Βασικά περί Δομών Δεδομένων

Διδάσκοντες:
Αρ. Παγουρτζής, Δ. Φωτάκης, Δ. Σούλιου
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/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): σύνολο (στιγμιότυπα) με λειτουργίες (μεθόδους) επί των στοιχείων του.
- **Δομή Δεδομένων**: Υλοποίηση ενός ADT
  - Αναπαράσταση – οργάνωση στιγμιοτύπων 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.

Πρόβλημα (ADT) Λεξικού

- Δυναμικά μεταβαλλόμενη συλλογή αντικειμένων που αναγνωρίζονται με «κλειδί» (π.χ. κατάλογοι, πίνακες ΒΔ).
- **Λεξικό** : συλλογή αντικειμένων με μοναδικό «κλειδί».
  - «Κλειδί»: αριθμός ή τύπος δεδομένων με ολική διάταξη.
  - Γενίκευση και για μη-μοναδικά κλειδιά.
- ADT λεξικού υποστηρίζει ακολουθίες λειτουργιών:
  - Αναζήτηση στοιχείου με κλειδί x
    - member(x): ελέγχει ύπαρξη στοιχείου με κλειδί x
    - search(x): επιστρέψει δείκτη σε θέσεις x
  - Εισαγωγή στοιχείου με κλειδί x
  - Διαγραφή στοιχείου με κλειδί x
  - Βοηθητικές λειτουργίες ...

Λειτουργίες Λεξικού

- Λεξικό μπορεί ακόμη να υποστηρίζει λειτουργίες που σχετίζονται με ταξινόμηση κλειδιών:
  - Εκτύπωση στοιχείων σε αύξουσα / φθίνουσα σειρά
  - Προηγούμενο και επόμενο στοιχείο.
  - Μέγιστο και ελάχιστο στοιχείο.
  - k-οστό μικρότερο στοιχείο

Υλοποιήσεις Λεξικού

- Μη-ταξινομημένη διασυνδεδεμένη λίστα:
  - Εισαγωγή: $O(1)$
  - Αναζήτηση / τυχαία διαγραφή: $O(n)$
  - Κατάλληλη όταν συχνές εισαγωγές, σπάνιες αναζητήσεις / διαγραφές μεμονωμένες ή στο τέλος (π.χ. log file).
- Ταξινομημένος πίνακας:
  - (Δυαδική) αναζήτηση: $O(\log n)$
  - Στατική συλλογή : «εισαγωγή» $O(\log n)$ / στοιχείο
    Χρόνος ταξινόμησης : $O(n \log n)$
  - Δυναμική συλλογή : εισαγωγή / διαγραφή $O(n)$
  - Κατάλληλη όταν συχνές αναζητήσεις και δεδομένα μεταβάλλονται σπάνια (π.χ. αγγλο-ελληνικό λεξικό).

Υλοποιήσεις Λεξικού

- Ζυγισμένο (Δυαδικό) Δέντρο Αναζήτησης:
  - Αναζήτηση / εισαγωγή / διαγραφή: $O(\log n)$
  - Μέγιστο / ελάχιστο / προηγούμενο / επόμενο / k-οστό: $O(\log n)$
  - Range queries σε γραμμικό χρόνο.
  - Πλήρως δυναμική – επιπλέον χώρος για δείκτες!
- Πίνακας Κατακερματισμού (hashing):
  - Αναζήτηση / διαγραφή : $O(1)$
  - Εισαγωγή : $O(1)$ expected amortized, $O(\log n)$ (ακόμη και $O(1)$) whp., $O(n)$ χ.π.
  - Δεν υποστηρίζει αποδοτικά άλλες λειτουργίες.
  - Δυναμική – επιπλέον χώρος στον πίνακα (util ≈ 50%)

