# Ουρά Προτεραιότητας: Heap

Επιμέλεια διαφανειών: **Δ. Φωτάκης**

(λίγες τροποποιήσεις: Α. Παγουρτζής)

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο

# Δομές Δεδομένων

- (Αναπαράσταση,) οργάνωση και διαχείριση συνόλων δεδομένων 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/median, first/last, ...
  - Εισαγωγή, διαγραφή, ενημέρωση.
- Λύσεις και τεχνικές για αποδοτική διαχείριση δεδομένων.
  - Ανάλυση για απαιτήσεις και καταλληλότητα.

# Γενικευμένος Τύπος Δεδομένων

- **Abstract Data Type** (ADT): μοντέλο περιγραφής δομών δεδομένων ή τύπων δεδομένων με παρόμοιες ιδιότητες. Έμμεση περιγραφή: με επιθυμητές λειτουργίες (μεθόδους) επί των στιγμιοτύπων του.
  - π.χ. `Array, Stack, List, Set, Tree`
- **Δομή Δεδομένων**: Υλοποίηση ενός ADT
  - Αναπαράσταση – οργάνωση δεδομένων 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 εξόδου / διαγραφής** καθορίζεται από **προτεραιότητα.**
- Δεδομένα στοιχείων: προτεραιότητα, πληροφορία.
- Επιθυμητές λειτουργίες:
  - insert(x): εισαγωγή x.
  - deleteMax(): διαγραφή και επιστροφή στοιχείου μέγιστης προτεραιότητας.
  - getMax(): επιστροφή στοιχείου μέγιστης προτεραιότητας (χωρίς διαγραφή).
  - changePriority(x,p): αλλαγή προτεραιότητας στοιχείου x σε p.
  - isEmpty(), size(): βοηθητικές λειτουργίες.

# Εφαρμογές

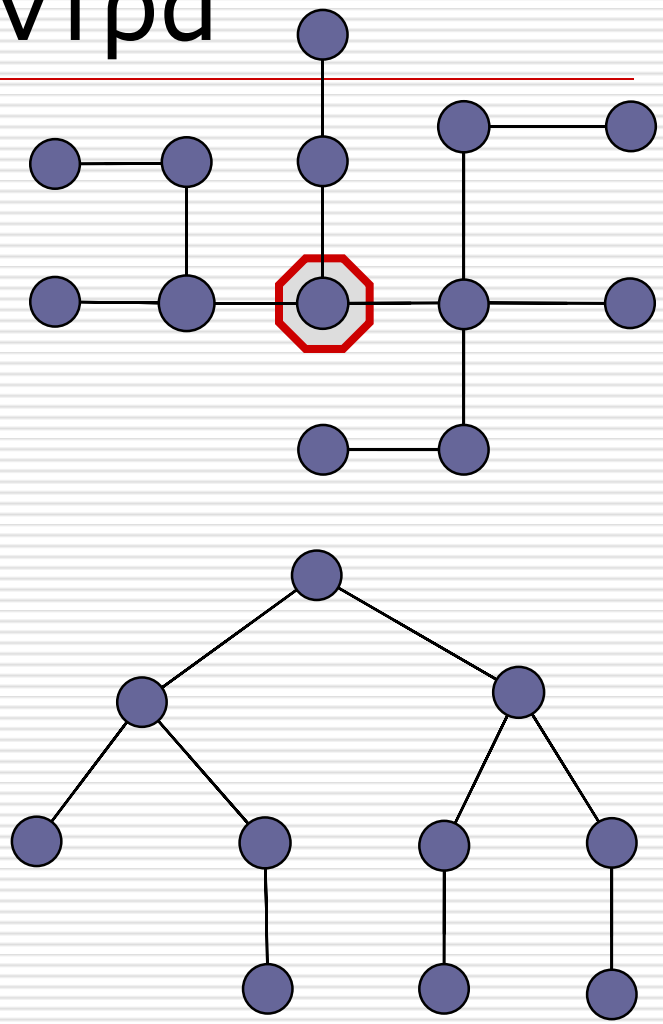
- Άμεσες εφαρμογές:
  - Υλοποίηση ουρών αναμονής με προτεραιότητες.
    - Δρομολόγηση με προτεραιότητες.
    - Largest (Smallest) Processing Time First.
- Έμμεσες εφαρμογές:
  - Βασικό συστατικό **πολλών** ΔΔ και αλγορίθμων:
    - HeapSort (γενικά ταξινόμηση με επιλογή).
    - Αλγόριθμος Huffman.
    - Αλγόριθμοι Prim και Dijkstra.
    - ...

