Σωροί

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Ουρές Προτεραιότητας

Σωροί – υλοποίηση και πράξεις

Ο αλγόριθμος ταξινόμησης HeapSort

Παραλλαγές Σωρών

Ουρά προτεραιότητας

- Η δομή δεδομένων *ουρά* υποστηρίζει FIFO (first in first out) στρατηγική για εισαγωγές και διαγραφές στοιχείων.
- Σε διάφορες εφαρμογές, όμως, υπάρχει η ανάγκη επιλογής στοιχείων από κάποιο σύνολο σύμφωνα με κάποια σειρά προτεραιότητας (π.χ. σε λειτουργικά συστήματα).
- Σε ουρές προτεραιότητας κύρια σημασία έχει η προτεραιότητα του κάθε στοιχείου, πρώτο βγαίνει πάντα το στοιχείο με τη μεγαλύτερη προτεραιότητα.
- Ουρά Προτεραιότητας

  Ο ΑΤΔ *ουρά προτεραιότητας* ορίζεται ως μια ακολουθία στοιχείων συνοδευόμενη από τις πράξεις
  - Delete_Min*, και
  - Insert.

*Θεωρούμε ότι το μικρότερο κλειδί έχει τη μεγαλύτερη προτεραιότητα

Ουρά προτεραιότητας

Πιθανές υλοποιήσεις:

1. *συνδεδεμένη λίστα*

   Insert: O(1),   Delete_Min: O(n)

2. *ταξινομημένη συνδεδεμένη λίστα*

   Insert: O(n),   Delete_Min: O(1)

3. *δυαδικό δένδρο αναζήτησης*

   Insert, Delete_Min: O(log n)

- Υπάρχει καλύτερη υλοποίηση;

  Ναι, μια ενδιαφέρουσα τάξη δυαδικών δένδρων, *οι σωροί*.

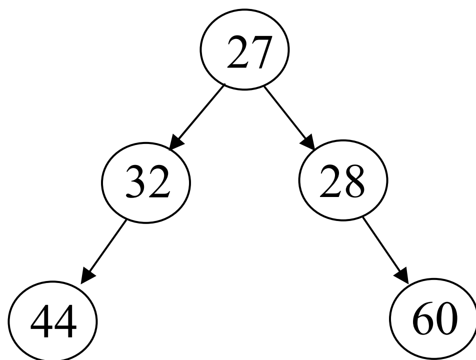
Σωρός

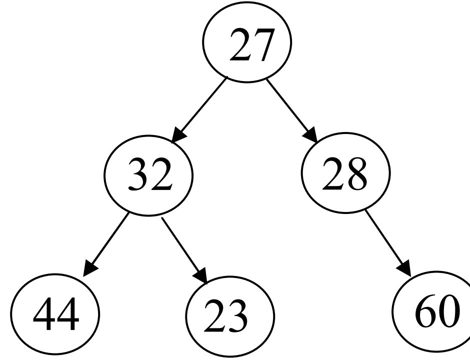
- Σωρός ελαχίστων (MinHeap) είναι ένα δυαδικό δένδρο που ικανοποιεί:
  - *δομική ιδιότητα*: είναι πλήρες
  - *ιδιότητα σειράς*: το κλειδί ενός κόμβου είναι μικρότερο από τα κλειδιά των παιδιών του

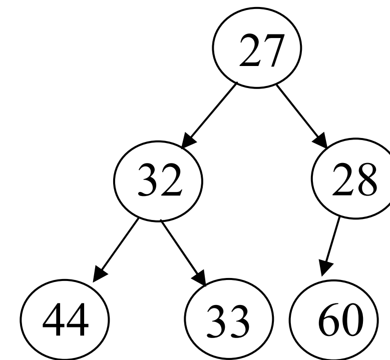
- Σε κάθε υπόδενδρο, το μικρότερο στοιχείο βρίσκεται στη ρίζα.

- Δεν υπάρχει καμιά σχέση μεταξύ κλειδιών αδελφών κόμβων.

- Ποια από τα πιο κάτω δένδρα είναι σωροί;







Πλήρη Δυαδικά Δένδρα

- Σε ένα πλήρες δυαδικό δένδρο, στο επίπεδο k υπάρχουν το πολύ 2^{k-1} κόμβοι.
- Σε ένα πλήρες δυαδικό δένδρο ύψους h όλα τα επίπεδα μέχρι το h-οστό είναι εντελώς γεμάτα, και το επίπεδο h+1 είναι γεμάτο από τα αριστερά στα δεξιά.
- Ο αριθμός των κόμβων μέχρι το επίπεδο h δίνεται από το άθροισμα

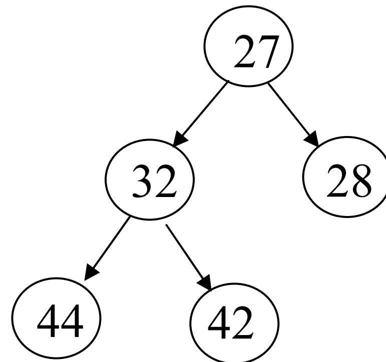
$$\sum_{i=1}^{h} 2^{i-1} = 2^{h} - 1$$

- Επομένως, ένα πλήρες δένδρο ύψους h έχει μεταξύ 2^h και $2^{h+1} - 1$ κόμβους.
- Ένα πλήρες δένδρο με n κόμβους έχει ύψος O(log n).

