



# Διάλεξη 16: Σωροί

---

**Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:**

- Ουρές Προτεραιότητας
- Ο ΑΤΔ Σωρός, Υλοποίηση και πράξεις

**Διδάσκων: Παναγιώτης Ανδρέου**

# Ουρά Προτεραιότητας

- Η δομή δεδομένων **Ουρά (Queue)** 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 (συν.)

- Ουρά Προτεραιότητας
- Ο ΑΤΔ ουρά προτεραιότητας ορίζεται ως μια ακολουθία στοιχείων συνοδευόμενη από τις πράξεις:
  - **Delete\_Min\***: Διαγράφει το ελάχιστο στοιχείο  
Θεωρούμε ότι το μικρότερο κλειδί έχει τη μεγαλύτερη προτεραιότητα
  - **Insert**: Εισάγει ένα καινούριο στοιχείο

# Υλοποίηση Ουράς 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

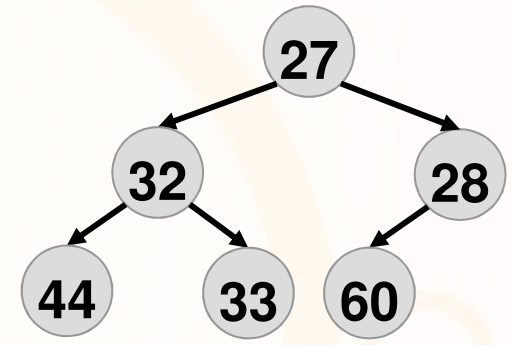
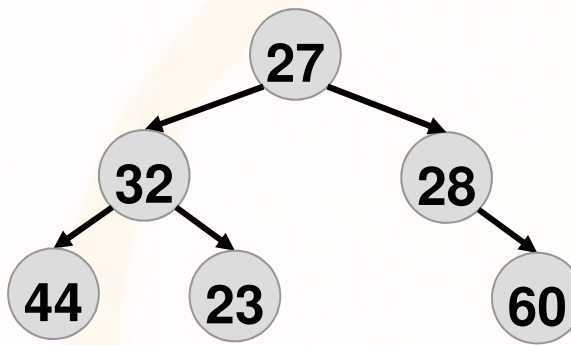
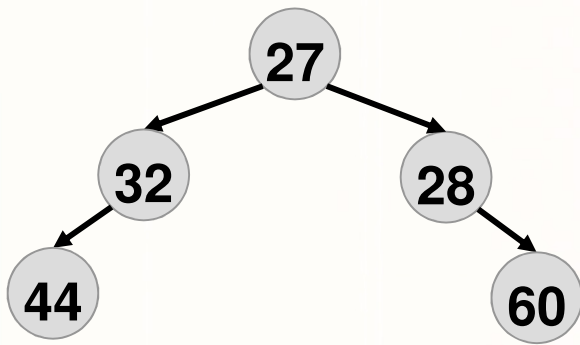
Insert, Delete\_Min:  $O(\log n)$

