

Ουρές Προτεραιότητας, Δυαδικός Σωρός

Δημήτρης Φωτάκης

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο



Ουρά Προτεραιότητας (Priority Queue)

- ❑ Ουρά όπου **σειρά διαγραφής** καθορίζεται από **προτεραιότητα (μεγαλύτερη – μικρότερη)**.
- ❑ Στοιχεία (**προτεραιότητα, πληροφορία**).
- ❑ Ακολουθία από λειτουργίες:
 - **insert(x)**: εισαγωγή x.
 - **deleteMax()**: διαγραφή και επιστροφή στοιχείου μέγιστης προτεραιότητας.
 - **max()**: επιστροφή στοιχείου μέγιστης προτεραιότητας (χωρίς διαγραφή).
 - **changePriority(k)**: αλλαγή προτεραιότητας θέσης k.
 - **isEmpty(), size()**: βοηθητικές λειτουργίες.

Εφαρμογές

□ Άμεσες εφαρμογές:

- Υλοποίηση ουρών αναμονής με προτεραιότητες.
 - Δρομολόγηση με προτεραιότητες.
 - Largest (Smallest) Processing Time First.

□ Έμμεσες εφαρμογές:

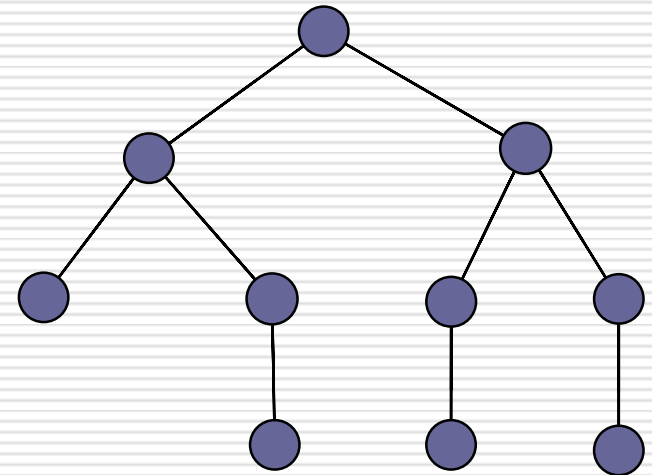
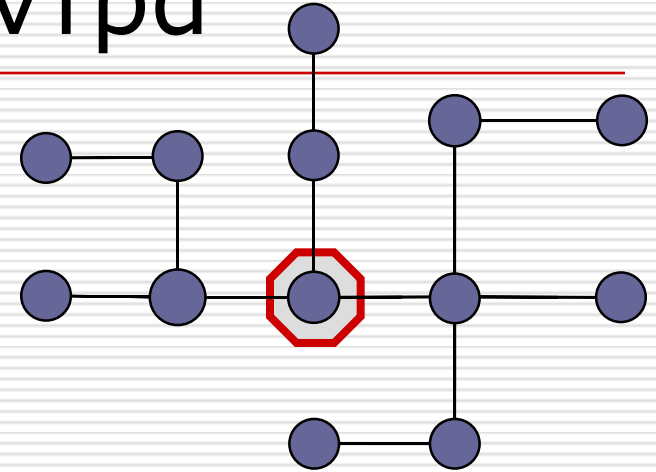
- Βασικό συστατικό **πολλών** ΔΔ και αλγορίθμων:
 - HeapSort (γενικά ταξινόμηση με επιλογή).
 - Αλγόριθμος Huffman.
 - Αλγόριθμοι Prim και Dijkstra.
 - ...

Στοιχεία Ουράς Προτεραιότητας

- Ουρές Προτεραιότητας:
 - **Ολική διάταξη** στοιχείων με βάση προτεραιότητα.
 - Στοιχεία είναι **αριθμοί** (με συνήθη διάταξη) που δηλώνουν προτεραιότητα.
 - Εφαρμογή για στοιχεία **κάθε συνόλου** με σχέση **ολικής διάταξης** (αριθμοί, λέξεις, εισοδήματα, ...).
- Ουρά Προτεραιότητας με **γραμμική λίστα**:
 - Διαγραφή μέγιστου ή εισαγωγή απαιτεί **γραμμικό χρόνο**.
- Υλοποίηση ουράς προτεραιότητας με **σωρό (heap)**.
 - **Δυαδικό δέντρο** με διάταξη σε κάθε μονοπάτι ρίζα – φύλλο.

