκρίσεις χρειάζονται στη **χειρότερη περίπτωση**;
  - Σε κάθε επίπεδο γίνονται μεταξύ  $n/2$  και  $n$  συγκρίσεις.
  - Υπάρχουν  $\log n$  επίπεδα
  - Άρα συνολικά έχουμε μεταξύ  $n/2 \log n$  και  $n \log n$  συγκρίσεις.
- **Δεν υπάρχει** «ασυμπτωτικά» πιο αποδοτικός αλγόριθμος ταξινόμησης από τον mergesort στη χειρότερη περίπτωση.

# Γραφήματα των μαθηματικών παραστάσεων $n$ , $\log n$ , $n \log n$ και $n^2$



α.  $n$  και  $\lg n$



β.  $n^2$  και  $n \lg n$

$\log n$ : υπο-γραμμική συνάρτηση (sub-linear)

$n^2$ : υπερ-γραμμική συνάρτηση (super-linear)

# Χρόνος εκτέλεσης αλγορίθμων

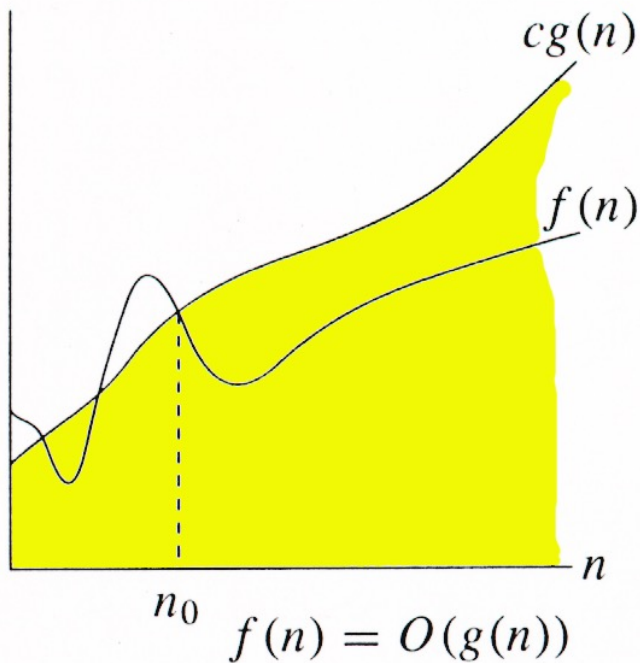
μέγεθος εισόδου  $n$

$n$	$\log_2 n$	$n$	$n^2$	$2^n$ πολυπλοκότητα αλγορίθμου
10	$3 \times 10^{-9}$	$10^{-8}$	$10^{-7}$	$10^{-6}$ sec
100	$7 \times 10^{-9}$	$10^{-7}$	0.00001	<u><math>4 \times 10^{11}</math></u> αιώνες
1000	$10^{-8}$	$10^{-6}$	0.001	*
10000	$1.3 \times 10^{-8}$	$10^{-5}$	0.1	*
100000	$1.7 \times 10^{-8}$	0.0001	10	*
1000000	$2 \times 10^{-8}$	0.001	0.27 ώρες	*

1 βήμα = 1 nsec

# Ασυμπτωτικός συμβολισμός O

- ΑΝΩ ΟΡΙΟ στην αύξηση μιας συνάρτησης.
- Έστω  $f(n)$  και  $g(n)$  συναρτήσεις, ώστε  $f, g : \mathbb{N}^+ \rightarrow \mathbb{R}^+$
- $f(n) = O(g(n))$  : Η  $f(n)$  δεν αυξάνεται πιο γρήγορα από την  $g(n)$
- Ένα (σταθερό) πολλαπλάσιο της  $g(n)$  είναι άνω όριο στην  $f(n)$  για αρκετά μεγάλα  $n$

