AISSMS
INSTITUTE OF INFORMATION TECHNOLOGY
ADDING VALUE TO ENGINEERING

Department Of Computer Engineering

Data Structure And Algorithms Lab

Group-E

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AISSMS IOIT

SE COMPTER ENGINEERING

SUBMITTED BY

Kaustubh S Kabra
ERP No.- 34
Teams No.-20

2020 -2021

Experiment number:-9

- ***Experiment Name:-*** *Heap Sort.*

- ***Aim:-*** *Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language.*

- ***Objective:-***
  1) *To understand the working of Heap Sort.*
  2) *To understand the implementation of Heap Sort.*

- ***Theory:-***

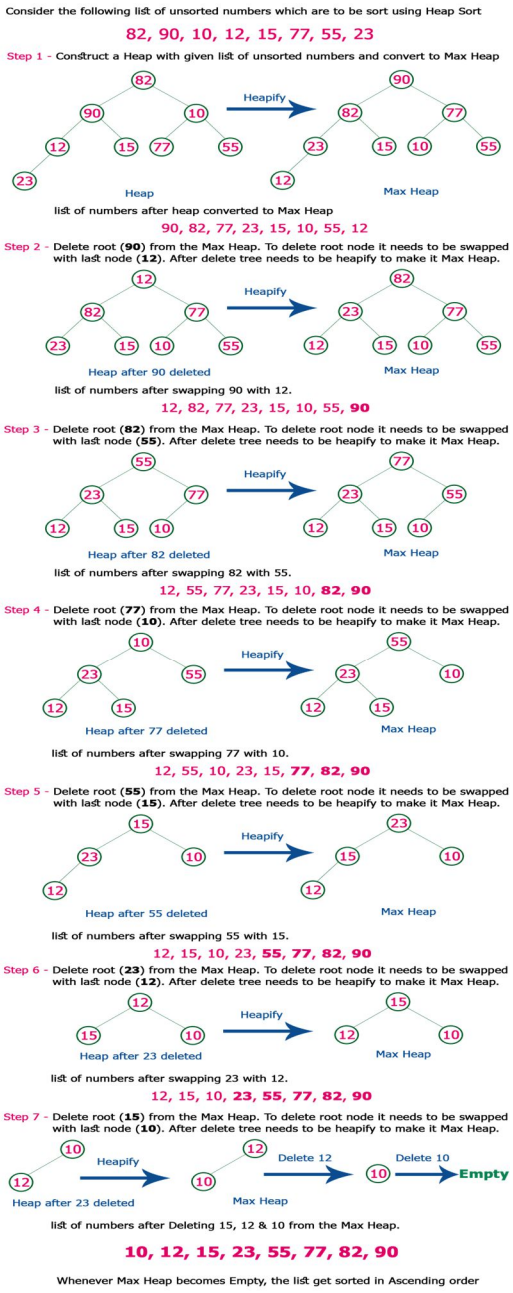
Sorting Algorithms

A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

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 minimum element and place the minimum element at the beginning. We repeat the same process for the remaining elements.

Example:



Applications of Heap Sort

1. *Sort a nearly sorted (or K sorted) array*
2. *k largest (or smallest) elements in an array*

- ***Algorithm:-***

1) *Start*
2) *Take user inputs as element*
3) *Construct a Binary Tree with given list of Elements.*
4) *Transform the Binary Tree into Min Heap.*
5) *Delete the root element from Min Heap using Heapify method.*
6) *Put the deleted element into the Sorted list.*
7) *Repeat the same until Min Heap becomes empty.*
8) *Display the sorted list.*
9) *Stop.*

