ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

# Εισαγωγή στους Αλγορίθμους

## Ενότητα 3η


Διδάσκων
Χρήστος Ζαρολιάγκης
Καθηγητής
Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
Email: zaro@ceid.upatras.gr





Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

# Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.

- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

# Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

# Σκοποί ενότητας

- Εισαγωγή στις δομές δεδομένων λίστας και πίνακα
- Εισαγωγή στο δυαδικό σωρό

# Περιεχόμενα ενότητας

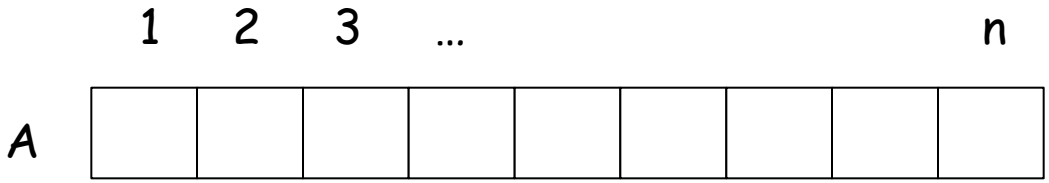
- Βασικές λειτουργίες σε λίστες, πίνακες και δυαδικό σωρό

- Παραδείγματα βασικών αλγορίθμων για αναζήτηση, απαρίθμηση, ταξινόμηση

- Μέθοδοι ανάλυσης της χρονικής και χωρικής πολυπλοκότητας των αλογρίθμων

# Δύο Απλές & Σημαντικές Δομές Δεδομένων

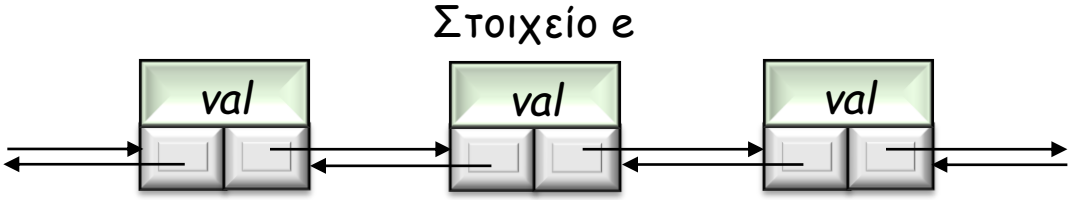
# Πίνακας (array)



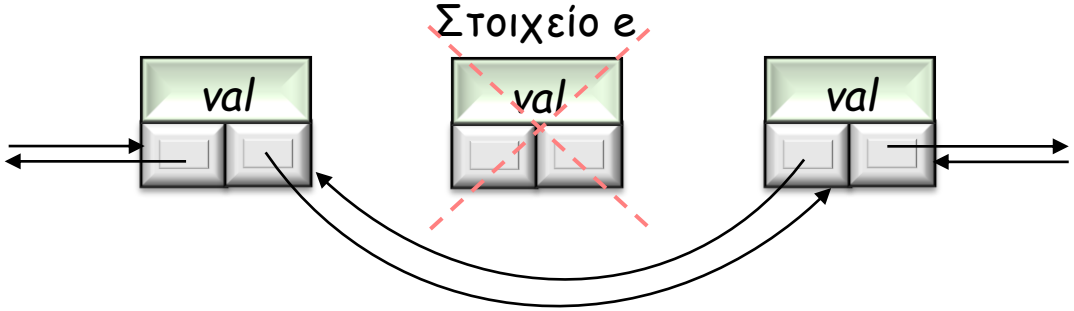

- Πίνακας: σταθερό μέγεθος.
- A[i]: προσπέλαση i-στού στοιχείου
- Γρηγορότερη προσπέλαση στοιχείων από οποιαδήποτε άλλη δομή.
- A[n+1]: σφάλμα! Βγήκαμε εκτός ορίου του πίνακα.

# Λίστες

- Πριν από τη διαγραφή του e :




- Μετά τη διαγραφή του e :




- Διπλά συνδεδεμένη λίστα: για κάθε στοιχείο e, διατηρείται ένας δείκτης προς το επόμενο στοιχείο
- Μεταβλητό μέγεθος
- Εισαγωγή/διαγραφή ενός στοιχείου e σε χρόνο $O(1)$

# Λίστες (C)

```
struct list_el {
   int val;
   struct list_el * next;
};

typedef struct list_el item;

void main() {
   item * curr, * head;
   int i;
   head = NULL;

   for(i=1;i<=10;i++) {
      curr = (item *)malloc(sizeof(item));
      curr->val = i;
      curr->next  = head;
      head = curr;
   }
}
```

# Απλοί Αλγόριθμοι και Συνηθισμένοι Χρόνοι Εκτέλεσης

# Γραμμικός χρόνος: $O(n)$

Γραμμικός χρόνος. Ο χρόνος εκτέλεσης είναι το πολύ ένας σταθερός παράγοντας επί το μέγεθος της εισόδου.