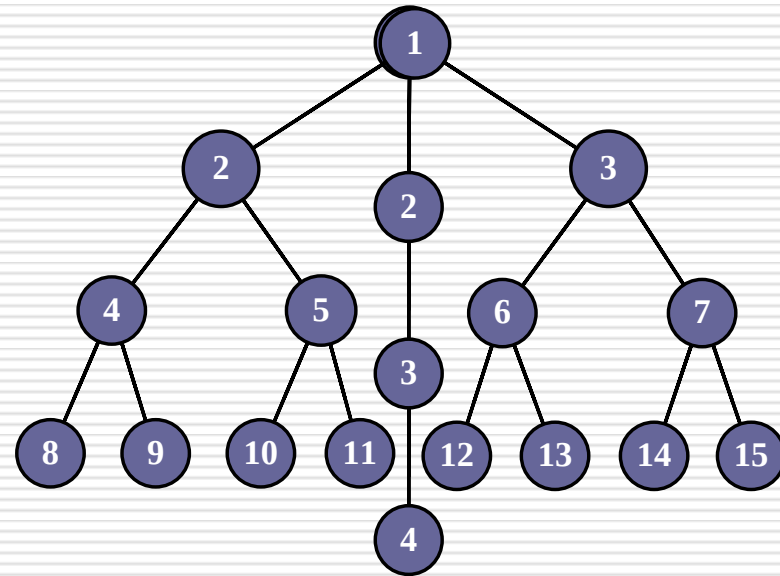
Ιεραρχικές Δομές: Δέντρα

- Γράφημα **ακυκλικό** και **συνεκτικό**.
- Δέντρο με **n κορυφές** έχει **$m = n - 1$ ακμές**.
- Δέντρο με **ρίζα** : **Ιεραρχία**
- **Ύψος** : μέγιστη απόσταση φύλλου από ρίζα.
- **Δυαδικό δέντρο** : έχει **ρίζα** και κάθε κορυφή **≤ 2 παιδιά** :
 - **Αριστερό** και **δεξιό**.
- Κάθε **υποδέντρο** είναι δυαδικό δέντρο.



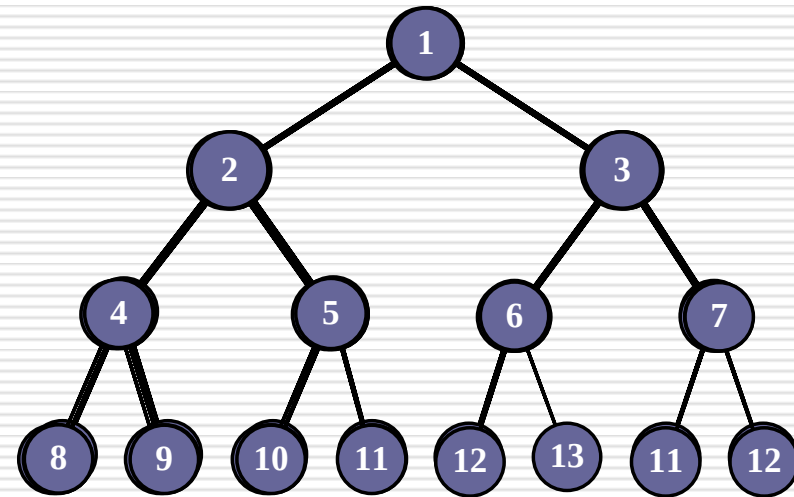
Δυαδικά Δέντρα

- #κορυφών για ύψος = h :
 $h+1 \leq \#κορυφών \leq 2^{h+1} - 1$
 - $h+1$ επίπεδα, ≥ 1 κορ. / επίπ.
 - $\leq 2^i$ κορυφές στο επίπεδο i .
 $1 + 2 + \dots + 2^h = 2^{h+1} - 1$
- Ύψος για #κορυφών = n :
 $\log_2(n+1) - 1 \leq \text{ύψος} \leq n - 1$
- **Γεμάτο** (full):
 - Κάθε κορυφή είτε φύλλο είτε 2 παιδιά.
- **Πλήρες** (complete) :
 - Όλα τα επίπεδα συμπληρωμένα (εκτός ίσως τελευταίο).
- **Τέλειο** (perfect) : $n = 2^{h+1} - 1$



Πλήρες

- Όλα τα επίπεδα **συμπληρωμένα εκτός ίσως από τελευταίο** που γεμίζει από αριστερά προς τα δεξιά.
- $n(h)$: #κορυφών για ύψος h :
 $2^h \leq n(h) \leq 2^{h+1} - 1$
 - $\text{τέλειο}(h) : 2^{h+1} - 1$
 - $\text{τέλειο}(h-1) + 1 : (2^h - 1) + 1 = 2^h$.
- $h(n)$: ύψος για n κορυφές:
 $\log_2(n+1) - 1 \leq h(n) \leq \log_2 n$
- Ύψος : $h(n) = \lfloor \log_2 n \rfloor$
- #φύλλων = $\lceil n / 2 \rceil$



Αναπαράσταση

□ **Δείκτες** σε παιδιά, πατέρα (δυναμική).

