MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Second Year B. Tech

Academic Year 2022-2023. (SEM-II)

Subject: Advance Data Structures Laboratory

Assignment 12

Assignment Title: Implement Heap sort.

Aim: Implement heap sort.

Prerequisite:

1. Basic concepts of heap.
2. Basic concept of max heap and min heap.

Objectives:

Implement a Program in C++ for the following operations:

1. Create an array of elements.
2. Sort elements in array using heap sort.

Outcomes:

Upon Completion of the assignment the students will be able to

1. Create array of elements using data structure.

2. Sort elements in array using heap sort.

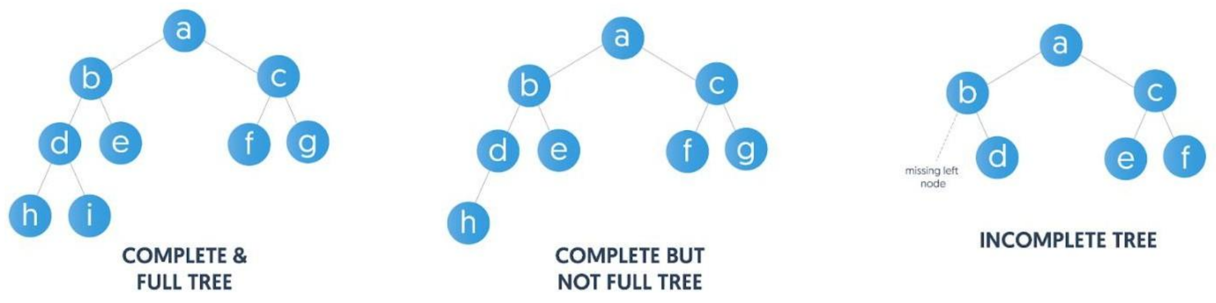
.

Theory:

A heap is a complete binary tree, and the binary tree is a tree in which the node can have the utmost two children. A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node, should be completely filled, and all the nodes should be left-justified.

Heap is a tree-based data structure in which all the tree nodes are in a particular order, such that the tree satisfies the heap properties (that is, a specific parent-child relationship is followed throughout the tree).

A heap data structure where the tree is a complete binary tree is referred to as a binary heap.



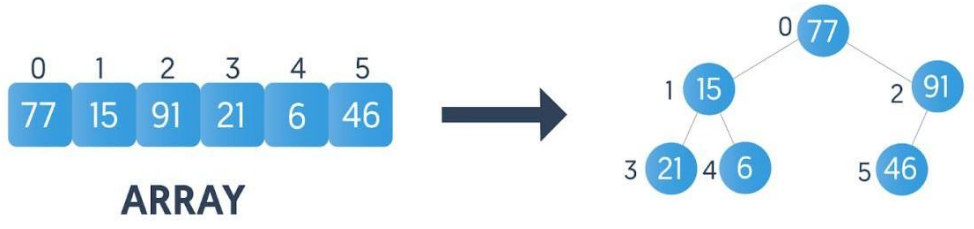
A complete binary tree is a binary tree in which:

- ➢ All levels except the bottom-most level are completely filled.
- ➢ All nodes in the bottom-most level are as far left as possible.
- ➢ The last level may or may not be completely filled.
- ➢ A full binary tree is a binary tree where every node has 0 or 2 children.

Properties of a Binary Heap

1. They are complete binary trees: This means all levels are totally filled (except maybe the last level), and the nodes in the last level are as left as possible. This property makes arrays a suitable data structure for storing binary heaps.