Υλοποίηση με πίνακες

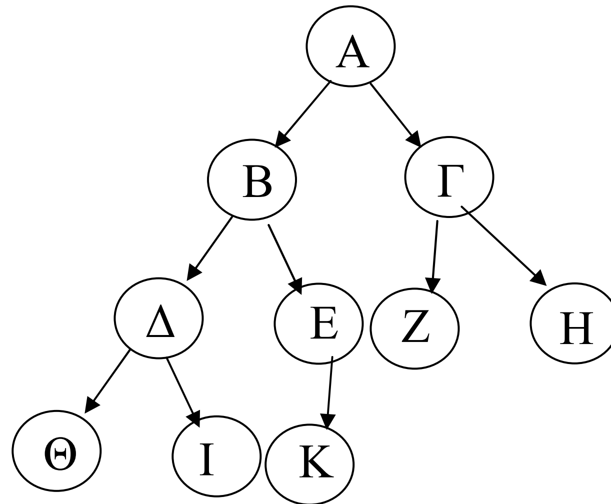
- Ένα πλήρες δυαδικό δένδρο μπορεί να αποθηκευτεί σε πίνακα ως εξής:
  - στη θέση 1 βάζουμε το στοιχείο της ρίζας
  - αν κάποιος κόμβος u βρίσκεται στη θέση i, τότε τοποθετούμε το αριστερό του παιδί στη θέση 2i, και το δεξιό του παιδί στη θέση 2i +1.
- Ο πατέρας ενός κόμβου στη θέση i (εκτός από τη ρίζα) βρίσκεται στη θέση $\lfloor i/2 \rfloor$.
- Πλεονέκτημα: Δεν χρειάζονται δείκτες, έτσι εξοικονομούμε μνήμη και έχουμε πιο απλές διαδικασίες.
- Μειονέκτημα: πρέπει να γνωρίζουμε από την αρχή το μέγιστο μέγεθος του σωρού.

Παράδειγμα αναπαράστασης σωρού (1)



Θέση	0	1	2	3	4	5	6
Στοιχείο		27	32	28	44	42	

Παράδειγμα αναπαράστασης σωρού (2)



Θέση	0	1	2	3	4	5	6	7	8	9	10
Στοιχείο		Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ

Υλοποίηση Σωρού

- Ένας σωρός μπορεί να υλοποιηθεί ως μια εγγραφή `heap` με τρία πεδία
  1. `size`, τύπου int, όπου αποθηκεύεται το μέγεθος του σωρού.
  2. `maxSize`, τύπου int, που δηλώνει το μέγεθος του πίνακα, και
  3. `contents`, τύπου πίνακα, όπου αποθηκεύουμε τα στοιχεία του σωρού.

- Αυτή η δομή θα πρέπει να υποστηρίζει τις πράξεις: MakeEmpty, Insert, DeleteMin, IsEmpty, IsFull.

Εισαγωγή κόμβου

- Σε ένα πλήρες δυαδικό δένδρο υπάρχει μία μόνο θέση όπου μπορεί να εισαχθεί κόμβος και η εισαγωγή να διατηρήσει το δένδρο πλήρες.

- *Αυτή η θέση είναι η πιο δεξιά στο τελευταίο επίπεδο του δένδρου, και αντιστοιχεί στη θέση size+1 του πίνακα.*

- Για να εισάγουμε ένα κλειδί k σε ένα σωρό σκεφτόμαστε ως εξής:

  Πιθανόν το k να μην μπορεί να μπει στην κενή θέση size+1, γιατί μια τέτοια εισαγωγή να παραβιάζει τη δεύτερη ιδιότητα του σωρού.

  Έστω ότι η κενή θέση είναι η x, ο πατέρας αυτής της θέσης είναι ο u, και k´ είναι το κλειδί του u. Τότε εφαρμόζουμε τα εξής:

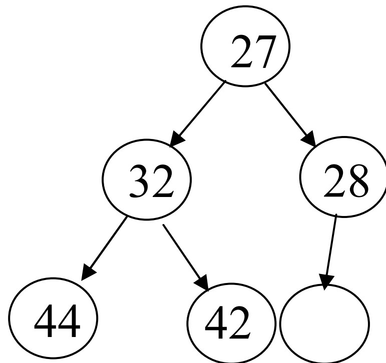
  1. *αν k>k´, ή, η θέση x αντιστοιχεί στη ρίζα,* τότε `contents[x] = k`
  2. αν k<k´, τότε βάλε το k´ στη θέση x, και ανάλαβε να γεμίσεις τη θέση u, δηλαδή `contents[x] =k´; x=u;` και
  3. επανάλαβε τη διαδικασία.