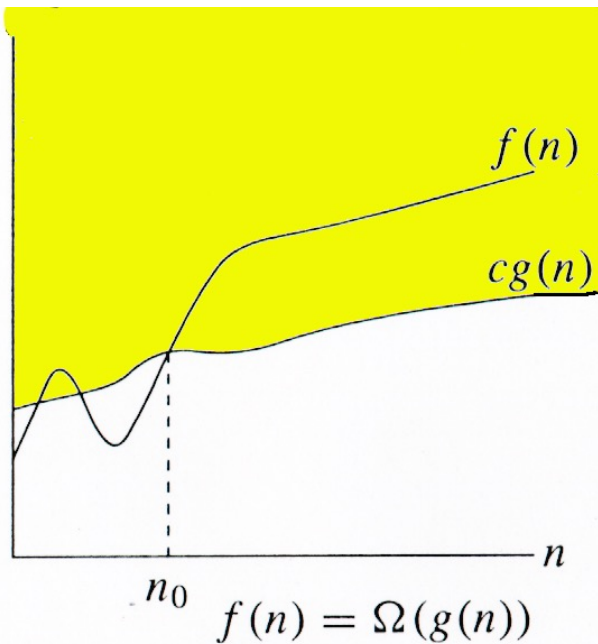


Πχ.  $2n+1 = O(n)$

$$\exists c > 0, n_0 > 0, \text{ τέτοια ώστε } f(n) \leq c \cdot g(n), \forall n > n_0$$

# Ασυμπτωτικός συμβολισμός $\Omega$

- ΚΑΤΩ ΟΡΙΟ στην αύξηση της συνάρτησης
- Έστω  $f(n)$  και  $g(n)$  συναρτήσεις, ώστε  $f, g : \mathbb{N}^+ \rightarrow \mathbb{R}^+$
- $f(n) = \Omega(g(n))$  : Η  $f(n)$  δεν αυξάνεται πιο αργά από την  $g(n)$
- Ένα (σταθερό) πολλαπλάσιο της  $g(n)$  είναι κάτω όριο στην  $f(n)$   
για αρκετά μεγάλα  $n$



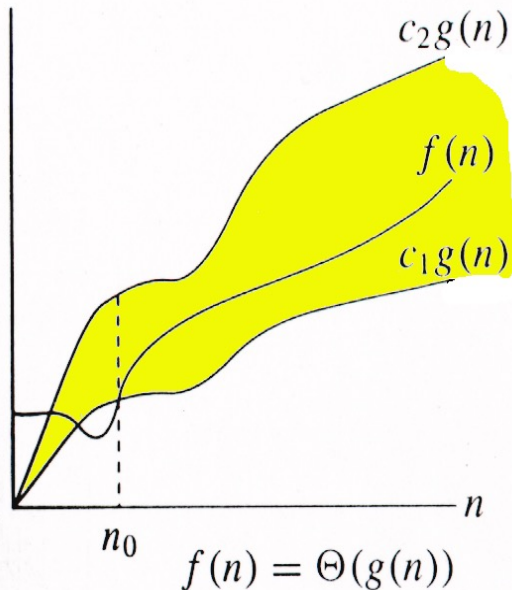
Πχ.  $n/2 - 1 = \Omega(n)$

$\exists c > 0, n_0 > 0$ , τέτοια ώστε  $f(n) \geq c \cdot g(n), \forall n > n_0$



# Ασυμπτωτικός συμβολισμός $\Theta$

- ΑΝΩ ΚΑΙ ΚΑΤΩ ΟΡΙΟ στην αύξηση της συνάρτησης
- $f(n) = \Theta(g(n))$  : Η  $f(n)$  δεν αυξάνεται ούτε πιο γρήγορα ούτε πιο αργά από την  $g(n)$
- (σταθερά) πολλαπλάσια της  $g(n)$  είναι άνω και κάτω όρια στην  $f(n)$  για αρκετά μεγάλα  $n$



$$f(n) = \Theta(g(n))$$

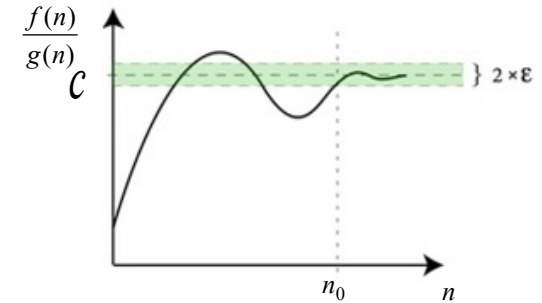
$\Leftrightarrow$

$$f(n) = O(g(n)) \text{ και } f(n) = \Omega(g(n))$$

$$\exists c_1, c_2 > 0, n_0 > 0, \text{ τέτοια ώστε } c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n), \forall n > n_0$$