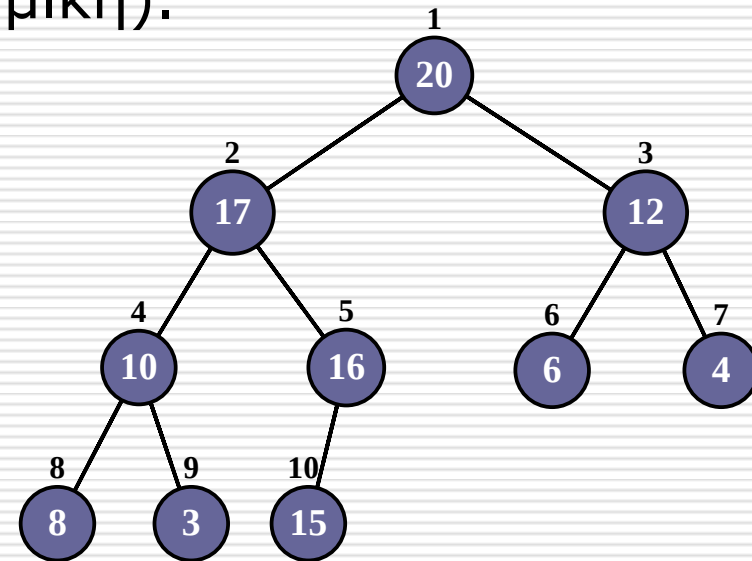
□ **Πλήρη** δυαδικά δέντρα :

■ **Πίνακας** (στατική).

■ Αρίθμηση αριστερά → δεξιά
και πάνω → κάτω.

■ **Ρίζα** : $\Pi[1]$

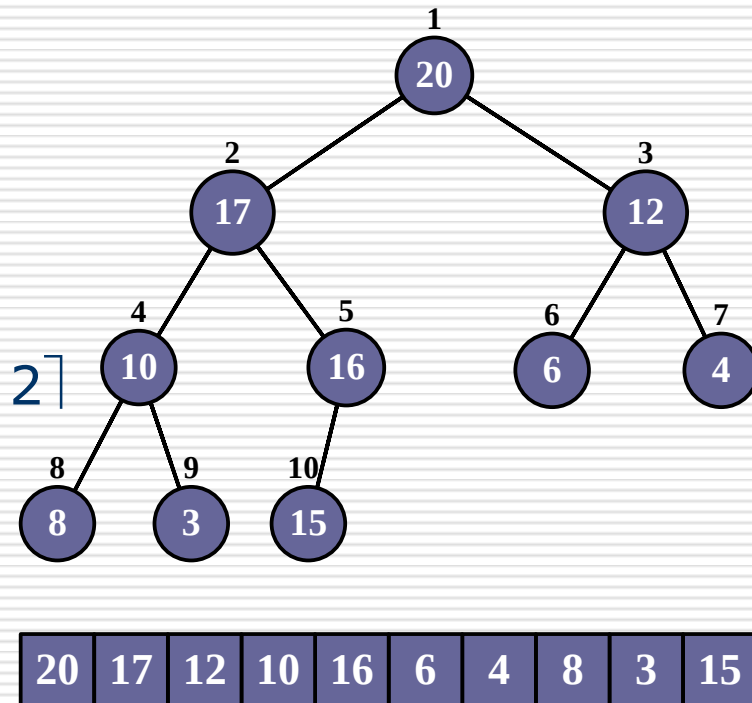
■ $\Pi[i]$: πατέρας $\Pi[i/2]$
αριστερό παιδί $\Pi[2i]$
δεξιό παιδί $\Pi[2i+1]$



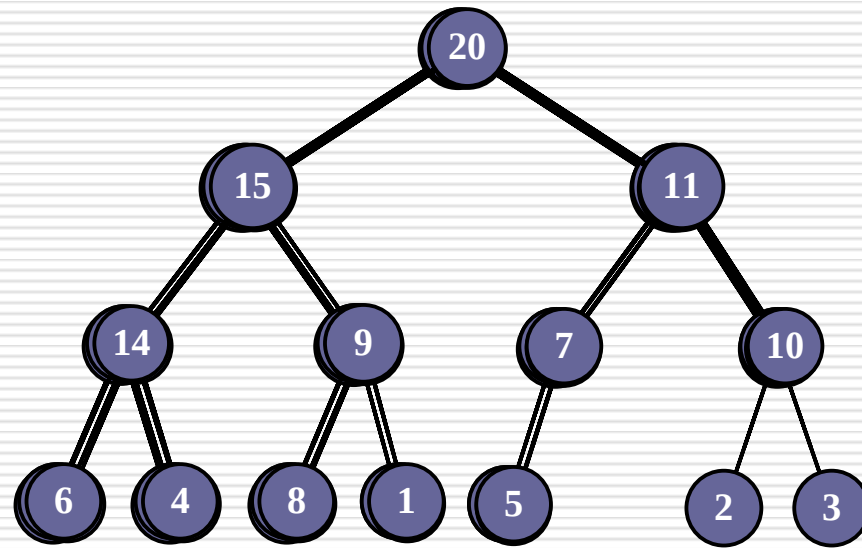
20	17	12	10	16	6	4	8	3	15
----	----	----	----	----	---	---	---	---	----