# Υλοποίηση ουρών προτεραιότητας

- Ουρές Προτεραιότητας:
  - **Ολική διάταξη** στοιχείων με βάση προτεραιότητα.
  - Προτεραιότητα καθορίζεται με ακέραιο ριθμό (με συνήθη διάταξη) που περιέχεται σε κάθε στοιχείο.
  - Επεκτείνεται σε κάθε σύνολο με σχέση ολικής διάταξης (ρητοί, πραγματικοί, 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.

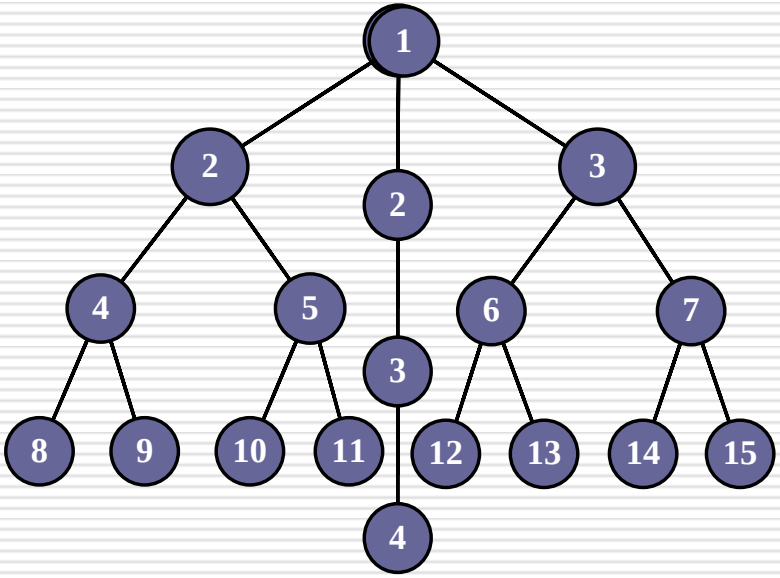
# Ιεραρχικές Δομές: Δέντρα

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



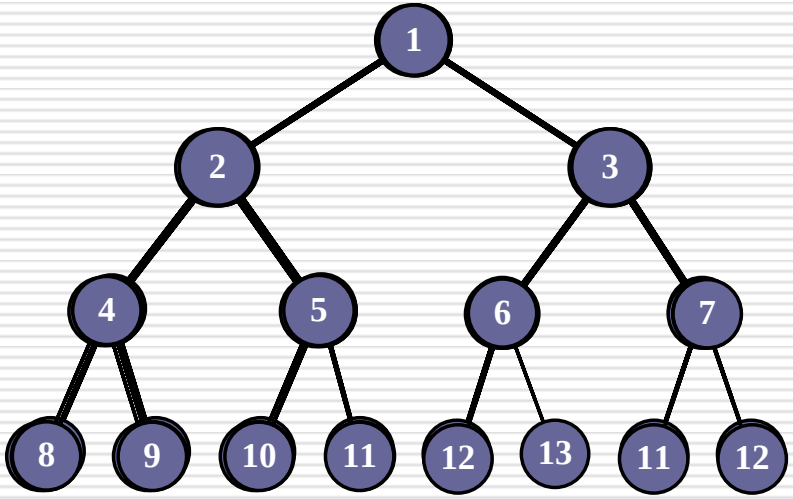
# Δυαδικά Δέντρα

- $n(h)$: #κορυφών σε ΔΔ ύψους $h$.
  $h+1 \le n(h) < 2^{h+1}$
  - $h+1$ επίπεδα, ≥ 1 κορ. / επίπ.
  - $\le 2^i$ κορυφές στο επίπεδο $i$ .
    $1 + 2 + \ldots + 2^h = 2^{h+1} - 1$
- $h(n)$: ύψος ΔΔ με $n$ κορυφές:
  $\log_2 n - 1 < h(n) \le n - 1$
- **Γεμάτο** (full):
  - Κάθε κορυφή είτε φύλλο είτε 2 παιδιά.
- **Πλήρες** (complete) :
  - Όλα τα επίπεδα συμπληρωμένα (εκτός ίσως από τελευταίο).
- **Τέλειο** (perfect): $n = 2^{h+1} - 1$