# Ασυμπτωτικοί συμβολισμοί με χρήση ορίου

1) If  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$ , then  $f(n) = \Theta(g(n))$



2) If  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ , then  $f(n) = O(g(n))$  and  $f(n) \neq \Theta(g(n))$

3) If  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ , then  $g(n) = O(f(n))$  and  $g(n) \neq \Theta(f(n))$

# Ασυμπτωτικοί συμβολισμοί

## Τι τα θέλουμε όλα αυτά?

- Για να συγκρίνουμε αλγορίθμους
- Για να ξέρουμε αν ο αλγόριθμός μας αξίζει να υλοποιηθεί και να δοκιμαστεί
- Να ξέρουμε αν είναι γρήγορος (πολυωνυμικός =  $O(n^k)$ )

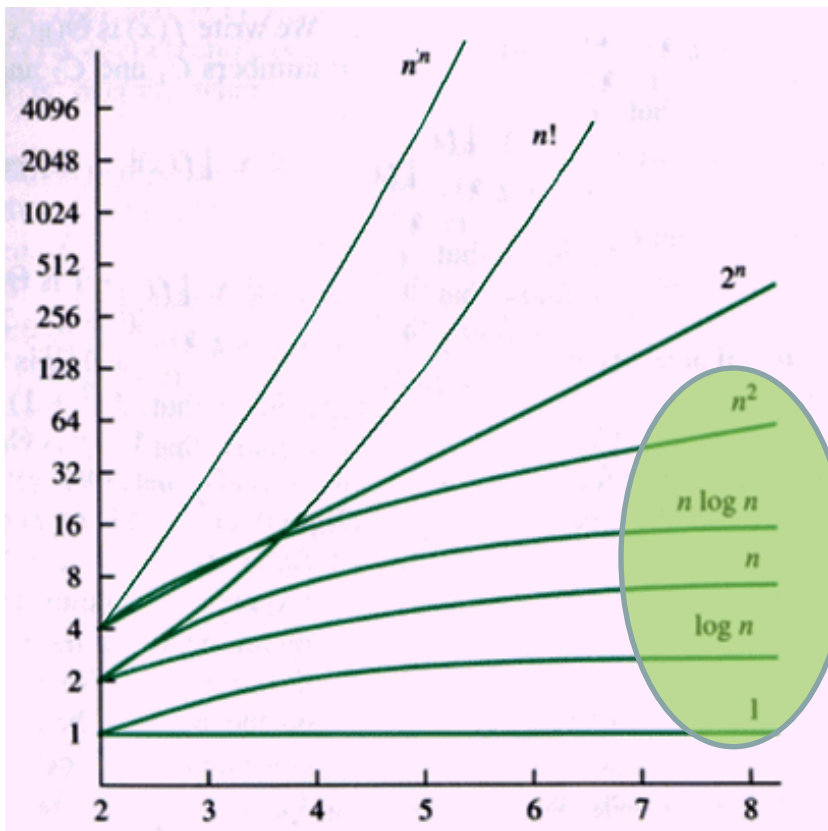
## Ο υπολογισμός είναι απλός τελικά:

- Μετράμε τα βήματα του αλγορίθμου
- Πετάμε τις σταθερές
- Από τα πολυώνυμα κρατάμε μόνο τον όρο με το μεγαλύτερο βαθμό

Με λίγη εξάσκηση και κατανόηση, δεν χρειάζονται πράξεις, γίνεται αυτόματα...

# Πολυωνυμική πολυπλοκότητα

- Αλγόριθμοι με **πολυωνυμική** πολυπλοκότητα  $O(n^k)$ :  
Γρήγοροι/αποτελεσματικοί/αποδοτικοί αλγόριθμοι



## Περί πολυπλοκότητας...

Time complexity function	Size $n$					
	10	20	30	40	50	60
$n$	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
$n^2$	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
$n^3$	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
$n^5$	.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
$2^n$	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
$3^n$	.059 second	58 minutes	6.5 years	3855 centuries	$2 \times 10^8$ centuries	$1.3 \times 10^{13}$ centuries