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

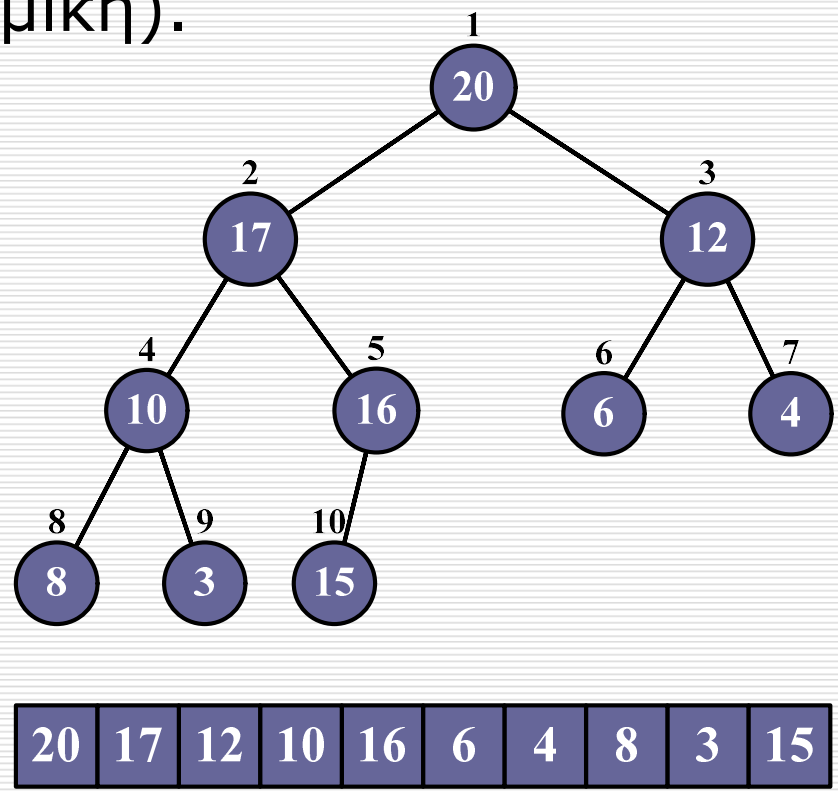
- Άμεσες εφαρμογές:
  - Υλοποίηση ουρών αναμονής με προτεραιότητες.
    - Δρομολόγηση με προτεραιότητες.
    - Largest (Smallest) Processing Time First.
- Έμμεσες εφαρμογές:
  - Βασικό συστατικό **πολλών** ΔΔ και αλγορίθμων:
    - HeapSort (γενικά ταξινόμηση με επιλογή).
    - Αλγόριθμος Huffman.
  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