Σωρός (heap)

- Δέντρο **μέγιστου** (ελάχιστου):
Τιμές στις κορυφές και **τιμή κάθε κορυφής** $\geq (\leq)$ **τιμές παιδιών της**.
- **Σωρός** : πλήρες δυαδικό δέντρο μέγιστου (ελάχιστου).
 - Ύψος $\Theta(\log n)$, #φύλλων = $\lceil n / 2 \rceil$
- Πίνακας **A[]** ιδιότη. **σωρού** :
 $\forall i \ A[i] \geq A[2i], A[2i+1]$.
- **Μέγιστο** : ρίζα
Ελάχιστο : κάποιο φύλλο

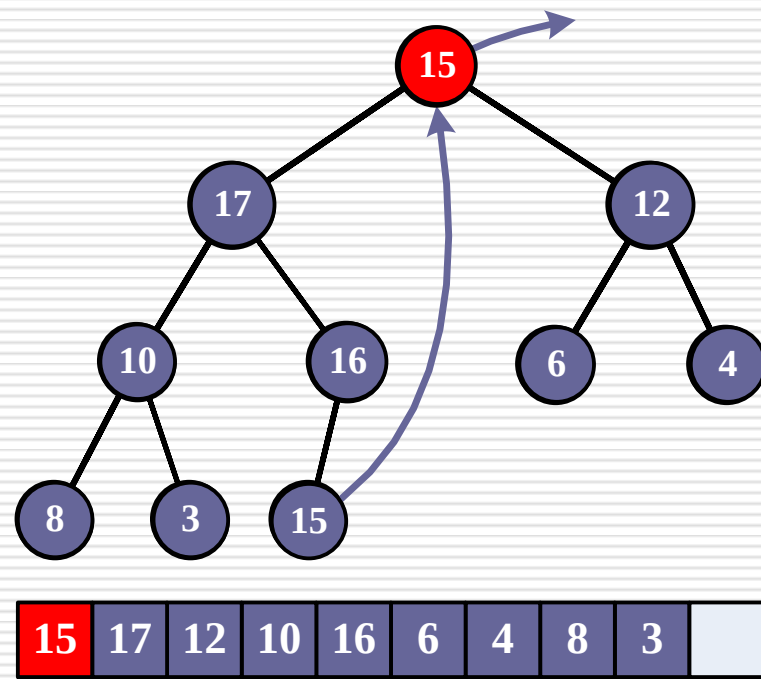


Σωροί και Μη-Σωροί



Σωρός σαν Ουρά Προτεραιότητας

- ❑ `int A[n], hs;`
- ❑ `max() : O(1)`
`int max() { return(A[1]); }`
- ❑ `deleteMax() :`
`int deleteMax() {`
 `if (isEmpty()) return(EMPTY);`
 `max = A[1]; A[1] = A[hs--];`
 `combine(1);`
 `return(max); }`



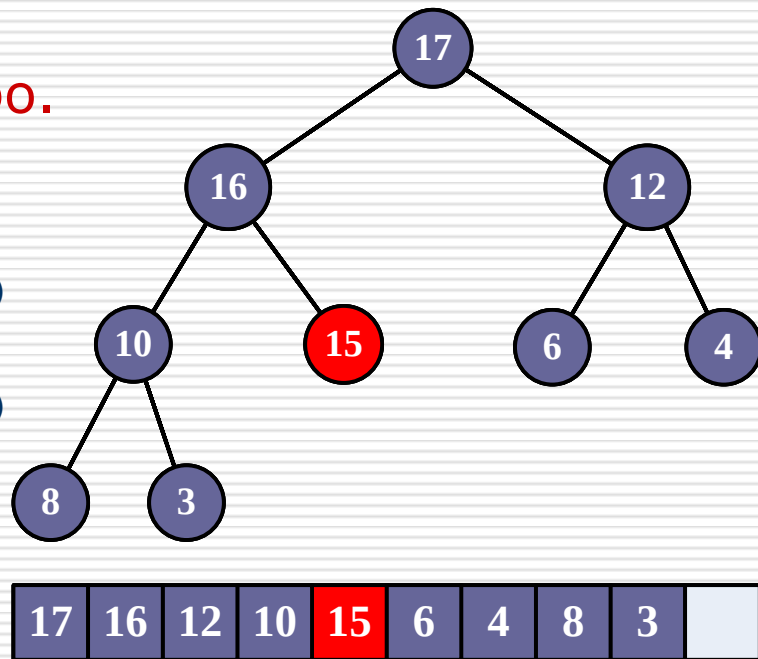
Αποκατάσταση Προς-τα-Κάτω

□ `combine(i)` :