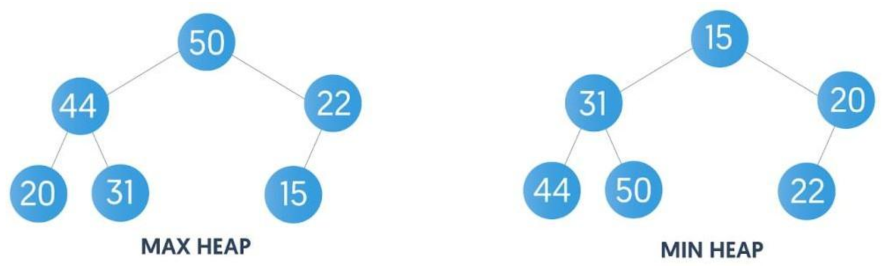
We can easily calculate the indices of a node's children. So, for parent index i, the left child will be found at index 2*i+1, and the right child will be found at index 2*i+2 (for indices that start with 0). Similarly, for a child at index i, its parent can be found at index floor((i-1)/2).



2. Heaps are mainly of two types — max heap and min heap:

In a max heap, the value of a node is always >= the value of each of its children.

In a min heap, the value of a parent is always <= the value of each of its children.



3. Root element: In a max heap, the element at the root will always be the maximum. In a min heap, the root element will always be the smallest. The heap sort algorithm takes advantage of this property to sort an array using heaps.

Heap sort

Heapsort is a popular and efficient sorting algorithm. The concept of heap sort is to eliminate the elements one by one from the heap part of the list, and then insert them into the sorted part of the list. Heapsort is the in-place sorting algorithm.

Heap sort processes the elements by creating the min-heap or max-heap using the elements of the given array. Min-heap or max-heap represents the ordering of array in which the root element represents the minimum or maximum element of the array.

Heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the smallest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the minimum.

Heap Sort :

Heap sort is an efficient comparison-based sorting algorithm that:

- Creates a heap from the input array.
- Then sorts the array by taking advantage of a heap's properties.

Heapify Method

Before going into the workings of heap sort, we'll visualize the array as a complete binary tree. Next, we turn it into a max heap using a process called heapification.

The brilliance of heapification lies in the following fact:

If all the subtrees in a binary tree are MaxHeaps themselves, the whole tree is a MaxHeap.

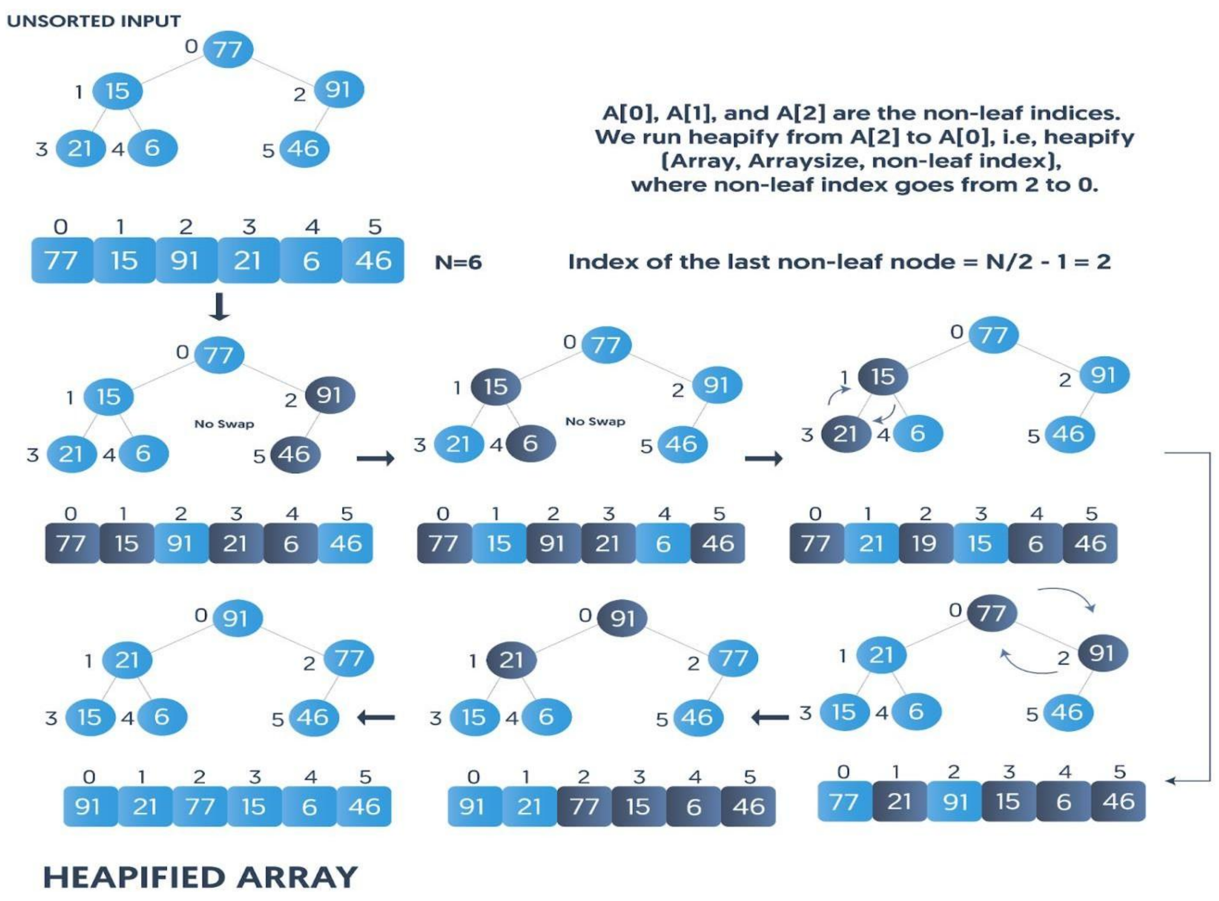
- One way to implement this idea would be:
- Start at the bottom of the tree.
- Iterate through all the nodes as we travel to the top.
- At each step, ensure that the node and all its children form a valid max heap.