**Ερώτηση: Υπάρχει καλύτερη υλοποίηση;**

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

# Ο ΑΤΔ 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.
- Ποια από τα πιο κάτω δένδρα είναι σωροί;



# Πλήρη Δυαδικά Δέντρα

- 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

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

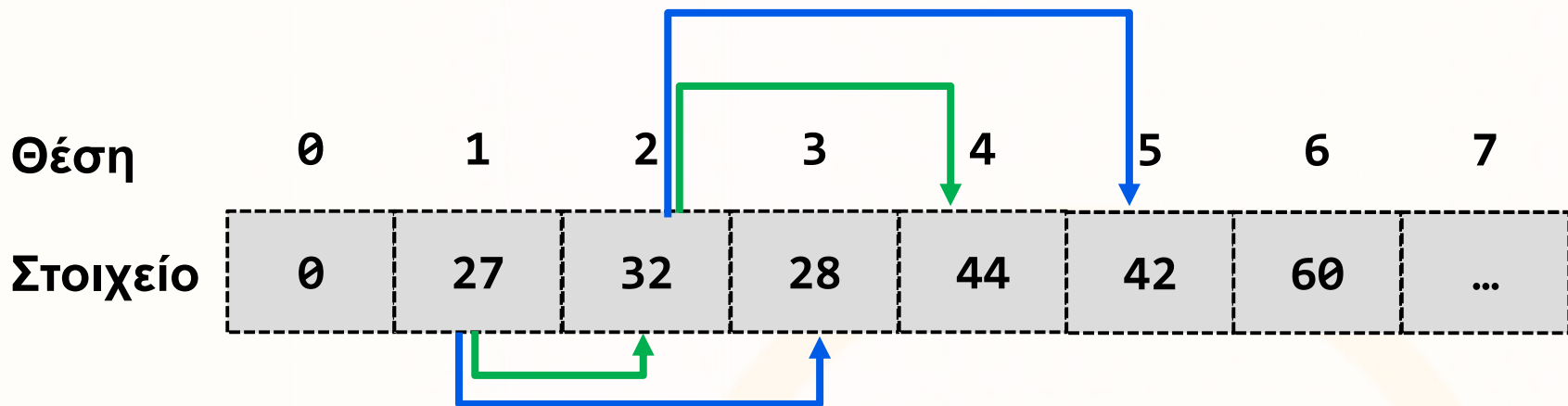
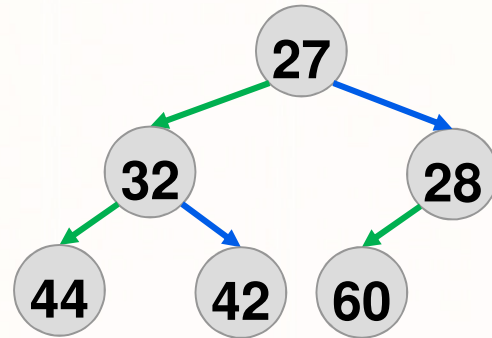
- Επομένως, ένα πλήρες δένδρο ύψους  $h$  έχει μεταξύ  $2^h$  και  $2^{h+1} - 1$  κόμβους.
- Ένα πλήρες δένδρο με  $n$  κόμβους έχει ύψος  $O(\log n)$ .