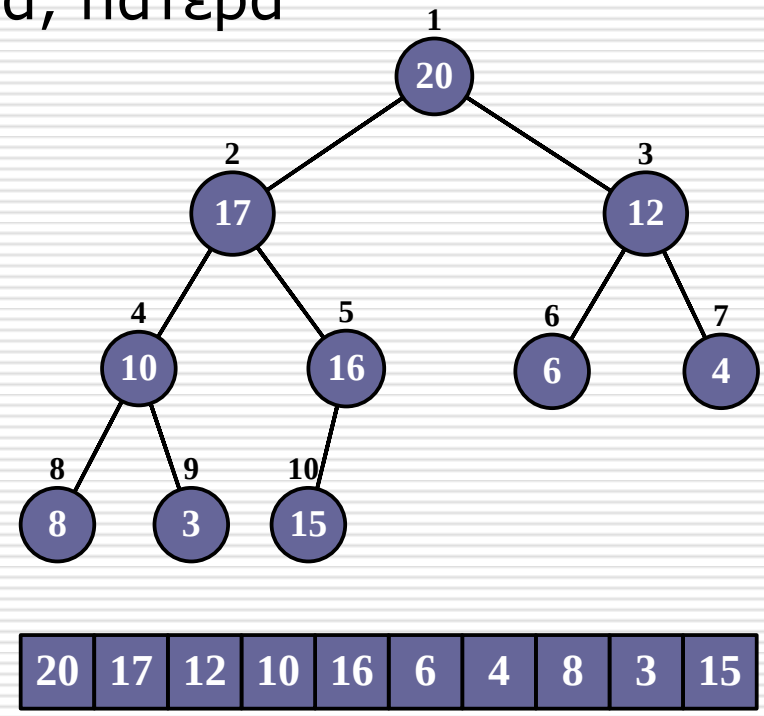
# Πλήρες δυαδικό δέντρο

- Όλα τα επίπεδα συμπληρωμένα εκτός ίσως από τελευταίο που γεμίζει από αριστερά προς τα δεξιά.
- $n(h)$: #κορυφών για ύψος $h$: $2^h \le n(h) < 2^{h+1}$
  - Τέλειο($h$) : $2^{h+1} - 1$
  - Τέλειο($h-1$) + 1 : $(2^h - 1) + 1 = 2^h$ .
- $h(n)$: ύψος για $n$ κορυφές: $\log_2 n - 1 < h(n) \le \log_2 n \Rightarrow h(n) = \lfloor \log_2 n \rfloor$
- Επομένως : $n(h) = \Theta(2^h)$ $h(n) = \Theta(\log_2 n)$
- #φύλλων $= \lceil n / 2 \rceil$



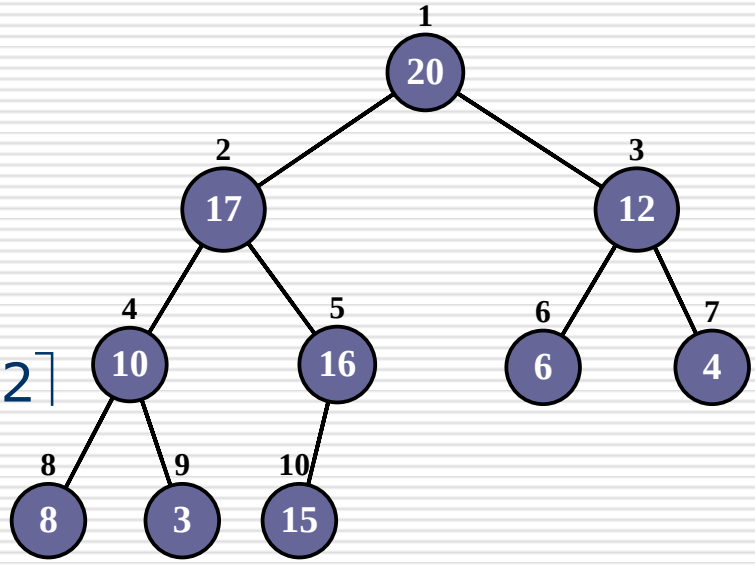
# Αναπαράσταση

- **Γενικά** δέντρα: **δείκτες** σε παιδιά, πατέρα (δυναμική).
- **Πλήρη** δυαδικά δέντρα :
  - **Πίνακας** (στατική).
  - Αρίθμηση αριστερά → δεξιά και πάνω → κάτω.
  - **Ρίζα** : **Π[1]**
  - **Π[*i*]** : πατέρας **Π[*i* / 2]**
    αριστερό παιδί **Π[2*i*]**
    δεξιό παιδί **Π[2*i*+1]**



