CL-210
Data Structures
Lab # 12

Objectives:

- Binary Heap
- Max, Min Heap
- Heapify
- Heap Sort

Note: Carefully read the following instructions (*Each instruction contains a weightage*)

1. There must be a block of comments at start of every question's code by students; the block should contain brief description about functionality of code.
2. Comment on every function and about its functionality.
3. Mention comments where necessary such as comments with variables, loop, classes etc to increase code understandability.
4. Use understandable name of variables.
5. Proper indentation of code is essential.
6. Write a code in C++ language.
7. Make a Microsoft Word file and paste all of your C++ code with all possible screenshots of every task **outputs in Microsoft Word and submit word file.**
8. First think about statement problems and then write/draw your logic on copy.
9. After copy pencil work, code the problem statement on MS Studio C++ compiler.
10. At the end when you done your tasks, attached C++ created files in MS word file and make your submission on Google Classroom. (Make sure your submission is completed).
11. Please submit your file in this format **19F1234_L11**.
12. Do not submit your assignment after deadline. Late and email submission is not accepted.
13. Do not copy code from any source otherwise you will be penalized with negative marks.

Problem: 1| Binary heap

1) **getMini():** It returns the root element of Min Heap.
2) **extractMin():** Removes the minimum element from MinHeap..
3) **insert():** Inserting a new key. We add a new key at the end of the tree. If new key is greater than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

Problem: 2 | Heap Sort

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

What is Binary Heap?

Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

A Binary Heap is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes.

The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

Heap Sort Algorithm for sorting in increasing order:

1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps while size of heap is greater than 1.

Problem: 3 | K largest(or smallest) elements in an array

Write an efficient program for printing k largest elements in an array. Elements in array can be in any order.

For example, if given array is [1, 23, 12, 9, 30, 2, 50] and you are asked for the largest 3 elements i.e., k = 3 then your program should print 50, 30 and 23.

Problem: 4 | Convert BST to Min Heap

Given a binary search tree which is also a complete binary tree. The problem is to convert the given BST into a Min Heap with the condition that all the values in the left subtree of a node should be less than all the values in the right subtree of the node. This condition is applied on all the nodes in the so converted Min Heap.

Examples:

```
Input:          4
              /   \
             2     6
           /  \   /  \
          1   3  5    7


Output:         1
              /   \
             2     5
           /  \   /  \
          3   4  6    7
```

The given **BST** has been transformed into a
Min Heap.
All the nodes in the Min Heap satisfies the given condition, that is, values in the left subtree of a node should be less than the values in the right subtree of the node.

1. Create an array **arr[]** of size **n**, where n is the number of nodes in the given BST.
2. Perform the inorder traversal of the BST and copy the node values in the **arr[]** in sorted order.
3. Now perform the preorder traversal of the tree.

While traversing the root during the preorder traversal, one by one copy the values from the array **arr[]** to the nodes

Problem: 5 | Convert min Heap to max Heap

Given array representation of min Heap, convert it to max Heap

Example:

```
Input: arr[] = [3 5 9 6 8 20 10 12 18 9]
         3
      /     \
     5       9
   /   \    /  \
  6     8  20   10
 /  \   /
12   18 9



Output: arr[] = [20 18 10 12 9 9 3 5 6 8] OR
       [any Max Heap formed from input elements]


         20
       /    \
     18      10
   /    \    /  \
  12     9  9    3
 /  \   /
5    6 8
```

The problem might look complex at first look. But our final goal is to only build the max heap. The idea is very simple – we simply build Max Heap without caring about the input. We start from bottom-most and rightmost internal mode of min Heap and heapify all internal modes in bottom up way to build the Max heap.

Heap Data Structures

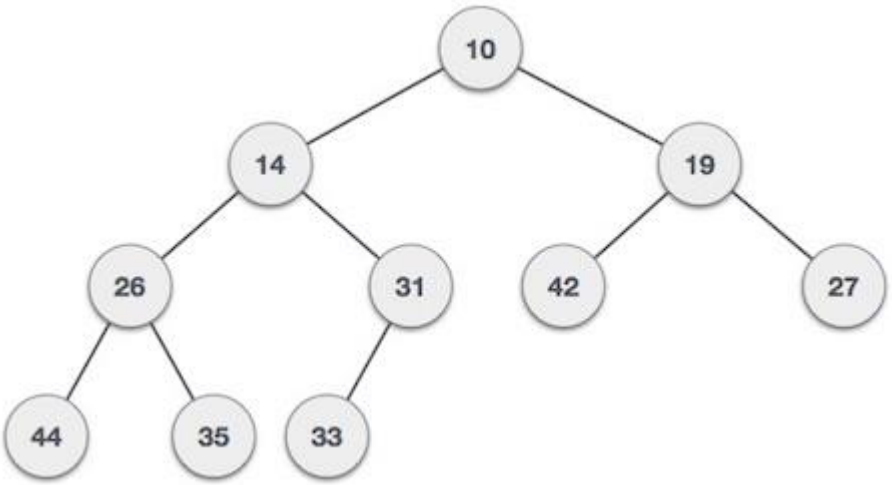
Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If **α** has child node **β** then −

key(α) ≥ key(β)