## Διάταξη πολυπλοκότητας

- $O(1)$  – **constant**
- $O(\log n)$  – **logarithmic**
- $O(\log^k n)$  – **poly-logarithmic**
- $O(n^{1/2})$
- $O(n)$  – **linear**
- $O(n \log n)$
- $O(n^2)$  – **quadratic**
- $O(n^3)$  - **cubic**
- $O(n^k)$  – **polynomial**
- $O(2^n)$  – **exponential**
- $O(a^n)$
- $O(n!)$
- $O(n^n)$

# Παραδείγματα πολυπλοκότητας αλγορίθμων

1. Εσωτερικό γινόμενο δυο διανυσμάτων  $a$  και  $b$ , **διάστασης  $n$**  το καθένα
  - $O(n)$ . Κάθε όρος του  $a$  πολλαπλασιάζεται με έναν μόνο όρο του  $b$ .
2. Δίνεται σύνολο  $S$   $n$  θετικών ακεραίων και ένας ακέραιος  $M$ . **Υπάρχει υποσύνολο  $A$  του  $S$ , με  $k$  στοιχεία και άθροισμα στοιχείων =  $M$ ;**
  - Βρίσκουμε **όλα τα υποσύνολα  $k$  στοιχείων** του  $S$  και για καθένα, προσθέτουμε τα στοιχεία του και **ελέγχουμε** αν το άθροισμα ισούται με  $M$
  - **Το πλήθος των υποσυνόλων** που πρέπει να ελέγξουμε είναι όλοι οι συνδυασμοί των  $n$  ανά  $k$  που είναι  $n!/(n-k)!k! \leq n^k / k! = O(n^k)$ .
3. Δίνεται σύνολο  $S$ ,  $n$  θετικών ακεραίων και ένας ακέραιος  $M$ . **Υπάρχει υποσύνολο  $A$  του  $S$  με άθροισμα στοιχείων =  $M$ ;**
  - Βρίσκουμε **όλα τα υποσύνολα** του  $S$  και για κάθε ένα προσθέτουμε τα στοιχεία του και **ελέγχουμε** αν το άθροισμα ισούται με  $M$ .
  - Πρέπει να ελέγξουμε **όλα τα δυνατά υποσύνολα του  $S$** , που είναι  $2^n$  και για κάθε ένα να προσθέσουμε τα στοιχεία του, άρα πολυπλοκότητα  $\Omega(2^n)$ .

# Θεωρία Παιγνίων

- Είναι η επιστήμη που μελετά **ανταγωνιστικά** περιβάλλοντα, π.χ. διαμοιρασμός πόρων, όπου οι οντότητες που συμμετέχουν μπορούν να δρουν **στρατηγικά**.
- Η συμπεριφορά ενός συστήματος εξαρτάται από την **αλληλεπίδραση** αυτών των οντοτήτων.
- **Γιατί ενδιαφέρει την πληροφορική;**
  - **Αλγοριθμική θεωρία παιγνίων**: η τομή της θεωρίας παιγνίων με την πληροφορική, π.χ. πολυπλοκότητα εύρεσης της βέλτιστης στρατηγικής.
  - Τέτοιου είδους συστήματα αποτελούν μέρος της **τεχνολογικής υποδομής** και εμφανίζονται και στην πληροφορική, π.χ. διαμοιρασμός πόρων στο Internet.



# Παράδειγμα1: Prisoner's Dilemma



- Δύο ύποπτοι ανακρίνονται σε ξεχωριστά δωμάτια για ένα έγκλημα που έχουν διαπράξει
- Αν δεν ομολογήσουν (**defect: D**), η αστυνομία έχει κάποια στοιχεία για να τους καταδικάσει για ένα πλημμέλημα, δηλαδή **1 χρόνο φυλακή ο καθένας**
- Αν ομολογήσουν και οι 2 (**confess: C**), πάνε φυλακή για **3 χρόνια ο καθένας**
- Αν **ομολογήσει μόνο ο ένας** από τους 2, τότε αυτός αφήνεται **ελεύθερος**, και ο άλλος πάει φυλακή **5 χρόνια**
- Οι 2 ύποπτοι **δεν μπορούν να επικοινωνήσουν** κατά τη διάρκεια της ανάκρισης και ενδιαφέρονται **μόνο για τη δική τους ποινή**