# Υλοποίηση με πίνακες

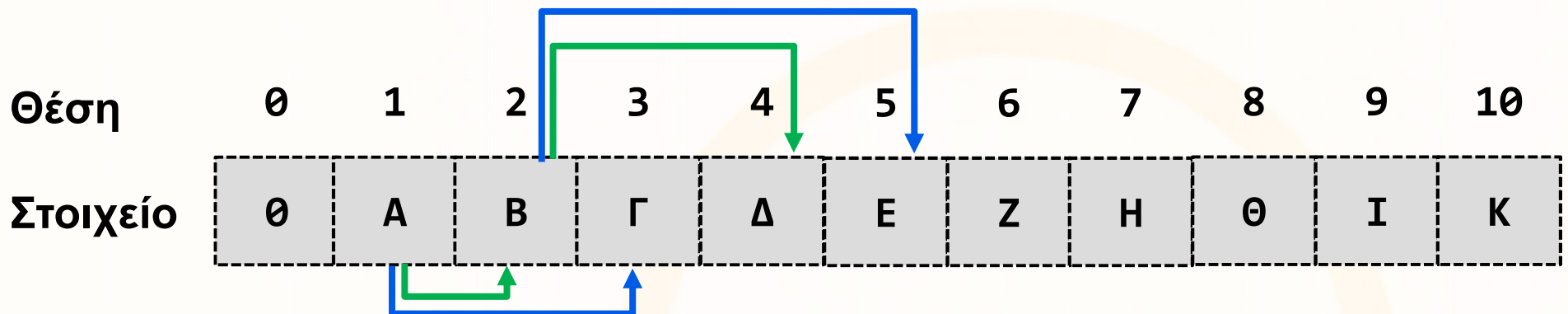
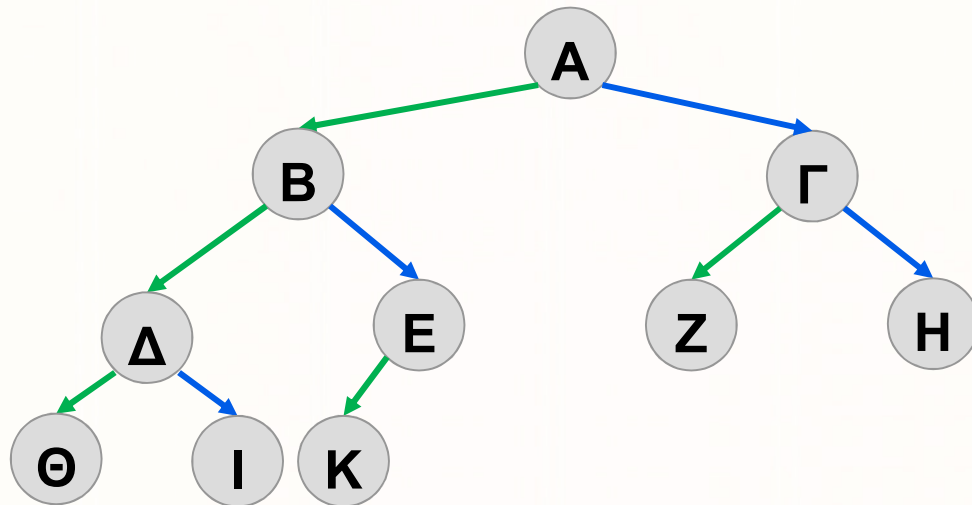
- Ένα πλήρες δυαδικό δένδρο μπορεί να αποθηκευτεί σε πίνακα ως εξής:
  - στη θέση 1 βάζουμε το στοιχείο της ρίζας
  - αν κάποιος κόμβος  $u$  βρίσκεται στη θέση  $i$ , τότε τοποθετούμε το αριστερό του παιδί στη θέση  $2i$ , και το δεξιό του παιδί στη θέση  $2i + 1$ .
- Ο πατέρας ενός κόμβου στη θέση  $i$  (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





# Παράδειγμα αναπαράστασης σωρού



# Υλοποίηση Σωρού

- Ένας σωρός μπορεί να υλοποιηθεί ως μια εγγραφή **Heap** με τρία πεδία:
1. **size** (τύπου int): αποθηκεύει το μέγεθος του σωρού.
  2. **maxSize** (τύπου int): το μέγιστο μέγεθος του πίνακα,
  3. **contents** (τύπου πίνακα): τα στοιχεία του σωρού.
- Αυτή η δομή θα πρέπει να υποστηρίζει τις πράξεις: `makeEmpty`, `isEmpty`, `isFull`, `insert`, `deleteMin`.

```
public class Heap {  
  
    private int contents[];  
    private int size;  
    private int maxsize;  
  
    public Heap(int n) {  
        this.contents = new int[n];  
        this.size = 0;  
        this.maxsize = n-1;  
    }  
  
    public boolean isEmpty(){  
        return  
            this.size==0;  
    }  
  
    public boolean isFull(){  
        return  
            this.size==this.maxsize;  
    }  
}
```