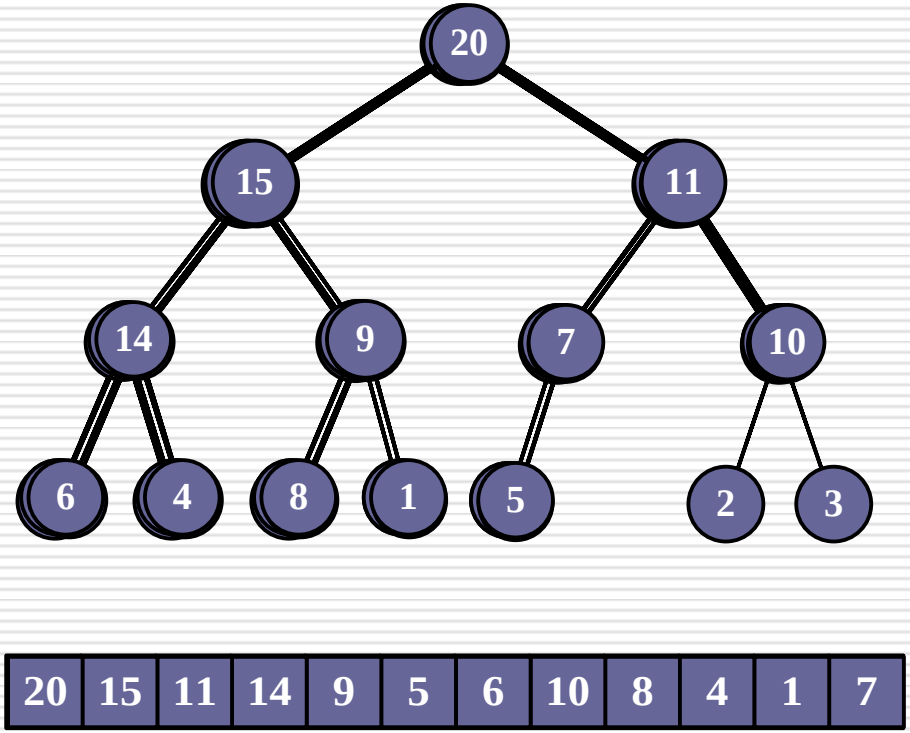
# Σωρός (heap)

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




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

# Σωροί και Μη-Σωροί



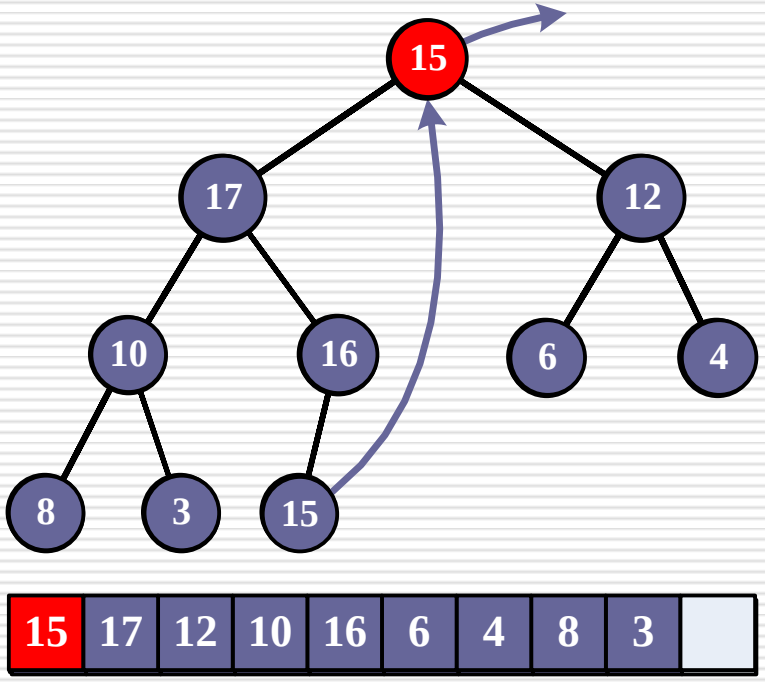
# Σωρός σαν Ουρά Προτεραιότητας

- `int A[n], hs;`
- getMax() : O(1)

```
int getMax() { return(A[1]); }
```

- deleteMax() :

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




- combine(*i*) : αποκατάσταση ιδιότητας σωρού σε υποδέντρο θέσης *i* αν μόνο στοιχείο A[*i*] λάθος