If we successfully do that, we will have transformed the whole binary tree into a valid MaxHeap after processing all the nodes.

One way to optimize this process is by ignoring all the leaf nodes since they don't have any children:

- Go to the right-most node in the second bottom-most layer, which has any children.
- Process that right-most node to make sure it forms a MaxHeap with its children
- Traverse to the node to its left and repeat the process.
- At the end of the level, we jump to the right-most node of the level above it.

This journey ends when we eventually reach the topmost node and process it.


A[0], A[1], and A[2] are the non-leaf indices. We run heapify from A[2] to A[0], i.e, heapify (Array, Arraysize, non-leaf index), where non-leaf index goes from 2 to 0.

Index of the last non-leaf node = N/2 - 1 = 2

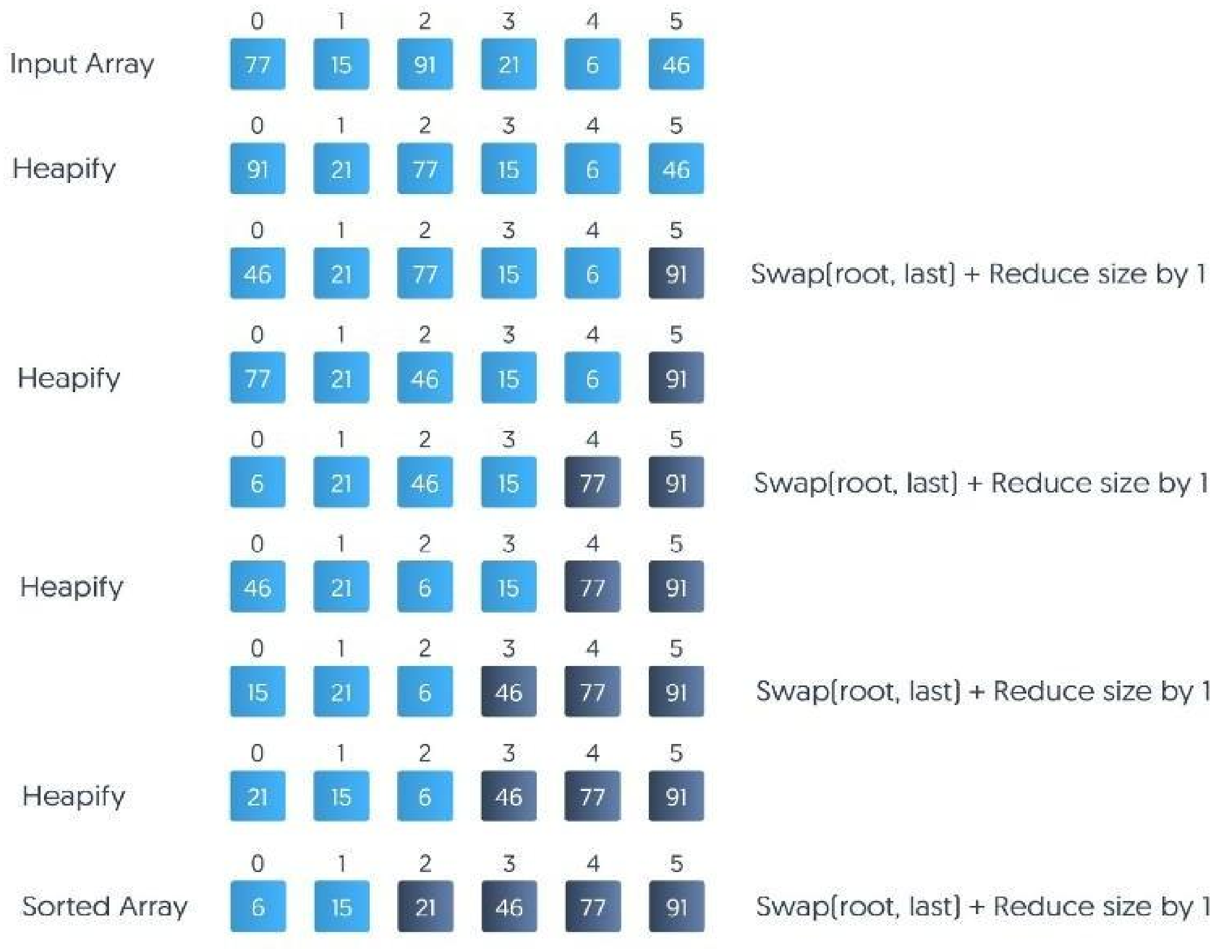
How Does Heap Sort Work?

Now that we've learned how to create a heap from an array using the heapify method, we will look into using the heap to sort the array.

After the heap formation using the heapify method, the sorting is done by:

- ➢ Swapping the root element with the last element of the array and decreasing the length of the heap array by one. (In heap representation, it is equivalent to swapping the root with the bottom-most and right-most leaf and then deleting the leaf.)
