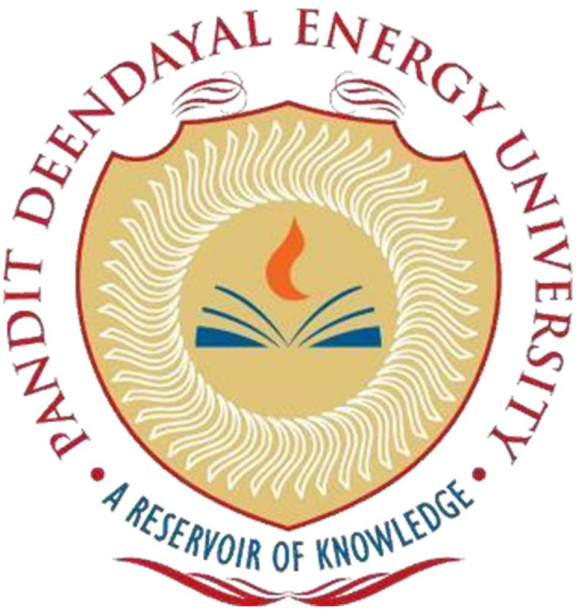
Roll No.: 21BCP359

Name: Harsh Shah

Semester: III

Division: 6-G11

DATA STRUCTURES LAB
(20CP201P)



Pandit Deendayal Energy University, Gandhinagar

School of Technology

Computer Engineering Department

Practical 1: Revision of Arrays

1. Write a program in C to perform linear and binary search.

```
// Linear Search
#include <stdio.h>
int main(){
   int n,i,ele,loc=-1;
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   int a[n];
   printf("\nEnter Array Elements: ");
   for (i=0 ; i<n ; i++){
      scanf("%d", &a[i]);
   }
   printf("\nArray: ");
   for (i=0 ; i<n ; i++){
      printf("%d ", a[i]);
   }
   printf("\nEnter element to be searched: ");
   scanf("%d", &ele);

    for(i=0 ; i<n ; i++){
      if(a[i] == ele){
         loc = i;
         break;
      }
    }
   if (loc != -1){
      printf("\nElement %d found at index %d", ele,loc);
   }
   else{
      printf("\nElement not found!");
   }

  return 0;
}
```

Output:

```
Enter the number of elements: 6

Enter Array Elements: 6
5
8
9
1
3

Array: 6 5 8 9 1 3
Enter element to be searched: 8

Element 8 found at index 2
```

```
// Binary Search
#include <stdio.h>
int main(){
  int n, c, array[100], ele, first, last, mid;
  printf("Enter the number of elements: ");
  scanf("%d", &n);

  printf("\nEnter Array Elements: ", n);
  for (c = 0; c < n; c++){
    scanf("%d", &array[c]);
  }
  printf("\nEnter element to be searched: ");
  scanf("%d", &ele);
  first = 0;
  last = n - 1;
  mid = (first+last)/2;

  while (first <= last) {
    if (array[mid] < ele){
      first = mid + 1;
    }
    else if (array[mid] == ele) {
      printf("\n%d found at location %d.", ele, mid);
      break;
    }
    else{
      last = mid - 1;
    }
    mid = (first + last)/2;
  }
  if (first > last){
    printf("\nElement not found!", ele);
  }

  return 0;
}
```

Output:

```
Enter the number of elements: 6

Enter Array Elements: 1
5
6
8
4
3

Enter element to be searched: 6

6 found at location 2.
```

2. Write a program in C to perform bubble sort, insertion sort and selection sort. Take the array size and array elements from user.

```
#include <stdio.h>
int main(){
   int n;
   printf("Enter the size of the array: ");
   scanf("%d", &n);
   int arr [n];
   //Inputing the array
   printf("Enter elements of array");
   for(int i = 0; i < n; i ++) {
      scanf("%d", &arr[i]);
   }
   //Bubble Sort algorithm
   for(int i = 0; i < n; i ++) {
      for(int j = 0; j < n; j ++) {
         if(arr[j + 1] > arr[j]) {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
   //Insertion Sort algorithm
   for (int i = 1; i < n; i++) {
      int element = arr[i];
         int j = i - 1;
         while (j >= 0 && arr[j] > element) {
            arr[j + 1] = arr[j];
            j = j - 1;
      }
      arr[j + 1] = element;
   }
   //Selection Sort algorithm
   for(int i = 0; i < n - 1; i++)    {
      int position=i;
      for(int j = i + 1; j < n; j++)       {
         if(arr[position] > arr[j])
            position=j;
      }
      if(position != i)       {
         int swap;
         swap=arr[i];
         arr[i]=arr[position];
         arr[position]=swap;
      }
   }
```

```
    //Printing the sorted array
    for(int i = 0; i < n; i ++) {
       printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
Enter the size of the array: 6
Enter elements of array: 6 5 8 3 2 1
Sorted array in ascending order:
1 2 3 5 6 8
```