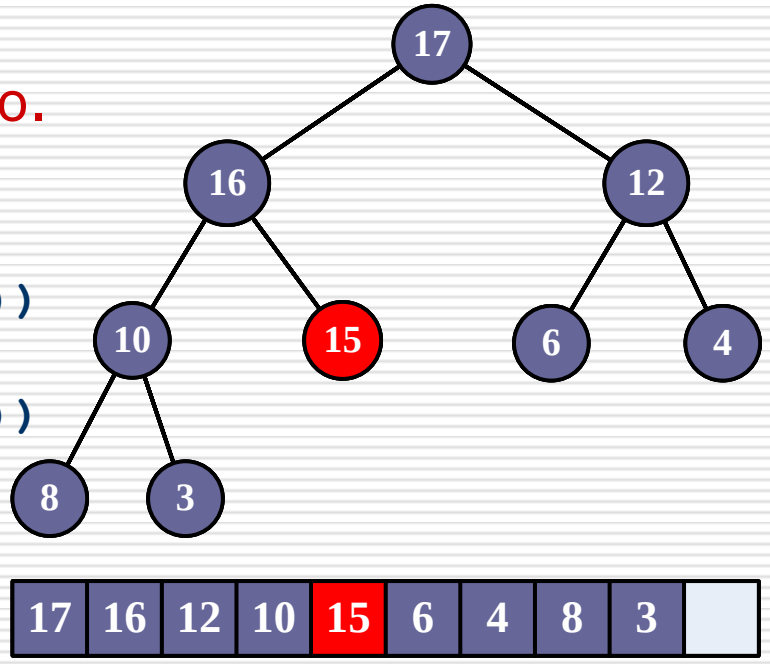
# Αποκατάσταση Προς-τα-Κάτω

- combine(*i*) :

  Ενόσω όχι σωρός (A[i] < max(A[2*i*], A[2*i*+1]))

  - A[i] $\overset{\text{swap}}{\longleftrightarrow}$ max(A[2*i*], A[2*i*+1])
  - συνέχεια στο αντίστοιχο υποδέντρο.

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

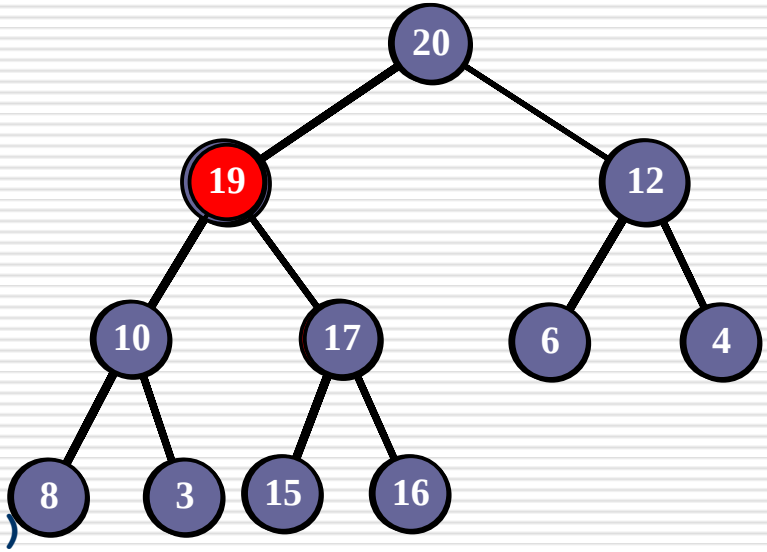



- Χρόνος για deleteMax() : O(ύψος) = O(log *n*)

# Εισαγωγή: Αποκατάσταση Προς-τα-Πάνω

swap ⟷

- insert( *k* ) :
  - Εισαγωγή στο τέλος.
  - Ενόσω όχι σωρός (A[i] > A[*i* /2])
    A[*i*] ⟷ (swap) 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(ύψος) = O(log *n*)