- ➢ Restoring heap properties (reheapification) after each deletion, where we need to apply the heapify method only on the root node. The subtree heaps will still have their heap properties intact at the beginning of the process.
- ➢ Repeating this process until every element in the array is sorted: Root removal, its storage in the position of the highest index value used by the heap, and heap length decrement.

- On a max heap, this process will sort the array in ascending order.
- On a min heap, this process will sort in descending order.


Input Array 0 1 2 3 4 5 | 77 15 91 21 6 46
Heapify 0 1 2 3 4 5 | 91 21 77 15 6 46
0 1 2 3 4 5 | 46 21 77 15 6 91 Swap[root, last] + Reduce size by 1
Heapify 0 1 2 3 4 5 | 77 21 46 15 6 91
0 1 2 3 4 5 | 6 21 46 15 77 91 Swap[root, last] + Reduce size by 1
Heapify 0 1 2 3 4 5 | 46 21 6 15 77 91
0 1 2 3 4 5 | 15 21 6 46 77 91 Swap[root, last] + Reduce size by 1
Heapify 0 1 2 3 4 5 | 21 15 6 46 77 91
Sorted Array 0 1 2 3 4 5 | 6 15 21 46 77 91 Swap[root, last] + Reduce size by 1

The process above ends when heap size = 2 because a two-element heap is always considered sorted.

