Data Structures Lab 10

Course: Data Structures (CL2001) **Semester:** Fall 2023

Instructor: Sameer Faisal **T.A:** N/A

Note:

- Lab manual cover following topics
  {Heaps, Min Heap, Max Heap, Heap Sort, Priority Queue}
- Maintain discipline during the lab.
- Just raise your hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

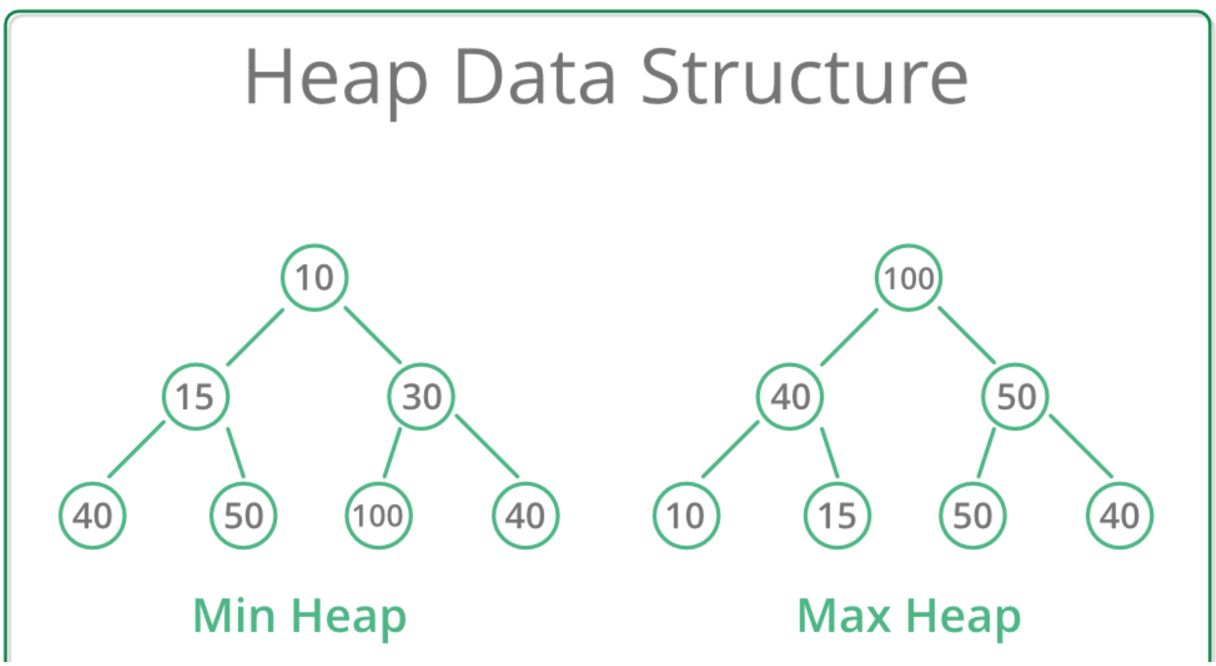
HEAP Data Structures

What is Heap Data Structure?

A Heap is a special Tree-based data structure in which the tree is a complete binary tree.

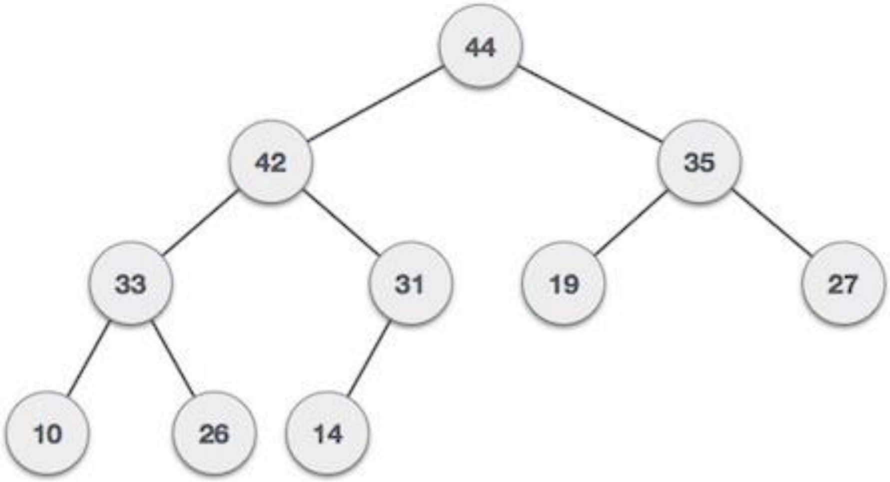
Operations of Heap Data Structure:

1. Heapify: a process of creating a heap from an array.
2. Insertion: process to insert an element in existing heap time complexity O(log N).
3. Deletion: deleting the top element of the heap or the highest priority element, and then organizing the heap and returning the element with time complexity O(log N).
4. Peek: to check or find the most prior element in the heap, (max or min element for max and min heap).

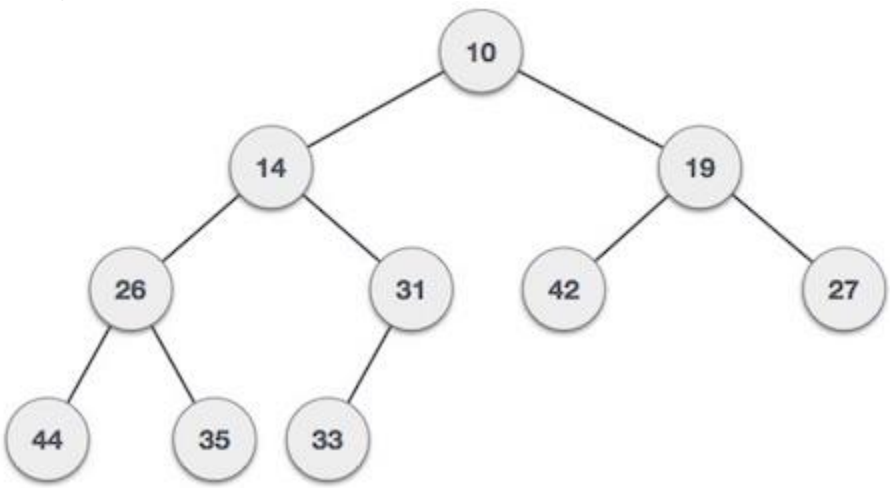


Types of Heap Data Structure
Generally, Heaps can be of two types:

Max-Heap: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.



Min-Heap: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.



How to construct a HEAP:

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.

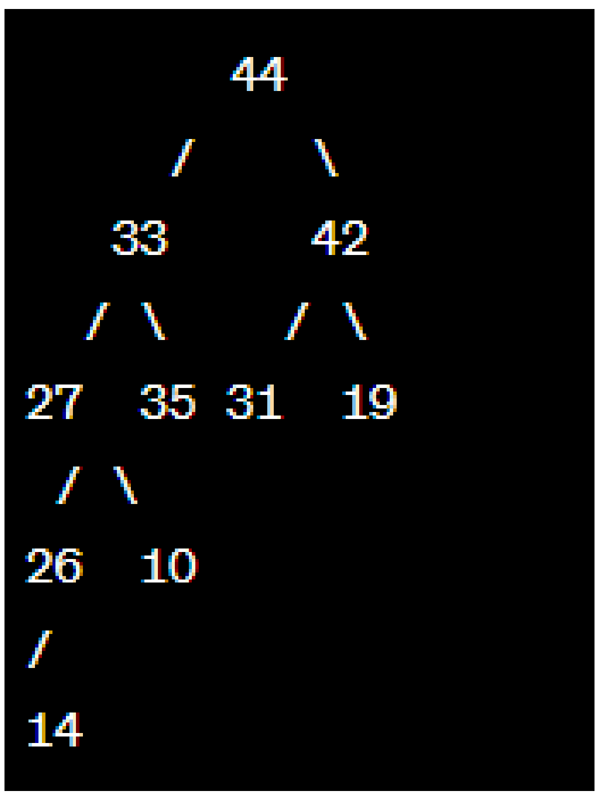
We are going to derive an algorithm for max heap by inserting one element at a time. At any point of time, heap must maintain its property. While insertion, we also assume that we are inserting a node in an already heapified tree.