Ενώσω όχι σωρός,

- $A[i] \leftrightarrow \max\{A[2i], A[2i+1]\}$
- συνεχίζω στο αντίστοιχο υποδέντρο.

```
combine(int i) {  
    l = 2*i; r = 2*i+1; mp = i;  
    if ((l <= hs) && (A[l] > A[mp]))  
        mp = l;  
    if ((r <= hs) && (A[r] > A[mp]))  
        mp = r;  
    if (mp != i) {  
        swap(A[i], A[mp]);  
        combine(mp); } }
```



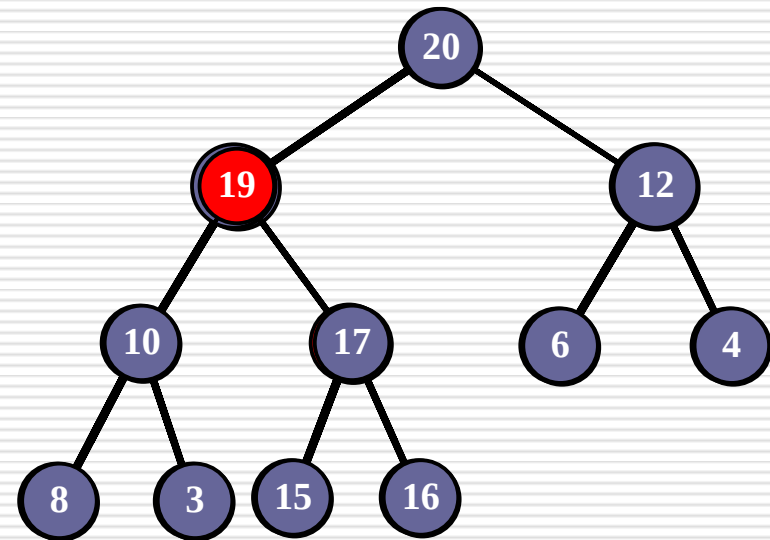
□ Χρόνος για `deleteMax()` : $O(\text{ύψος}) = O(\log n)$

Εισαγωγή: Αποκατάσταση Προς-τα-Πάνω

□ `insert(k)` :

- Εισαγωγή στο **τέλος**.
- Ενώσω όχι σωρός, $A[i] \leftrightarrow A[i/2]$

```
insert(int k) {  
    A[++hs] = k;  
    i = hs; p = i / 2;  
    while ((i > 1) && (A[p] < A[i]))  
    {  
        swap(A[p], A[i]);  
        i = p; p = i / 2; } }
```

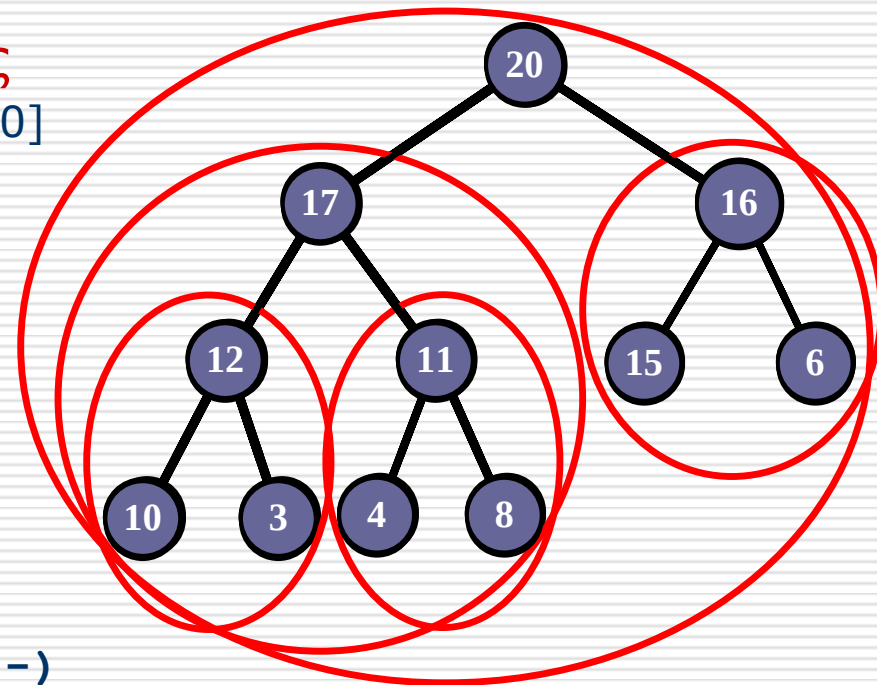


- Χρόνος για `insert()` : $O(\text{ύψος}) = O(\log n)$
- Αύξηση προτεραιότητας : **εισαγωγή** (αποκατ. προς-τα-πάνω).
Μείωση προτεραιότητας : **διαγραφή** (αποκατ. προς-τα-κάτω).

