Ταξινόμηση Σωρού

Ορισμός Σωρού

- ***(Δυαδικός) σωρός:*** συστοιχία η οποία μπορεί να θεωρηθεί ως σχεδόν πλήρες δυαδικό δένδρο
- Κάθε κόμβος του δένδρου: ένα στοιχείο της συστοιχίας.
- Το δένδρο είναι συμπληρωμένο σε όλα τα επίπεδα, εκτός ίσως από το χαμηλότερο, το οποίο είναι συμπληρωμένο από τα αριστερά μέχρι κάποιου σημείου.
- Μια συστοιχία Α που αντιπροσωπεύει έναν σωρό είναι ένα αντικείμενο με δύο πεδία:
  - το Α.*μήκος*, το πλήθος των στοιχείων της συστοιχίας,
  - το Α.*πλήθος-σωρού*, το πλήθος των στοιχείων του σωρού αποθηκευμένα εντός της συστοιχίας Α.

Ορισμός Σωρού

- Η ρίζα του δένδρου είναι το A[1]
- Ο κόμβος *i* είναι ο A[*i*]
- Ο πατέρας του κόμβου *i* είναι ο A[*i*/2] (Σημείωση: ακέραια διαίρεση)
- Το αριστερό παιδί του κόμβου *i* είναι ο A[2*i*]
- Το δεξί παιδί του κόμβου *i* είναι ο A[2*i* + 1]

Ορισμός Σωρού

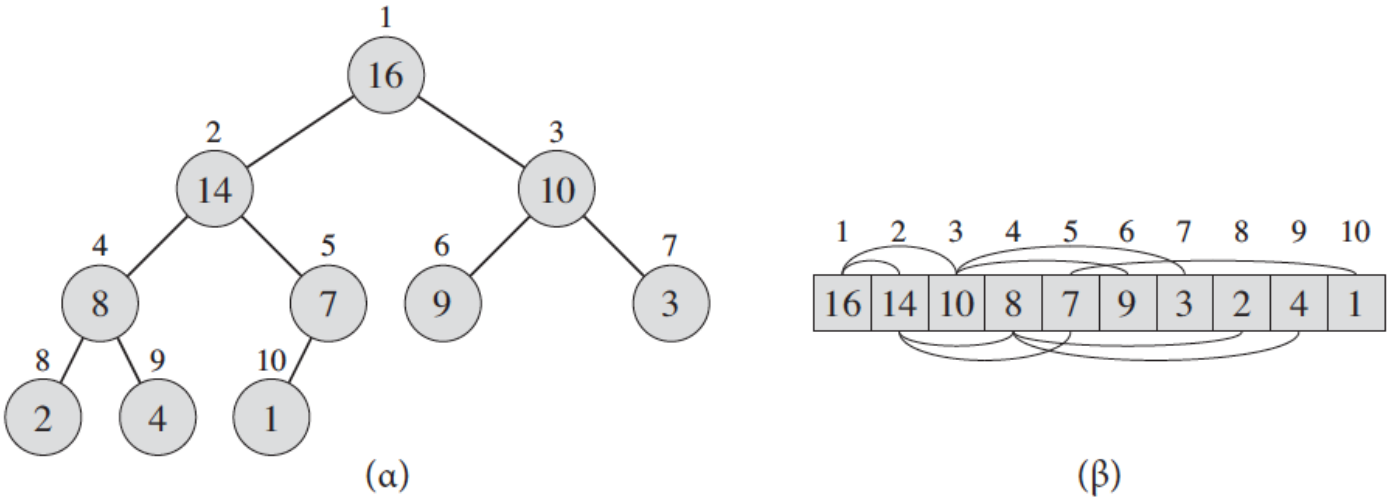
Έτσι

```
Parent(i) { return ⌊i/2⌋; }
Left(i) { return 2*i; }
right(i) { return 2*i + 1; }
```

Οι σωροί ικανοποιούν την *ιδιότητα σωρού*:

$A[Parent(i)] \geq A[i]$ για όλους τους κόμβους $i > 1$

- *Το μεγαλύτερο στοιχείο είναι αποθηκευμένο στη ρίζα του σωρού.*



Σχήμα 6.1 Ένας σωρός μεγίστου θεωρούμενος ως (α) δυαδικό δένδρο και (β) συστοιχία. Ο αριθμός μέσα στον κύκλο σε κάθε κόμβο του δένδρου δηλώνει την τιμή που είναι αποθηκευμένη στον κόμβο. Ο αριθμός επάνω από τον κάθε κόμβο είναι ο αντίστοιχος αύξων αριθμός στη συστοιχία. Οι γραμμές στο επάνω και το κάτω μέρος της συστοιχίας υποδεικνύουν σχέσεις πατρικού-θυγατρικού· οι πατρικοί κόμβοι βρίσκονται πάντοτε αριστερά από τους θυγατρικούς τους. Το δένδρο έχει ύψος τρία· ο κόμβος στη θέση υπ' αριθμ. 4 (με τιμή 8) έχει ύψος ένα.