# Εισαγωγή κόμβου

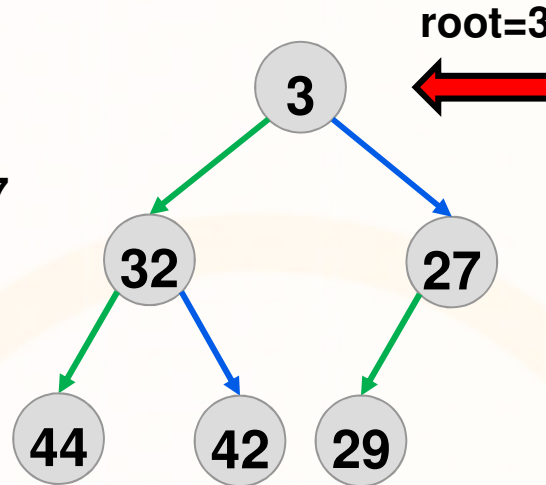
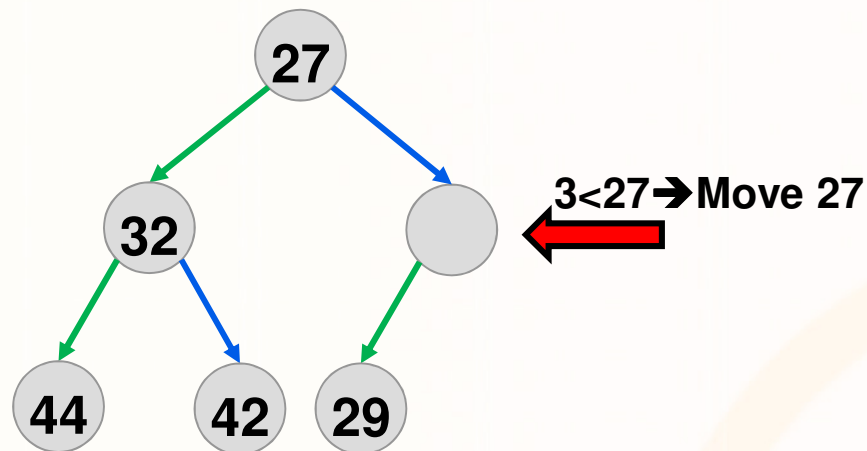
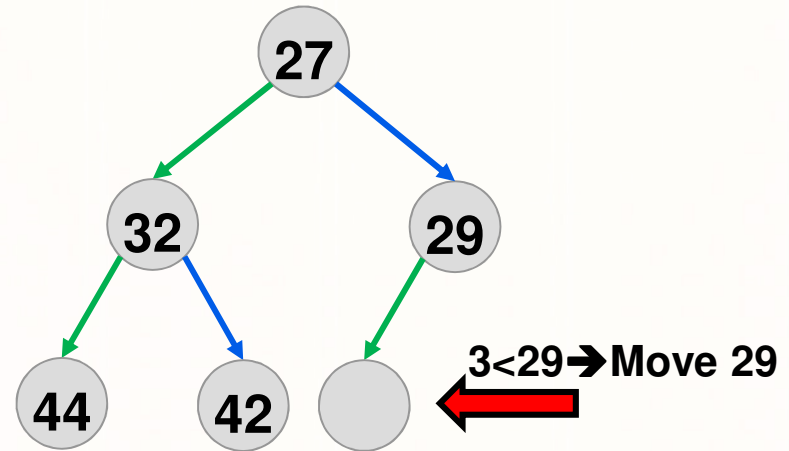
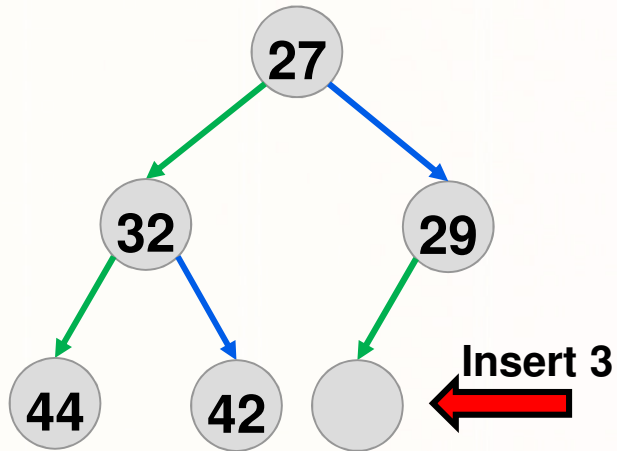
- Σε ένα πλήρες δυαδικό δένδρο υπάρχει μία μόνο θέση όπου μπορεί να εισαχθεί κόμβος και η εισαγωγή να διατηρήσει το δένδρο πλήρες.
- Αυτή η θέση είναι η πιο δεξιά στο τελευταίο επίπεδο του δένδρου, και αντιστοιχεί στη θέση  $size+1$  του πίνακα.
- Για να εισάγουμε ένα κλειδί  $k$  σε ένα σωρό σκεφτόμαστε 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.

