Priority Queues

- Definitions:
    - A priority queue is a restricted form of a list, where items are arranged according to their priorities (or keys). The key assigned to an item may not be unique.

    - The item with highest priority is removed in O(1).

    - Each node stores prioritized key-item(s) pairs

- Priority Queue ADT operations:
    - insertItem (key, data): inserts data in the priority queue according to key
    - removeItem (): removes the item with the smallest key (in the min priority queue), or the item with the largest key (in the max priorityqueue).
    - size()
    - empty()
    - minItem() / maxitem(): (returns the item with the smallest/largest key)
    - minKey() / maxKey() :(returns the smallest/largest key)

- Heap
  - Motivations:
    - Get an object with highest priority in a constant of time O(1).

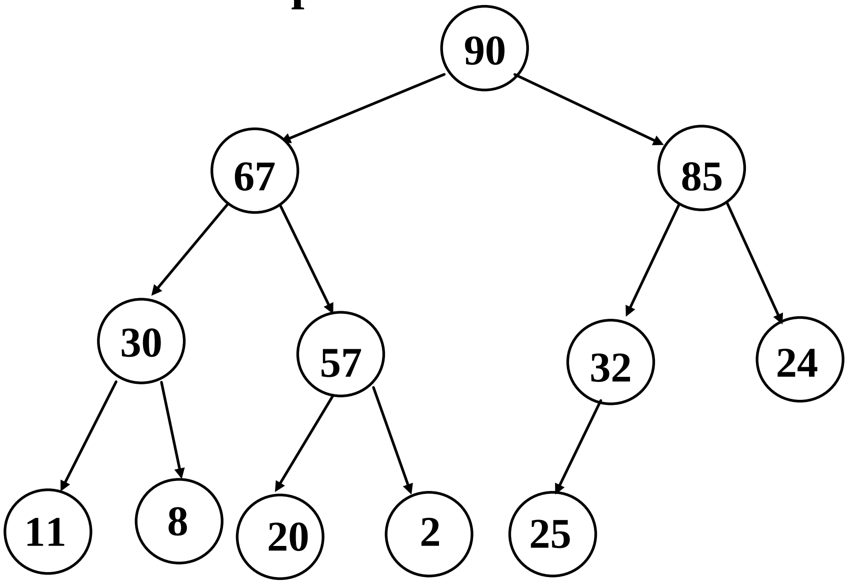
  - Definition
    - Heap is a priority queue.

  - Examples:
    - Heaps
    - Deaps
    - Etc.

- Heap Structures
  - **Definition**
    - A **max-heap** (**min-heap**) is a complete BT with the property that the key (priority) of each node is at least as **large** (**small**) as the values at its children (if they exist).

  - **Implementation**:
    - Sequential representation

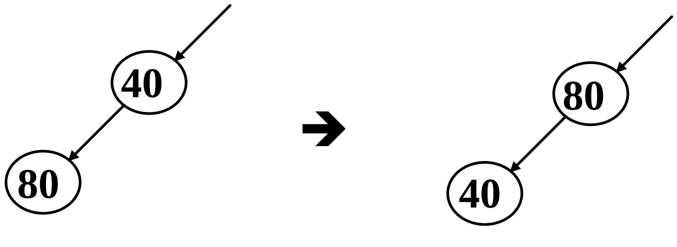
    - **Example**:



- **Insertion of a heap**
  - Procedure Insert(A[1..n],i)

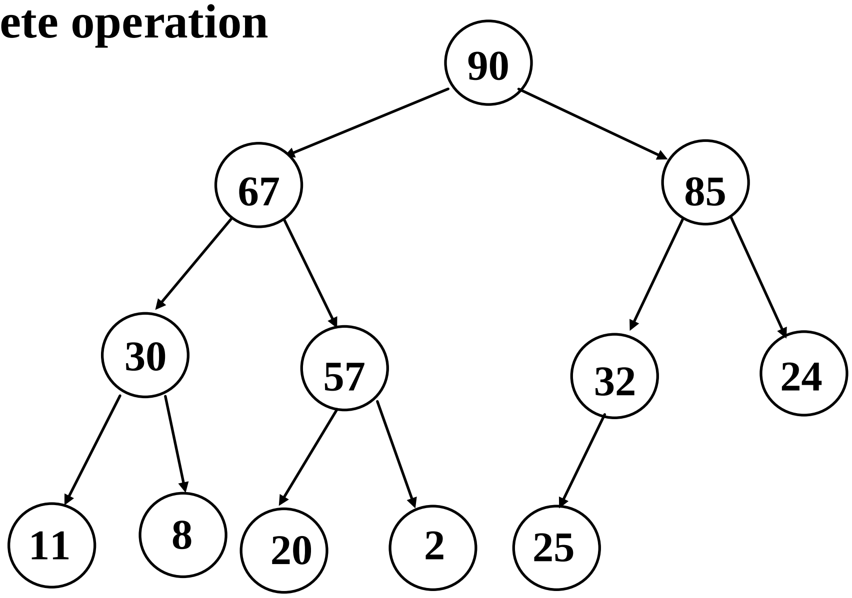
```
/* Insert A[i] into the already hear A[1..n] */
Begin
    While (I>1) and (A[i]> A[$\lfloor \frac{i}{2} \rfloor$]) do
    Begin
        swap(A[i], A[$\lfloor \frac{i}{2} \rfloor$]);
        i = $\lfloor \frac{i}{2} \rfloor$;
    Endwhile
End;
```