Υπολογισμός μέγιστου. Υπολογίστε το μέγιστο από n αριθμούς $a_1, \ldots, a_n$.

```
max ← a_1
for i = 2 to n {
    if (a_i > max)
        θέσε max ← a_i
}
```

# Γραμμικός χρόνος: O(n)

| A | 7 | 15 | 8 | 20 | 3 | 9 | 45 | 2 | 6 |
|---|---|---|---|---|---|---|---|---|---|

**max ← 7**

**1η Επανάληψη (i=2)**
**if ( 15 > 7 ) true**
**Θέσε max ← 15**

**max ← 15**

**2η Επανάληψη (i=3)**
**if ( 8 > 15 ) false**
**Θέσε max ← $a_i$**

**max ← 15**

**3η Επανάληψη (i=4)**
**if ( 20 > 15 ) true**
**Θέσε max ← 20**

**max ← 20**

**4η Επανάληψη (i=5)**
**if ( 3 > 20 ) false**
**Θέσε max ← $a_i$**

**max ← 20**

**5η Επανάληψη (i=6)**
**if ( 9 > 20 ) false**
**Θέσε max ← $a_i$**

**max ← 20**

**6η Επανάληψη (i=7)**
**if ( 45 > 20 ) true**
**Θέσε max ← 45**

**max ← 45**

**7η Επανάληψη (i=8)**
**if ( 2 > 45 ) false**
**Θέσε max ← $a_i$**

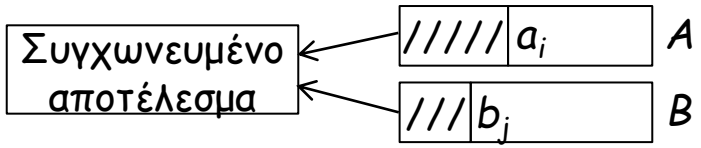
**max ← 45**

**8η Επανάληψη (i=9)**
**if ( 6 > 45 ) false**
**Θέσε max ← $a_i$**

# Γραμμικός χρόνος: O(n)

Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.

Προσάρτηση του μικρότερου από τα $a_i$ και $b_j$ στην έξοδο.



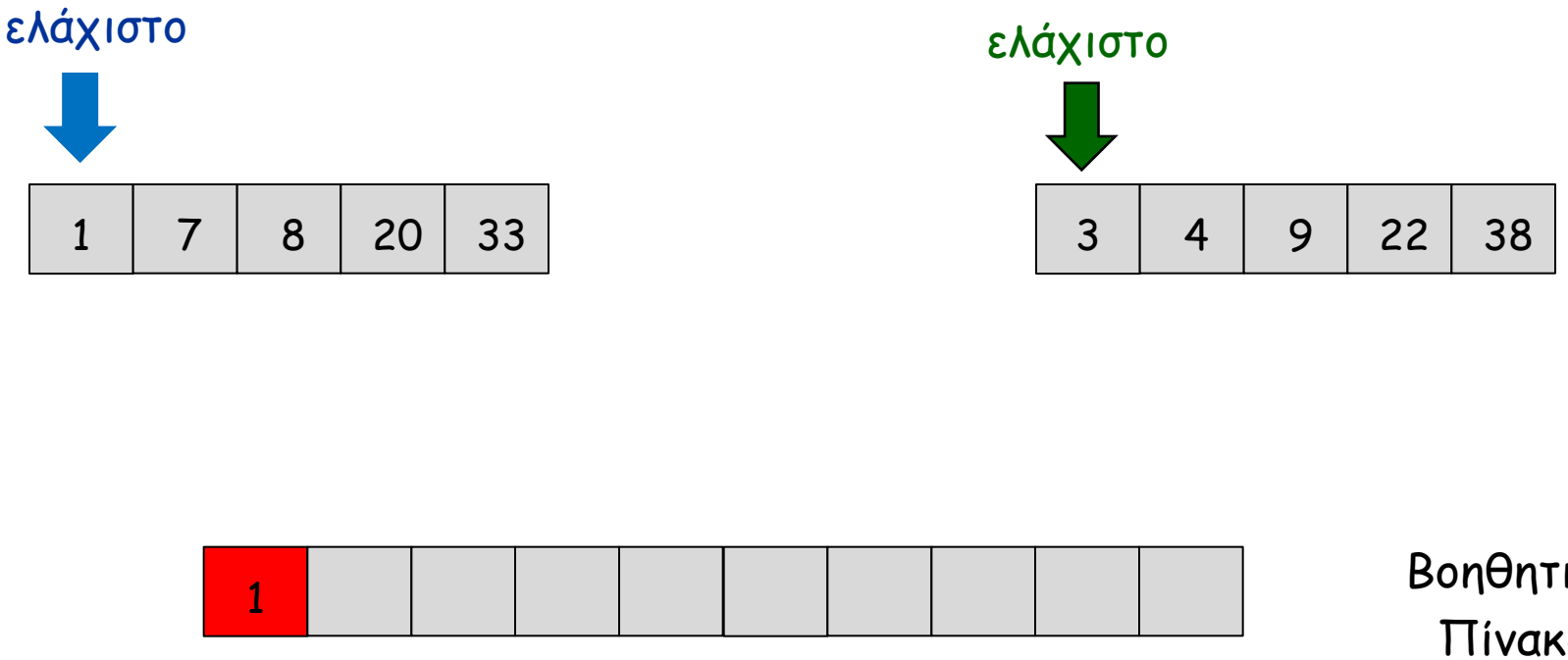

```
i = 1, j = 1
while (και οι δύο λίστες δεν είναι κενές) {
   if (a_i ≤ b_j) Προσάρτηση του a_i στη λίστα εξόδου και i=i+1
   else           Προσάρτηση του b_j στη λίστα εξόδου και j=j+1
}
προσάρτηση των υπόλοιπων στοιχείων της μη άδειας λίστας στην έξοδο
```

Ισχυρισμός. Η συγχώνευση δύο λιστών μεγέθους n παίρνει χρόνο O(n).

