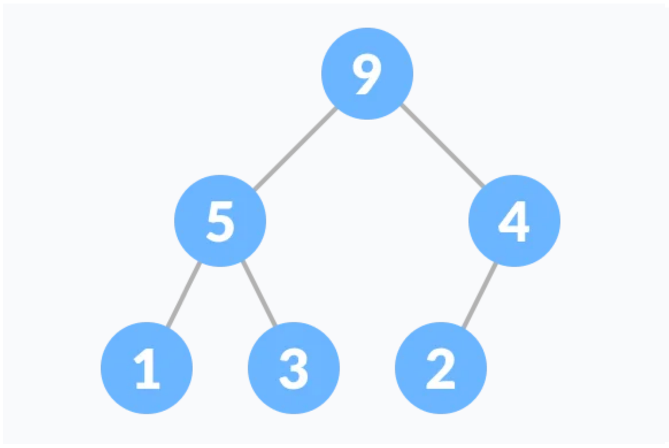
The School Electrical Engineering and Information Technology
Computer Science Department

Heap

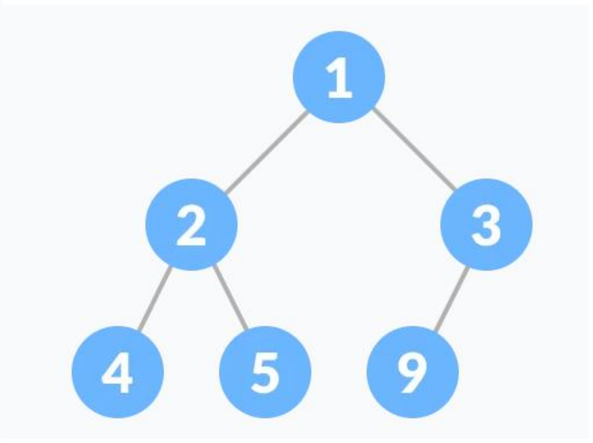
Definition:

Heap data structure is a complete binary tree that satisfies the **heap property**, where any given node is:

- Always greater than its child node/s and the key of the root node is the largest among all other nodes. This property is also called **max heap property**.
- Always smaller than the child node/s and the key of the root node is the smallest among all other nodes. This property is also called **min heap property**.



Max Heap



Min Heap

Created by Hend Bataineh

Main Operations

- **Adding a Node to a Heap**
- **Remove the Top Operation**
- **Heapify**
- **Removing a value from the a Heap**

Lab Work

The following code contains functions to implement a **Max-Heap** using a partially filled array, Fill the body of all functions:

```
#include<iostream>
using namespace std;

   static const int MAX_SIZE = 15;
   int heap[MAX_SIZE];
   int size=0;

   // returns the index of the parent node
   static int parent(int i) {
      return (i - 1) / 2;
   }
   // return the index of the left child
   static int leftChild(int i) {
      return 2*i + 1;
   }
   // return the index of the right child
   static int rightChild(int i) {
      return 2*i + 2;
   }

   // insert the item at the appropriate position
   void insert(int data) {



   }
   // returns the  maximum item of the heap
   int getMax() {




   }
```

```
    // deletes the max item

    void DeleteTop() {



    }

    void maxHeapify(int i){
        // find left child node
        int left = leftChild(i);

        // find right child node
        int right = rightChild(i);


        // find the largest among 3 nodes
        int largest = i;
        // check if the left node is larger than the current node
        if (left <= size && heap[left] > heap[largest]) {
            largest = left;
        }


        // check if the right node is larger than the current node
        // and left node
        if (right <= size && heap[right] > heap[largest]) {
            largest = right;
        }

        // swap the largest node with the current node
        // and repeat this process until the current node is larger than
        // the right and the left node
        if (largest != i) {
            int temp = heap[i];
            heap[i] = heap[largest];
            heap[largest] = temp;
            maxHeapify(largest);
        }

    }


int main() {


}
```