

Ουρά Προτεραιότητας: Heap

Διδάσκοντες: **Σ. Ζάχος, Δ. Φωτάκης**

Επιμέλεια διαφανειών: **Δ. Φωτάκης**

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο



Δομές Δεδομένων

- (Αναπαράσταση,) **οργάνωση** και **διαχείριση** συνόλων αντικειμένων για αποδοτική **ενημέρωση** και **ανάκτηση** πληροφορίας.
 - Αποδοτική υλοποίηση αλγορίθμων και Βάσεων Δεδομένων.
- (Αποδοτική) **αναπαράσταση – οργάνωση «σύνθετων»** αντικειμένων με χρήση:
 - Βασικών τύπων δεδομένων (ints, floats, chars, strings, arrays).
 - Μηχανισμών που παρέχονται από γλώσσες προγραμματισμού (structs – records, objects).
- **Διαχείριση:** υλοποίηση στοιχειωδών λειτουργιών
 - Ταξινόμηση, αναζήτηση, min/max, first/last, ...
 - Εισαγωγή, διαγραφή, ενημέρωση.
- Λύσεις και τεχνικές για **αποδοτική διαχείριση** δεδομένων.
 - Ανάλυση για απαιτήσεις και καταλληλότητα.

Γενικευμένος Τύπος Δεδομένων

- Γ.Τ.Δ.: **σύνολο** (στιγμιότυπα) με **λειτουργίες** (μεθόδους) επί των στοιχείων του.
- Δομή Δεδομένων: **Υλοποίηση** ενός Γ.Τ.Δ.
 - **Αναπαράσταση – οργάνωση** στιγμιότυπων και **υλοποίηση λειτουργιών** με κατάλληλους αλγόριθμους.
 - **Διατύπωση:** ορισμός αναπαράστασης και περιγραφή υλοποίησης λειτουργιών (ψευδο-κώδικας).
 - **Ανάλυση:** προσδιορισμός απαιτήσεων σε χώρο αποθήκευσης και χρόνο εκτέλεσης για κάθε (βασική) λειτουργία.

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

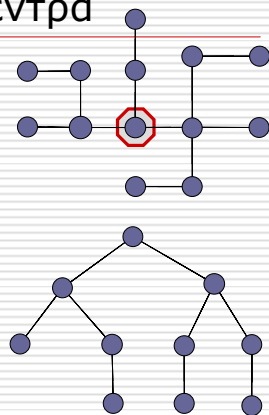
- Άμεσες εφαρμογές:
 - Υλοποίηση ουρών αναμονής με προτεραιότητες.
 - Δρομολόγηση με προτεραιότητες.
 - Largest (Smallest) Processing Time First.
- Έμμεσες εφαρμογές:
 - Βασικό συστατικό **πολλών** ΔΔ και αλγορίθμων:
 - HeapSort (γενικά ταξινόμηση με επιλογή).
 - Αλγόριθμος Huffman.
 - Αλγόριθμοι Prim 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, ...).
- Γραμμικές Δομές Δεδομένων: ολικά διατεταγμένα στοιχεία.
- Υλοποίηση ουράς προτεραιότητας με σωρός (heap).
 - Δυαδικό δέντρο με διάταξη σε κάθε μονοπάτι ρίζα - φύλλο.

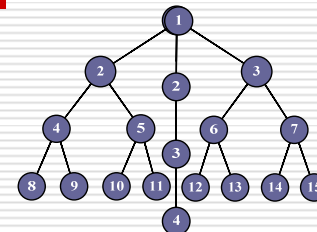
Ιεραρχικές Δομές: Δέντρα

- Γράφημα **ακυκλικό** και **συνεκτικό**.
- Δέντρο με **n κορυφές** έχει **$m = n - 1$ ακμές**.
- Δέντρο με **ρίζα** : **Ιεραρχία**
- **Ύψος** : μέγιστη απόσταση 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 :
 - $h+1 \leq \# \text{κορυφών} \leq 2^{h+1} - 1$
 - $h+1$ 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) :
 - Γεμάτο και όλα τα επίπεδα συμπληρωμένα.
 - $n = 2^{h+1} - 1$



Σχεδόν Πλήρες

□ Όλα τα επίπεδα **συμπληρωμένα** εκτός από **τελευταίο** που πληρώνεται από αριστερά προς τα δεξιά.