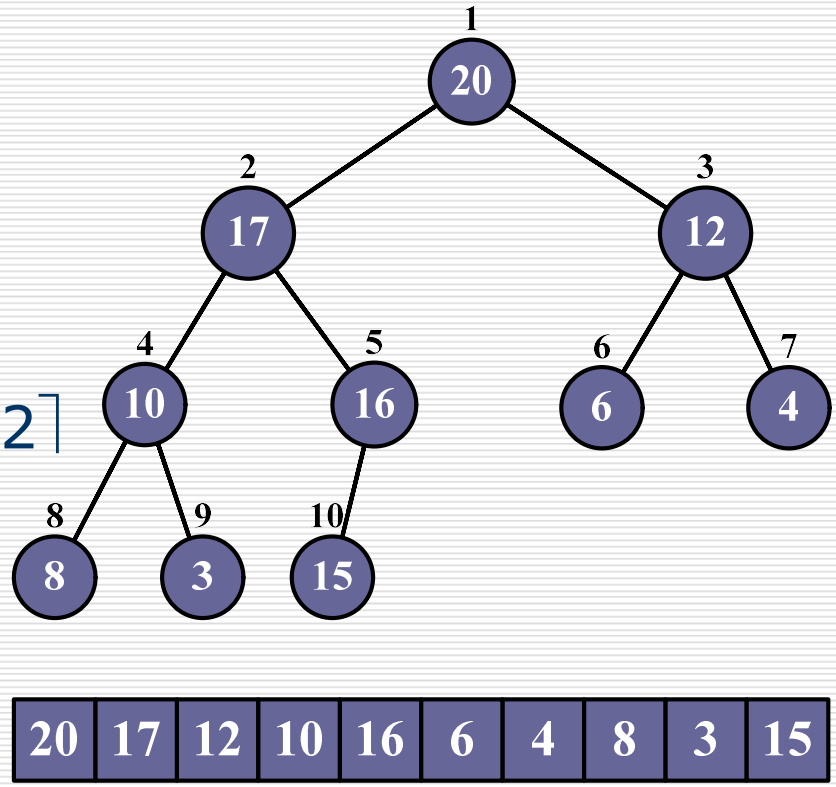
Αναπαράσταση

- **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[ ]** ιδιότ. **σωρού** :
  $\forall i$ $A[i] \geq A[2i], A[2i+1]$.
- **Μέγιστο** : ρίζα
  Ελάχιστο : κάποιο φύλλο



20	17	12	10	16	6	4	8	3	15

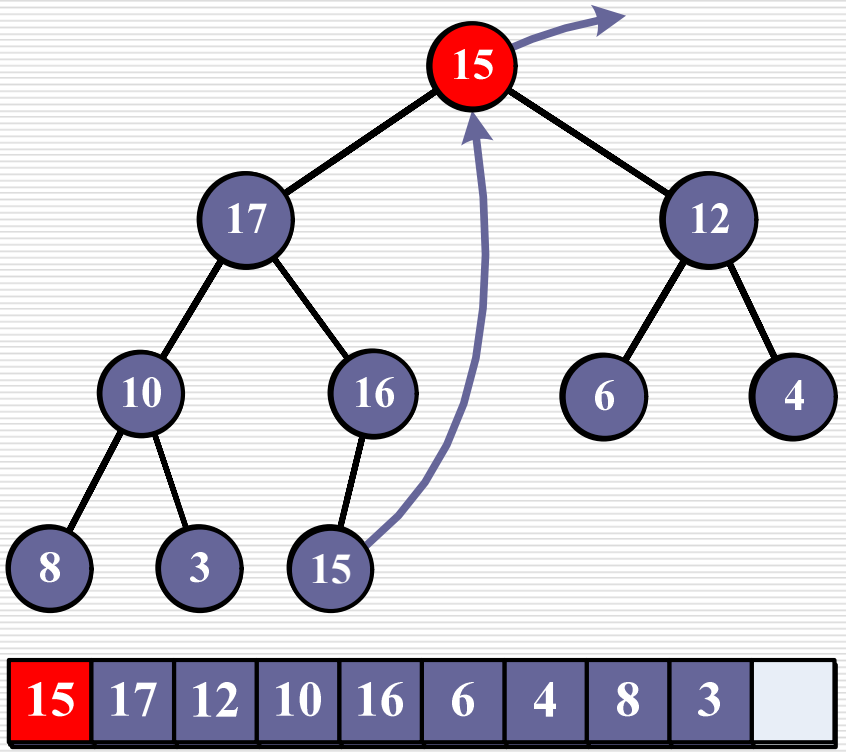
Σωρός σαν Ουρά Προτεραιότητας

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

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

- deleteMax() :

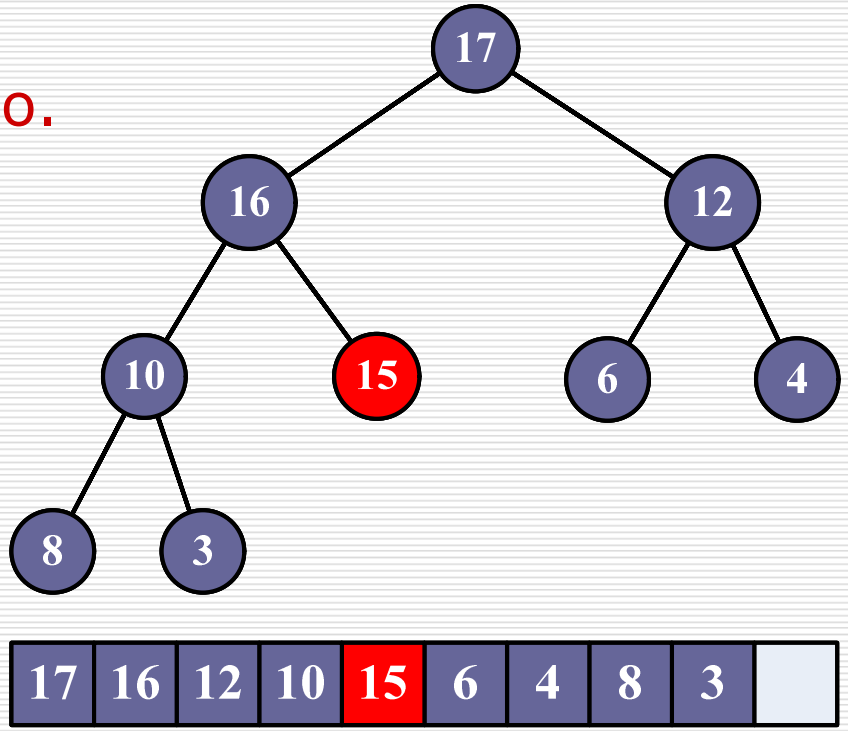
```
int deleteMax() {
   if (isEmpty()) return(EMPTY);
   max = A[1]; A[1] = A[hs--];
   combine(1);
   return(max);  }
```