Δημιουργία Σωροῦ

- $A[n] \rightarrow$ σωρός με n εισαγωγές
[3, 4, 6, 10, 8, 15, 16, 17, 12, 11, 20]
- Χρόνος $O(n \log n)$.
- **Ιεραρχικά** (bottom-up):
Υποδέντρα-σωροί **ενώνονται**
σε δέντρο-σωρό.

```
constructHeap(int n) {  
    hs = n;  
    for (i = n / 2; i > 0; i--)  
        combine(i);  
}
```



Χρόνος Δημιουργίας

```
for (i = n / 2; i > 0; i--)  
    combine(i);
```

□ Χρόνος $\text{combine}(i) = O(\text{ύψος } i)$.

$n / 4$ στοιχεία χρόνος $1 \times c$

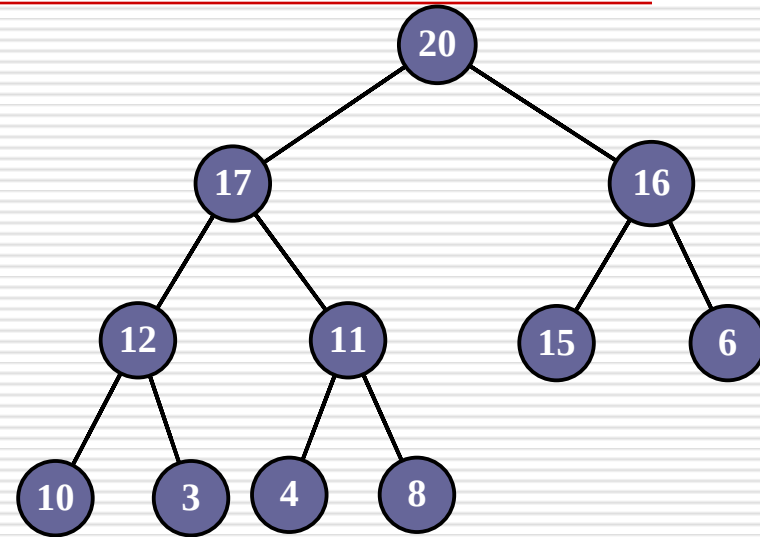
$n / 8$ στοιχεία χρόνος $2 \times c$

.....

$n / 2^k$ στοιχεία χρόνος $(k-1) \times c, k \leq \log_2 n$

□ $\sum_{k=2}^{\log n} \frac{n \cdot k \cdot c}{2^k} = O\left(n \cdot \sum_{k=2}^{\log n} \frac{k}{2^k}\right) = O(n)$, γιατί $\sum_{k=0}^{\infty} \frac{k}{2^k} = 2$

□ Χρόνος $\text{constructHeap}() = \Theta(n)$.



Απόδοση Σωρού

- Χώρος : $\Theta(1)$ (in-place)
- Χρόνοι :
 - createHeap : $\Theta(n)$
 - insert, deleteMax : $O(\log n)$
 - max, size, isEmpty : $\Theta(1)$
- Εξαιρετικά εύκολη υλοποίηση!
- Συμπέρασμα:
 - Γρήγορη και ευρύτατα χρησιμοποιούμενη ουρά προτεραιότητας.

