**HEAP :**

Heap is a special tree based data structure that satisfies the any one of the heap properties.

**Max Heap:**

1 . the root node always has the max value

2 . the parent node always > the children .

**Min Heap :**

1 . root node is always the min value

2 . the parent node always < the child

**Heap Tree :**

1 . the complete binary tree (all levels except the leaf are fully filled and nodes are left align are possible )

2 . root always contains the smallest(minheap) or largest (max heap)

3 used in priority queue , graph algor , dizgistra , …etc :

**Heapify up algorithm ( or) insert algorithm :**

**1 . added to available pos in the array**

**2 . compare it with its parent**

**3. if the heap properties is swap the nod to parent node**

**4. repeat the process util the heap properties is satisfied .**

**DELETION IN THE HEAP : 🡪 HEAPIFY DOWN :**

delete the element from usually removing the root node .(max element in max heap or min element in min heap )

**algorithm :**

1. **Swap the root to last element in the array**
2. **Remove the last element**
3. **Heapify down 🡪 compare the new root which is largest child and swap is need**
4. **Reap the util heap property**

**BUILD A HEAP IN UNSORTED ARRAY .**

The program only sort the array.

package day22;

class MaxHeap{

int heap[];

int size;

public MaxHeap(int capacity)

{

heap = new int[capacity];

size = 0;

}

public void heapifyDown(int index)

{

int leftchild = 2\*index+1;

int rightchild = 2\*index+2;

int lar = index;

if(leftchild<size && heap[leftchild]>heap[lar])

{

lar=leftchild;

}

if(rightchild<size && heap[rightchild]>heap[lar])

{

lar=rightchild;

}

if(lar!=index)

{

swap(index,lar);

heapifyDown(lar);

}

}

public void swap(int index , int lar )

{

int temp =heap[index];

heap[index]=heap[lar];

heap[lar]=-temp;

}

public void buildHeap(int arr[])

{

heap = arr;

size=arr.length;

for(int i=size/2-1;i>=0;i--) // heapify down

{

heapifyDown(i);

}

}

}

public class task1 {

public static void main(String args[])

{

MaxHeap h = new MaxHeap(10);

int arr[] = {10,50,30,20,40,60,70,90}; // unsorted array to build a heap

h.buildHeap(arr);

}

}

**MAX HEAP :**

package day22;

class MaxHeap{

int heap[];

int size;

public MaxHeap(int capacity)

{

heap = new int[capacity];

size = 0;

}

public void heapifyDown(int index)

{

int leftchild = 2\*index+1;

int rightchild = 2\*index+2;

int lar = index;

if(leftchild<size && heap[leftchild]>heap[lar])

{

lar=leftchild;

}

if(rightchild<size && heap[rightchild]>heap[lar])

{

lar=rightchild;

}

if(lar!=index)

{

swap(index,lar);

heapifyDown(lar);

}

}

public void swap(int index , int lar )

{

int temp =heap[index];

heap[index]=heap[lar];

heap[lar]=temp;

}

public void buildHeap(int arr[])

{

heap = arr;

size=arr.length;

for(int i=(size/2)-1;i>=0;i--) // heapify down

{

heapifyDown(i);

}

}

public void printHeap()

{

for(int i=0;i<size;i++)

{

System.***out***.println(heap[i]+" ");

}

}

public void insert(int val) {

if (size == heap.length) {

System.***out***.println("Heap full");

return; // Exit to prevent overflow

}

heap[size] = val; // Insert value at the next available index

size++; // Increase the heap size

heapifyup(size - 1); // Restore heap property

}

public void heapifyup(int index)

{

while(index > 0 && heap[index]>heap[index-1]/2)

{

swap(index,(index-1)/2);

index=(index-1)/2;

}

}

public void delete() {

if(size==0)

{System.***out***.println("heap if empty");return;}

heap[0]=heap[size-1];

size--;

heapifyDown(0);

}

}

public class task1 {

public static void main(String args[])