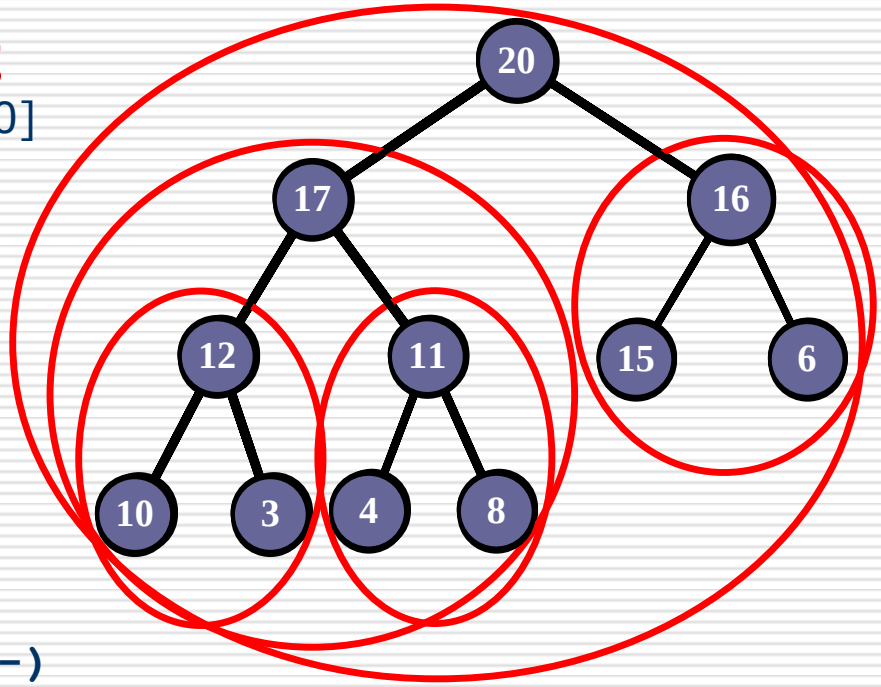
# Αλλαγή προτεραιότητας

- Αύξηση προτεραιότητας : παρόμοια με εισαγωγή (αποκατ. προς-τα-πάνω).
- Μείωση προτεραιότητας : παρόμοια με διαγραφή (αποκατ. προς-τα-κάτω).
- Χρόνος: O(ύψος) = O(log $n$)

