Name-Harshita Totala Roll No-SEITA14 Div-IT-A

ASSIGNMENT-9
Heap Sort

AIM: Implement the heap sort to sort given set of values using max or min heap.

Objectives:

1.Understand the concept of Heap Sort.

2. Understand the types of heap sort and its applications.

THEORY :

There are several types of heaps, however in this chapter, we are going to discuss binary heap. A **binary heap** is a data structure, which looks similar to a complete binary tree. Heap data structure obeys ordering properties discussed below. Generally, a Heap is represented by an array. In this chapter, we are representing a heap by ***H***.

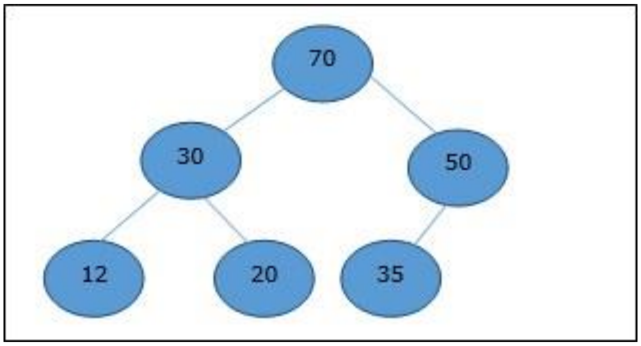
As the elements of a heap is stored in an array, considering the starting index as ***1***, the position of the parent node of i^{th} element can be found at $\lfloor i/2 \rfloor$. Left child and right child of i^{th} node is at position ***2i*** and ***2i + 1***.

A binary heap can be classified further as either a ***max-heap*** or a ***min-heap*** based on the ordering property.

Max-Heap

In this heap, the key value of a node is greater than or equal to the key value of the highest child.

Hence, $H[Parent(i)] \geq H[i]$

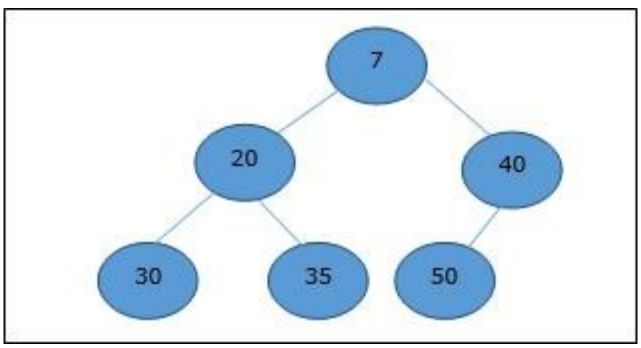


Min-Heap

In mean-heap, the key value of a node is lesser than or equal to the key value of the lowest child.

Hence, ***H[Parent(i)] ≤ H[i]***

In this context, basic operations are shown below with respect to Max-Heap. Insertion and deletion of elements in and from heaps need rearrangement of elements. Hence, **Heapify** function needs to be called.

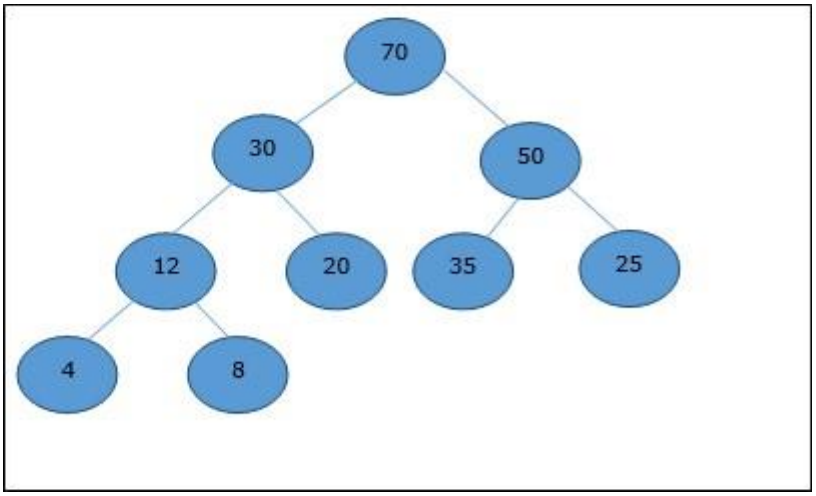


Array Representation

A complete binary tree can be represented by an array, storing its elements using level order traversal.

Let us consider a heap (as shown below) which will be represented by an array **H**.

Name-Harshita Totala Roll No-SEITA14 Div-IT-A



Considering the starting index as **0**, using level order traversal, the elements are being kept in an array as follows.

Index	0	1	2	3	4	5	6	7	8
elements	70	30	50	12	20	35	25	4	8

In this context, operations on heap are being represented with respect to Max-Heap.

To find the index of the parent of an element at index **i**, the following algorithm ***Parent (numbers[], i)*** is used.

Algorithm: Parent (numbers[], i)

```
if i == 1     return NULL  else
[i / 2]
```

The index of the left child of an element at index **i** can be found using the following algorithm, ***Left-Child (numbers[], i)***.

Algorithm: Left-Child (numbers[],
i)
```
  If 2 * i ≤ heapsize     return
[2 * i]  else     return NULL
```

The index of the right child of an element at index **i** can be found using the following algorithm, ***Right-Child(numbers[], i)***.

```
Algorithm: Right-Child (numbers[], i)
if 2 * i < heapsize     return [2 * i
+ 1]  else     return NULL
```

Max-Heapify

Heapify method rearranges the elements of an array where the left and right sub-tree of **i**[th] element obeys the heap property.