Απόδειξη. Μετά από κάθε σύγκριση, το μέγεθος της λίστας εξόδου αυξάνεται κατά 1.

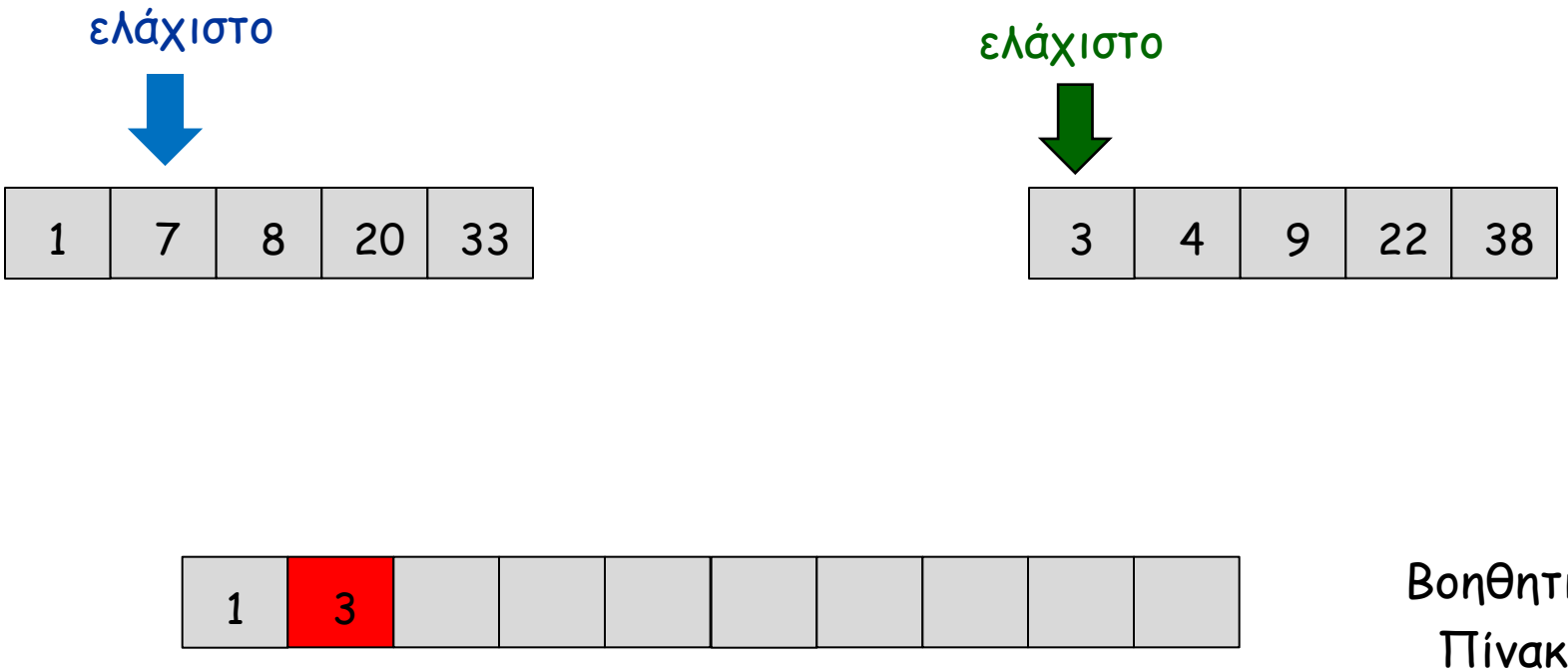
# Γραμμικός χρόνος: O(n)

Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.



# Γραμμικός χρόνος: O(n)

Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.



# Γραμμικός χρόνος: O(n)

Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.



Βοηθητικός Πίνακας

# Γραμμικός χρόνος: O(n)

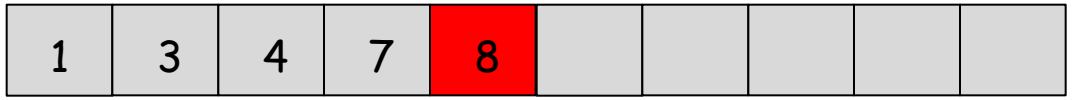
Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.



Βοηθητικός Πίνακας

# Γραμμικός χρόνος: O(n)

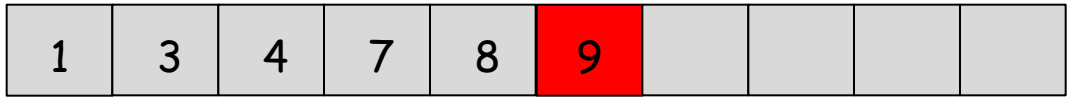
Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.