Ύψος Σωρού

- Ορισμοί:
  - Το *ύψος* ενός κόμβου στο δένδρο = το πλήθος των ακμών στη μακρύτερη απλή καθοδική διαδρομή από τον κόμβο μέχρι κάποιο φύλλο
  - Το ύψος του δένδρου = το ύψος της ρίζας του
- Ο Σωρός n στοιχείων βασίζεται σε ένα πλήρες δυαδικό δένδρο:
  - το ύψος του είναι Θ(lg n)
  - Οι βασικές λειτουργίες σωρού απαιτούν το πολύ χρόνο ανάλογο του ύψους του σωρού.

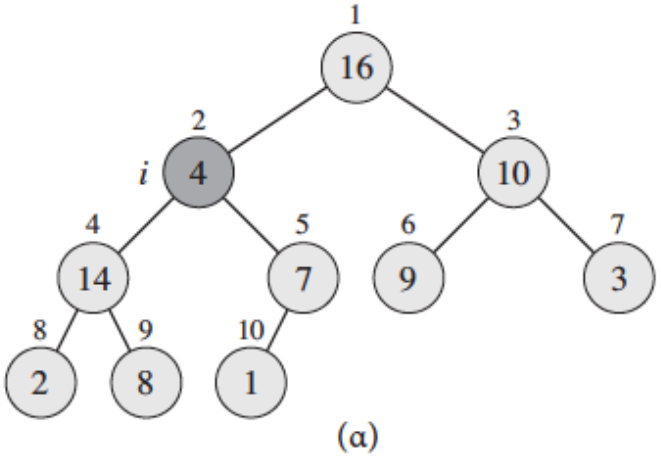
Λειτουργίες Σωρού

- **`Heapify()`:** Διατηρεί την ιδιότητα του σωρού
  - Είσοδος: ένας κόμβος *i* στο σωρό με παιδιά *l* και *r*
  - Τα δύο υποδένδρα με ρίζες τα *l* και *r* είναι σωροί
  - Το υποδένδρο με ρίζα το *i* μπορεί να παραβιάζει την ιδιότητα σωρού
- Ενέργεια: «μετακυλύουμε» την τιμή του A[i] προς τα κάτω στον σωρό μεγίστου έτσι ώστε το υπόδενδρο με ρίζα το στοιχείο στη θέση i να ικανοποιεί την ιδιότητα του σωρού μεγίστου.

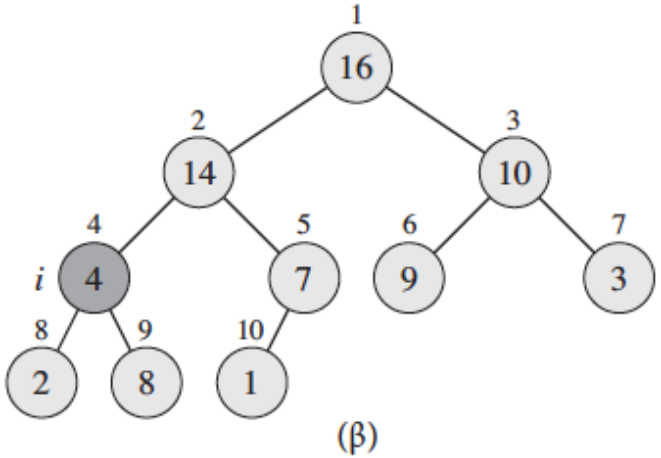
Λειτουργίες Σωρού

```
Heapify(A, i)
{
   l = Left(i); r = Right(i);
   if (l <= heap_size(A) && A[l] > A[i])
       largest = l;
   else
       largest = i;
   if (r <= heap_size(A) && A[r] > A[largest])
       largest = r;
   if (largest != i)
       Swap(A, i, largest);
       Heapify(A, largest);
}
```