Algorithm:

```
Max-Heapify(numbers[], i)
leftchild := numbers[2i]  rightchild
:= numbers [2i + 1]
if leftchild ≤ numbers[].size and numbers[leftchild] >
numbers[i]     largest := leftchild  else     largest
:= i
if rightchild ≤ numbers[].size and numbers[rightchild] >
numbers[largest]     largest := rightchild  if largest ≠
i
   swap numbers[i] with numbers[largest]
   Max-Heapify(numbers, largest)
```

Build-Max-Heap

When the provided array does not obey the heap property, Heap is built based on the following algorithm ***Build-Max-Heap (numbers[])***.

Algorithm:

```
Build-Max-Heap(numbers[])
numbers[].size := numbers[].length  fori
= ⌊ numbers[].length/2 ⌋ to 1 by -1
   Max-Heapify (numbers[], i)
```

Max-Heap-Sort

Algorithm:

1.Build Max Heap from unordered array;2.Find maximum element A[1];
3.Swap elements A[*n*] and A[1]:
now max element is at the end of the array!
4.Discard node *n* from heap(by decrementing heap-size variable)
5.New root may violate max heap property, but itschildren are max heaps. Run max_heapify to fix this.
6.Go to Step 2 unless heap is empty.

Min-Heapify

Heapify method rearranges the elements of an array where the left and right sub-tree of **i**th element obeys the heap property.

Algorithm:

```
Min-Heapify(numbers[], i) kg
leftchild := numbers[2i]  rightchild
:= numbers [2i + 1]
if leftchild ≤ numbers[].size and numbers[leftchild]<
numbers[i]
   smallest := leftchild
else     smallest := i
if rightchild ≤ numbers[].size and numbers[rightchild] <
numbers[smallest]     smallest := rightchild  if
smallest ≠ i
   swap numbers[i] with numbers[smallest]
   Min-Heapify(numbers, smallest)
```

Build-Min-Heap

When the provided array does not obey the heap property, Heap is built based on the following algorithm ***Build-Min-Heap (numbers[])***.

Algorithm:

```
Build-Min-Heap(numbers[])
numbers[].size := numbers[].length  fori
= ⌊ numbers[].length/2 ⌋ to 1 by -1
```

```
Min-Heapify (numbers[], i)
```

Min-Heap-Sort

Algorithm:

1.Build Min Heap from unordered array;2.Find maximum element A[1]; 3.Swap elements A[*n*] and A[1]:
now max element is at the end of the array!
4.Discard node *n* from heap(by decrementing heap-size variable)
5.New root may violate min heap property, but its children are min heaps. Run min_heapify to fix this.
6.Go to Step 2 unless heap is empty.

Name-Harshita Totala Roll No-SEITA14 Div-IT-A

CODE-

```
#include <iostream>
using namespace std;
void maxHeapify(int a[], int i, int n){
 int j, temp;
 temp = a[i];
 j = 2*i;
 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 min_heapify(int a[], int i, int n){
 int j, temp;
 temp=a[i];
 j=2*i;
 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 build_maxheap(int a[], int n){
```

```
 int i;
 for(i=n/2 ; i>=1; i--){
 maxHeapify(a,i,n);
 }
}
void max_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_minheap(int a[], int n){
 int i;
 for(i=n/2; i>=1; i--){
 min_heapify(a,i,n);
 }
}

void min_HeapSort( int a[], int n){
 int i, temp;
 for(i=n; i>=2; i--){
 temp = a[i];
 a[i] = a[1];
 a[1] = temp;
 min_heapify(a, 1, i-1);
 }
}
void print(int arr[], int n){
 cout<<"Sorted data : ";
 for(int i=1; i<=n; i++){
 cout<<"->"<<arr[i];
 }
 return;
}
int main(){
 int n, i, ch;
 cout<<"\t*** Heap Sort ***\n"<<endl;
 cout<<"Enter the number of elements to be sorted: " ;
 cin>>n;
 int arr[n];
 for(i=1; i<=n; i++) {
```

```
 cout<<"Enter element "<<i<<" :";
 cin>>arr[i];
 }
 do{
 cout<<"\n\n1]Heap sort using Max Heap";
 cout<<"\n2]Heap sort using Min heap";
 cout<<"\n3]Exit";
 cout<<"\nEnter your choice: ";
 cin>>ch;
 switch(ch){

case 1:
build_maxheap(arr, n);
max_HeapSort(arr, n);
print(arr, n);

break;
case 2:
 build_minheap(arr, n);
 min_HeapSort(arr, n);
 print(arr, n);
 break;
 case 3:
 cout<<"\nProgram Exited Successfully !!"<<endl;
}
 }while(ch!=3);
return 0;
}
```

Name-Harshita Totala Roll No-SEITA14 Div-IT-A

OUTPUT-

*** Heap Sort ***

Enter the number of elements to be sorted: 5

Enter element 1 :34

Enter element 2 :76

Enter element 3 :55

Enter element 4 :21

Enter element 5 :8

1]Heap sort using Max Heap

2]Heap sort using Min heap

3]Exit

Enter your choice: 1

Sorted data : ->76->55->34->21->8

1]Heap sort using Max Heap

2]Heap sort using Min heap

3]Exit

Enter your choice: 2

Sorted data : ->8->21->34->55->76

1]Heap sort using Max Heap

2]Heap sort using Min heap

3]Exit

Enter your choice: 3

Program Exited Successfully !!

Conclusion :

We have constructed max and min heap sort to sort array and perform deletion operation on it.