Βοηθητικός Πίνακας

# Γραμμικός χρόνος: $O(n)$

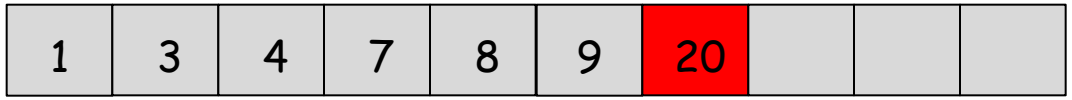
Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.


Βοηθητικός Πίνακας

# Γραμμικός χρόνος: O(n)

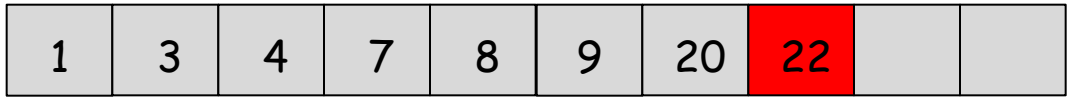
Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.


Βοηθητικός Πίνακας

# Γραμμικός χρόνος: O(n)

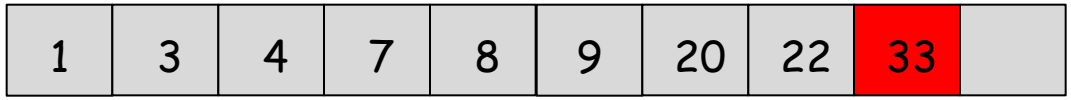
Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.


Βοηθητικός Πίνακας

# Γραμμικός χρόνος: O(n)

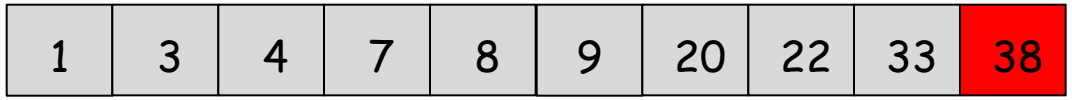
Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.


Βοηθητικός Πίνακας

# Γραμμικός χρόνος: O(n)

Συγχώνευση. Συγχωνεύστε δύο ταξινομημένες λίστες $A = a_1, a_2, \ldots, a_n$ και $B = b_1, b_2, \ldots, b_n$ σε μια μόνο ταξινομημένη λίστα.


Βοηθητικός Πίνακας

# Υπογραμμικός χρόνος

*Αναζήτηση σε ταξινομημένο πίνακα.* Δεδομένου ενός ταξινομημένου πίνακα *A*[1:n] με n στοιχεία, προσδιορίστε αν ένα συγκεκριμένο στοιχείο p ανήκει στον *A*.

n = 11 , p = 82

| 1 | 3 | 8 | 12 | 24 | 45 | 78 | 82 | 93 | 97 | 99 |
|---|---|---|---|---|---|---|---|---|---|---|

```
Bin-Search(A, 82, 1, 11)
imid = (1 + 11)/2 = 6
if A[6] > 82  false
   Bin-Search(A, p, imin, imid-1)
 else if A[6] < 82
         Bin-Search(A, 82, 7, 11)
 else return imid
```

# Υπογραμμικός χρόνος

*Αναζήτηση σε ταξινομημένο πίνακα.* Δεδομένου ενός ταξινομημένου πίνακα $A[1:n]$ με n στοιχεία, προσδιορίστε αν ένα συγκεκριμένο στοιχείο p ανήκει στον $A$.

n = 11 , p = 82


```
Bin-Search(A, 82, 7, 11)
imid = (7 + 11)/2 = 9
if A[9] > 82   true
   Bin-Search(A, 82, 7, 8)
 else if A[9] < 82
         Bin-Search(A, 82, 9, 11)
 else return imid
```

# Υπογραμμικός χρόνος

*Αναζήτηση σε ταξινομημένο πίνακα.* Δεδομένου ενός ταξινομημένου πίνακα *A*[1:n] με n στοιχεία, προσδιορίστε αν ένα συγκεκριμένο στοιχείο p ανήκει στον *A*.

n = 11 , p = 82


```
Bin-Search(A, 82, 7, 8)
imid = (7 + 8)/2 = 8 (στρογγυλοποίηση)
if A[8] > 82  false
   Bin-Search(A, p, imin, imid-1)
 else if A[8] < 82
        Bin-Search(A, p, imid+1, imax)
 else return imid
```

# Υπογραμμικός χρόνος

*Αναζήτηση σε ταξινομημένο πίνακα.* Δεδομένου ενός ταξινομημένου πίνακα $A[1:n]$ με n στοιχεία, προσδιορίστε αν ένα συγκεκριμένο στοιχείο p ανήκει στον $A$.

*Λύση O(log n).* Δυαδική αναζήτηση. Κλήση **`Bin-Search (A,p,1,n)`**.

```
Bin-Search(A, p, imin, imax)
imid = (imin + imax)/2
if A[imid] > p
   Bin-Search(A, p, imin, imid-1)
 else if A[imid] < p
         Bin-Search(A, p, imid+1, imax)
 else return imid
```

Μέγεθος «ενεργής» περιοχής αναζήτησης μειώνεται κατά $\frac{1}{2}$ σε κάθε επανάληψη.

Μετά από k επαναλήψεις, μέγεθος «ενεργής» περιοχής $\leq (1/2)^k n$

Τερματίζουμε όταν $(1/2)^k n = 1$, δηλ. όταν $k = \log n$.

