Lecture - 76 (Heaps)

CLASSTIME | Pg. No.
Date / /

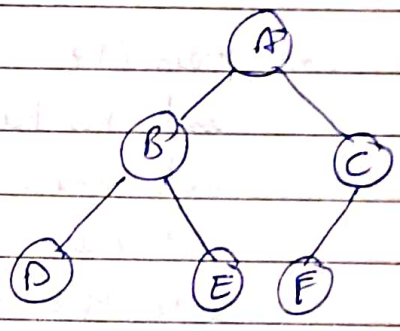
⭐ Binary Heap → Min Heap
Max Heap

1) Binary tree which is a complete binary tree (all levels will be completely filled except may be last level & last level has keys as left as possible).

2) Can be • Min Heap :- Parent Node ≤ ChildNode
• Max Heap :- ParentNode ≥ ChildNode

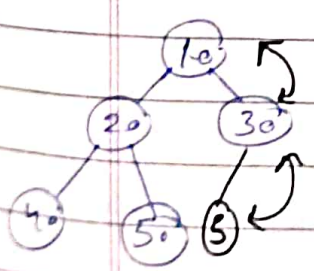
⭐ Representation of Heap Using an Array.

1) Root node will be at index 1
2) For Node at index i
left child → 2*i
right child → 2*i+1
3) For node at index i
Parent node → i/2

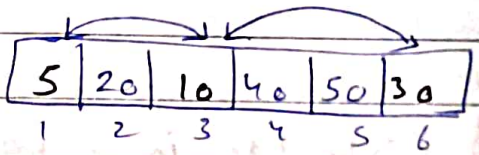


A	B	C	D	E	F
1	2	3	4	5	6

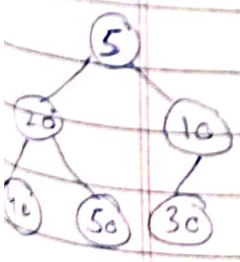
⭐ Insertion in a min heap :-



n = 5
Best case → B.C = O(1)
Worst Case → O(log n) (height)



Parent(5) = arr[6/2] = arr[3]
= 30
Parent(5) = arr[3/2] = arr[1]
= 10



⇒ Last Pos. pe insert karo or Parent se compare kro. Agr Vo Heapify ki property ko Violate kr rha hai to swap karo.

Code

```
#include<iostream>
using namespace std;
int const N=1e3

Void insertMinHeap(int minHeap[], int &size, int Value){
    size++;
    minHeap[size] = value;
    int curr = size;

    while(curr/2>0 && minHeap[curr/2] > minHeap[curr]){
        swap(minHeap[curr/2], minHeap[curr]);
        curr = curr/2;
    }
}
int main(){
    int minHeap[N] = {-1, 10, 20, 30, 40, 50};
    int size = 5;
    int value = 5;
    insertMinHeap(minHeap, size, Value);
    for(int i=0; i<=size; i++){
        cout<<minHeap[i]<<" ";
    } cout<<endl;
    return 0;
}
```

(*) Insertion in a max heap.

All same → Now Parent should be greater than child.

Condition:- while(curr/2 >0 && arr[curr/2] < arr[curr]){
swap(arr[curr/2], arr[curr])
Curr = Curr/2;
}

CLASSTIME Pg. No.
Date / /

(4) Deletion in a min heap:

(1) Swap Root Node and last Node
(2) Del. Last Node.
(3) Heapify the Tree :→ curr = 3
left child = 7
Right child =
Min child = ⇒ Swap with curr until curr reach the leaf Node (right Pos.)

Code

```
Void removeMinHeap(int minHeap[], int &size){

    minHeap[1] = minHeap[size];
    size--;
    int curr = 1;
    while(2*curr <= size){
        int leftchild = 2*curr;
        int rightChild = 2*curr+1;
        int minchild = leftchild;
        if (rightchild <= size && minHeap[rightchild] < minHeap[leftchild]){
            minchild = rightchild;
        }
        if(minHeap[minchild] >= minHeap[curr]){
            return;
        }
        swap(minHeap[minchild], minHeap[curr]);
        curr = minchild;
    }
}
```

⭐ Deletion in Max Heap :-

All same → But swapping of curr with max child

⭐ Heapify :-

→ Arranging the nodes in correct order so that they follow properties of minHeap / maxHeap.

⭐ Using heapify to generate a min Heap

last Parent Node

→ leaf nodes start from (n/2 + 1), last non-leaf node → n/2

⇒ Code ⇒

```
Void Heapify(int arr[], int &size, int curr) {

    while (2*curr <= size) {

        int leftchild = 2*curr;
        int rightchild = 2*curr + 1;
        int minchild = leftchild;

        if (rightchild <= size && arr[rightchild] < arr[minchild]) {
            minchild = rightchild;
        }
        if (arr[minchild] >= arr[curr]) {
            return;
        }
        swap(arr[minchild], arr[curr]);
        curr = minchild;
    }
}
```

CLASSTIME Pg No:
Date / /

```
int main() {
    int arr[N] = {-1, 60, 10, 20, 50, 5, 20, 70};
    int size = 7;
    for (int i = size/2; i > 0; i--) {
        // i -> parent node
        heapify(arr, size, i);
    }
    for (int i = 1; i <= size; i++) {
        cout << arr[i] << " ";
    } cout << endl;

}
```

(+) Using heapify to generate a max heap T.C = O(log N)

⇒ All same

⇒ Find max child instead of min child.