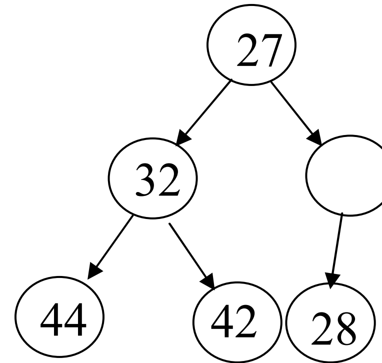
- Αυτή η διαδικασία σύγκρισης με τον πατρικό κόμβο και αναρρίχησης μπορεί να συνεχιστεί μέχρι τη ρίζα του δένδρου.

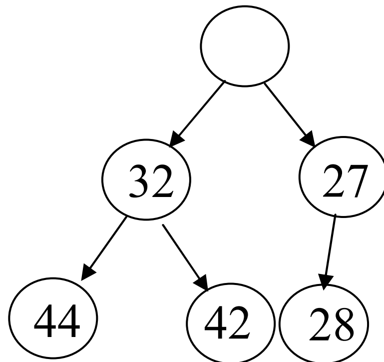
Διαδικασία Εισαγωγής

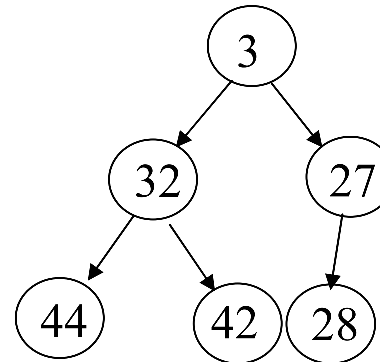
```
Insert(int k, heap E){

 check heap.size < heap.maxsize;
 int x = heap.size + 1;
 while(x>1 && contents[⌊x/2⌋]>k){
      contents[x] = contents[⌊x/2⌋];
      x = ⌊x/2⌋;
 }
 contents[x] = k;
}
```

Παράδειγμα: Εισαγωγή του 3









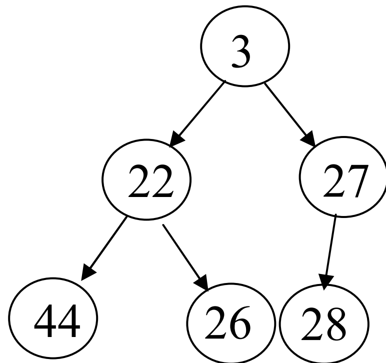
Διαγραφή του ελάχιστου στοιχείου

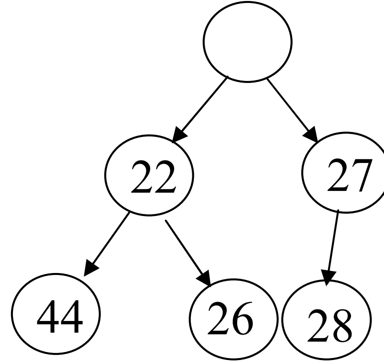
- Το ελάχιστο στοιχείο βρίσκεται πάντοτε στην κορυφή και η διαγραφή του προκαλεί μια κενή θέση στη ρίζα.
- Θα πρέπει να κατεβάσουμε αυτή την κενή θέση προς τα κάτω και δεξιά.
- Σε κάθε βήμα ελέγχουμε τα παιδιά της εκάστοτε κενής θέσης. Έστω ότι x είναι η κενή θέση,
  1. Αν το κλειδί που βρίσκεται στην τελευταία θέση του σωρού είναι μικρότερο από τα κλειδιά των παιδιών του x τότε μεταφέρουμε το κλειδί αυτό στην κενή θέση και μειώνουμε το μέγεθος του σωρού
     ```
     contents [x] = contents [size];  size--
     ```
     και τερματίζουμε τη διαδικασία.
  2. Διαφορετικά, διαλέγουμε το παιδί u του x το οποίο έχει το μικρότερο κλειδί, μεταφέρουμε το κλειδί του u στο x και κάνουμε κενή θέση τη u:
     ```
     contents[x] = u;    x = u
     ```
  3. και επαναλαμβάνουμε τη διαδικασία.

