Name: Pavanikumari

Roll no: 320022720007

1st semester

Sub: DS and Algorithms lab

7.)a.) write a program for heap sort.

Aim: The heap sort algorithm to arrange a list of elements in ascending order is performed using following steps... · Step 1 - Construct a Binary Tree with given list of Elements. · Step 2 - Transform the Binary Tree into Min heap. · Step 3 - Delete the root element from Min heap using heapify method. · Step 4 - Put the deleted element into the sorted list. · Step 5 - Repeat the same until Min heap becomes empty. · Step 6 - Display the sorted list.

Program:

```
public class heapSort {
public void sort(int arr[])
{
int n = arr.length;
for (int i = n  /2 - 1; i >= 0; i--)
heapify(array(i,0)):
for (int i = n - 1; i > 0;
int temp = arr[0];
arr[0] = arr[i];
arr[i] = temp;
```

```
heapify(arr, i, 0);
}
}
void heapify(int arr[], int n, int i)
{
int largest = i;
int l = 2 * i + 1;
int r = 2 * i + 2;
if (l < n && arr[l] > arr[largest])
largest = l;
if (r < n && arr[r] > arr[largest])
largest = r;
if (largest != i) {
int swap = arr[i];
arr[i] = arr[largest];
arr[largest] = swap;
heapify(arr, n, largest);
}
}
static void printArray(int arr[])
{
int n = arr.length;
for (int i = 0; i < n; ++i)
System.out.print(arr[i] + " ");
```

```
System.out.println();
}
public static void main(String args[])
{
int arr[] = { 21, 11, 13, 5, 6, 7 };
int n = arr.length;
HeapSort ob = new HeapSort();
ob.sort(arr);
System.out.println("Sorted array is");
printArray(arr);
}
}
```

Output: Sorted array is 5 6 7 11 12 13

7.) b.) write a program to implement the operations on stacks?

Aim: Stack can be easily implemented using an Array or a Linked List. Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and deallocate, but is not limited in size. Here we will implement Stack using arrays. There are two ways to implement a stack: · using array · using linked list Implements stacks using Array: Algorithm for PUSH() OPERATION: Step 1: Start Step 2: Declare stack[MAX]

Step 3: Check if the stack is full or not by comparing top with (MAX-1)

If the stack is full, Then print "Stack Overflow" i.e, stack is full and cannot be pushed with another element
Step 4: Else, the stack is not full Increment top by 1 and set, a[top] = p which pushes the element p into the address pointed by top.
Step 5: Stop Algorithm for POP() OPERATION:

Algorithm for POP() OPERATION:
Step 1: Start
Step 2: Declare stack[MAP]
Step 3: Push the elements into the stack
Step 4: Check if the stack is empty or not by comparing top with base of array i.e 0 If top is less than 0, then stack is empty, print "stack underflow"
Step 5: Else, If top is greater than zero the stack is not empty, then store the value pointed by top in a variable p=a[top] and decrement top by 1. The popped element is p Algorithm for PEEK() OPERATION:

Step 1: Start
Step 2: Declare stack[MAP]
Step 3: Push the elements into the stack
Step 4: Print the value stored in the stack pointed by top.
Step 5: Stop
PROGRAM:

```
class stack {
static final int MAP = 1000;
```

```
int top;
int a[] = new int[MAX];
boolean isEmpty()
{
return (top < 0);
}
stack()
{
top = -1;
}
boolean push(int x)
{
if (top >= (MAX - 1)) {
System.out.println("stack Overflow");
return false;
}
else {
a[++top] = x;
System.out.println(x + " pushed into stack");
return true;
}
}
int pop()
{
```

```
if (top < 0) {
System.out.println("Stack underflow");
return 0;
}
else {
int p = a[top--];
return p;
}
}
int peek()
{
if (top < 0) {
System.out.println("Stack underflow");
return 0;
}
else {
int p = a[top];
return p;
}
}
}
class Main {
public static void main(String args[])
{
```

```
Stack s = new Stack();
s.push(10);
s.push(20);
s.push(30);
System.out.println(s.pop() + " Popped from stack");
}
}
```

OUTPUT: 10 pushed into stack
20 pushed into stack
30 pushed into stack
30 Popped from stack
Top element is 20
Elements present in stack 20 10
Implements stack using linked list:
Algorithm for PUSH() OPERATION:
Create a newNode with the given data.
Check whether the stack is empty (TOP == NULL).
If it is empty, then set the pointer of the node to NULL.
If it is not empty, then make the node point to TOP.
Finally, make the newNode as TOP.
Algorithm for POP() OPERATION: Check whether stack is empty (top == NULL). If it is empty, then display "EMPTY STACK" If it is not empty, then create a temporary node and set it to TOP. Print the data of TOP. Make TOP to point to the next node. Delete the temporary node

PROGRAM:

```
public class stackAsLinkedList {
stackNode root;
static class stackNode {
int data;
stackNode next;
stackNode(int data) {  his.data = data;   }
}
public boolean isEmpty()
{
if (root == null) {
return true;
}
else
return false;
}
public void push(int data)
{
stackNode newNode = new stackNode(data);
if (root == null) {
root = newNode;
}
else {
stackNode temp = root;
```

```
root = newNode;
newNode.next = temp;
}
System.out.println(data + " pushed to stack");
}
public int pop()
{
int popped = Integer.MIN_VALUE;
if (root == null) {
System.out.println("Stack is Empty");
}
else {
popped = root.data;
root = root.next;
}
return popped;
}
public int peek()
{
if (root == null) {
System.out.println("Stack is empty");
return Integer.MIN_VALUE;
}
else {
```

```
return root.data;
}
}
public static void main(String[] args)
{
StackAsLinkedList sll = new StackAsLinkedList();
sll.push(10);
sll.push(20);
sll.push(30);
System.out.println(sll.pop()
+ " popped from stack");
System.out.println("Top element is " + sll.peek());
}
} Output: 10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
Top element is 20
Elements present in stack  20
```