Άρα, συνολικός χρόνος $O(\log n)$

# Χρόνος O(n log n)

Χρόνος O(n log n). Συναντάται σε αλγόριθμους «διαίρει-και-βασίλευε».

Ταξινόμηση. Ο mergesort και ο heapsort είναι αλγόριθμοι ταξινόμησης που εκτελούν O(n log n) συγκρίσεις.

Μέγιστο άδειο διάστημα. Δεδομένων n χρονοσφραγίδων $x_1, \ldots, x_n$ στις οποίες φθάνουν αντίγραφα ενός αρχείου σε ένα διακομιστή, ποιό είναι το μέγιστο χρονικό διάστημα στην διάρκεια του οποίου δεν έφτασε κανένα αντίγραφο αρχείου στον διακομιστή;

Λύση O(n log n). Ταξινόμησε τις χρονοσφραγίδες. Σάρωσε την ταξινομημένη λίστα στη σειρά, προσδιορίζοντας το μέγιστο κενό μεταξύ διαδοχικών χρονοσφραγίδων.

# Τετραγωνικός χρόνος: $O(n^2)$

Τετραγωνικός χρόνος. Απαριθμήστε όλα τα ζευγάρια στοιχείων.

Κοντινότερο ζευγάρι σημείων. Δεδομένης μιας λίστας n σημείων σε ένα επίπεδο $(x_1, y_1), \ldots, (x_n, y_n)$, βρείτε το ζευγάρι με την μικρότερη απόσταση.

Λύση $O(n^2)$. Δοκιμάστε κάθε ζευγάρι σημείων.

```
min ← (x1 - x2)^2 + (y1 - y2)^2
for i = 1 to n {
   for j = i+1 to n {
      d ← (xi - xj)^2 + (yi - yj)^2
      if (d < min)
         min ← d
   }
}
```

← Δεν χρειάζεται να πάρουμε την ρίζα

# Κυβικός χρόνος: $O(n^3)$

Κυβικός χρόνο. Απαριθμήστε όλες τις τριάδες στοιχείων.

Συμπληρωματικότητα συνόλων. Δεδομένων n συνόλων $S_1$, …, $S_n$ καθένα από τα οποία είναι υποσύνολο του 1, 2, …, n, υπάρχουν κάποια ζευγάρια συνόλων που να είναι ξένα;

Λύση $O(n^3)$. Για κάθε ζευγάρι συνόλων, ελέγξτε αν είναι ξένα.

```
foreach σύνολο S_i {
   foreach άλλο σύνολο S_j {
      foreach στοιχείο p του S_i {
         Βρές αν το p ανήκει και στο S_j
      }
      if (κανένα στοιχείο του S_i δεν ανήκει στο S_j)
         Ανέφερε ότι τα S_i και S_j είναι ξένα
   }
}
```

Υπόθεση: κατάλληλη δομή υποσυνόλου, έτσι ώστε ο έλεγχος αν ένα στοιχείο ανήκει σε κάποιο υποσύνολο γίνεται σε $O(1)$ χρόνο.

# Πολυωνυμικός χρόνος: Χρόνος $O(n^k)$

Ανεξάρτητο σύνολο μεγέθους k. Δεδομένου ενός γραφήματος, υπάρχουν k κορυφές τέτοιες ώστε ανά δύο να μην συνδέονται με μια πλευρά;

Το k είναι σταθερά

Λύση $O(n^k)$. Απαριθμήστε όλα τα υποσύνολα k κορυφών.

```
foreach υποσύνολο S με k κόμβους {
   έλεγξε αν το S αποτελεί ανεξάρτητο σύνολο
   if (S είναι ένα ανεξάρτητο σύνολο){
      δήλωσε ότι το S είναι ένα ανεξάρτητο σύνολο
   }
}
```

- Έλεγχος αν το S είναι ανεξάρτητο σύνολο = $O(k^2)$.
- Αριθμός υποσυνόλων k στοιχείων = $\binom{n}{k} = \frac{n\,(n-1)\,(n-2)\cdots(n-k+1)}{k\,(k-1)\,(k-2)\cdots(2)\,(1)} \leq \frac{n^k}{k!}$
- $O(k^2\, n^k / k!) = O(n^k)$

Πολυωνυμικού χρόνου για k=17, αλλά όχι χρήσιμο στην πράξη

# Εκθετικός χρόνος

Υποσύνολα Συνόλου. Δεδομένου ενός συνόλου S βρείτε όλα τα υποσύνολά του.

Λύση $O(2^n)$. Απαριθμήστε όλα τα υποσύνολα.