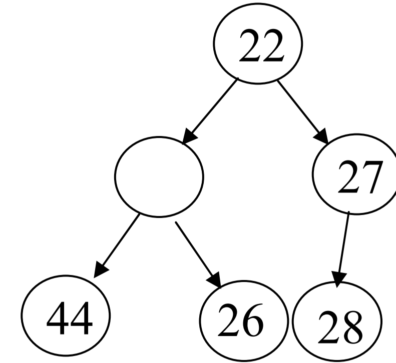
Διαδικασία Διαγραφής ελάχιστου στοιχείου

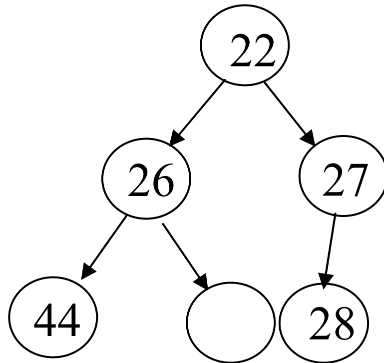
```
int Delete_Min(heap E)

 if IsEmpty(E) report error; return;

 min = elements[1];
 last = elements[size];

 size=size-1; x=1;

 while(x*2 <= size){
   child = x*2;
   if (child!= size && elements[child+1]<elements[child])
  child++; }
   if (last>elements[child])
       elements[x]=elements[child];
       x=child;
    else break;
 }
 elements[x] = last;
 return min;
```

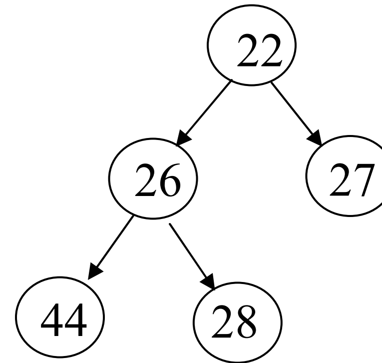
Παράδειγμα: Διαγραφή του 3











Μερικά Σχόλια

- Ο χρόνος εκτέλεσης των διαδικασιών Insert και Delete_min είναι της τάξης O(h) δηλαδή O(log n). (h: ύψος, n: αριθμός κόμβων)
- Ποιο είναι το όφελος της δομής σε σύγκριση με δυαδικά δένδρα αναζήτησης;
- Οι σωροί χρησιμοποιούνται ευρέως σε λειτουργικά συστήματα, συστήματα όπου γίνεται διαμερισμός του χρόνου του υπολογιστή σε > 1 εργασίες (task scheduling) και σε μεταγλωττιστές.
- Συμμετρικά, μπορούμε να ορίσουμε τη δομή maxHeap, όπου η ρίζα περιέχει το μέγιστο στοιχείο.
- Εκτός από δυαδικούς σωρούς, μπορούμε να ορίσουμε τους δ-σωρούς (d-heaps), όπου κάθε κόμβος έχει d παιδιά.

Διαδικασία Καθόδου

- Έστω ένας πίνακας A[1..n] και μια τιμή i, θα ορίσουμε διαδικασία `PercoladeDown(i)`, η οποία μετακινεί το στοιχείο A[i] μέσα στον σωρό προς τα κάτω όσο χρειάζεται.
- Έστω ότι A[i] = k.
- Θεωρούμε πως η i είναι άδεια θέση.
- Αν η άδεια θέση έχει παιδί που περιέχει στοιχείο μικρότερο του k και x είναι το μικρότερο τέτοιο παιδί, τότε μετακινούμε το στοιχείο του x στην κενή θέση και μετακινούμε την κενή θέση στο x.
- Επαναλαμβάνουμε την ίδια διαδικασία μέχρι τη στιγμή που η κενή θέση δεν έχει παιδιά με στοιχεία μικρότερα του k. Τότε αποθηκεύουμε το k στην θέση αυτή.
- Ο χρόνος εκτέλεσης είναι ανάλογος του ύψους του κόμβου που αντιστοιχεί στη θέση i του σωρού. Δηλαδή, στη χείριστη περίπτωση, όπου i=n, O(lg n).