3. **Write a program in C that obtains the minimum and maximum element from the array. Modify this program to give the second largest and second smallest element of the array.**

```
#include <stdio.h>
void secMinMax(int arr[], int n){
    int count = n;
    int a;
    for (int i = 0; i < count; i++) {
      for (int j = i + 1; j < count; j++) {
        if (arr[i] > arr[j]) {
          a = arr[i];
          arr[i] = arr[j];
          arr[j] = a;
        }
      }
    }
    printf("\nMinimum: %d" , arr[0]);
    printf("\nMaximum: %d" , arr[n-1]);
    printf("\nSecond Minimum: %d" , arr[1]);
    printf("\nSecond Maximum: %d" , arr[n-2]);
}
int main(){
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int arr [n];
    //Inputing the array
    printf("Enter elements of array: ");
    for(int i = 0; i < n; i ++) {
       scanf("%d", &arr[i]);
    }
    secMinMax(arr, n);
    return 0;
}
```

Output:

```
Enter the size of the array: 6
Enter elements of array: 8
9
23
1
1
4

Minimum: 1
Maximum: 23
Second Minimum: 1
Second Maximum: 9
```

Practical 2: Revision of Structures

1. **Create a structure Student in C with student name, student roll number and student address as its data members. Create the variable of type student and print the values.**

```
#include <stdio.h>
struct student {
   char name[15];
   char address[50];
   int rno;
};

int main() {
   struct student s1;
   printf("Enter Name of student: ");
   scanf("%s", s1.name);
   printf("Enter Address of student: ");
   scanf("%s", s1.address);
   printf("Enter Roll number of student: ");
   scanf("%d", &s1.rno);

   printf("Name of student: %s", s1.name);
   printf("\nAddress of student: %s", s1.address);
   printf("\nRoll number of student: %d", s1.rno);

   return 0;
}
```

Output:

```
Enter Name of student: Harsh
Enter Address of student: Hostel
Enter Roll number of student: 359
Name of student: Harsh
Address of student: Hostel
Roll number of student: 359
```

2. **Modify the above program to implement arrays of structure. Create an array of 5 students and print their values.**

```
#include <stdio.h>

struct student {
   char name[15];
   char address[50];
   int rno;
};

int main(){
   struct student s[5];
   for (int i=0;i<=4;i++){
      printf("\nEnter name of student %d: ",i+1);
      scanf("\n%s", s[i].name);
```

```
        printf("Enter address of student %d: ",i+1);
        scanf("%s", s[i].address);
        printf("Enter roll number of student %d: ",i+1);
        scanf("%d", &s[i].rno);
    }
    for (int i=0;i<=4;i++){
        printf("\n%d :: %s :: %s", s[i].rno, s[i].name, s[i].address);
    }

    return 0;
}
```

Output:

```
Enter name of student 1: Derek
Enter address of student 1: Gandhinagar
Enter roll number of student 1: 13

Enter name of student 2: Nakia
Enter address of student 2: Surat
Enter roll number of student 2: 89

Enter name of student 3: Emerald
Enter address of student 3: Ahmedabad
Enter roll number of student 3: 65

13 :: Derek :: Gandhinagar
89 :: Nakia :: Surat
65 :: Emerald :: Ahmedabad
```

