Randomized Meldable Priority Queues

Group Members:
Sameera Shahzad Alam (18b-046-SE)
Fiza Khalid (18b-116-SE)
Humemah Khalid (18b-106-SE)

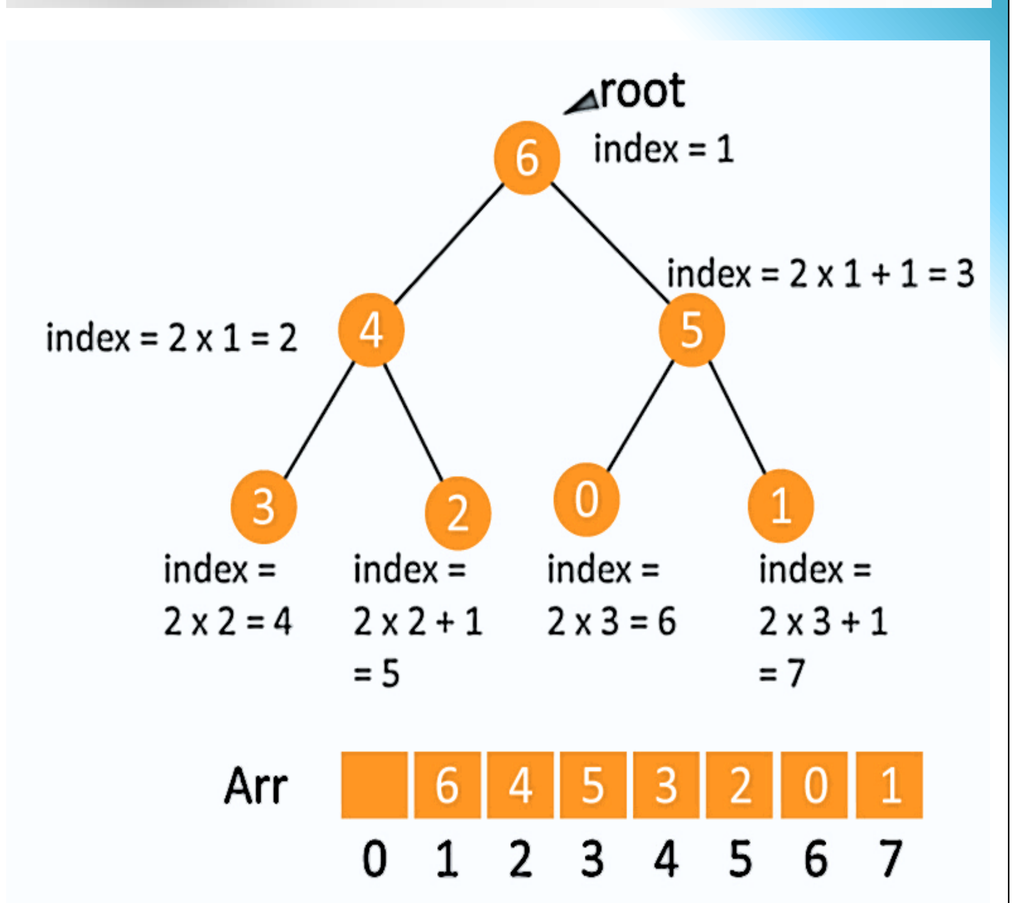
Course Facilitator:
Miss. Asma Idris Mala
Miss. Saba Saeed

Introduction:

In computer science, a randomized meldable heap (also Meldable Heap or Randomized Meldable Priority Queue) is a priority queue based data structure in which the underlying structure is also a heap-ordered binary tree.However, there are no restrictions on the shape of the underlying binary tree.

Operation:

The randomized meldable heap supports a number of common operations. These are insertion, deletion, and a searching operation, findMin. The insertion and deletion operations are implemented in terms of an additional operation specific to the Meldable Heap ,Meld(Q1, Q2).



Meld:

```
function Meld(Node Q1, Node Q2)
 if Q1 is nil => return Q2
 if Q2 is nil => return Q1
 if Q1 > Q2 => swap Q1 and Q2
 if coin_toss is 0 => Q1.left = Meld(Q1.left, Q2)
 else Q1.right = Meld(Q1.right, Q2)
 return Q1
```

Worst-Case Time Efficiency: Meld(Q1, Q2) = O(logn)

Insert:

```
function Insert(x)
 Node u = new Node
 u.x = x
 root = Meld(u, root)
 root.parent = nil
 increment node count
```

Worst-Case Time Efficiency:
Insert(x) = O(logn)

Remove:

```
function Remove()
  root = Meld(root.left, root.right)
  if root is not nil => root.parent = nil
  decrement node count
```

Worst-Case Time Efficiency:
Remove() = O(logn)

Additonal Operations:

>**Remove(u)** - Remove the node u and its key from the heap.
Worst-Case Time Efficiency: Remove(x) =O(logn)

>**Absorb(Q)** - Add all elements of the meldable heap Q to this heap, emptying Q in the process.
Worst-Case Time Efficiency: Absorb(Q) O(logn)

>**DecreaseKey(u, y)** - Decreases the key in node u to y (pre-condition: y <= u.x).

FindMin:

FindMin() simply returns the element currently stored in the heap's root node.

Worst-Case Time Efficiency:
FindMin() O(logn)