# Παράδειγμα1: Prisoner's Dilemma



NYCHA-Spotlight.com



NYCHA-Spotlight.com

		Prisoner 2	
		D	C
Prisoner 1	D	1, 1	0, 5
	C	5, 0	3, 3

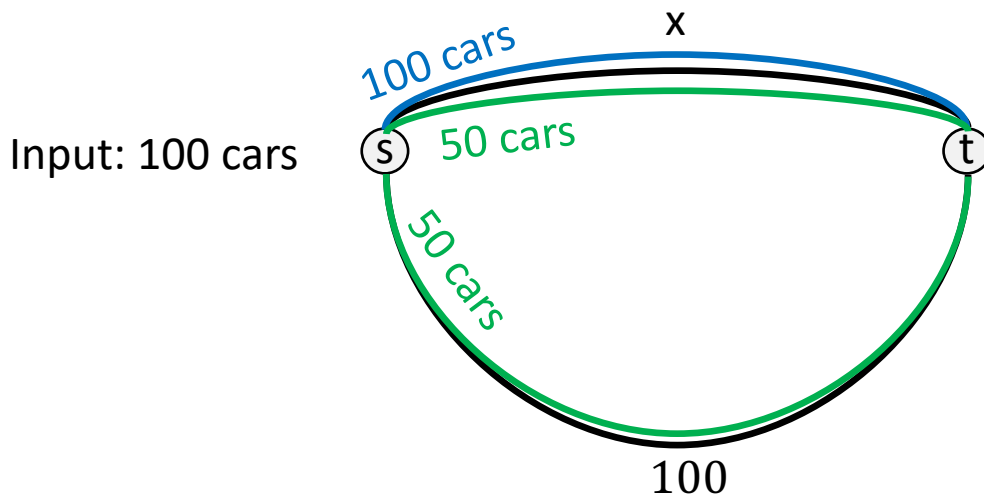
Οι 2 ύποπτοι **δεν μπορούν να επικοινωνήσουν** κατά τη διάρκεια της ανάκρισης και ενδιαφέρονται **μόνο για τη δική τους ποινή**.

Ο ύποπτος 1 προτιμάει να ομολογήσει (C) σε κάθε περίπτωση, δηλαδή **ό,τι** και να επιλέξει ο άλλος. Το ίδιο και ο ύποπτος 2.

Το στρατηγικό προφίλ (C,C) λέγεται **σημείο ισορροπίας κατά Nash (Nash equilibrium)**.

# Παράδειγμα2: Congestion Games

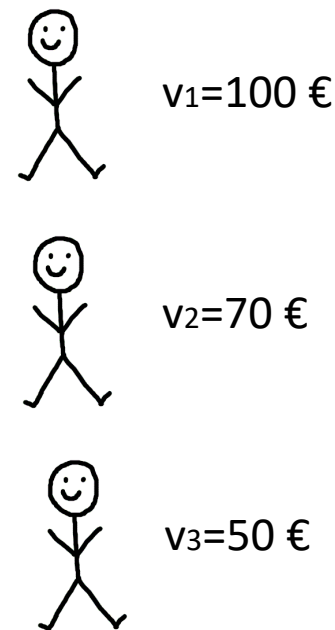
- 100 αυτοκίνητα θέλουν να πάνε από το s στο t.
- Υπάρχουν δύο επιλογές:
  - Ένας σύντομος αλλά στενός δρόμος, όπου ο χρόνος για να τον διανύσει κάποιος εξαρτάται από την κίνηση: αν υπάρχουν  $x$  αυτοκίνητα χρειάζεται χρόνο  $x$  λεπτά.
  - Ένας φαρδύς περιφερειακός αυτοκινητόδρομος που δεν πιάνει κίνηση, αλλά είναι πολύ πιο μακρύτες: για να τον διανύσει κάποιος χρειάζεται χρόνο 100 λεπτά.
- Ποιον δρόμο θα διαλέγατε για να πάτε **πιο γρήγορα**;



Αθροιστική καθυστέρηση:  
 $100 * 100 = 10000$   
(Nash equilibrium)

Αθροιστική καθυστέρηση:  
 $50 * 50 + 50 * 100 = 7500$   
(βέλτιστη λύση)