# Σωρός με Στατική Δέσμευση Μνήμης: Υλοποίηση

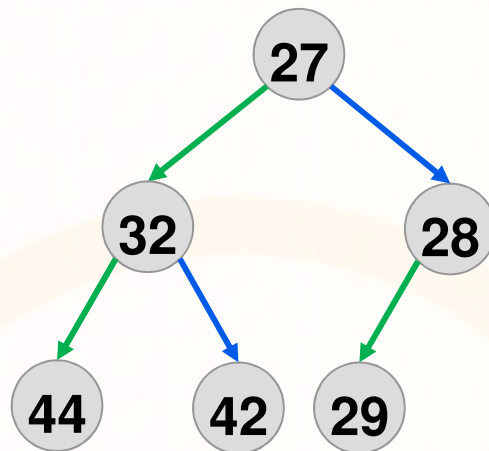
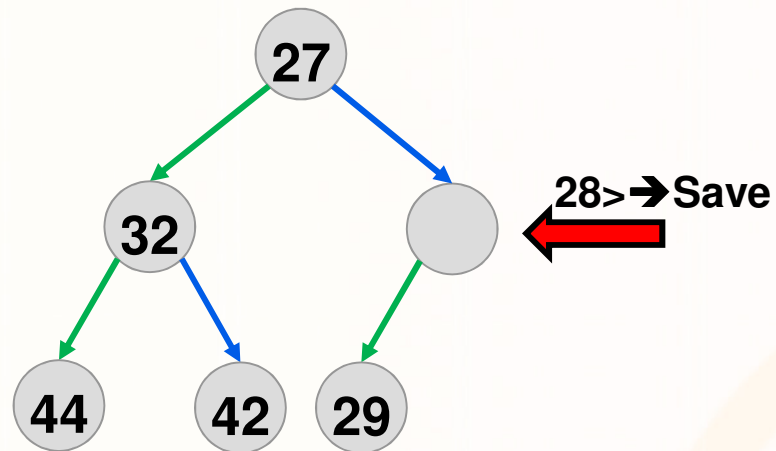
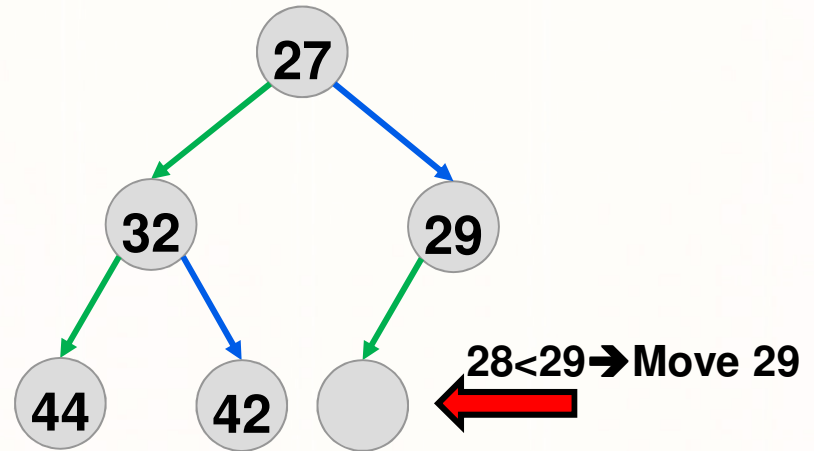
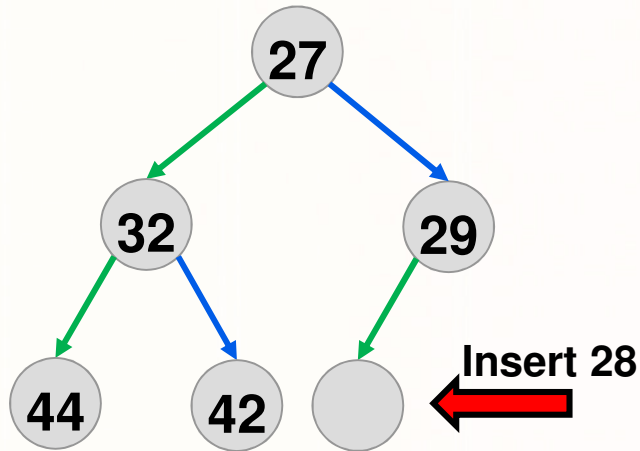
- Συνάρτηση `public void insert(int k);` :

```
public void insert(int k) {  
  
    if (this.size < this.maxsize) {  
        int index = this.size + 1;  
  
        while (index > 1 && this.contents[(index / 2)] > k) {  
            this.contents[index] = this.contents[(index / 2)];  
            index = index / 2;  
        }  
  
        this.contents[index] = k;  
        this.size++;  
  
        this.contents[0] = this.size;  
    }  
}
```