So basically, the heap sort algorithm has two parts that run recursively till heap size >= 2:

- Creating a heap from the currently unsorted elements of the array.
- Swapping the root element with the last element of the heap (right-most leaf node)
- Reducing heap size by 1.

Conclusion:

We have created array and sort elements in array using heap sort.

Frequently Asked Questions:

1. What is heap tree, min heap, max heap?
2. List out properties of heap tree.
3. Sort following element using heap sort
   01, 04, 03, 08, 07, 02

Name: Sourav Shailesh Toshniwal
Class: SY CSE - I
Roll no: 2213047
Subject: ADSL

Assignment - 12

1. what is heap tree, min heap, man heap?

Ans Heap tree: It is a specialized tree-based data structure in which the parent node is either greater than or less then the child nodes, depending on whethere it is a max heap or a min heap.
Min heap: It is a binary tree in which the parent node is always less than or equal to its child nodes.
Max heap: It ~~is~~ a binary tree in which the parent node is always greater then or equal to its child nodes.

2. List out properties of heap tree.

Ans - Complete binary tree structure.
- Satisfy either the max-heap or min-heap property.
- Root node contains max or min element.
- Parent - child relationships determined by array indexing.
- Height is logarithmic with respect to no. of elements
- Efficient time complexity for insertion, deletion, and search operations.

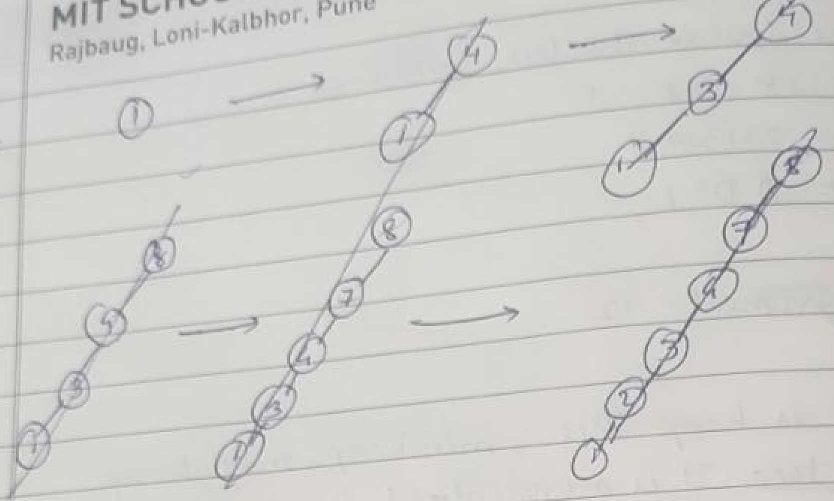
3. Sort following element using heap sort.

01, 04, 03, 08, 07, 02

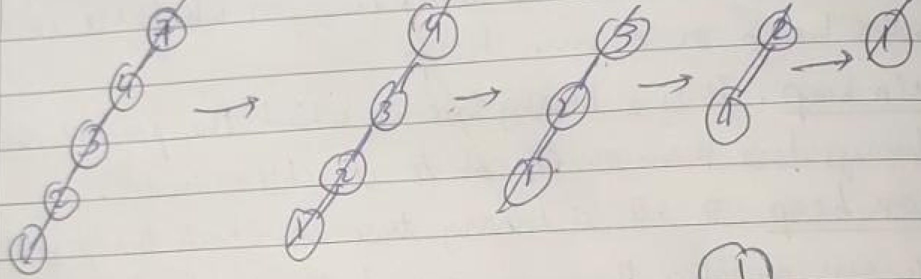
MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