3. **Create a structure Organization with organization name and organization ID as its data members. Next, create another structure Employee that is nested in structure Organization with employee ID, employee salary and employee name as its data members. Write a program in such a way that there are two organizations and each of these contains two employees.**

```
#include <stdio.h>
struct organization {
    char name[15];
    int ID;
    struct employee {
        char name[15];
        long salary;
        int ID;
    } e1, e2;
};
int main(){
    struct organization o1, o2;
    printf("Enter details of first Organization ");
```

```
    scanf("%s", o1.name);
    scanf("%d", &o1.ID);
    printf("Enter details of first employee of first Organization ");
    scanf("%s", o1.e1.name);
    scanf("%ld", &o1.e1.salary);
    scanf("%d", &o1.e1.ID);

    printf("Enter details of second employee of first Organization ");
    scanf("%s", o1.e2.name);
    scanf("%ld", &o1.e2.salary);
    scanf("%d", &o1.e2.ID);

    printf("Enter details of second Organization ");
    scanf("%s", o2.name);
    scanf("%d", &o2.ID);

    printf("Enter details of first employee of second Organization ");
    scanf("%s", o2.e1.name);
    scanf("%ld", &o2.e1.salary);
    scanf("%d", &o2.e1.ID);

    printf("Enter details of second employee of second Organization ");
    scanf("%s", o2.e2.name);
    scanf("%ld", &o2.e2.salary);
    scanf("%d", &o2.e2.ID);

    printf("\nDetails of first Organization are %s %d", o1.name, o1.ID);
    printf("\nDetails of first employee of first Organization is %s %ld %d", o1.e1.name,
o1.e1.salary, o1.e1.ID);
    printf("\nDetails of second employee of first Organization is %s %ld %d", o1.e2.name,
o1.e2.salary, o1.e2.ID);

    printf("\nDetails of second Organization are %s %d", o2.name, o2.ID);
    printf("\nDetails of first employee of second Organization is %s %ld %d", o2.e1.name,
o2.e1.salary, o2.e1.ID);
    printf("\nDetails of second employee of second Organization is %s %ld %d", o2.e2.name,
o2.e2.salary, o2.e2.ID);
    return 0;
}
```

Output:

```
Enter details of first Organization
PDPU
21
Enter details of first employee of first Organization
Harsh
3658689
359
Enter details of second employee of first Organization
Karan
8716
377
Enter details of second Organization
PDEU
2
Enter details of first employee of second Organization
Kushagra
81698
387
Enter details of second employee of second Organization
Vikas
45668
372
```

```
Details of first Organization are PDPU 21
Details of first employee of first Organization is Harsh 3658689 359
Details of second employee of first Organization is Karan 8716 377
Details of second Organization are PDEU 2
Details of first employee of second Organization is Kushagra 81698 387
Details of second employee of second Organization is Vikas 45668 372
```

Practical 3: Revision of Pointers

1. **Write a program in C to implement arrays of pointers and pointers to arrays**.

```
#include <stdio.h>
int main(){
   //pointer to an array
   int arr[5] = {6, 2, 1, 7, 5};
   int *ptr = &arr[0];
   printf("Array: ");
   for(int i = 0; i < 5; i ++) {
      printf("%d ", *(ptr + i));
   }

   //array of pointers
   int  var[] = {10, 100, 200};

   printf("\n\nArray of Pointer: ");
   for (int i = 0; i < 3; i ++) {
     printf("\nValue of var[%d] = %d", i, var[i] );
   }

   return 0;
}
```

Output:

```
Array: 6 2 1 7 5

Array of Pointer:
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

2. **Write a program in C to implement pointers to structures.**

```
#include <stdio.h>
struct person{
  int age;
  float weight;
  int height;
};
int main(){
   struct person *Ptr, person1;
   Ptr = &person1;
   printf("Enter age of Person: ");
   scanf("%d", &Ptr->age);
   printf("Enter weight of Person: ");
   scanf("%f", &Ptr->weight);
   printf("Enter height of Person: ");
   scanf("%d", &Ptr->height);
   printf("\nAge of Person is: %d", Ptr->age);
```

```
    printf("\nWeight of Person is: %f", Ptr->weight);
    printf("\nHeight of Person is: %d", Ptr->height);

    return 0;
}
```

Output:

```
Enter age of Person: 18
Enter weight of Person: 65
Enter height of Person: 170

Age of Person is: 18
Weight of Person is: 65.000000
Height of Person is: 170
```

3. **Write a program in C to perform swapping of two numbers by passing addresses of the variables to the functions.**