Έστω: $S \leftarrow \{2,8,9\}$

$S_0 \leftarrow \{\varnothing\}$
$S_1 \leftarrow \{2\}$
$S_2 \leftarrow \{8\}$
$S_3 \leftarrow \{9\}$
$S_4 \leftarrow \{2,8\}$
$S_5 \leftarrow \{2,9\}$
$S_6 \leftarrow \{8,9\}$
$S_7 \leftarrow \{2,8,9\}$

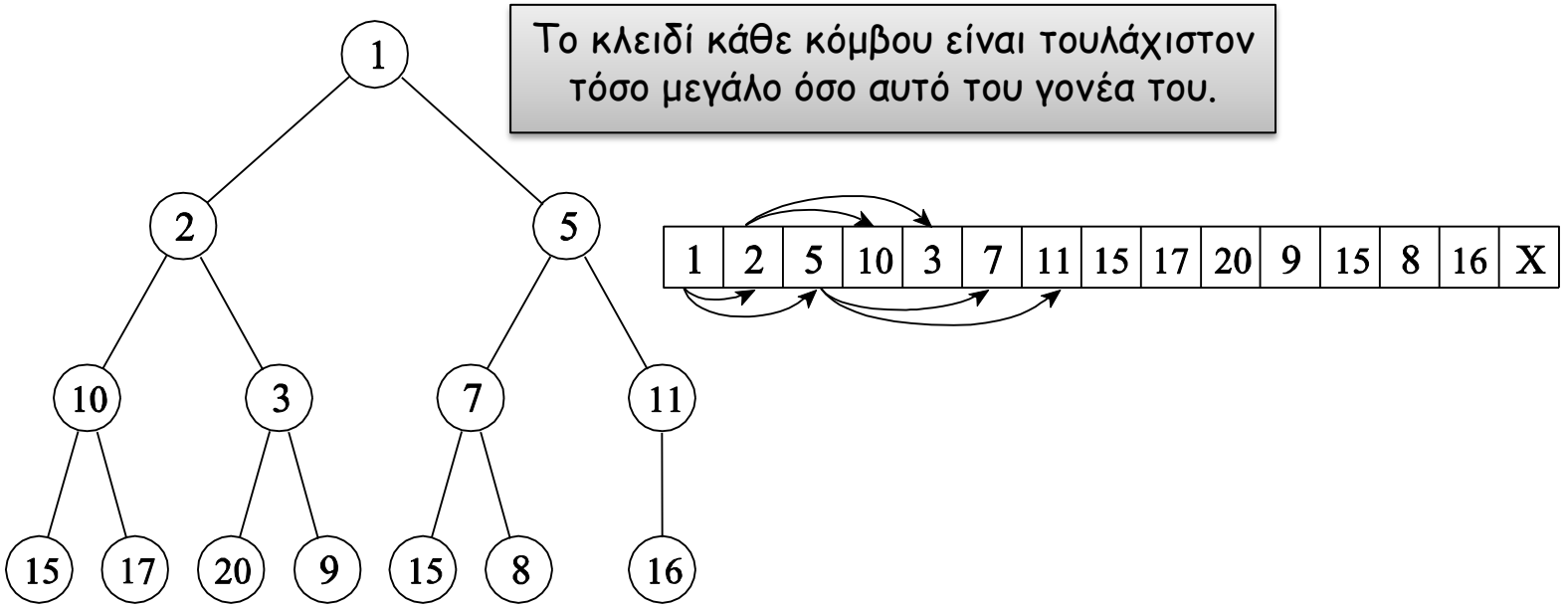
# Σωρός

## Θεμελιώδης Δομή Δεδομένων

# Σωρός (Heap)

Ορισμός: ισοστασθμισμένο δυαδικό δένδρο.

Συνδυάζει πλεονεκτήματα ταξινομημένου πίνακα και λίστας.




Διάταξη σωρού: για κάθε στοιχείο v, σε έναν κόμβο i, το στοιχείο w στο γονέα του i ικανοποιεί τη σχέση: $key(w) \leq key(v)$

Για κάθε κόμβο στη θέση i: $parent(i) = \lfloor i/2 \rfloor$, $leftChild(i) = 2i$, $rightChild(i) = 2i+1$

# Η διαδικασία Heapify-up

Στόχος: διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τη ρίζα

Σχεδόν σωρός με πολύ μικρό κλειδί στη θέση H[i]: το στοιχείο v στη θέση i, έχει πολύ μικρό κλειδί H[i], που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού.

```
Heapify-up(H, i):
  If i > 1 then
    έστω j = parent(i) = ⌊i/2⌋
    If key[H[i]] < key[H[j]] then
      Κάνε εναλλαγή των στοιχείων H[i] και H[j]
      Heapify-up(H, j)
    Endif
  Endif
```