Heap-Sort

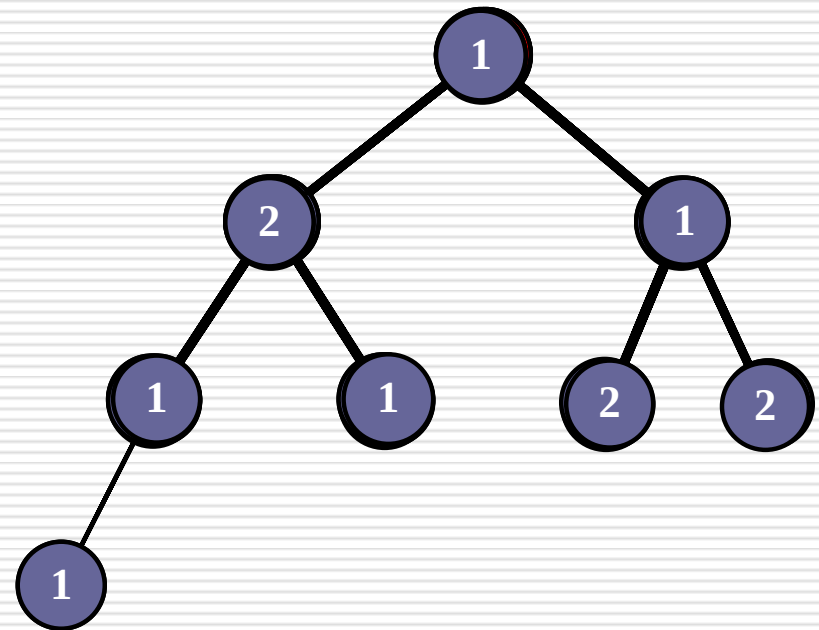
- **Αρχικοποίηση** : δημιουργία **σωρού** με n στοιχεία.
 - `constructHeap()` : χρόνος $\Theta(n)$.
- **Εξαγωγή μέγιστου** και τοποθέτηση **στο τέλος** ($n - 1$ φορές).
 - `deleteMax()` : χρόνος $\Theta(\log n)$.
- **Χρόνος** : $\Theta(n) + n \Theta(\log n) = \Theta(n \log n)$.

```
hs = n;  
constructHeap(n);  
for (i = n; i > 1; i--) {  
    swap(A[1], A[i]); hs--;  
    combine(1); }
```

- Χρονική Πολυπλοκότητα Ταξινόμησης: $O(n \log n)$.

Heap-Sort : Παράδειγμα

```
constructHeap(n);  
for (i = n; i > 1; i--) {  
    swap(A[1], A[i]); hs--;  
    combine(1);  
}
```

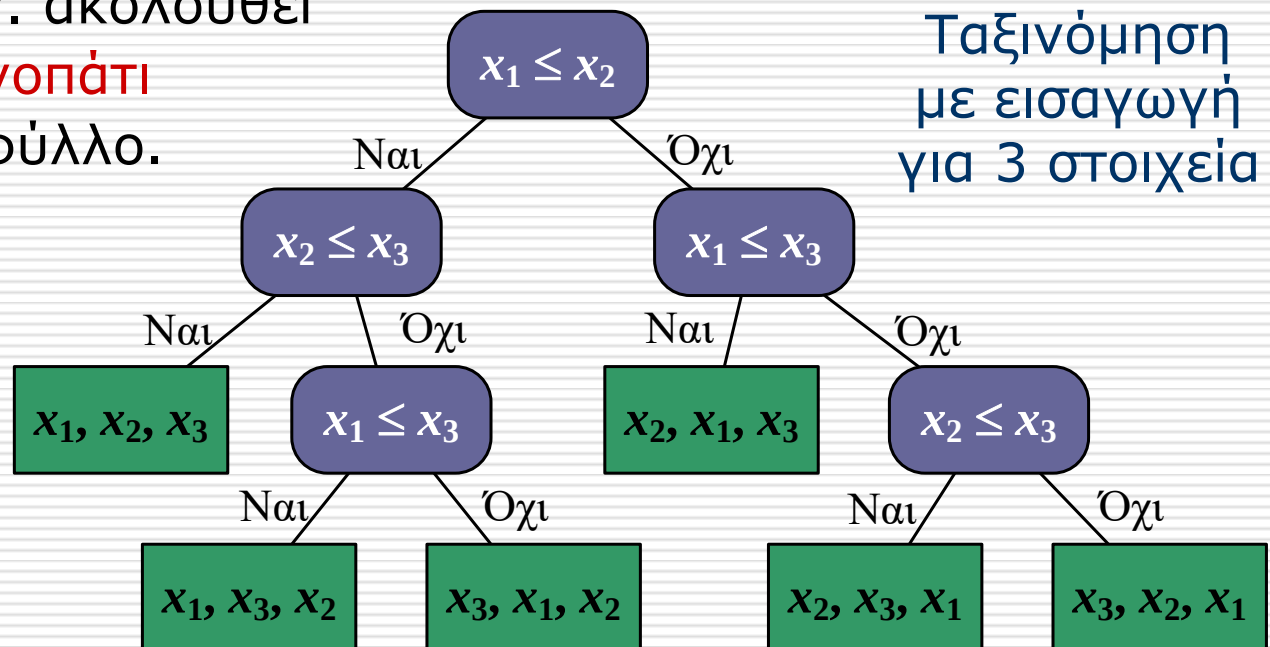


Συγκριτικοί Αλγόριθμοι

- Ταξινόμηση μόνο με συγκρίσεις και αντιμεταθέσεις στοιχείων.
 - Καμία άλλη ενέργεια στα στοιχεία (π.χ. ομαδοποίηση με βάση δυαδική αναπαράσταση).
- Κάθε ντετερμινιστικός συγκριτικός αλγ. ταξινόμησης χρειάζεται $\Omega(n \log n)$ συγκρίσεις μεταξύ στοιχείων.
 - Αντίστοιχο κάτω φράγμα για πιθανοτικούς αλγόριθμους.
- Χρονική Πολυπλοκότητα Ταξινόμησης: $\Theta(n \log n)$
- Υπάρχουν αλγόριθμοι με γραμμικό χρόνο για συγκεκριμένους τύπους δεδομένων (π.χ. αριθμούς).