Αποκατάσταση Προς-τα-Κάτω

- combine(*i*) :
  Ενόσω όχι σωρός,
  - A[i] ↔ max{A[2*i*], A[2*i*+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); } }
```

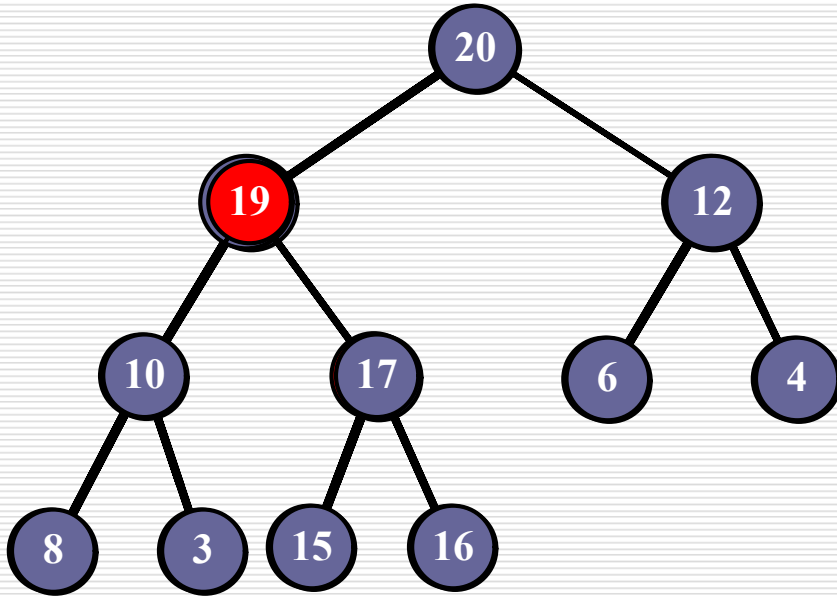


- Χρόνος για deleteMax() : O(ύψος) = O(log n)

Εισαγωγή: Αποκατάσταση Προς-τα-Πάνω

- insert(*k*) :
  - Εισαγωγή στο τέλος.
  - Ενόσω όχι σωρός, A[*i*] ↔ 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; }  }
```