{

MaxHeap h = new MaxHeap(10);

int arr[] = {10,50,30,20,40,60,70,90}; // unsorted array to build a heap

h.buildHeap(arr);

h.printHeap();

h.insert(100);

h.printHeap();

System.***out***.println("delete");

h.delete();

h.printHeap();

}

}

**MIN HEAP :**

package day22;

class MinHeap{

int heap[];

int size;

public MinHeap(int capacity)

{

heap = new int[capacity];

size = 0;

}

public void heapifyDown(int index)

{

int leftchild = 2\*index+1;

int rightchild = 2\*index+2;

int small = index;

if(leftchild<size && heap[leftchild]<heap[small])

{

small=leftchild;

}

if(rightchild<size && heap[rightchild]<heap[small])

{

small=rightchild;

}

if(small!=index)

{

swap(index,small);

heapifyDown(small);

}

}

public void swap(int index , int lar )

{

int temp =heap[index];

heap[index]=heap[lar];

heap[lar]=temp;

}

public void buildHeap(int arr[])

{

heap = arr;

size=arr.length;

for(int i=(size/2)-1;i>=0;i--) // heapify down

{

heapifyDown(i);

}

}

public void printHeap()

{

for(int i=0;i<size;i++)

{

System.***out***.println(heap[i]+" ");

}

}

public void insert(int val) {

if (size == heap.length) {

System.***out***.println("Heap full");

return;

}

heap[size] = val; // Insert value at the next available index

size++; // Increase the heap size

heapifyup(size - 1); // Restore heap property

}

public void heapifyup(int index)

{

while(index > 0 && heap[index]>heap[index-1]/2)

{

swap(index,(index-1)/2);

index=(index-1)/2;

}

}

public void delete() {

if(size==0)

{System.***out***.println("heap if empty");return;}

heap[0]=heap[size-1];

size--;

heapifyDown(0);

}

}

public class task2 {

public static void main(String args[])

{

MinHeap h = new MinHeap(10);

int arr[] = {10,50,30,20,40,60,70,90}; // unsorted array to build a heap

h.buildHeap(arr);

h.printHeap();

h.insert(100);

h.printHeap();

System.***out***.println("delete");

h.delete();

h.printHeap();

}

}

**BACKTRACKING :**

IT IS a recursive algorithmic technique .

Back tracking explore the all sol and eliminate the invalid once

**Idea behind back tracking :**

**1.select the possible sol step**

**2.verify the current is temp least to the valid solution**

**3. if valid move froward and explore the solution**

**4. if not valid undo the previous and try a different path**

**5. repeat the process all possible solution are found**

**Used in :**

N queen problem ,graph coloring , word searching , sub set problem .

**Permutation combination :**

🡪 find all possible ways to arrange

**1.try placing each place in the pos**

**2.if the num is already use skip**

**3.if all position are filled print the result**

**4.back track (undo the last choice) and try another also**

package day22;

public class task4 {

public static void backtrack(int arr[],boolean used[],String res,int n)

{

if(res.length()==arr.length)

{

System.***out***.println(res);

return;

}

for(int i=0;i<n;i++)

{

if(!used[i])

{

used[i]=true;

*backtrack*(arr,used,res+arr[i],n);

used[i]=false;

}

}

}

public static void main(String args[])

{

int arr[] = {1,2,3};

boolean used[]=new boolean[arr.length];

*backtrack*(arr,used,"",arr.length);

}}

123

132

213

231

312

321

**SUB SET :**

Find all the sub set .

package day22;

public class task6{

public static void backtrack(String str , String res , int index)

{

for(int i=index;i<str.length();i++)

{

res+=str.charAt(i);

System.***out***.println(res);

*backtrack*(str,res,i+1);

res=res.substring(0,res.length()-1);

}

}

public static void main(String args[])

{

String str = "abc";

*backtrack*(str,"",0);

}

}

a

ab

abc

ac

b

bc

c