# Παράδειγμα 1: Εισαγωγή σε Σωρό



# Παράδειγμα 2: Εισαγωγή σε Σωρό



0	1	2	3	4	5	6	7
0	27	32	28	44	42	29	...



# Διαγραφή 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. Επαναλαμβάνουμε τη διαδικασία.



# Σωρός με Στατική Δέσμευση Μνήμης: Υλοποίηση

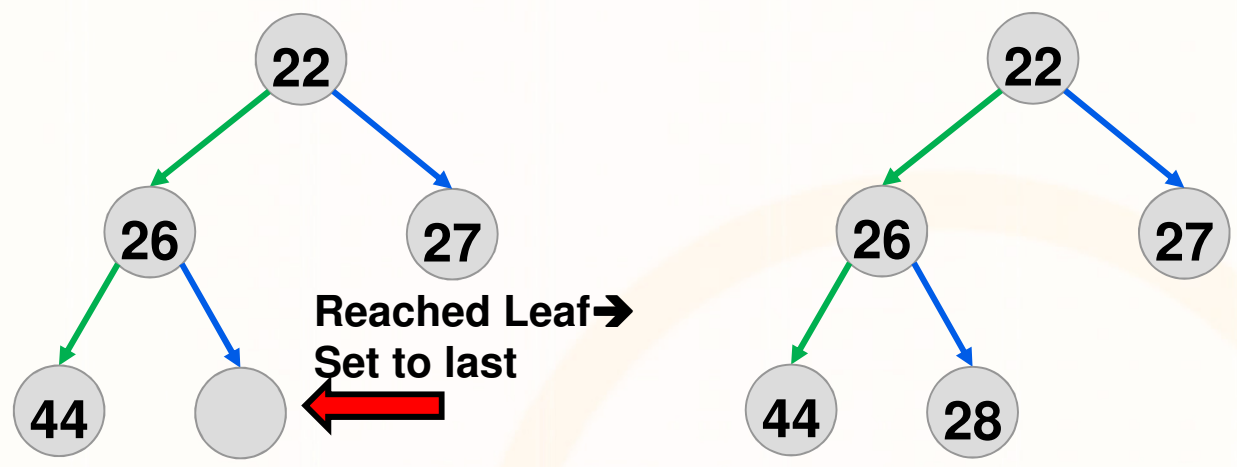
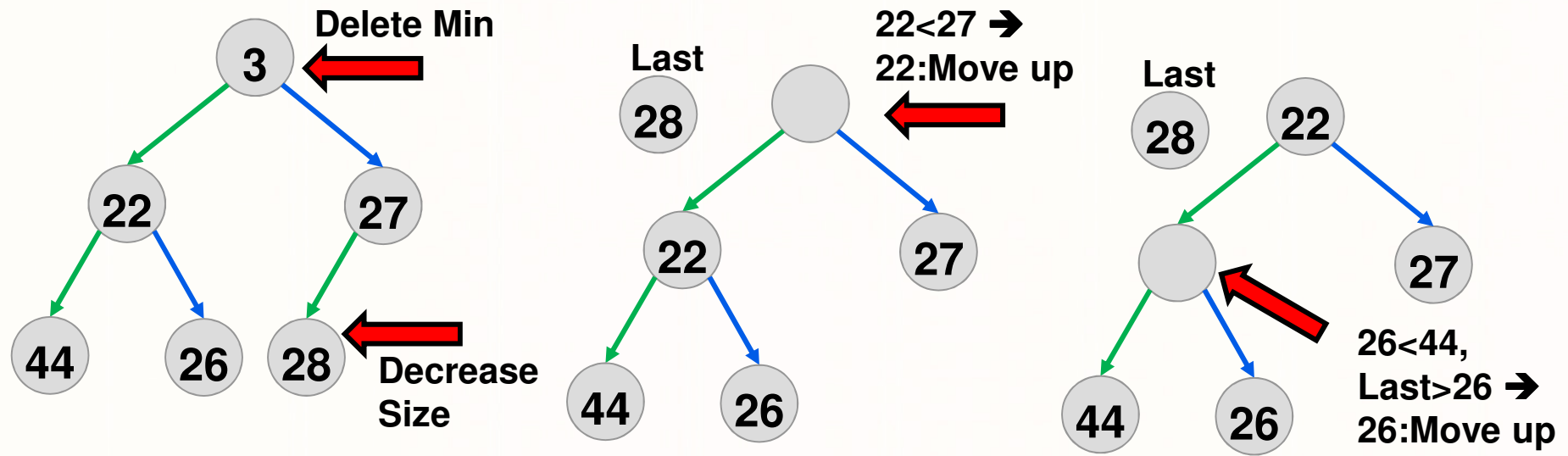
- Συνάρτηση `public int deleteMin()` :

```
public int deleteMin() {
    int min = 0, last;
    int x = 1, child = 0;

    if (!isEmpty()) {