```
#include <stdio.h>
void swap(int *x,int *y){
    int t;
     t = *x;
    *x = *y;
    *y =  t;
}
int main(){
    int a, b;

    printf("Enter value of first number: ");
    scanf("%d",&a);
    printf("Enter value of second number: ");
    scanf("%d",&b);
    printf("Before Swapping: %d, %d\n",a,b);
    swap(&a,&b);
    printf("After  Swapping: %d, %d\n",a,b);

    return 0;
}
```

Output:

```
Enter value of first number: 6
Enter value of second number: 8
Before Swapping: 6, 8
After  Swapping: 8, 6
```

Practical 4: Linked List

1. **Write a program that takes two sorted lists as inputs and merge them into one sorted list. For example, if the first linked list A is 5 =>10 =>15, and the other linked list B is 2 => 3 => 20, then output should be 2 => 3 => 5 => 10 => 15 => 20.**

```
import java.util.Scanner;
class Node{
   int data;
   Node next;
   Node(){}
   Node(int data){
      this.data = data;
   }
   Node(int data, Node next) {
      this.data = data;
      this.next = next;
   }
}

public class MergeSortedList {
   Node head;
   public void addLast(Node node){
      if (head == null){
         head = node;
      }
      else{
         Node tempNode = head;
         while(tempNode.next != null){
            tempNode = tempNode.next;
         }
         tempNode.next = node;
      }
   }
   void printList() {
      System.out.print("\nMerged Linked list is :  ");
      Node tempNode = head;
      while(tempNode != null){
         System.out.print(tempNode.data + " -> ");
         tempNode = tempNode.next;
      }
      System.out.println("null");
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      MergeSortedList msl1 = new MergeSortedList();
      MergeSortedList msl2 = new MergeSortedList();

      System.out.print("Enter number of elements for 1st List: ");
      int n1 = sc.nextInt();
```

```
        for (int i=0 ; i<n1 ; i++){
            System.out.print("Enter data: ");
            int data = sc.nextInt();
            msl1.addLast(new Node(data));
        }
        System.out.print("Enter number of elements for 2nd List: ");
        int n2 = sc.nextInt();
        for (int i=0 ; i<n2 ; i++){
            System.out.print("Enter data: ");
            int data = sc.nextInt();
            msl2.addLast(new Node(data));
        }
        msl1.head = new MergeList().mergeList(msl1.head, msl2.head);
        msl1.printList();
    }
}
class MergeList{
    public Node mergeList(Node A, Node B){
        if (A == null){
            return B;
        }
        if (B == null){
            return A;
        }
        if (A.data < B.data) {
            A.next = mergeList(A.next, B);
            return A;
        }
        else {
            B.next = mergeList(B.next, A);
            return B;
        }
    }
}
```

Output:

```
Enter number of elements for 1st List: 3
Enter data: 5
Enter data: 10
Enter data: 15
Enter number of elements for 2nd List: 3
Enter data: 2
Enter data: 3
Enter data: 20

Merged Linked list is :  2 -> 3 -> 5 -> 10 -> 15 -> 20 -> null
```

2. **Write a program to insert a new node into the linked list. A node can be added into the linked list using three ways:**

a. At the front of the list
b. After a given node
c. At the end of the list

```
import java.io.*;
public class LinkedList
{
    Node head;
    class Node{
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    public void push(int new_data){
        Node new_node = new Node(new_data);
        new_node.next = head;
        head = new_node;
    }

    public void insertAfter(Node prev_node, int new_data){
        if (prev_node == null){
            System.out.println("The given previous node cannot be null");
            return;
        }
        Node new_node = new Node(new_data);
        new_node.next = prev_node.next;
        prev_node.next = new_node;
    }
   public void append(int new_data){
        Node new_node = new Node(new_data);
        if (head == null){
            head = new Node(new_data);
            return;
        }
        new_node.next = null;
        Node last = head;
        while (last.next != null)
            last = last.next;
        last.next = new_node;
        return;
    }
    public void printList()
    {
        Node tnode = head;
        while (tnode != null){
            System.out.print(tnode.data+" ");
            tnode = tnode.next;
```

```
            }
        }

        public static void main(String[] args){
            /* Start with the empty list */
            LinkedList llist = new LinkedList();
            llist.append(6);
            llist.push(7);
            llist.push(1);
            llist.append(4);
            llist.insertAfter(llist.head.next, 8);
            System.out.println("\nCreated Linked list is: ");
            llist.printList();
        }
    }
```

Output:

```
Created Linked list is:
1 7 8 6 4
```

3. **Write a program to delete a node from the linked list. A node can be deleted from the linked list using three ways:**
   a. **Delete from the beginning**
   b. **Delete from the end**
   c. **Delete from the middle.**

```
public class LL{
    Node head;
    class Node{
        String data;
        Node next;
        Node(String data){
            this.data = data;
            this.next = null;
        }
    }
    public void addFirst(String data){
        Node newNode = new Node(data);
        if (head==null){
            head = newNode;
            return;
        }
```

```
        newNode.next = head;
        head = newNode;
    }
    public void addLast(String data){
        Node newNode = new Node(data);
        if (head==null){
            head = newNode;
            return;
        }
        //traverse
        Node currNode = head;
        while(currNode.next!=null){
            currNode = currNode.next;
        }
        currNode.next = newNode;
    }
    public void deleteFirst(){
        if (head==null){
            System.out.println("The list is empty");
            return;
        }
        head = head.next;
    }
    public void deleteLast(){
        if (head==null){
            System.out.println("The list is empty");
            return;
        }
        if (head.next == null){ //Single node only
            head = null;
            return;
        }
        Node secondLast = head;
```

```
        Node lastNode = head.next;

        while(lastNode.next != null){
            lastNode = lastNode.next;
            secondLast = secondLast.next;
        }
        secondLast.next = null;
    }
    public void deleteLoc(int loc){
        if (head == null){
            return;
        }
        Node temp = head;

        if (loc == 0){
            head = temp.next;
            return;
        }
        int i=0;
        while(temp != null && i<loc-1){
            temp = temp.next;
            i++;
        }
        if(temp == null || temp.next == null){
            return;
        }
        Node next = temp.next.next;
        temp.next = next;
    }
    public void printList(){
        if (head == null){
            System.out.println("List is empty");
            return;
```

```
        }
        Node currNode = head;
        while(currNode!=null){
            System.out.print(currNode.data + " -> ");
            currNode = currNode.next;
        }
        System.out.println("null");
    }
    public static void main(String[] args) {
        LL list  = new LL();
        list.addFirst("3");
        list.addFirst("2");
        list.addFirst("1");
        list.addLast("4");
        list.printList();
        list.deleteFirst();
        list.printList();
        list.deleteLast();
        list.printList();
        list.deleteLoc(2);
        list.printList();
    }
}
```

Output:

```
1 -> 2 -> 3 -> 4 -> null
2 -> 3 -> 4 -> null
2 -> 3 -> null
2 -> 3 -> null
```

4. **Implement the circular linked list and perform the operation of traversal on it. In a conventional linked list, we traverse the list from the head node and stop the traversal when we reach NULL. In a circular linked list, we stop traversal when we reach the first node again.**

```
public class CircularLinkedList {
    private Node head;
    private Node tail;
```

```
private class Node {
    int val;
    Node next;

    public Node(int val) {
        this.val = val;
    }
}

public CircularLinkedList() {
    this.head = null;
    this.tail = null;
}

public void insert(int val){
    Node node = new Node(val);
    if (head == null){
        head = node;
        tail = node;
        return;
    }

    tail.next = node;
    node.next = head;
    tail = node;
}

public void delete(int val){
    Node node = head;
    if (node == null){
        return;
    }
    if (val == node.val){
        head = head.next;
        tail.next = head;
        return;
    }
    do{
        Node n = node.next;
        if (n.val == val){
            node.next = n.next;
            break;
        }
        node = node.next;
    } while (node != head);
}
```

```
    public void display(){
        Node node = head;
        if (head != null){
            do{
                System.out.print(node.val + " -> ");
                node = node.next;
            } while (node != head);
            System.out.println("head");
        }
    }

    public static void main(String[] args) {
        CircularLinkedList list  = new CircularLinkedList();
        list.insert(1);
        list.insert(2);
        list.insert(3);
        list.insert(4);
        list.display();
        list.delete(4);
        list.display();
    }
}
```

Output:

```
1 -> 2 -> 3 -> 4 -> head
1 -> 2 -> 3 -> head
```

5. **Implement the doubly linked list and perform the deletion and/ or insertion operation on it. Again, you can perform insertion deletion according to the three ways as given above. Implement all of them according to availability of time.**