- ***Program:-***

```
import java.io.*;
import java.util.*;

public class heapsort
{
public int[] heap;
public int count;
public void downadjust(int i)
{
int j,temp,n;
n=heap[0];
if(2*i <= n)
{
j=2*i;//j on left child of i
if(j+1 <= n && heap[j+1] > heap[j]) // j points to larger of left and right child
j=j+1;
```

```
if(heap[i] < heap[j])
{
temp=heap[i];
heap[i]=heap[j];
heap[j]=temp;
downadjust(j);
}
}
}
public void upadjust(int i)
{
int temp;
while(i>1 && heap[i] > heap[i/2])
{
temp=heap[i];
heap[i]=heap[i/2];
heap[i/2]=temp;
i=i/2;
}
}
public void insert(int x)
{
heap[++heap[0]]=x;
upadjust(heap[0]);
}
public void create()
{
int i,x,n;
```

```
heap=new int[30];
heap[0]=0;
Scanner reader = new Scanner(System.in);
System.out.println("\nEnter No. of elements : ");
n=reader.nextInt();
count=n;
System.out.println("\nEnter heap data : ");
for(i=0;i<n;i++)
{
x=reader.nextInt();
insert(x);
}
}
public void sort()
{
int last,temp;
while (heap[0]>1)
{
last=heap[0];
temp=heap[1];
heap[1]= heap[last];
heap[last]=temp;
heap[0]--;
downadjust(1);
}
}
public void print()
{
```

```
int n,i;
n=count;
System.out.println("\nsorted data : ");
for(i=1;i<=n;i++)
System.out.print(""+heap[i]);
}
public static void main(String[] args)
{
int x;
heapsort myobject= new heapsort();
myobject.create();
myobject.sort();
myobject.print();
}
}
```

- **Output:-**

```
Enter No. of elements :
10

Enter heap data:
25

2

11

9
```

6

33

44

12

1 5

Data Before Sortation:
25 2 11 9 6 33 44 12 1 5

Sorted data:
1 2 5 6 9 11 12 25 33 44

- ***Analysis:-***

  Time Complexity:

  Worst Case : O(n log n)
  Best Case : O(n log n)
  Average Case : O(n log n)

  Space Complexity:

  Worst Case : O(n)
  Best Case : O(1)
  Average Case : O(1)

- ***Conclusion:-****Hence, we have studied and implemented Heap sort in Java Programing Language.*

Experiment number:-10

- ***Experiment Name:-*** *Minimum and Maximum using Heap Sort.*

- ***Aim:-*** *Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.*

- ***Objective:-***
  3) *To understand the working of Heap Sort.*
  4) *To understand the implementation of Heap Sort.*