# Δημιουργία Σωρού

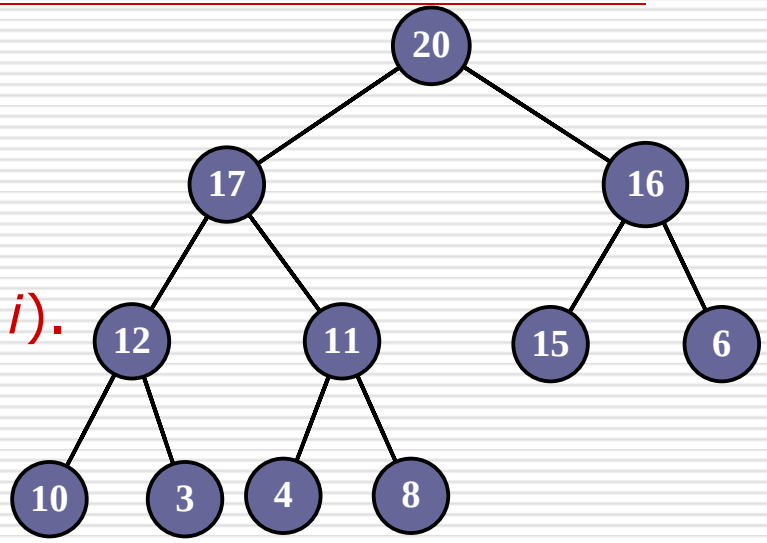
- A[n] → σωρός με *n* εισαγωγές
  [3, 4, 6, 10, 8, 15, 16, 17, 12, 11, 20]
- Χρόνος O(*n*log*n*) .
- **Ιεραρχικά** (bottom-up):
  Υποδέντρα-σωροί ενώνονται σε δέντρο-σωρό.

```
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 combine( *i* ) = O(*n* - ύψος *i*).

  *n* / 4 στοιχεία χρόνος 1×*c*

  *n* / 8 στοιχεία χρόνος 2×*c*

  . . . . . .

  *n* / $2^k$ στοιχεία χρόνος (*k*–1)×*c*, $k \le \log_2 n$

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

- Χρόνος constructHeap( ) = Θ(*n*).

# Απόδοση Σωρού

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

# Heap-Sort

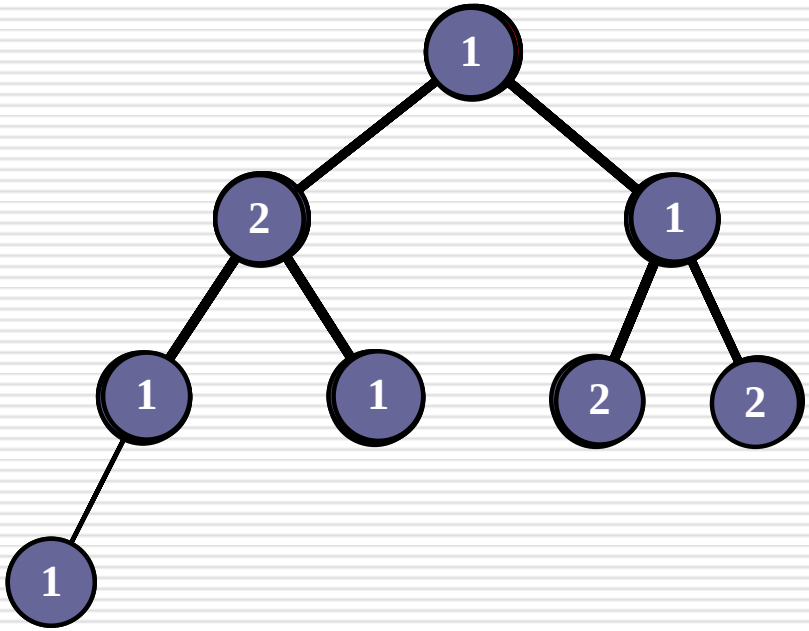
- **Αρχικοποίηση** : δημιουργία σωρού με $n$ στοιχεία.
  - constructHeap() : χρόνος $\Theta(n)$.
- **Εξαγωγή μέγιστου** και τοποθέτηση στο τέλος ($n - 1$ φορές).
  - deleteMax() : χρόνος $\Theta(\log n)$.
- **Χρόνος** : $\Theta(n) + n\,\Theta(\log n) = \Theta(n \log n)$.

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

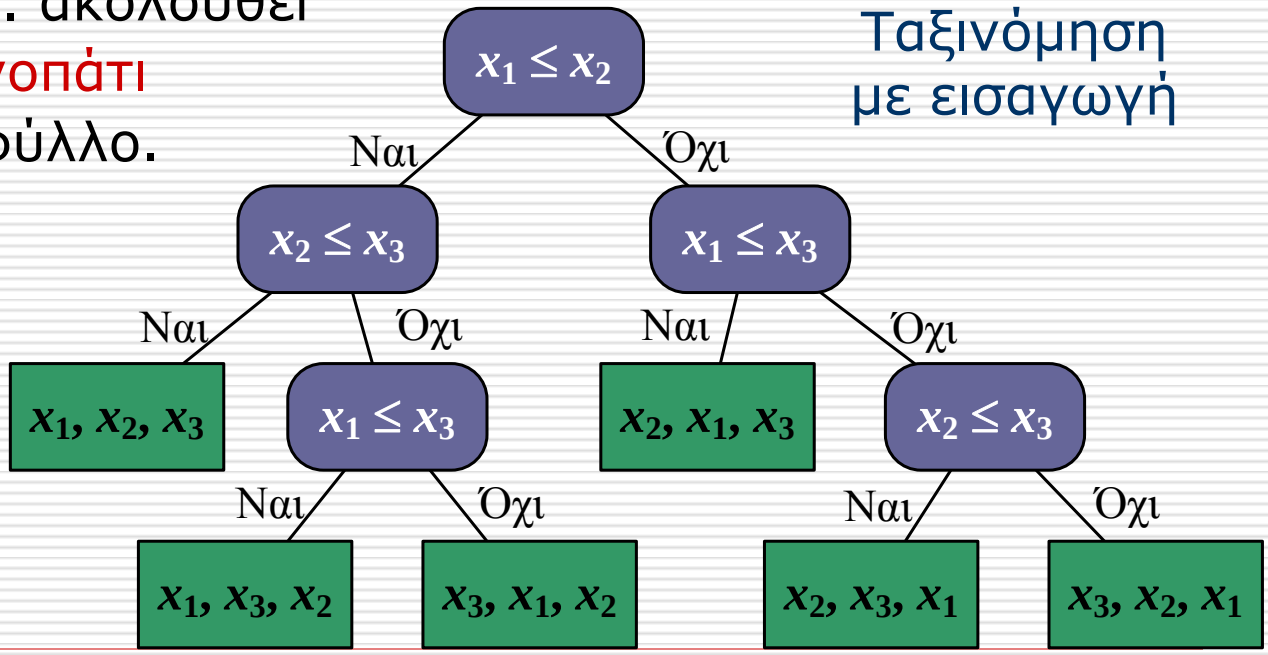



# Συγκριτικοί Αλγόριθμοι

- 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

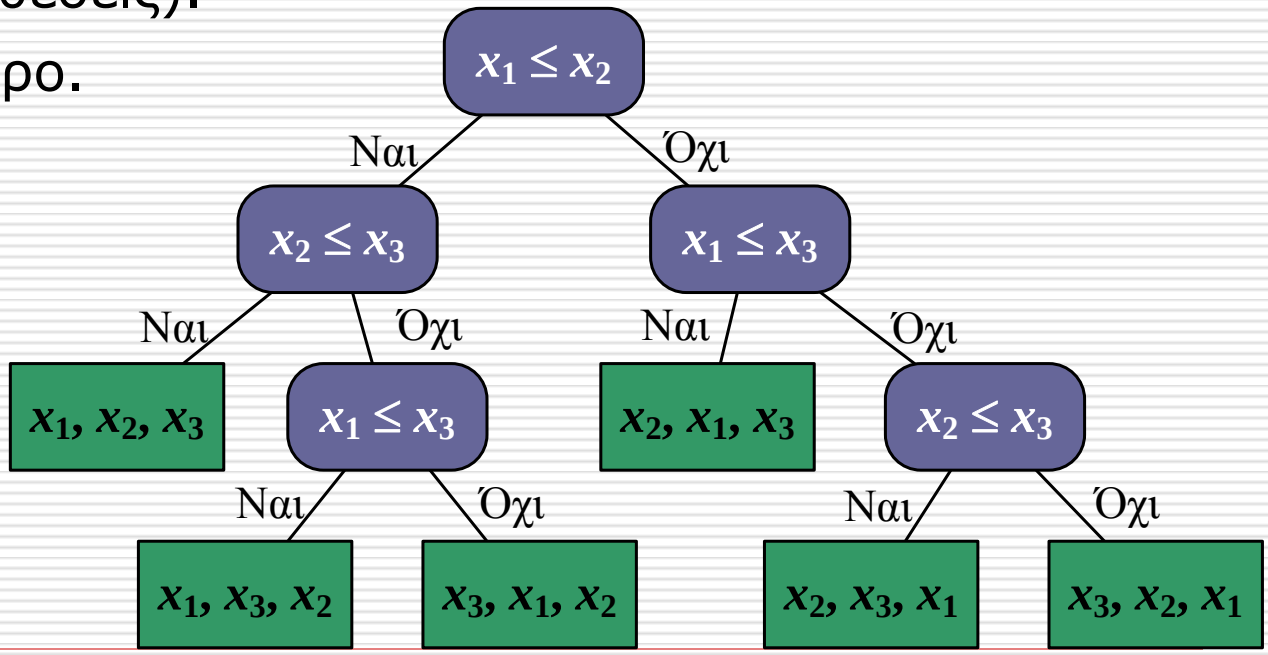
- Λειτουργία συγκριτικών αλγορίθμων αναπαρίσταται με δέντρο συγκρίσεων (ή αποφάσεων).
- Αλγόριθμος ↔ δέντρο συγκρίσεων.
- ∀είσοδο: αλγ. ακολουθεί μοναδικό μονοπάτι από ρίζα σε φύλλο.