# Παράδειγμα3: Δημοπρασίες (Auctions)



- $v_i$  : αξία (value) του agent  $i$  για το αντικείμενο
- Στόχος: Να δώσουμε το αντικείμενο στον agent με το **μεγαλύτερο value**.
- Πρόβλημα: **Δεν γνωρίζουμε τα values!**
- Μπορούμε να **ρωτήσουμε τα values**

# Παράδειγμα3: Δημοπρασίες (Auctions)



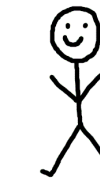
Το value μου  
είναι **110 €**



$v_1=100 \text{ €}$



$v_2=70 \text{ €}$



$v_3=50 \text{ €}$

- $v_i$  : αξία (value) του agent  $i$  για το αντικείμενο
- Στόχος: Να δώσουμε το αντικείμενο στον agent με το **μεγαλύτερο value**.
- Πρόβλημα: **Δεν γνωρίζουμε τα values!**
- Μπορούμε να **ρωτήσουμε τα values**
  - Αλλά μπορεί να μας πουν **ψέματα**. Ποιος;
  - Μπορούμε να ζητήσουμε να **πληρώσουν**. Πόσο;
  - Χρησιμότητα (utility):  $u_i = v_i - \text{payment}_i$

# Παράδειγμα3: First Price Auction



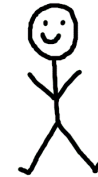
Το value μου είναι 71 €

Άλλαξα γνώμη, 56+ε €

Το value μου είναι 56 €

Άλλαξα γνώμη, 56+2ε €

Το value μου είναι 40 €



$v_1=100$  €



$v_2=70$  €



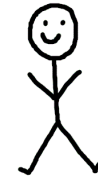
$v_3=50$  €

- Χρησιμότητα (**utility**):  $u_i = v_i - \text{payment}_i$
- Να πληρώσει (αυτός που δήλωσε το μεγαλύτερο) το **ποσό που δήλωσε**
- Αν καταλήξουν σε σημείο ισορροπίας, όντως θα δώσουμε το αντικείμενο σε αυτόν με το **μεγαλύτερο value**.
- **Μπορούμε όμως να τους «αναγκάσουμε» να πουν την αλήθεια;;;**

# Παράδειγμα3: Second Price Auction



To value μου είναι 100 €



$v_1=100$  €

To value μου είναι 70 €



$v_2=70$  €

To value μου είναι 50 €



$v_3=50$  €

- Χρησιμότητα (**utility**):  $u_i = v_i - \text{payment}_i$
- Να πληρώσει (αυτός που δήλωσε το μεγαλύτερο) τη **2<sup>η</sup> μεγαλύτερη δήλωση!**
- **Δεν μπορούν να αυξήσουν το utility τους λέγοντας ψέματα!**
- Αυτός ο μηχανισμός είναι **truthful!**
- **Vickrey auction** (William Vickrey - Nobel Οικονομικών 1996).
  - Generalization to many items: **VCG** (Vickrey-Clarke-Groves)

# Για όποιον/α ενδιαφέρεται να το ψάξει περισσότερο

- Ποιους άλλους αλγόριθμους ταξινόμησης γνωρίζουμε;
- Να βρείτε κι άλλα προβλήματα που χρησιμοποιούν την τεχνική **διαίρει και βασίλευε**. Ποια είναι η πολυπλοκότητά τους.
- Τι είναι οι **άπληστοι** (greedy) αλγόριθμοι;
- Τι είναι ο **δυναμικός προγραμματισμός**; Υπάρχει πιο αποδοτικός αλγόριθμος για το πρόβλημα 3 της διαφάνειας 55;
- Τι είναι ο **γραμμικός προγραμματισμός**;
- Τι σημαίνει **P vs NP**;
- Αναζητήστε **εφαρμογές της θεωρίας παιγνίων**



# Τέλος Κεφαλαίου 5

- Μιλήσαμε για αλγορίθμους: έννοιες, βασικοί αλγόριθμοι και πολυπλοκότητα
- Στο επόμενο κεφάλαιο θα ασχοληθούμε με γλώσσες προγραμματισμού και προγραμματιστικά μοντέλα.