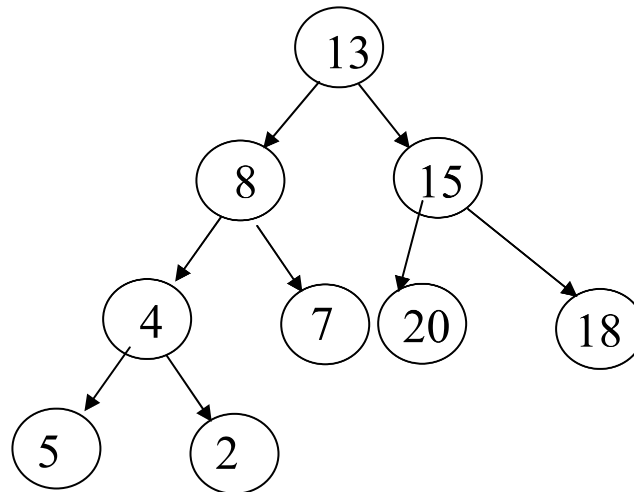
Διαδικασία Καθόδου

Μη αναδρομική διαδικασία PercolateDown

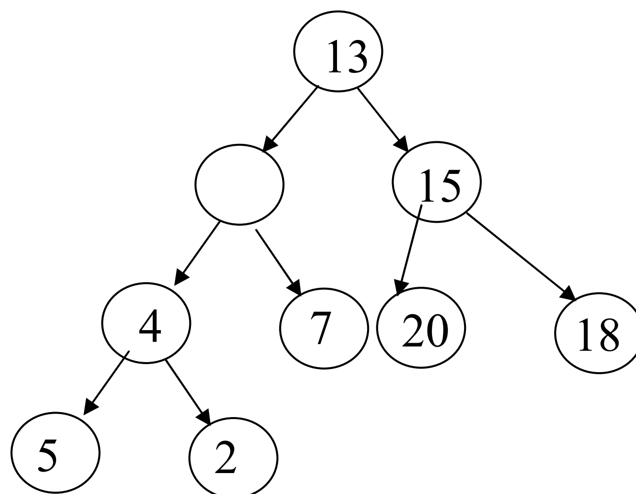
```
PercolateDown(int A[],int n,int i){
   int k = A[i];
   while (2*i <= n) {
       j = 2*i;
       if (j<n && A[j+1]<A[j]) j++;
       if (k > A[j])
           A[i] = A[j]; i=j;
       else
           break;
       }
       A[i] = k;
   }
```

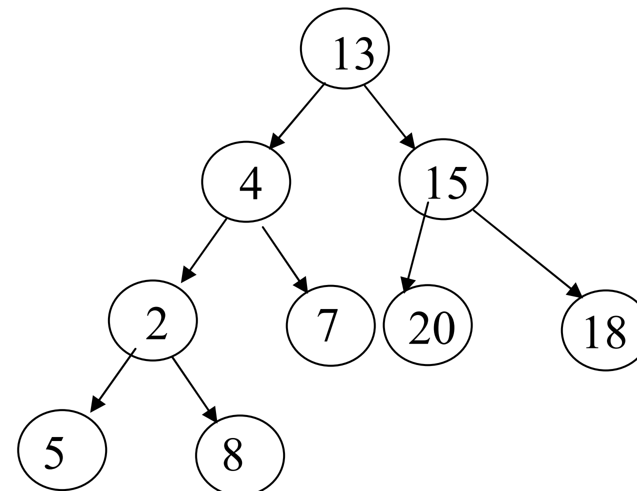
Παράδειγμα PercoladeDown

- Δεδομένα Εισόδου: i = 2, n = 9, A = [- , 13, 8, 15, 4, 7, 20, 18, 5, 2]



- k=8





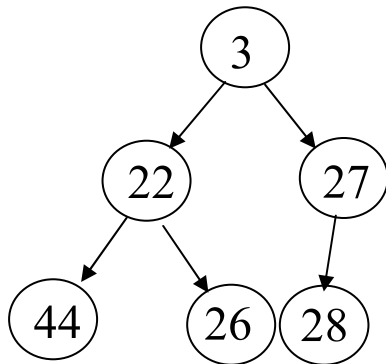
Διαδικασία DeleteMin (2)

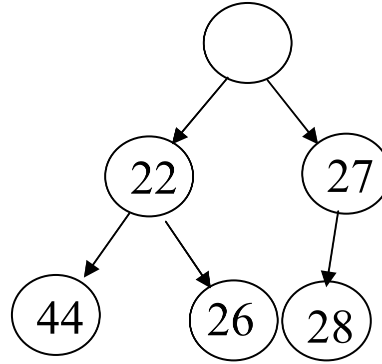
- Αφαιρούμε το στοιχείο της ρίζας (είναι το μικρότερο κλειδί του σωρού).
- Μεταφέρουμε το τελευταίο κλειδί στη ρίζα, και εφαρμόζουμε τη διαδικασία PercoladeDown(A, n, 1):

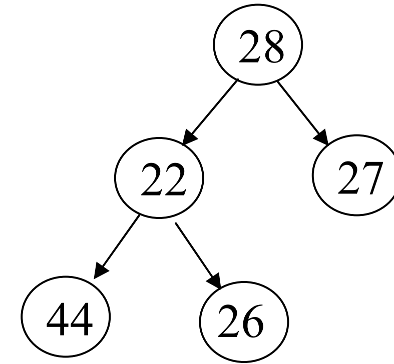
```
int DeleteMin2(heap A){
    check that size > 0;
    return (contents[1]);
    swap(contents[1],contents[size]);
    size = size - 1;
    PercoladeDown(contents, size, 1);
}
```