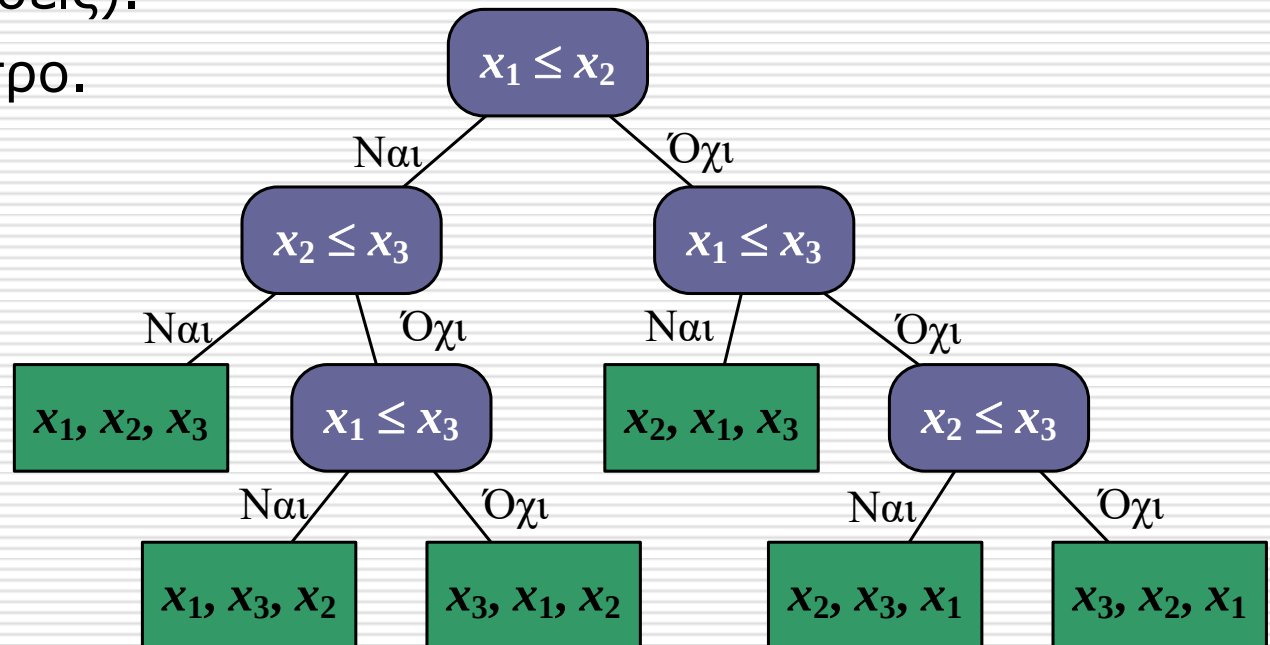
Δέντρο Συγκρίσεων

- ❑ Λειτουργία συγκριτικών αλγορίθμων αναπαρίσταται με **δέντρο συγκρίσεων (ή αποφάσεων)**.
- ❑ Αλγόριθμος $\leftrightarrow$ δέντρο συγκρίσεων.
- ❑ $\forall$ είσοδο: αλγ. ακολουθεί **μοναδικό μονοπάτι** από ρίζα σε φύλλο.



Δέντρο Συγκρίσεων

- Ύψος δέντρου καθορίζει #συγκρίσεων (χ.π.) και αποτελεί κάτω φράγμα στο χρόνο εκτέλεσης.
- Ταξινόμηση n στοιχείων: τουλάχιστον $n!$ φύλλα (όλες μεταθέσεις).
- Δυαδικό δέντρο.



Δέντρο Συγκρίσεων

- Δυαδικό δέντρο ύψους h έχει $\leq 2^h$ φύλλα.
- Χρόνος εκτέλεσης = $\Omega(h)$.
- Ταξινόμηση n στοιχείων: $2^h \geq n!$

$$2^h \geq n! \Rightarrow$$

$$h \geq \log(n!) = \sum_{k=1}^n \log k$$

$$\geq \sum_{k=n/2}^n \log k \geq \sum_{k=n/2}^n \log \frac{n}{2}$$

$$\geq \frac{n}{2} \log \frac{n}{2} = \Omega(n \log n)$$