Δομές Δεδομένων & Ανάλυση Αλγορίθμων
3ο Εξάμηνο

- **Πλήρη και Σχεδόν Πλήρη ΔΔ**
- **Σωρός (Heap)**
- **Ουρές Προτεραιοτήτων (Priority Queues)**

Δημοσθένης Σταμάτης
http:/www.iee.ihu.gr/~demos
Τμήμα Μηχανικών Πληροφορικής & Ηλεκτρονικών Συστημάτων

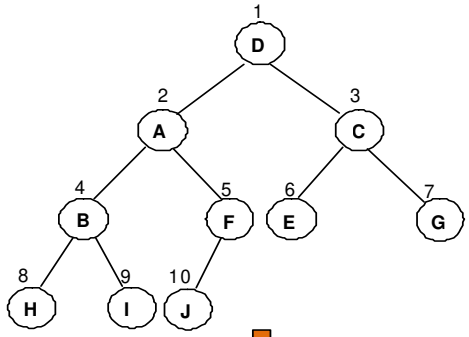
Σωρός (Heap)

Ορισμός 1:
Ένα δυαδικό δέντρο βάθους **N** ονομάζεται **πλήρες (*complete*)** όταν έχει όλους τους κόμβους του επιπέδου **N** συμπληρωμένους.

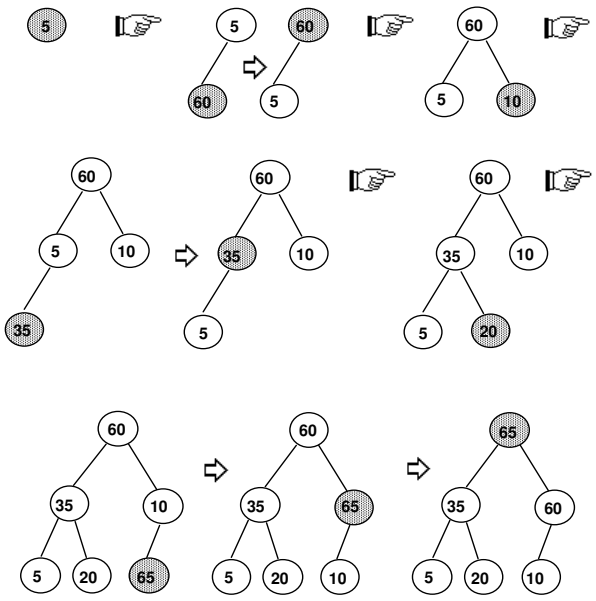
Ορισμός 2:
Ένα δυαδικό δέντρο βάθους **N** ονομάζεται **σχεδόν πλήρες (*almost complete*)** όταν έχει όλους τους κόμβους του επιπέδου **N-1** συμπληρωμένους και οι κόμβοι που υπάρχουν στο **N-οστό** επίπεδο είναι τοποθετημένοι όσο το δυνατόν πιο αριστερά.

Ορισμός 3:
Ένα δυαδικό δέντρο ονομάζεται **σωρός (*heap*)** μεγίστων (ελαχίστων) όταν ισχύουν επιπλέον οι παρακάτω δύο προϋποθέσεις:
(α) το δυαδικό δέντρο είναι σχεδόν πλήρες και
(β) οι τιμές των κόμβων είναι τοποθετημένες με τέτοιο τρόπο ώστε ο κάθε κόμβος πατέρας να έχει μεγαλύτερη (μικρότερη) τιμή από τα παιδιά του.

Σχεδόν Πλήρες Δυαδικό Δέντρο



D	A	C	B	F	E	G	H	I	J
1	2	3	4	5	6	7	8	9	10



Διαδικασία δημιουργίας Σωρού με επαναληπτική χρήση της insert για την εισαγωγή των παρακάτω δεδομένων:

5 60 10 35 20 65

Υλοποίηση Σωρού με σχεδόν πλήρες δυαδικό δέντρο

```
public class Heap {

        private Object[ ] btree;
        private int index;
        private int capacity;

        public Heap( ) {
                this(100);
        }
        public Heap(int cap) {
                capacity = cap;
                btree = new Object[capacity+1];
                index = 0;
        }
. . .
```


//class Heap (συνέχεια)

```
public int size( ) {
    return index;
}

public boolean isEmpty () {
    return index==0;
}

public boolean isFull() {
    return index==capacity;
}
 . . .
```

//class Heap (συνέχεια)

```
public void insert (Object item) throws HeapFullException {
    if (isFull()) throw new HeapFullException("Heap is now Full");
    int father, son;
    son = ++index;
    btree[son] = item;  // αρχικά το item στο τέλος της heap
    father = son/2;
    while(son>1 && ((Comparable)btree[son]).compareTo
                                    ((Comparable)btree[father])>0) {
        Object b = btree[father];
        btree[father] = btree[son];
        btree[son] = b;
        son = father;
        father = son/2;
     }
  }
```


//class Heap (συνέχεια)

```
public Object remove( ) throws HeapEmptyException {
   if (isEmpty( )) throw new HeapEmptyException("No items in Heap");
   int father, son;
   Object lastItem = btree[index];
   btree[index] = btree[1];
   Object removeItem = btree[1];
   index--;
   father = 1;
   if ( (index > 2) && ((Comparable)btree[2]).compareTo((Comparable)btree[3])>0 )
        son = 2;
   else son = 3;
   while(son <= index && ((Comparable)btree[son]).compareTo
                                                ((Comparable)lastItem)>0 ) {
        btree[father] = btree[son];
        father = son;
        son = father * 2;
        if (son+1 <= index && ((Comparable)btree[son+1]).compareTo
                                        ((Comparable)btree[son])>0) son = son +1;
   }
   btree[father] = lastItem;
   return removeItem;
  }
```