⇒ Introduction to heapsort

T.C = O(n log n)

2 Steps

- Convert the array into heap data structure using heapify.
- One by one delete the root node of the heap and replace it with the last node in the heap and heapify the root of the heap. Repeat this process till the ~~weight~~ size of the heap is greater than 1.

```
#include <iostream>
using namespace std;
int const N = 1e3;
```

```
Void heapify(int arr[], int n, int curr){

    while (2*curr <= n){

        int leftchild = 2*curr;
        int rightchild = 2*curr+1;
        int maxchild = leftchild;

        if (rightchild <= n && arr[rightchild] > arr[maxchild]){
            maxchild = rightchild;
        }
        if (arr[maxchild] <= arr[curr]){
            return;
        }
        swap(arr[maxchild], arr[curr])
        curr = maxchild;
    }
}
Void remove(int arr[], int &size){
    swap(arr[i], arr[size]);
    size--;
    heapify(arr, size, 1);
}
Void heapSort(int arr[], int n){
    //1. heapify all parent nodes into max heaps
    for(int i = n/2; i>0; i--){
        heapify(arr, n, i);
    }
    //2. keep deleting elements from max heap until size becomes 0.
    while (n>0){
        remove(arr, n);
    }
}
```

```
int main(){
    int n=7;
    int arr[N]={-1,60,10,80,50,5,20,70};
    heapSort(arr,n);
    for(int i=1;i<=n;i++){
        cout<<arr[i]<<" ";
    }cout<<endl;
    return 0;
}
```

Q34 Given two binary max heaps as arrays, the task is to merge the given heaps.

for(auto ele:a){
---
this b

Sol: ① [Arr.1, Arr 2] Given
② Merge Both arrays in single Array [Normally]
③ Heapify it.

// Previously we have done the concept by using array with 1 base indexing.

// If we use array with zero base indexing then { leftchild = 2*curr+1; } & { rightchild = 2*curr+2; }

```
Vector<int> mergeheaps(Vector<int>&a, Vector<int>&b, int n, int m){
    Vector<int> mergeHeap;
    for(auto ele:a){
        mergeHeap.push_back(ele);
    }
    for(auto ele:b){
        mergeHeap.push_back(ele);
    }
    int lastParentNode=(n+m)/2-1;
```

Make function in class

```
for(int i = lastParentNode; i>=0, i--) {
        heapify(mergedHeap, n+m, i);
    }
}
return mergedHeap;
};
```

Q 1→ Given an array arr[] and an integer k where k is smaller than size of array, the task is to find the k^{th} smallest element in the given array. It is given that all array elements are distinct.

Note:- l and r denotes the starting and ending index of the array.

input:- N=6

arr[] = 7 10 4 3 20 15

k = 3

Output: 7

Sol→ Method ① → Brute Force Sol.

① Sort the array

② Get the k^{th} element.

T.C → O(N) (k-1) log N

Method ② → ① Make a min heap

② Remove k-1 nodes

③ Root Node will be the k^{th} Node.

```
int kthSmallestElement(int arr[], int size, int k){
    //1. Create min heap
    for(int i = size/2 - 1; i>=0; i--){
        heapify(arr, size, i);          // Same prev funct
    }
```

```
//2. Remove k-1 elements
    k-=1;
    while (k--) {
        remove(arr, size);          //Same Prev. Funct.
    }
    return arr[0];
}
```

Q 4> Given an array A[] of N positive integers and two positive integers k1 and k2. Find the sum of all elements between $k1^{th}$ and $k2^{th}$ smallest elements of the array. It may be assumed that (1 <= k1 < k2 <= n).

Input : N = 7
A[] = {20, 8, 22, 4, 12, 10, 14}
k1 = 3, k2 = 6

Output : 26

Sol:-

```
int remove(int arr[], int & size){
    int removedValue = arr[0];
    arr[0] = arr[size-1];
    size--;
    heapify(arr, size, 0);
    return removedValue;
}
int sumofsmallest( int arr[], int size, int k1, int k2){
    //1. min heap
    for (int i = size/2 - 1; i>=0; i--) {
        heapify(arr, size, i);
    }
    int Sum = 0;
    int elements = k2 - k1 - 1;
```

```
// 2. remove k1 elements
    while (k--) {
    remove (arr, size);
    }
// 3. calculate sum by removing elements.
    while (elements--) {
        sum += remove (arr, size);
    }
    return sum;
}
int main () {
    int arr[7] = {20, 8, 22, 4, 12, 10, 14};
    int size = 7;
    int k1 = 3;
    int k2 = 6;
    cout << sumOfSmallest (arr, size, k1, k2) << endl;
    return 0;
}
```