Ταξινόμηση με εισαγωγή

# Δέντρο Συγκρίσεων

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



# Δέντρο Συγκρίσεων

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

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

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

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

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

# Εναλλακτική απόδειξη

- Επιχείρημα αντιπάλου (adversary argument).
- Σωστή ταξινόμηση: μία από $n!$ δυνατές μεταθέσεις
- Σε κάθε σύγκριση "$x_i < x_j$ ?" σύνολο δυνατών μεταθέσεων χωρίζεται σε δύο υποσύνολα, αλγόριθμος συνεχίζει με το ένα.
- Ο αντίπαλος επιλέγει μεγαλύτερο από τα δύο.
- Θέτει κατάλληλα $x_i$, $x_j$ ώστε αλγόριθμος συνεχίζει με το μεγαλύτερο.
- Μέγεθος υποσυνόλου: $\geq n!/2^k$ μετά από $k$-οστή σύγκριση.
- Ελάχιστο πλήθος συγκρίσεων: $h$ τ.ώ. $n!/2^h \leq 1$
- … $\Rightarrow$ $\boldsymbol{h = \Omega(n \log n)}$

# Συνοψίζοντας

- (Δυαδικός) σωρός: αποδοτική υλοποίηση ουράς προτεραιότητας με δυαδικό δέντρο.
- Ιδιότητα σωρού: προτεραιότητα γονέα μεγαλύτερη από προτεραιότητα παιδιών.
- Εύκολη υλοποίηση με πίνακα.
- Πολλές εφαρμογές: δρομολόγηση, Prim, Dijkstra, …
- Χρήση σε ταξινόμηση Heapsort: πολυπλοκότητα $\Theta(n \log n)$.
- Βέλτιστη πολυπλοκότητα μεταξύ συγκριτικών αλγορίθμων: απόδειξη με δέντρο αποφάσεων ή επιχείρημα αντιπάλου.