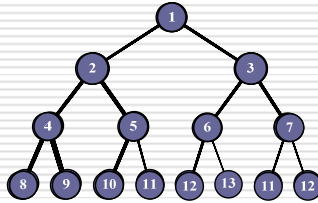
□ #κορυφών για ύψος = h :
 $2^h \leq n \leq 2^{h+1} - 1$

- Πλήρες(h) : $2^{h+1} - 1$
- Πλήρες($h - 1$) + 1 : $(2^h - 1) + 1 = 2^h$.

□ Ύψος για #κορυφών = n :
 $\log_2(n+1) - 1 \leq h \leq \log_2 n$

□ Ύψος : $h(n) = \lfloor \log_2 n \rfloor$

□ #φύλλων = $\lceil n/2 \rceil$



Αναπαράσταση

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

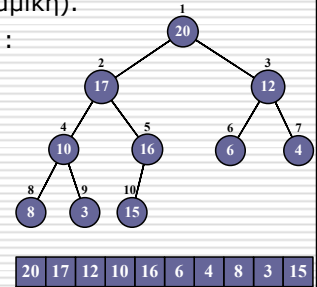
□ **Σχεδόν πλήρη** δυαδικά δέντρα :

■ **Πίνακας** (στατική).

■ Αρίθμηση αριστερά → δεξιά και πάνω → κάτω.

■ **Ρίζα** : $\Pi[1]$

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



Σωρός (heap)

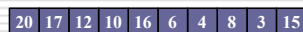
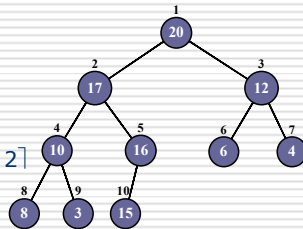
□ Δέντρο **μέγιστου** (ελάχιστου): Τιμές στις κορυφές και **τιμή κάθε κορυφής** \geq (\leq) τιμές παιδιών της.

□ **Σωρός** : σχεδόν πλήρες δυαδικό δέντρο μέγιστου (ελάχιστου).

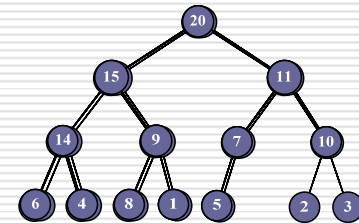
■ Ύψος $\Theta(\log n)$, #φύλλων = $\lceil n/2 \rceil$

□ Πίνακας **A[]** ιδιότ. **σωρού** :
 $\forall i \ A[i] \geq A[2i], A[2i+1]$.

□ **Μέγιστο** : ρίζα
Ελάχιστο : φύλλο

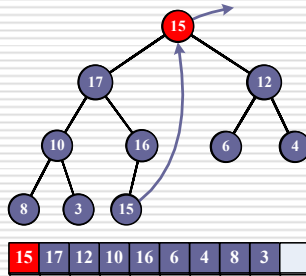


Σωροί και Μη-Σωροί



Σωρός σαν Ουρά Προτεραιότητας

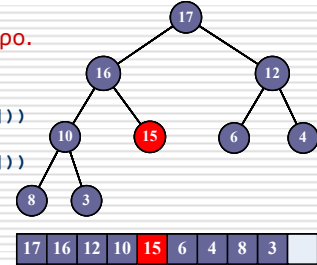
- `int A[n], hs;`
- `max() : O(1)`
`int max() { return(A[1]); }`
- `deleteMax() :`
`int deleteMax() {`
`if (isEmpty()) return(EMPTY);`
`max := A[1]; A[1] := A[hs--];`
`combine(1);`
`return(max); }`