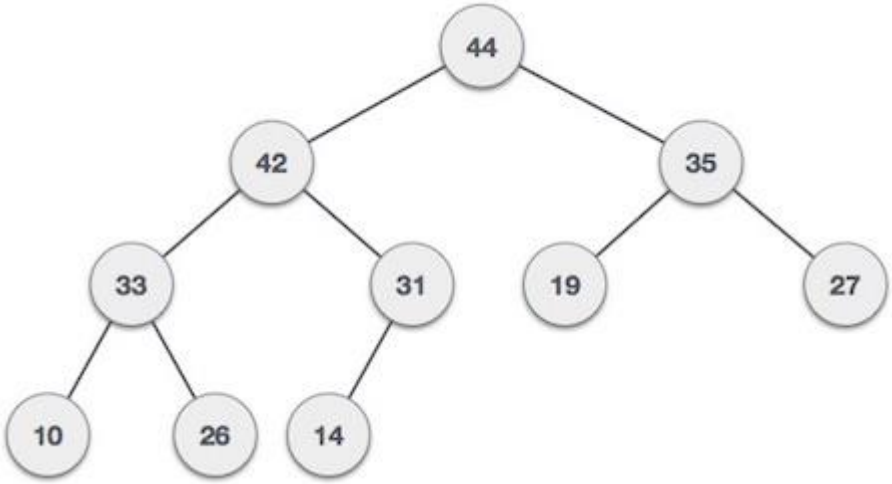
As the value of parent is greater than that of child, this property generates **Max Heap**. Based on this criteria, a heap can be of two types −

For Input → 35 33 42 10 14 19 27 44 26 31

Min-Heap − Where the value of the root node is less than or equal to either of its children.



Max-Heap − Where the value of the root node is greater than or equal to either of its children.



Both trees are constructed using the same input and order of arrival.

Max Heap Construction Algorithm

We shall use the same example to demonstrate how a Max Heap is created. The procedure to create Min Heap is similar but we go for min values instead of max values.

We are going to derive an algorithm for max heap by inserting one element at a time. At any point of time, heap must maintain its property. While insertion, we also assume that we are inserting a node in an already heapified tree.

Step 1 − Create a new node at the end of heap.
Step 2 − Assign new value to the node.
Step 3 − Compare the value of this child node with its parent.
Step 4 − If value of parent is less than child, then swap them.
Step 5 − Repeat step 3 & 4 until Heap property holds.

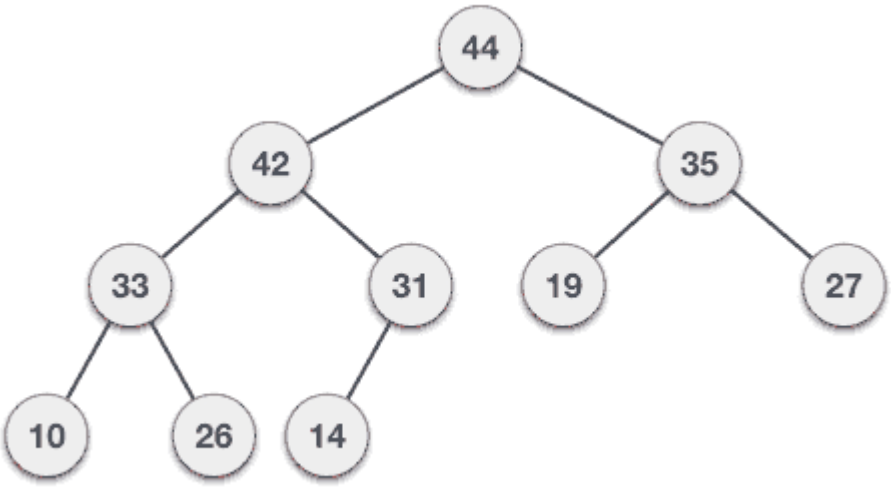
Note − In Min Heap construction algorithm, we expect the value of the parent node to be less than that of the child node.

Let's understand Max Heap construction by an animated illustration. We consider the same input sample that we used earlier.

Max Heap Deletion Algorithm

Let us derive an algorithm to delete from max heap. Deletion in Max (or Min) Heap always happens at the root to remove the Maximum (or minimum) value.

Step 1 − Remove root node.
Step 2 − Move the last element of last level to root.
Step 3 − Compare the value of this child node with its parent.
Step 4 − If value of parent is less than child, then swap them.
Step 5 − Repeat step 3 & 4 until Heap property holds.



Best of luck