```
public class DoublyLinkedList {
    private Node head;
    private class Node{
        int val;
        Node next;
        Node prev;
        public Node(int val, Node next, Node prev) {
            this.val = val;
            this.next = next;
            this.prev = prev;
        }

        public Node(int val) {
            this.val = val;
        }
```

```
    }
    public Node find(int value) {
        Node node = head;
        while (node != null) {
            if (node.val == value) {
                return node;
            }
            node = node.next;
        }
        return null;
    }
    public void insertFirst(int val){
        Node node = new Node(val);
        node.next = head;
        node.prev = null;
        if (head != null){
            head.prev = node;
        }
        head = node;
    }
    public void insert(int after, int val){
        Node p = find(after);
        if (p==null){
            System.out.println("Doesn't exist");
            return;
        }
        Node node = new Node(val);
        node.next = p.next;
        p.next = node;
        node.prev = p;
        if (node.next.prev != null){
            node.next.prev = node;
        }
    }

    public void insertLast(int val){
        Node node = new Node(val);
        Node last = head;
        node.next = null;

        if (head == null){
            node.prev = null;
            head = node;
            return;
        }

            while(last.next != null) {
```

```
            last = last.next;
        }
        last.next = node;
        node.prev = last;
    }
    public void display(){
        Node node = head;
        Node last = null;
        System.out.print("Printing in Normal Order: ");
        if (head != null){
            do {
                System.out.print(node.val + " -> ");
                last = node;
                node = node.next;
            } while(node != null);
            System.out.println("null");
        }

        // Displaying in reverse order
        System.out.print("Printing in Reverse Order: ");
        while (last != null){
            System.out.print(last.val + " -> ");
            last = last.prev;
        }
        System.out.println("start");
    }

    public static void main(String[] args) {
        DoublyLinkedList list = new DoublyLinkedList();
        list.insertFirst(23);
        list.insertFirst(56);
        list.insertFirst(89);
        list.insertLast(66);
        list.insert(89,78);
        list.display();
    }
}
```

Output:

```
Printing in Normal Order: 89 -> 78 -> 56 -> 23 -> 66 -> null
Printing in Reverse Order: 66 -> 23 -> 56 -> 78 -> 89 -> start
```

Practical 5: Stack Applications - I

1. **Implement a stack using an array and using a linked list.**

```
public class StackOperations {
    int size;
    int[] arr;
    int top;
```

```
StackOperations(int size) {
   this.size = size;
   this.arr = new int[size];
   this.top = -1;
}
public int getTop(){
   return top;
}
public void push(int Element) {
   if (!isFull()) {
      top++;
      arr[top] = Element;
      System.out.println("Pushed element:" + Element);
   }
   else {
      System.out.println("Stack is full !");
   }
}
public void pop() {
   if (!isEmpty()) {
      int returnedTop = top;
      top--;
      System.out.println("Popped element :" + arr[returnedTop]);
   }
   else {
      System.out.println("Stack is empty !");
   }
}
public boolean isEmpty() {
   return (top == -1);
}
public boolean isFull() {
   return (size - 1 == top);
}
public static void main(String[] args) {
   StackOperations StObj = new StackOperations(10);
   StObj.pop();
   System.out.println();
   StObj.push(10);
   StObj.push(30);
   StObj.push(50);
   StObj.push(40);
   System.out.println();
   StObj.pop();
   StObj.pop();
   StObj.pop();
   System.out.println();
```

```
    }
}
```

Output:

```
Stack is empty !

Pushed element:10
Pushed element:30
Pushed element:50
Pushed element:40

Popped element :40
Popped element :50
Popped element :30
```

2. **Given a stack, sort it using recursion. Use of any loop constructs like while, for, etc. is not allowed. We can only use the following functions on Stack S:**
   a. **isEmpty(S): Tests whether stack is empty or not.**
   b. **push(S): Adds new element to the stack.**
   c. **pop(S): Removes top element from the stack.**
   d. **top(S): Returns value of the top element.**

```
import java.util.ListIterator;
import java.util.Stack;

class Test{
    static void sortedInsert(Stack<Integer> s, int x) {
        if (s.isEmpty() || x > s.peek()) {
            s.push(x);
            return;
        }

        int temp = s.pop();
        sortedInsert(s, x);
        s.push(temp);
    }

    // Method to sort stack
    static void sortStack(Stack<Integer> s) {
        if (!s.isEmpty()) {
            int x = s.pop();
            sortStack(s);
            sortedInsert(s, x);
        }
    }
```