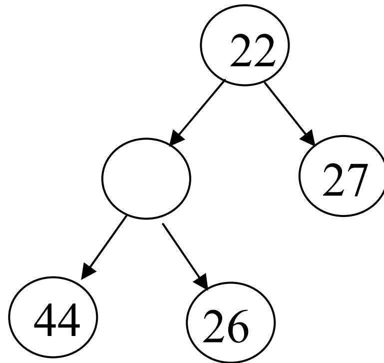
- Χρόνος Εκτέλεσης: O(h) = O(log n)

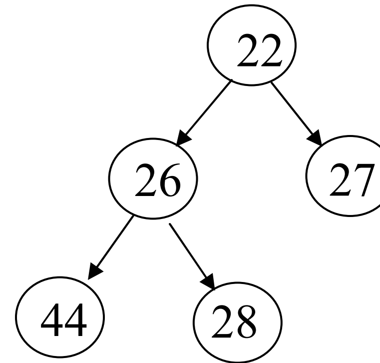
Παράδειγμα: Διαγραφή του 3





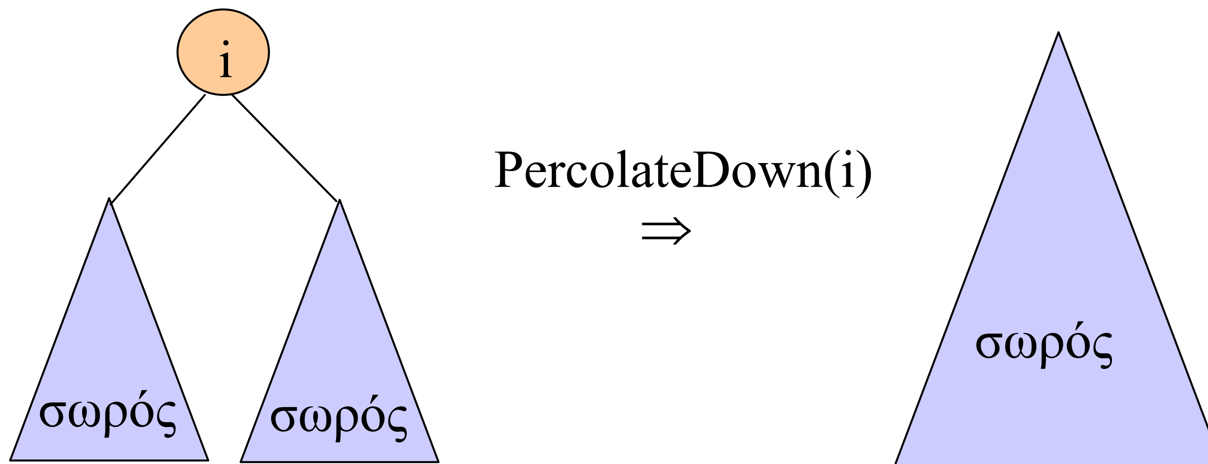






Από πίνακες σε σωρούς

- Έστω πίνακας A[1..n].
- Μπορούμε να θεωρήσουμε τον πίνακα ως ένα πλήρες δυαδικό δένδρο με n κόμβους.
- Αν για μια τιμή i το αριστερό και το δεξί υπόδενδρο του i ικανοποιούν τις ιδιότητες ενός σωρού, τότε, αν καλέσουμε τη διαδικασία PercolateDown(A, n, i) θα έχουμε σαν αποτέλεσμα το υπόδενδρο που ριζώνει στη θέση i να ικανοποιεί τις ιδιότητες ενός σωρού.



Κτίσιμο σωρού από ένα πίνακα

- Μπορούμε να μετατρέψουμε ένα πίνακα A[1..n] σε ένα σωρό με διαδοχική εφαρμογή της διαδικασίας PercoladeDown() από κάτω προς τα πάνω.
- Παρατήρηση: οι θέσεις > n/2 αντιστοιχούν σε φύλλα.
- Διαδικασία BuildHeap:

```
BuildHeap( int A[], int n) {
   for (i=n/2; i>0; i--)
    PercoladeDown(A,n,i);
}
```

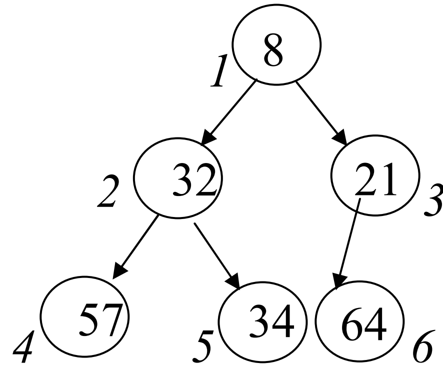
- *Ορθότητα.* Αποδεικνύεται με τη μέθοδο της επαγωγής: μετά από την εφαρμογή της διαδικασίας PercoladeDown(A,n,i), τα υπόδενδρα που ριζώνουν στις θέσεις i, ..., n, ικανοποιούν τις ιδιότητες του σωρού.
- *Ανάλυση του Χρόνου Εκτέλεσης.* Ο ολικός χρόνος εκτέλεσης είναι ανάλογος του αθροίσματος των υψών όλων των εσωτερικών κόμβων, το οποίο είναι O(n).