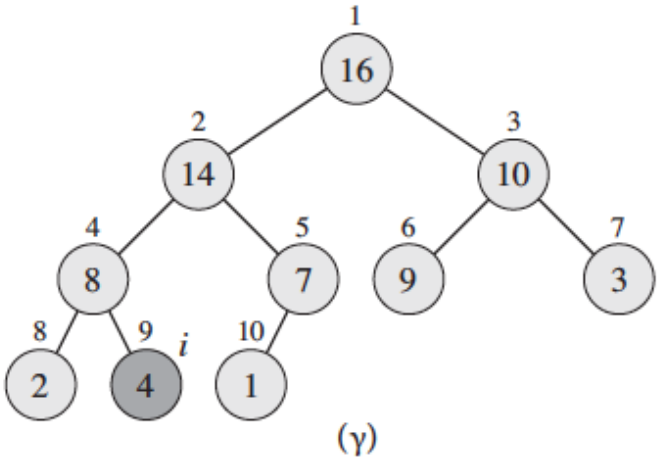
Λειτουργίες Σωρού – Αποκατάσταση Σωρού



(α)



(β)



(γ)

Ανάλυση της Heapify

- Η σύγκριση των *i*, *l*, και *r* απαιτεί χρόνο $\Theta(1)$
- Αν ο σωρός στο i έχει n στοιχεία, το μέγιστο πλήθος των στοιχείων των υποδένδρων στο l ή r είναι $2n/3$ (χειρότερη περίπτωση: η τελευταία γραμμή είναι 1/2 - γεμάτη)
- Ο χρόνος του **`Heapify()`**:

  $T(n) \leq T(2n/3) + \Theta(1)$

  $T(n) = O(\lg n)$

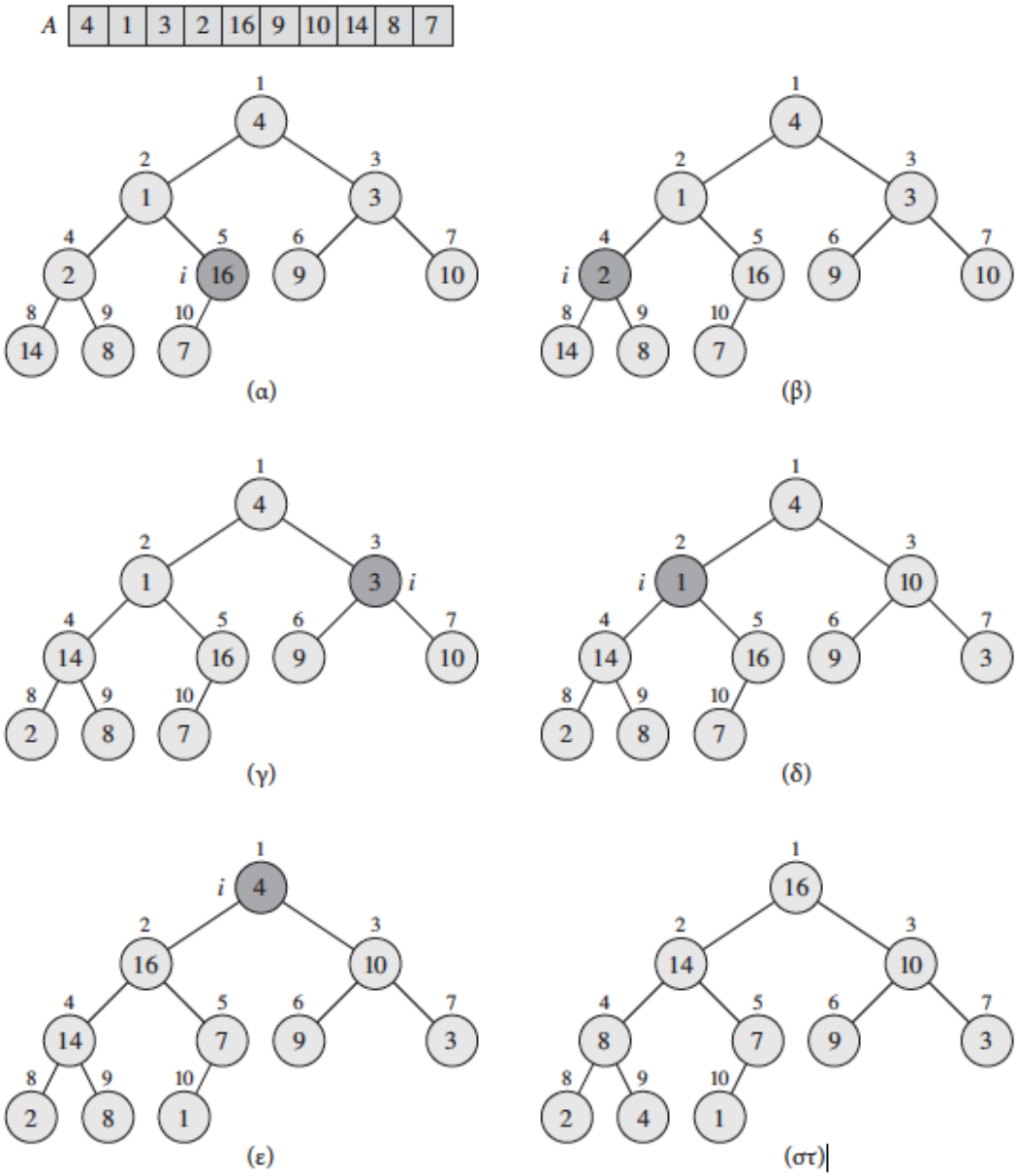
Κατασκευή Σωρού

- Μπορούμε να χτίσουμε ένα σωρό από κάτω προς τα πάνω εκτελώντας τη **`Heapify()`** σε διαδοχικές υποσυστοιχίες
  - Για κάθε συστοιχία μήκους *n*, όλα τα στοιχεία στην υποσυστοιχία $A[\lfloor n/2 \rfloor + 1 .. n]$ είναι σωροί
    Έτσι:
    - Κινούμαστε προς τα πίσω στη συστοιχία από το στοιχείο $\lfloor n/2 \rfloor$ στο 1, καλώντας τη **`Heapify()`** σε κάθε κόμβο.
    - Η σειρά επεξεργασίας εγγυάται ότι τα παιδιά του κόμβου *i* είναι σωροί όταν επισκεπτόμαστε το κόμβο *i.*

Κατασκευή Σωρού

```
Είσοδος: μίας μη ταξινομημένης
  συστοιχίας A,
Έξοδος: A ως σωρός
BuildHeap(A)
{
  heap_size(A) = length(A);
  for (i = ⌊length[A]/2⌋ downto 1)
     Heapify(A, i);
}
```

Κατασκευή Σωρού



Ανάλυση Κατασκευής Σωρού

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right)$$

$$\begin{aligned} \sum_{h=0}^{\infty} \frac{h}{2^h} &= \frac{1/2}{(1-1/2)^2} \\ &= 2 . \end{aligned}$$

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2}$$

Για |x|<1

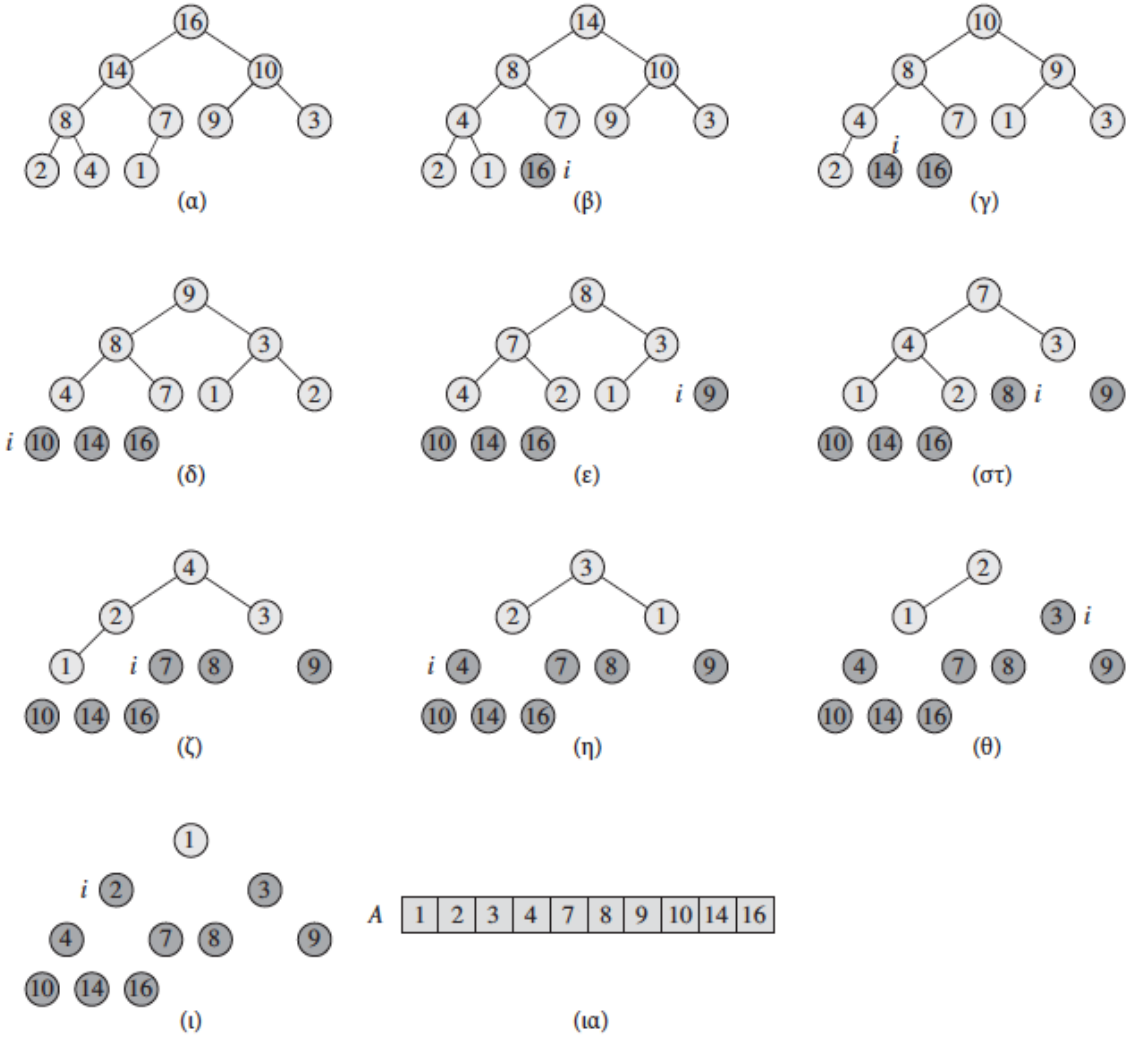
$$\begin{aligned} O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) &= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) \\ &= O(n) . \end{aligned}$$