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.

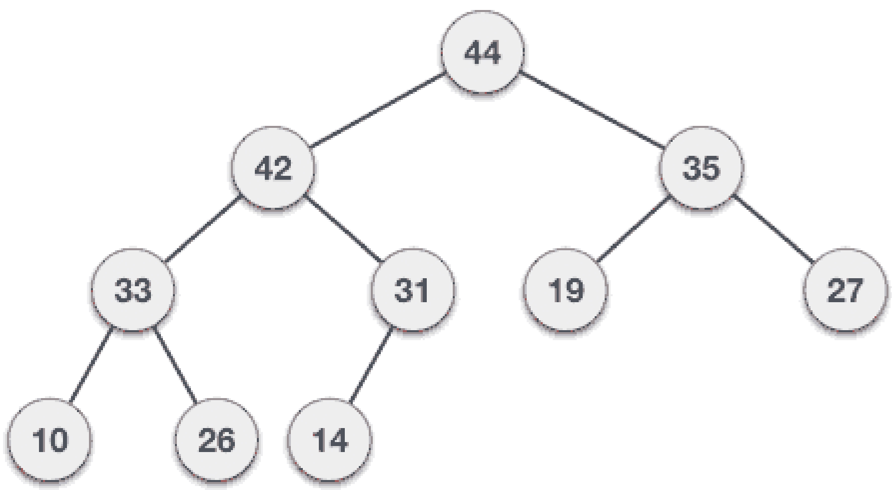
Input 35 33 42 10 14 19 27 44 26 31



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.



Priority Queue:

A priority queue is a special type of queue in which each element is associated with a priority value. And, elements are served on the basis of their priority. That is, higher priority elements are served first.

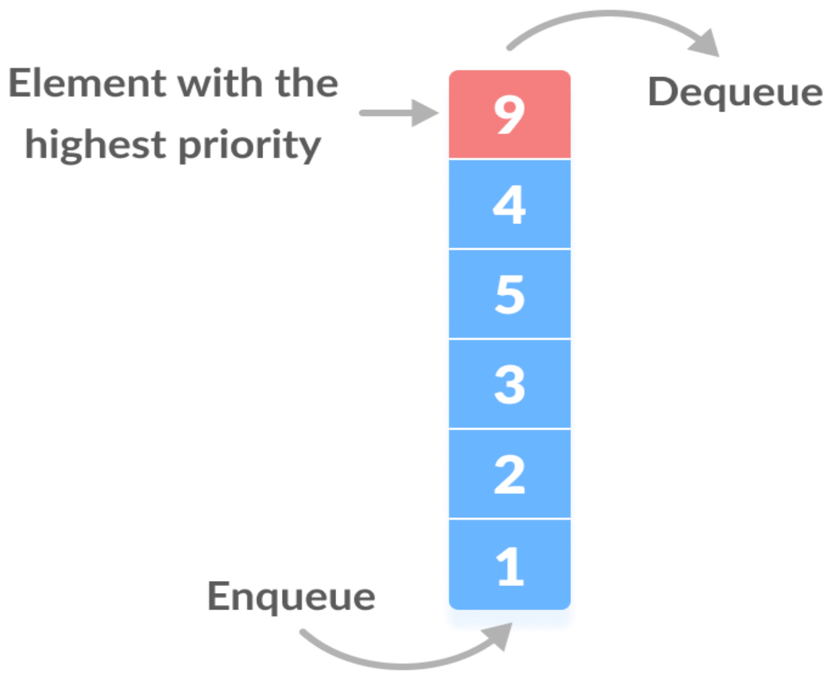
However, if elements with the same priority occur, they are served according to their order in the queue.