Τι κάνει ο πιο κάτω αλγόριθμος;

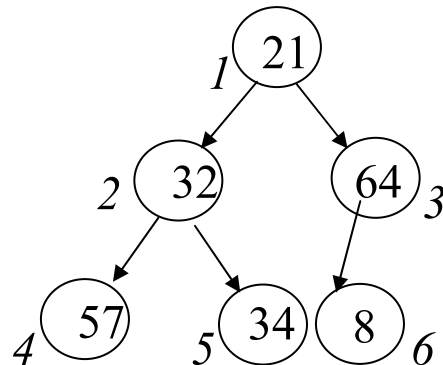
```
mystery (int A[], int n) {
 BuildHeap(A, n);
 for (i=n ; i>1; i--){
   swap (A[1], A[i]);
   PercoladeDown(A, i-1, 1);
  }
}
```

Παράδειγμα Εκτέλεσης

- Δεδομένα Εισόδου: A = [- , 34, 8, 64, 57, 32, 21]

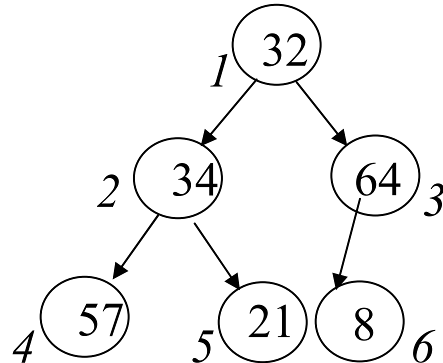


- Μετά από την πρώτη επανάληψη του for-loop: [21, 32, 64, 57, 34, 8]

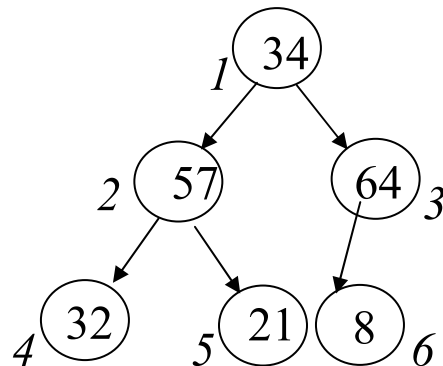


Παράδειγμα Εκτέλεσης

- Μετά από τη δεύτερη επανάληψη του for-loop: [32, 34, 64, 57, 21, 8]

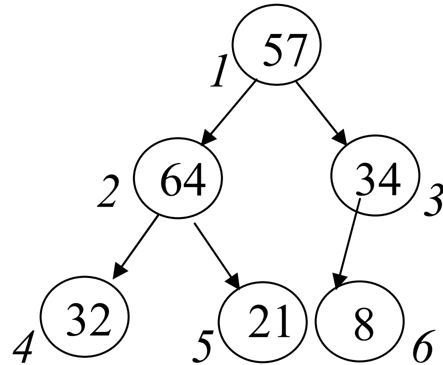


- Μετά από την τρίτη επανάληψη του for-loop: [34, 57, 64, 32, 21, 8]

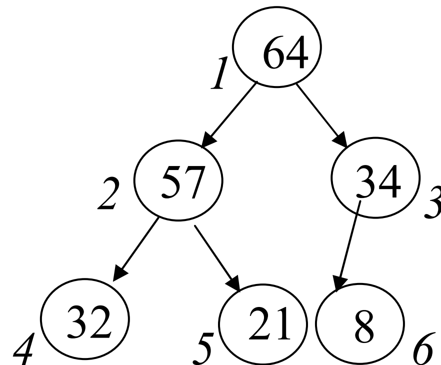


Παράδειγμα Εκτέλεσης

- Μετά από την τέταρτη επανάληψη του for-loop: [57, 64, 34, 32, 21, 8]



- Μετά από την πέμπτη επανάληψη του for-loop: [64, 57, 34, 32, 21, 8]



Δεδομένο Εξόδου

HeapSort

- Η διαδικασία mystery ταξινομεί ένα πίνακα σε φθίνουσα σειρά.
- Αρχικά δημιουργεί ένα σωρό σε χρόνο O(n).
- Στη συνέχεια επαναλαμβάνει το εξής: αφαιρεί το μικρότερο στοιχείο (της ρίζας του σωρού) και το μετακινεί στο τέλος (εκτελεί την PercolateDown). Κάθε εκτέλεση της PercoladeDown χρειάζεται χρόνο της τάξης O(log n).
- Ολικός Χρόνος Εκτέλεσης: O(n·log n)
- Ο αλγόριθμος ονομάζεται Heapsort.
- Μπορούμε εύκολα να αλλάξουμε τον κώδικα ώστε να επιστρέφεται η λίστα σε αύξουσα σειρά.