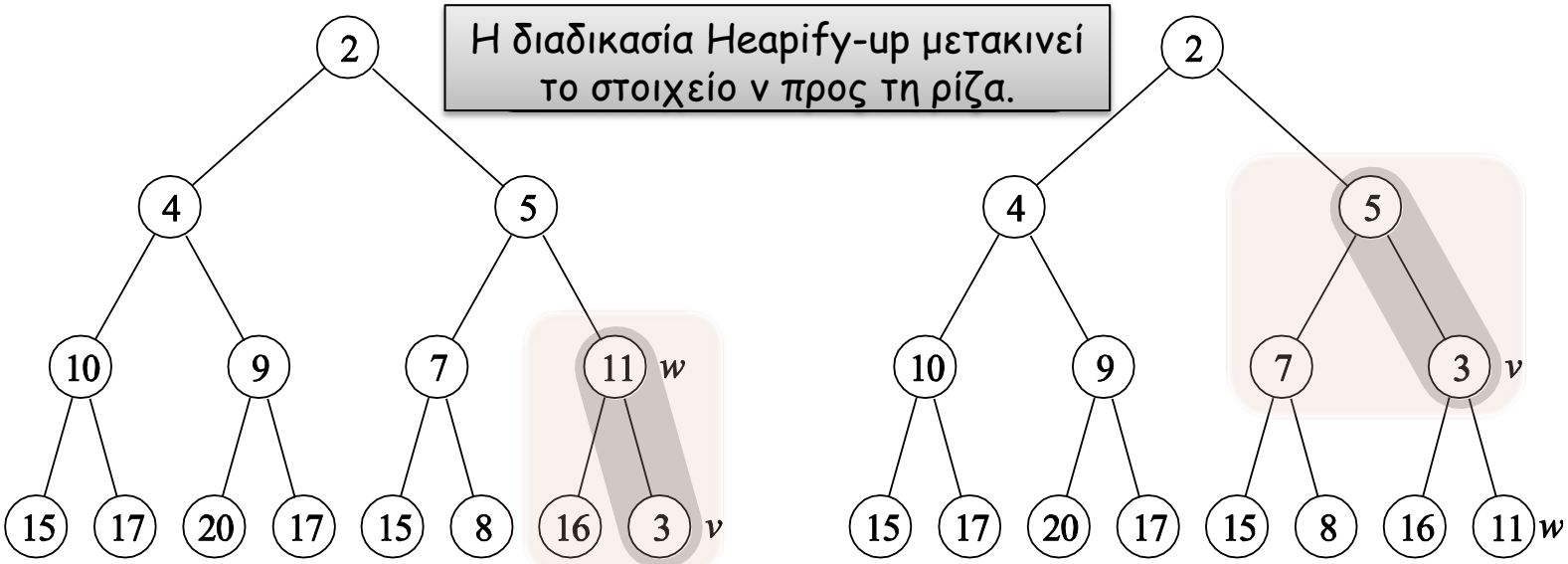
Χρόνος Heapify-up: $O(\log n)$

Εισαγωγή στοιχείου: τοποθέτηση του νέου στοιχείου ως τελευταίο στον σωρό και κλήση της Heapify-up.
Χρόνος: $O(\log n)$

# Η διαδικασία Heapify-up

Στόχος: διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τη ρίζα

Σχεδόν σωρός με πολύ μικρό κλειδί στη θέση H[i]: το στοιχείο v στη θέση i, έχει πολύ μικρό κλειδί H[i], που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού.




Εναλλαγή των κλειδιών 3 και 11.

Ομοίως για τα κλειδιά 3 και 5.

# Η διαδικασία Heapify-down

Στόχος: διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τα φύλλα

Σχεδόν σωρός με πολύ μεγάλο κλειδί στη θέση H[i]: το στοιχείο v στη θέση i, έχει πολύ μεγάλο κλειδί H[i], που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού.

```
Heapify-down(H, i):
  Έστω n = length(H)
  If 2i > n then
     Τερμάτισε με αμετάβλητο H
  Else if 2i < n then
    Έστω left = 2i, και right = 2i+1
    Έστω j ο δείκτης που ελαχιστοποιεί τα key[H[left]] και key[H[right]]
  Else if 2i = n then
     Έστω j = 2i
  Endif
  If key[H[j]] < key[H[i]] then
     Κάνε εναλλαγή των στοιχείων πίνακα H[i] και H[j]
     Heapify-down(H, j)
  Endif
```

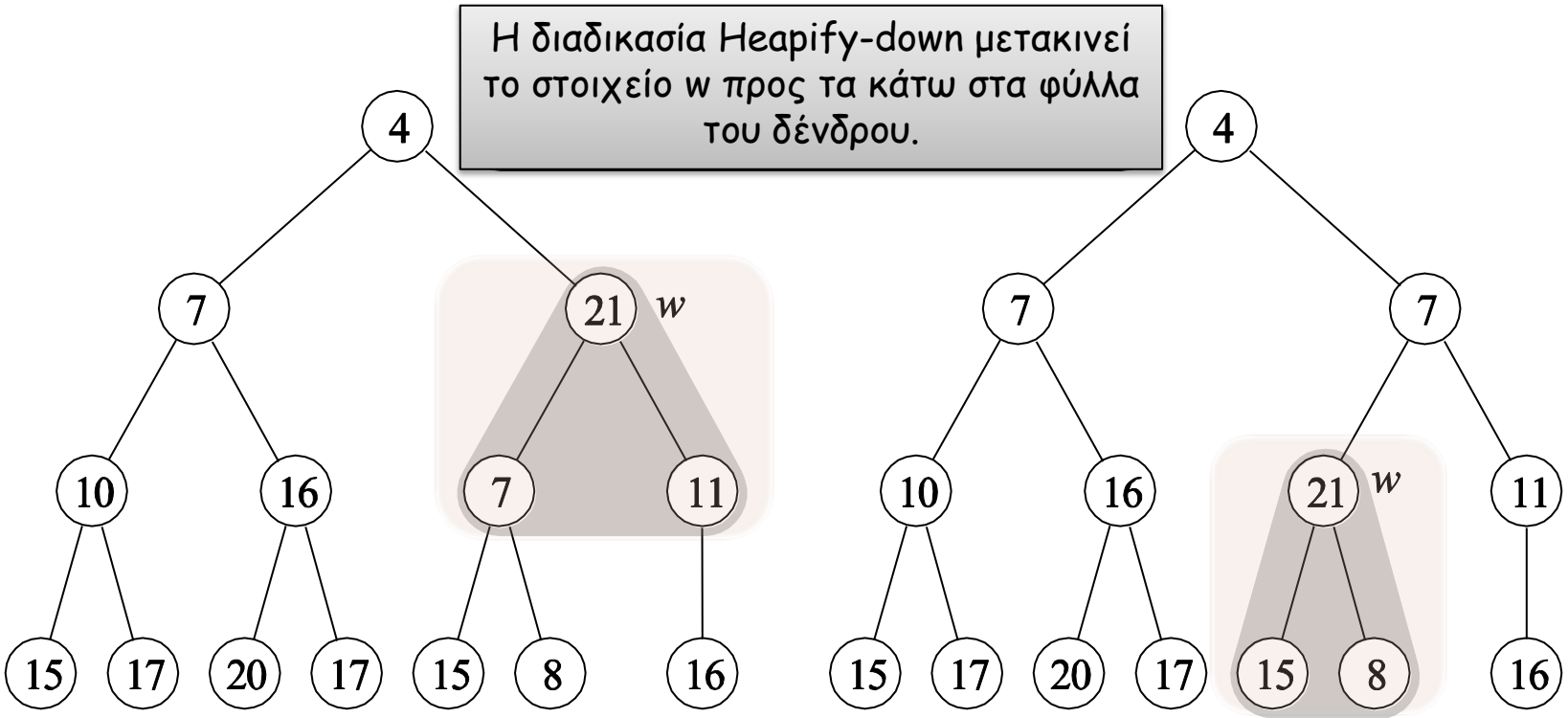
Χρόνος Heapify-down: O(log n)