Assigning Priority Value

Generally, the value of the element itself is considered for assigning the priority. For example,

The element with the highest value is considered the highest priority element. However, in other cases, we can assume the element with the lowest value as the highest priority element.

We can also set priorities according to our needs.



Difference between Priority Queue and Normal Queue
In a queue, the first-in-first-out rule is implemented whereas, in a priority queue, the values are removed based on priority. The element with the highest priority is removed first.

HEAP Sort

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

HEAP Sort Algorithm

First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heap is greater than 1.

- Build a heap from the given input array.
- Repeat the following steps until the heap contains only one element:
  - Swap the root element of the heap (which is the largest element) with the last element of the heap.
  - Remove the last element of the heap (which is now in the correct position).
  - Heapify the remaining elements of the heap.
- The sorted array is obtained by reversing the order of the elements in the input array.

```
HeapSort(arr):
    n = length of arr

    // Build a max heap
    for i from n/2 - 1 to 0:
        heapify(arr, n, i)

    // Extract elements one by one from the heap
    for i from n - 1 to 0:
        swap arr[0] with arr[i]
        heapify(arr, i, 0)

heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    // Check if left child is larger than the root
    if left < n and arr[left] > arr[largest]:
        largest = left

    // Check if right child is larger than the root or left child
    if right < n and arr[right] > arr[largest]:
        largest = right

    // If largest is not the root
    if largest != i:
        swap arr[i] with arr[largest]
        heapify(arr, n, largest)
```

Application of Heaps (Huffman Tree):
Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. It compresses data very effectively saving from 20% to 90% memory, depending on the characteristics of the data being compressed. We consider the data to be a sequence of characters. Huffman's greedy algorithm uses a table giving how often each character occurs (i.e., its frequency) to build up an optimal way of representing each character as a binary string.
These are called fixed-length codes. If all characters were used equally often, then a fixed-length coding scheme is the most space efficient method. But such thing isn’t possible in real word. If some characters are used more frequently than others, is it possible to take advantage of this fact and somehow assign them shorter codes? The price could be that other characters require longer codes, but this might be worthwhile if such characters appear rarely enough. Huffman coding variable-length codes approaches. While it is not commonly used in its simplest form for file compression, one motivation for studying Huffman coding is because it provides type of tree structure referred to as a search tree.