  - Example:
    - List of elements: 40, 80, 35, 90, 85, 100
    - The heap is empty
    - Insert(40)
    - Insert(80)

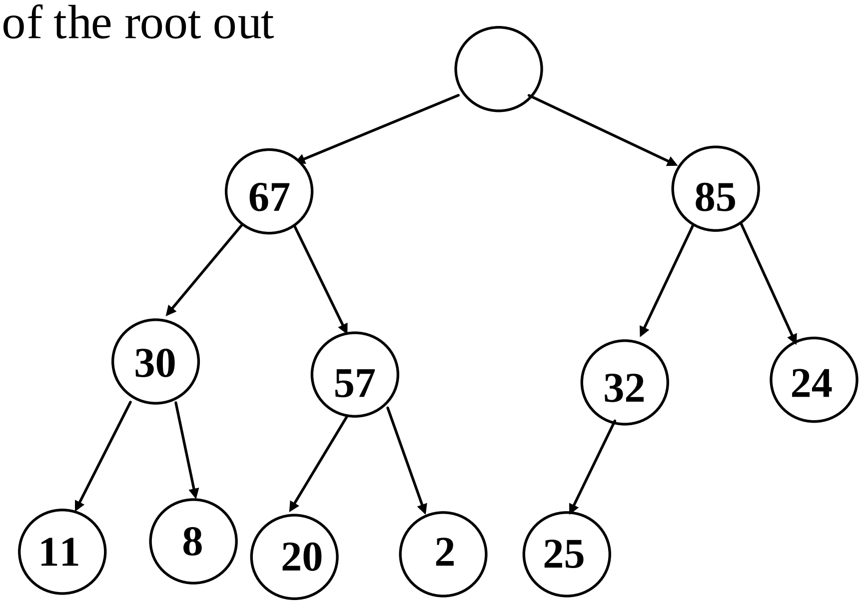


    - Etc.