MIT-ADT UNIVERSITY

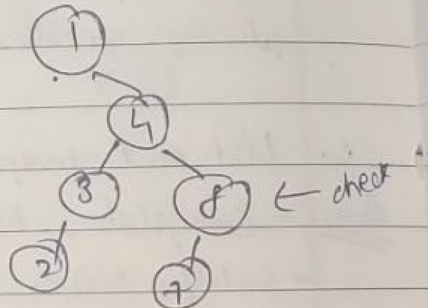
Ans

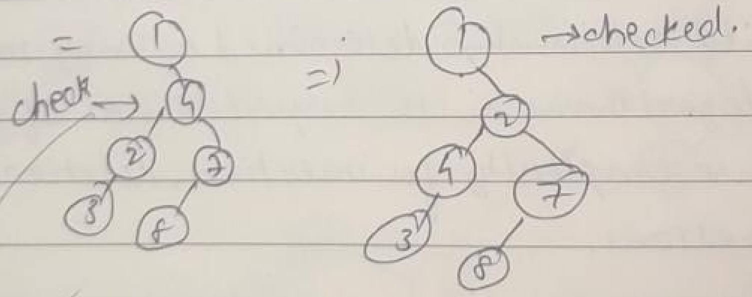


Now delete max heap node,



→ ~~8, 7, 4, 3, 2, 1.~~





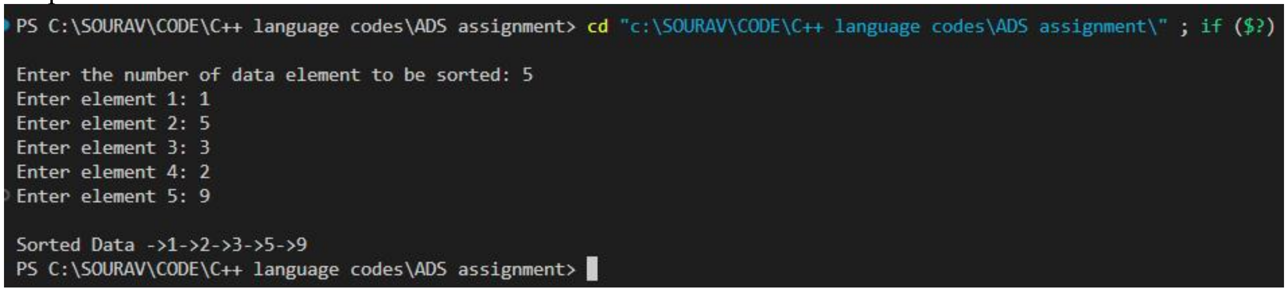
Code:

```
#include <iostream>
using namespace std;
void MaxHeapify(int a[], int i, int n)
{
    int j, temp;
    temp = a[i];
    j = 2*i;   //(left child)
    while (j <= n)
    {
        if (j < n && a[j+1] > a[j])
        j = j+1;
        if (temp > a[j])
            break;
        else if (temp <= a[j])
        {
            a[j/2] = a[j];
            j = 2*j;
        }
    }
    a[j/2] = temp;
    return;
}
void HeapSort(int a[], int n)
{
    int i, temp;
    for (i = n; i >= 2; i--)
    {
        temp = a[i];
        a[i] = a[1];
        a[1] = temp;
        MaxHeapify(a, 1, i - 1);
    }
}
void Build_MaxHeap(int a[], int n)
{
    int i;
    for(i = n/2; i >= 1; i--)
        MaxHeapify(a, i, n);
}
int main()
{
    int n, i;
    cout<<"\nEnter the number of data element to be sorted: ";
    cin>>n;
    n++;
    int arr[n];
    for(i = 1; i < n; i++)
    {
        cout<<"Enter element "<<i<<": ";
        cin>>arr[i];
    }
    Build_MaxHeap(arr, n-1);
    HeapSort(arr, n-1);
    cout<<"\nSorted Data ";

    for (i = 1; i < n; i++)
        cout<<"->"<<arr[i];

    return 0;
}
```

Output:



```
PS C:\SOURAV\CODE\C++ language codes\ADS assignment> cd "c:\SOURAV\CODE\C++ language codes\ADS assignment\" ; if ($?)

Enter the number of data element to be sorted: 5
Enter element 1: 1
Enter element 2: 5
Enter element 3: 3
Enter element 4: 2
Enter element 5: 9

Sorted Data ->1->2->3->5->9
PS C:\SOURAV\CODE\C++ language codes\ADS assignment>
```