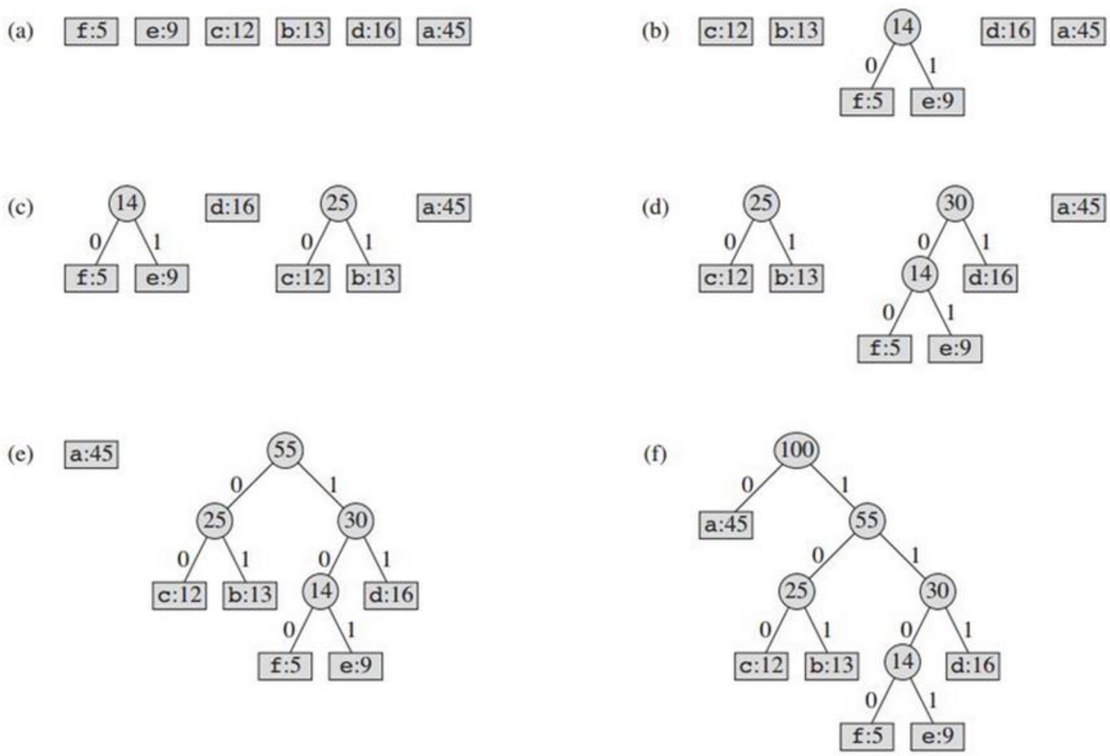
There are mainly two major parts in Huffman Coding

- Build a Huffman Tree from input characters.
- Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree (Compression Technique)

1. The technique works by creating a binary tree of nodes. Tree can stored in a regular array, the size of which depends on the number of symbols, n. A node can either be a leaf node or an internal node. Initially all nodes are leaf nodes, which contain the symbol itself, its frequency and optionally, a link to its child nodes. As a convention, bit '0' represents left child and bit '1' represents right child. Priority queue is used to store the nodes, which provides the node with lowest frequency when popped. The process is described below:

a. Create a leaf node for each symbol and add it to the priority queue.
b. While there is more than one node in the queue:
c. Remove the two nodes of highest priority from the queue.
d. Create a new internal node with these two nodes as children and with frequency equal to the sum of the two nodes' frequency.
e. Add the new node to the queue.
f. The remaining node is the root node and the Huffman tree is complete.



Decompression Technique:
The process of decompression is simply a matter of translating the stream of prefix codes to individual byte value, usually by traversing the Huffman tree node by node as each bit is read from the input stream. Reaching a leaf node necessarily terminates the search for that particular byte value. The leaf value represents the desired character. Usually the Huffman Tree is constructed using statistically adjusted data on each compression cycle, thus the reconstruction is fairly simple. Otherwise, the information to reconstruct the tree must be sent separately.

```
Procedure HuffmanDecompression(root, S):   // root represents the root of
Huffman Tree
n := S.length                              // S refers to bit-stream to be decompressed
for i := 1 to n
   current = root
   while current.left != NULL and current.right != NULL
      if S[i] is equal to '0'
         current := current.left
      else
         current := current.right
      endif
      i := i+1
   endwhile
   print current.symbol
endfor
```

Exercises:

1. Consider a tree 7,1,6,2,5,9,10,2. You are tasked with finding both the min heap and max heap of this tree.
2. Consider another tree 35,33,42,10,14,19,27,44,26,31 Make the tree into a min heap and delete its root node and rebalance the tree to max heap and print the tree in a sorted output.
3. Assume you are tasked to schedule computer tasks. Given task from t1 to tn you must schedule them in the given manner down below.
   Note:
   Each task that is being pooled comes with a priority (generate this priority randomly between values 1 to 10). The value with the highest priority gets the first treatment and then subsequent nodes will get later priorities. Once the tree is built up delete the nodes accordingly to the priority (Max to min) while also printing the order.
4. Encode the String BAI-3B via Huffman and display the encoded string
5. Decode the Encoded string back to the original string