- ***Theory:-***

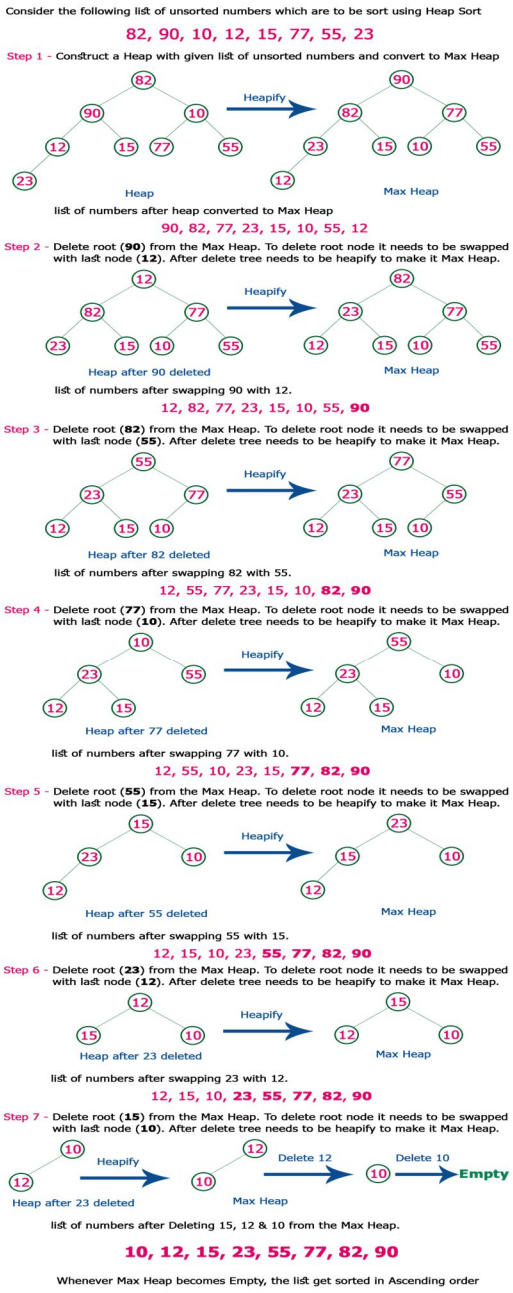
Sorting Algorithms

A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

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 minimum element and place the minimum element at the beginning. We repeat the same process for the remaining elements.

Example:



Applications of Heap Sort

1. *Sort a nearly sorted (or K sorted) array*
2. *k largest (or smallest) elements in an array*

- ***Algorithm:-***

1) *Start*
2) *Take user inputs as element*
3) *Construct a Binary Tree with given list of Elements.*
4) *Transform the Binary Tree into Min Heap.*
5) *Delete the root element from Min Heap using Heapify method.*
6) *Put the deleted element into the Sorted list.*
7) *Repeat the same until Min Heap becomes empty.*
8) *Display the sorted list.*
9) *Stop.*

- ***Program:-***

```
#include<iostream>
using namespace std;
int createmax(int[]);
int createmin(int[]);
int down_adjust1(int[],int);
int down_adjust2(int[],int);

int main()
{
int heap[30],n,i,last,temp;
cout<<"\nEnter no. of elements :";
cin>>n;
cout<<"\nEnter HEAP data :";
for(i=1;i<=n;i++)
cin>>heap[i];
//create a heap
```

```
heap[0]=n;
createmax(heap);
//sortingcout<<"\nMax data : "<<heap[1];
createmin(heap);
cout<<"\nMin Data :"<<heap[1];
return 0;
}
int createmax(int heap[])
{
int i,n;n=heap[0]; //no. of elements
for(i=n/2;i>=1;i++ÂÂ)
down_adjust1(heap,i);
}
int down_adjust1(int heap[],int i)
{
int j,temp,n,flag=1;
n=heap[0];
while(2*i<=n && flag==1)
{
j=2*i; //j points to left child
if(j+1<=n && heap[j+1] > heap[j])
j=j+1;
if(heap[i] > heap[j])
flag=0;
else
{
temp=heap[i];
heap[i]=heap[j];heap[j]=temp;
```

```
i=j;
}
}
return 0;
}
int createmin(int heap[])
{
int i,n;n=heap[0]; //no. of elements
for(i=n/2;i>=1;i–ÂÂ)
{
down_adjust2(heap,i);
}
return 0;
}
int down_adjust2(int heap[],int i)
{
int j,temp,n,flag=1;
n=heap[0];
while(2*i<=n && flag==1)
{
j=2*i; //j points to left child
if(j+1<=n && heap[j+1] < heap[j])
j=j+1;
if(heap[i] < heap[j])
flag=0;
else
{
temp=heap[i];heap[i]=heap[j];
```

```
heap[j]=temp;
i=j;
}
}
return 0;
}
```

- ***Output:-***

```
Enter no. of elements :7

Enter HEAP data :
90
72
60
52
88
92
48

Max data : 92
Min Data :48
```

- ***Analysis:-***

  Time Complexity:

  Worst Case : O(n log n)
  Best Case : O(n log n)
  Average Case : O(n log n)

  Space Complexity:

  Worst Case : O(n)
  Best Case : O(1)
  Average Case : O(1)

- ***Conclusion:-****Hence, we have studied and implemented Heap Sort on students marks to find minimum and maximum marks obtained.*