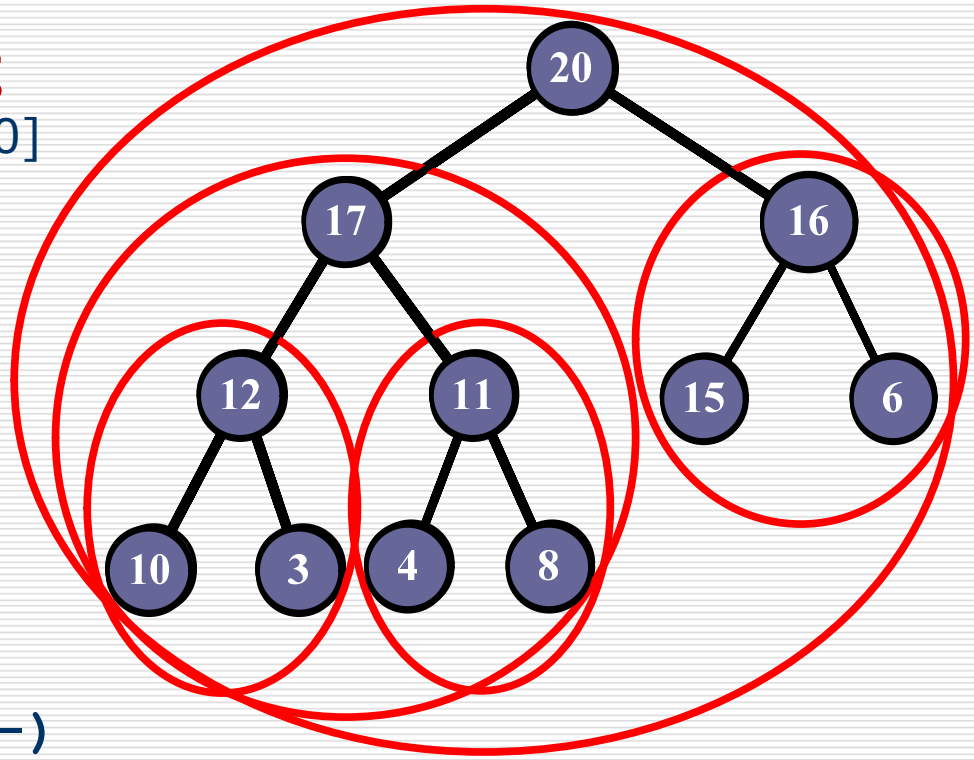


- Χρόνος για insert() : O(ύψος) = O(log n)
- Αύξηση προτεραιότητας : εισαγωγή (αποκατ. προς-τα-πάνω).
  Μείωση προτεραιότητας : διαγραφή (αποκατ. προς-τα-κάτω).

Δημιουργία Σωρού

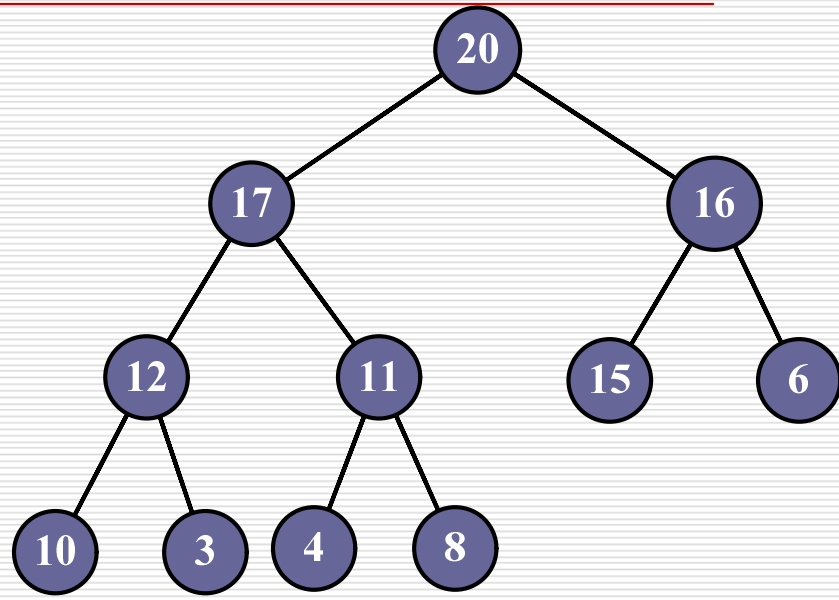
- A[n] → σωρός με *n* εισαγωγές
  [3, 4, 6, 10, 8, 15, 16, 17, 12, 11, 20]
- Χρόνος O(*n*log*n*) .
- **Ιεραρχικά** (bottom-up):
  Υποδέντρα-σωροί ενώνονται σε δέντρο-σωρό.

```
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(ύψος *i*).

  n / 4 στοιχεία χρόνος 1×*c*

  n / 8 στοιχεία χρόνος 2×*c*

  . . . . . .

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

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

- Χρόνος constructHeap( ) = Θ(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: $O(n \log n)$.