Ο αλγόριθμος της ταξινόμησης σωρού

- Δεδομένης της υπορουτίνας **`BuildHeap()`**, ένας αλγόριθμος ταξινόμησης μπορεί εύκολα να προκύψει:
  - Το μέγιστο στοιχείο είναι στο A[1]
  - Το αφαιρούμε με εναλλαγή με το στοιχείο A[n]
    - Μείωση του heap_size[A] κατά ένα
    - A[n] περιέχει την τελική σωστή τιμή
  - Αποκατέστησε την ιδιότητα του σωρού στο A[1] καλώντας τη **`Heapify()`**
  - Επανέλαβε, εναλλάσσοντας πάντα το A[1] με το A[heap_size(A)]

Ταξινόμηση Σωρού

```
Heapsort(A)
{
     BuildHeap(A);
     for (i = length(A) downto 2)
     {
          Swap(A[1], A[i]);
          heap_size(A) -= 1;
          Heapify(A, 1);
     }
}
```



Σχήμα 6.4 Η λειτουργία της ΤΑΞΙΝΟΜΗΣΗΣ ΣΩΡΟΥ. (α) Η δομή δεδομένων του σωρού μεγίστου αμέσως μετά την κατασκευή της μέσω της ΚΑΤΑΣΚΕΥΗΣ ΣΩΡΟΥ ΜΕΓΙΣΤΟΥ στη γραμμή 1. (β)–(ι) Οι διαδοχικοί σωροί μεγίστου αμέσως μετά την κάθε κλήση της ΑΠΟΚΑΤΑΣΤΑΣΗΣ ΣΩΡΟΥ ΜΕΓΙΣΤΟΥ στη γραμμή 5. Στο κάθε σχήμα υποδεικνύεται επίσης η τιμή του i τη δεδομένη χρονική στιγμή. Μόνο οι ελαφρά σκιασμένοι κόμβοι εξακολουθούν να ανήκουν στον σωρό. (ια) Η τελική ταξινομημένη συστοιχία A.

Ανάλυση κατασκευής σωρού

- Η κλήση της **`BuildHeap()`** παίρνει χρόνο $O(n)$
- Κάθε μία από τις $n - 1$ κλάσεις στη **`Heapify()`** παίρνει χρόνο $O(\lg n)$
- Έτσι ο συνολικός χρόνος που παίρνει η **`HeapSort()`**
  $= O(n) + (n - 1)\, O(\lg n)$
  $= O(n) + O(n \lg n)$
  $= O(n \lg n)$