        min = this.contents[1];
        last = this.contents[this.size];
        this.size--;
        this.contents[0] = this.size;

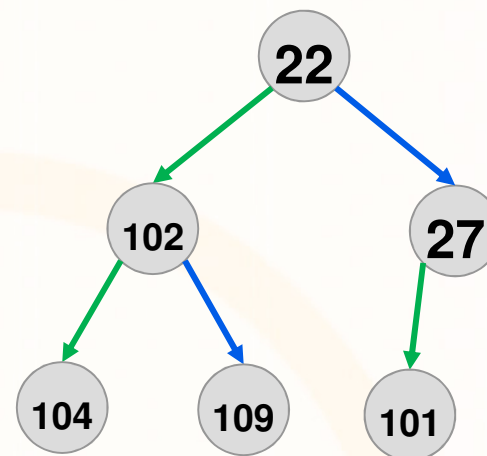
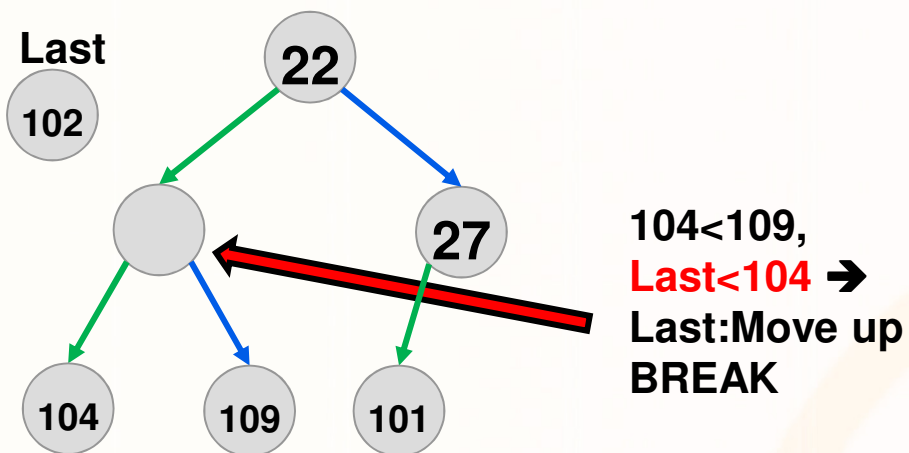
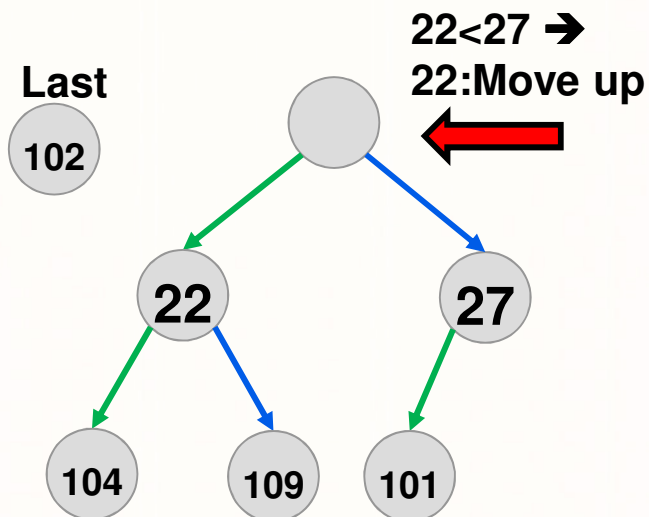
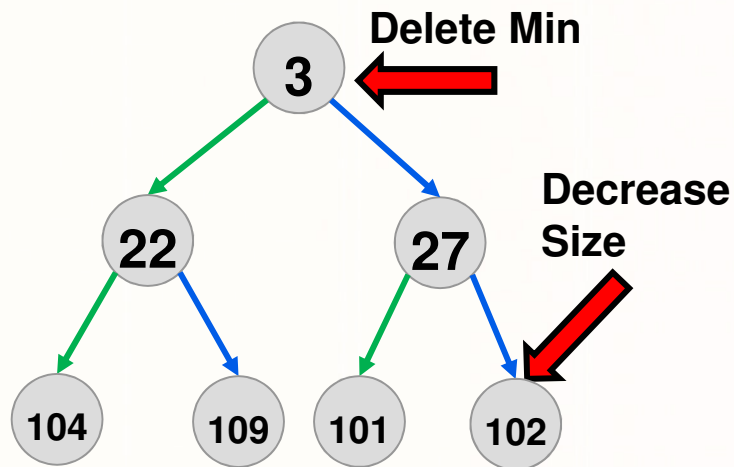
        while ((x * 2) <= this.size) {
            child = x * 2;
            if (child != this.size
                && this.contents[child + 1] < this.contents[child])
                child++;
            if (last > this.contents[child]) {
                this.contents[x] = this.contents[child];
          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

# Παράδειγμα 3: Εξαγωγή από Σωρό



0	1	2	3	4	5	6
0	22	26	27	44	28	...

# Παράδειγμα 3: Εξαγωγή από Σωρό



0	1	2	3	4	5	6
0	22	102	27	104	109	101

# Μερικά Σχόλια

- Ο χρόνος εκτέλεσης των διαδικασιών InsertHeap και DeleteMin 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  $d$ -σωρούς ( $d$ -heaps), όπου κάθε κόμβος έχει  $d$  παιδιά.