Διαγραφή στοιχείου v: αντικατάσταση του v με το τελευταίο στοιχείο του σωρού και κλήση της Heapify-down.
Χρόνος: O(log n)

# Η διαδικασία Heapify-down

Στόχος: διόρθωση ενός σχεδόν σωρού με μετακίνηση ενός στοιχείου προς τα φύλλα

Σχεδόν σωρός με πολύ μεγάλο κλειδί στη θέση H[i]: το στοιχείο v στη θέση i, έχει πολύ μεγάλο κλειδί H[i], που πιθανόν παραβιάζει την ιδιότητα διάταξης σωρού.




Εναλλαγή των κλειδιών 21 και 7.

Ομοίως για τα κλειδιά 21 και 8.

# Ουρά Προτεραιότητας

- Δομή δεδομένων που διατηρεί ένα σύνολο στοιχείων S
- Κάθε στοιχείο v ε S → key(v)
- Υποστήριξη εισαγωγής/διαγραφής στοιχείου και επιλογής στοιχείου με το μικρότερο key()

# Ουρά Προτεραιότητας – Υλοποίηση με Σωρό

- **CreateHeap(H):** επιστρέφει κενό σωρό H έτοιμο να αποθηκεύσει $\leq$ N στοιχεία
  Αρχικοποίηση: $O(N)$

- **Insert(H,v):** εισάγει το στοιχείο v στον H (Heapify-up) – $O(\log N)$
- **Delete(H,i):** διαγράφει το στοιχείο στη θέση i του H (Heapify-down) – $O(\log N)$
- **FindMin(H):** εύρεση μικρότερου στοιχείου στον H – $O(1)$
- **ExtractMin(H):** εύρεση και διαγραφή μικρότερου στοιχείου από τον H – $O(\log N)$

## Εφαρμογή: ταξινόμηση n στοιχείων με χρήση σωρού – Heapsort

- Αρχικοποίησε έναν σωρό H με τα n στοιχεία
- Κάλεσε n φορές την ExtractMin(H)
- Χρόνος: $O(n \log n)$

# Βιβλιογραφία


1. J. Kleinberg and E. Tardos, *Σχεδιασμός Αλγορίθμων*, ελληνική έκδοση, Εκδόσεις Κλειδάριθμος, 2008
2. T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Εισαγωγή στους Αλγορίθμους*, ελληνική έκδοση, Πανεπιστημιακές Εκδόσεις Κρήτης, 2012
3. K. Mehlhorn and P. Sanders, *Αλγόριθμοι και Δομές Δεδομένων – Τα βασικά εργαλεία*, ελληνική έκδοση, Εκδόσεις Κλειδάριθμος, 2014
4. S. Dasgupta, C. Papadimitriou, and U. Vazirani, *Αλγόριθμοι*, ελληνική έκδοση, Εκδόσεις Κλειδάριθμος, 2008
5. Θ. Παπαθεοδώρου, *Αλγόριθμοι: Εισαγωγικά Θέματα και Παραδείγματα*, Εκδόσεις Πανεπιστημίου Πατρών, 1999

# Τέλος Ενότητας

# Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

# Σημείωμα Αναφοράς


«Εισαγωγή στους Αλγορίθμους». Έκδοση: 1.0. Πάτρα 2014.
Διαθέσιμο από τη δικτυακή διεύθυνση:

https://eclass.upatras.gr/courses/CEID1083

# Σημείωμα Αδειοδότησης

# Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει) μαζί με τους συνοδευόμενους υπερσυνδέσμους.