Άλλες διαδικασίες σε σωρούς

- Παρόλο που εύρεση του ελάχιστου κλειδιού σε ένα σωρό μπορεί να πραγματοποιηθεί σε σταθερό χρόνο, η εύρεση τυχαίου στοιχείου στη χειρότερη περίπτωση επιβάλλει διερεύνηση ολόκληρης της δομής (δηλαδή, είναι της τάξης O(n)).
- Αν όμως γνωρίζουμε τη θέση στοιχείων με κάποιο άλλο τρόπο, διαδικασίες σε σωρούς πραγματοποιούνται εύκολα, π.χ. οι πιο κάτω εκτελούνται σε χρόνο λογαριθμικό.
- `Increase_Key(P,Δ),` αυξάνει την προτεραιότητα του κλειδιού P, κατά Δ. Χρησιμοποιείται από χειριστές λειτουργικών συστημάτων για αύξηση της προτεραιότητας σημαντικών διεργασιών. Η συμμετρική διαδικασία `Decrease_Key(P,Δ)` συχνά εκτελείται αυτόματα σε λειτουργικά συστήματα σε περίπτωση που κάποια δουλειά χρησιμοποιεί υπερβολικά μεγάλη ποσότητα χρόνου του CPU.
- `Remove(I),` αφαιρεί τον κόμβο της θέσης I (χρήσιμη σε περίπτωση τερματισμού διαδικασίας).

Παραλλαγές Σωρών

Leftist Heaps

- Ένα δυαδικό δένδρο T είναι leftist heap, αν για κάθε κόμβο u του T

  nullpath(u.left) ≥ nullpath(u.right)

  όπου nullpath(v) είναι η μικρότερη απόσταση του κόμβου v από κόμβο που έχει το πολύ ένα παιδί.

- Ο ορισμός αυτός επιτρέπει μη-ισοζυγισμένα δένδρα που είναι πιο βαθιά προς τα αριστερά.

- Ένα leftist heap T ικανοποιεί την πιο κάτω σημαντική ιδιότητα: αν το T έχει n κόμβους τότε το δεξί του μονοπάτι έχει το πολύ $\lfloor \log(n+1) \rfloor$ κόμβους.

- Το πλεονέκτημα αυτής της δομής είναι ότι επιτρέπει το συνδυασμό δύο σωρών σε ένα (διαδικασία Merge) σε χρόνο λογαριθμικό.

- Οι υπόλοιπες διαδικασίες (DeleteMin, Insert) επίσης πραγματοποιούνται σε λογαριθμικό χρόνο.

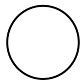
Παραλλαγές Σωρών

Διωνυμικές Ουρές Προτεραιότητας (Binomial Queues)

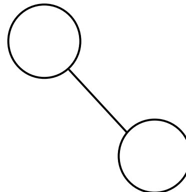
- Η δομή Binomial queue είναι ένα δάσος που αποτελείται από ένα αριθμό δένδρων τα οποία ονομάζονται διωνυμικά δένδρα (binomial trees).

- Ένα διωνυμικό δένδρο ύψους 0 έχει ένα κόμβο. Ένα διωνυμικό δένδρο ύψους k, B_k, κτίζεται από ένα διωνυμικό δένδρο ύψους k-1, με την εισαγωγή στη ρίζα του δένδρου ενός διωνυμικού υποδένδρου ύψους k-1.

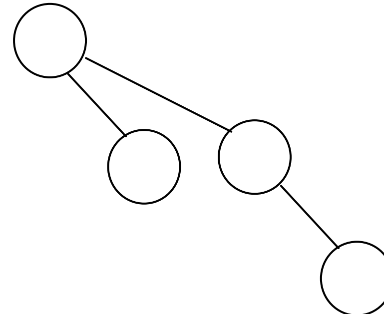
B_0



B_1



B_2



Παραλλαγές Σωρών

- Μια διωνυμική ουρά είναι ένα δάσος από διωνυμικά δένδρα $T_1, \ldots, T_n$
  - που ικανοποιούν την ιδιότητα σειράς ενός σωρού (δηλ. το στοιχείο κάθε κόμβου είναι μεγαλύτερο από αυτό των παιδιών του), και
  - για κάθε i, j, $height(T_i) = height(T_j) \Rightarrow i=j$.

π.χ. Μια ουρά προτεραιότητας μήκους 6, μπορεί να αναπαρασταθεί ως το δάσος που αποτελείται από τα δένδρα B_1 και B_2.

- Μια διωνυμική ουρά προτεραιότητας υποστηρίζει τις διαδικασίες DeleteMin, και Merge σε χρόνο λογαριθμικό και επιπλέον τη διαδικασία Insert σε σταθερό χρόνο μέσης περίπτωσης.