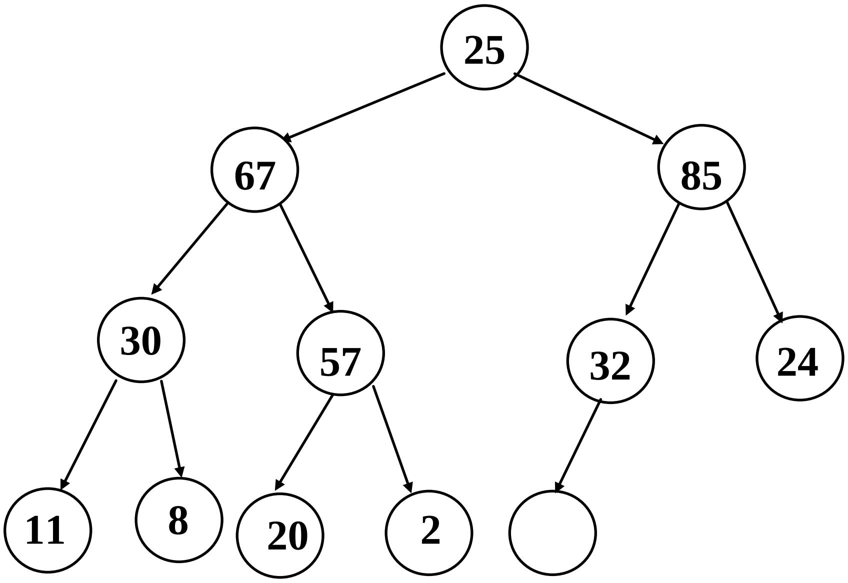
- **Delete operation**



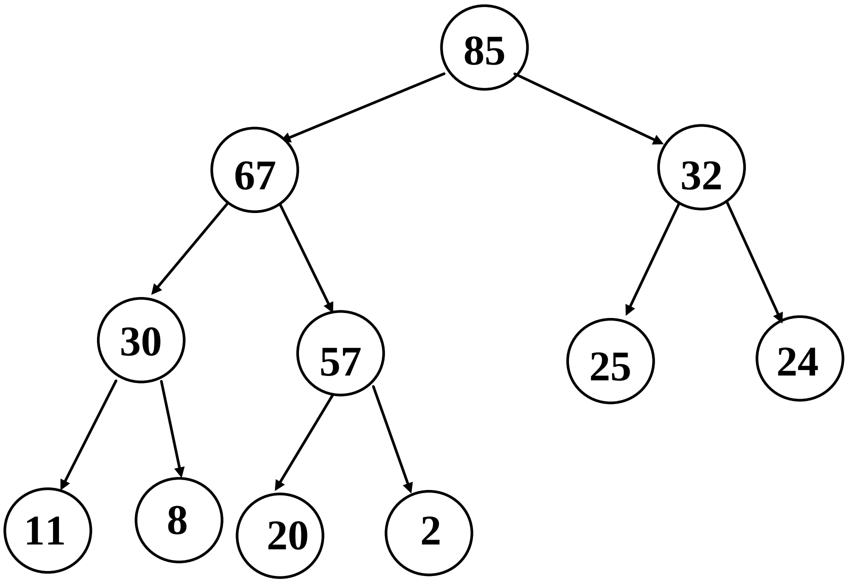
  - Take the content of the root out



- Put the last node in the heap in the root



- **Adjust the heap**:



- Procedure:

```
Procedure adjust_heap()
/* Move the last value in the heap to the root */
Boolean done=false;
type element;
Begin
    j = 2i; element = A[i];
    While ((j≤n) && (!done))
    Begin
        /* j points to the largest child of A[$\left\lfloor \frac{j}{2} \right\rfloor$] */
        If ((j<n) && (A[j]<A[j+1])) then
            j = j + 1;
        endif;
        If (element ≥ A[j]) then
            done = TRUE;
        else  begin
                A[$\left\lfloor \frac{j}{2} \right\rfloor$] = A[j];  j = 2*j;
            end;
        endif;
    Endwhile;
    A[$\left\lfloor \frac{j}{2} \right\rfloor$] = element;
end;
```

- Complexity:
  - **O(logn)** where n is the number of elements in the heap.

- **Sorting: HeapSort**
  - Motivation:
    - The worst case is O(nlogn)
  - Implementation:

```
Procedure Heapsort(A[1..n])
int i;
Begin
    construct_heap(A[1..n])
    for i=n to 2 step -1 do
        swap(A[1], A[i]);
        Adjust_heap((A[1..(i-1)]))
    endfor;
end;
```

  - Complexity:
    - Let n be the number of element to be sorted.
    - Heap construction takes O(n)
    - Adjust heap takes O(logn)
    - The for loop takes O(n)
      - Therefore, O(nlogn).

- Code Example

```
public class Element {

   private int inData;

   public Element(int data){
         inData = data;
   }

   public int getData(){
      return inData;
   }

   public void setData(int data){
      inData = data;
   }
}


 public class MaxHeap {

   private Element[] heapArray;
   private int maxSize;
   private int currentSize;
   public MaxHeap(int max){
         maxSize = max;
       currentSize = 0;
       heapArray = new Element[maxSize];  // create the heap
   }
   public boolean isEmpty(){
         return currentSize==0;
   }
   // Move an element up in the heap tree.
   public void adjustHeap(int index){
         int parent = (index-1) / 2;
       Element bottom = heapArray[index];

       while( index > 0 &&
              heapArray[parent].getData() < bottom.getData() ){
          heapArray[index] = heapArray[parent];  // move it down
          index = parent;
          parent = (parent-1) / 2;
       }
       heapArray[index] = bottom;
   }

   public boolean insert(int key) {
         if(currentSize==maxSize)
               return false;
         Element newElement = new Element(key);
         heapArray[currentSize] = newElement;
         adjustHeap(currentSize++);
         return true;
    }
   public void getMaxHeap() {


   }
```

```
    public void printHeap() {


    }
    public void deleteMax(){

    }

}
```

- **Programming Assignment:**
  - Is the sequence {23;17;14;6;13;10;1;5;7;12} a max-heap?
  - Design and implement the missing operations in the MaxHeap ADT:
    - getMaxHeap
    - deleteMax: delete the max value of the heap.
    - printHeap

  - Test your implementation.