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; lc := i;
  if ((l <= hs) AND (A[l] > A[lc]))
    lc := l;
  if ((r <= hs) AND (A[r] > A[lc]))
    lc := r;
  if (lc != i) {
    swap(A[i], A[lc]);
    combine(lc); } }
```

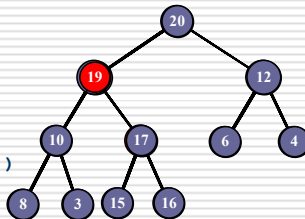


- Χρόνος για `deleteMax()` : $O(\text{ύψος}) = O(\log n)$

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) AND (A[p] < A[i]))
  { swap(A[p], A[i]);
    i := p; p := i / 2; } }
```



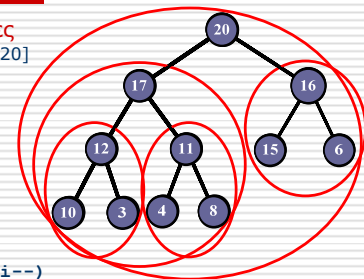
- Χρόνος για `insert()` : $O(\text{ύψος}) = O(\log n)$
- **Αύξηση προτεραιότητας :** εισαγωγή (αποκατ. προς-τα-πάνω).
Μείωση προτεραιότητας : διαγραφή (αποκατ. προς-τα-κάτω).

Δημιουργία Σωρού

- $A[n] \rightarrow$ 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 $O(1)$

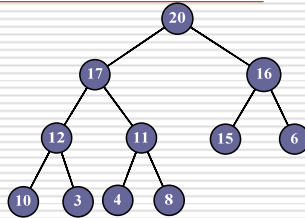
$n/8$ στοιχεία χρόνος $O(2)$

.....

$n/2^k$ στοιχεία χρόνος $O(k), k \leq \log_2 n$

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

- Χρόνος $\text{constructHeap}() = O(n)$.



Απόδοση Σωρού

- Χώρος : $\Theta(n)$ (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** : δημιούργησε σωρό με όλα τα στοιχεία.

- constructHeap() : χρόνος $\Theta(n)$.

- Εξαγωγή μέγιστου** και τοποθέτηση στο τέλος ($n-1$ φορές).

- deleteMax() : χρόνος $\Theta(\log n)$.

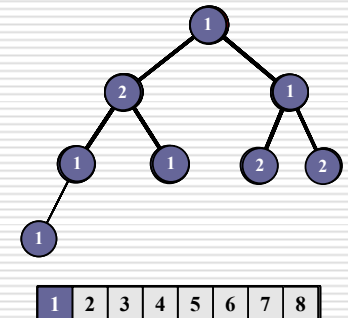
- Χρόνος** : $\Theta(n) + n \Theta(\log n) = \Theta(n \log n)$.

```
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); }
```

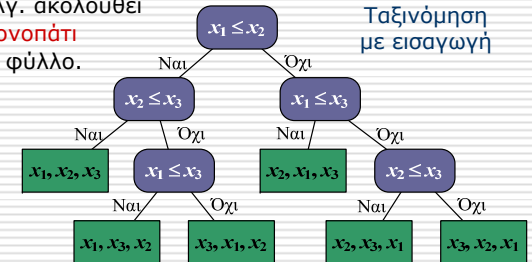


Συγκριτικοί Αλγόριθμοι

- ❑ Ταξινόμηση **μόνο με συγκρίσεις και μετακινήσεις** 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).

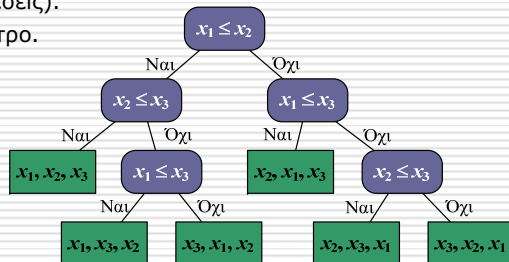
Δέντρο Συγκρίσεων

- ❑ 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).
- ❑ **Διαδικό δέντρο**.



Δέντρο Συγκρίσεων

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

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

$$\begin{aligned}
 h &\geq \log(n!) = \sum_{k=1}^n \log k \\
 &\geq \sum_{k=n/2}^n \log k \geq \sum_{k=n/2}^n \log \frac{n}{2} \\
 &\geq \frac{n}{2} \log \frac{n}{2} = \Omega(n \log n)
 \end{aligned}$$