```
    static void printStack(Stack<Integer> s) {
      ListIterator<Integer> lt = s.listIterator();
      while(lt.hasNext())
         lt.next();
      while(lt.hasPrevious())
         System.out.print(lt.previous()+" ");
    }
    public static void main(String[] args){
       Stack<Integer> s = new Stack<>();
       s.push(30);
       s.push(-5);
       s.push(18);
       s.push(14);
       s.push(-3);

       System.out.println("\nStack elements before sorting: ");
       printStack(s);

       sortStack(s);

       System.out.println("\nStack elements after sorting:");
       printStack(s);
    }
}
```

Output:

```
Stack elements before sorting:
-3 14 18 -5 30
Stack elements after sorting:
30 18 14 -3 -5
```

Practical 6: Stack Applications - II

1. **Convert the given infix expression into postfix expression using stack.**

   Example - Input: $a + b * (c\hat{}d - e)\hat{}(f + g * h) - i$

   Output: $abcd\hat{}e - fgh * +\hat{} * +i -$

```
import java.util.Stack;
public class InfixToPostfix {
```

```
static int precedence(char c){
    switch (c){
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return -1;
}

static String infixToPostFix(String expression){
    String result = "";
    Stack<Character> stack = new Stack<>();
    for (int i = 0; i <expression.length() ; i++) {
        char c = expression.charAt(i);

        //check if char is operator
        if(precedence(c)>0){
            while(stack.isEmpty()==false && precedence(stack.peek())>=precedence(c)){
                result += stack.pop();
            }
            stack.push(c);
        }
        else if(c==')'){
            char x = stack.pop();
            while(x!='('){
                result += x;
                x = stack.pop();
            }
        }
        else if(c=='('){
            stack.push(c);
        }
        else{
            result += c;
        }
    }

    for (int i = 0; i <=stack.size() ; i++) {
        result += stack.pop();
    }
    return result;
```

```
    }

    public static void main(String[] args) {
        InfixToPostfix ip = new InfixToPostfix();
        String exp = "(a+b)*(c+d)";
        System.out.println("Infix Expression: " + exp);
        System.out.println("Postfix Expression: " + ip.infixToPostFix(exp));
    }
}
```

Output:

```
Infix Expression: (a+b)*(c+d)
Postfix Expression: ab+cd+*
```

2. **Write a program to evaluate the following given postfix expressions:**
   Input: 2 3 1 * + 9 – **Output: -4**
   Input: 2 2 + 2 / 5 * 7 + **Output: 17**

```
import java.util.Scanner;
import java.util.Stack;

public class EvaluatePostfix{
    static int evaluatePostfix(String exp)  {
            Stack<Integer> stack=new Stack<>();
            for(int i=0;i<exp.length();i++)                    {
                    char c=exp.charAt(i);
                    if(Character.isDigit(c)){
                            stack.push(c - '0');
                    }
                    else {
                            int val1 = stack.pop();
                            int val2 = stack.pop();

                            switch(c) {
                                    case '+':
                                    stack.push(val2+val1);
                                    break;

                                    case '-':
                                    stack.push(val2- val1);
                                    break;

                                    case '/':
                                    stack.push(val2/val1);
                                    break;

                                    case '*':
```

```
                    stack.push(val2*val1);
                    break;
                }
            }
        }
        return stack.pop();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("\nEnter your Postfix expression without space: ");
        String exp = sc.nextLine();

        System.out.println("Postfix evaluation: " + evaluatePostfix(exp));
    }
}
```

Output:

```
Enter your Postfix expression without space: 231*+9-
Postfix evaluation: -4
```

```
Enter your Postfix expression without space: 22+2/5*7+
Postfix evaluation: 17
```

3. **Given an expression, write a program to examine whether the pairs and the orders of "{", "}", "(", ")", "[", "]" are correct in the expression or not.**

Example - **Input: exp = "[( )]{ }{[( )( )]( )}"** **Output: Balanced**
Input: exp = "[( ])" **Output: Not Balanced**

```
import java.util.*;
public class BalancedBrackets {
    static boolean areBracketsBalanced(String expr){
        Deque<Character> stack = new ArrayDeque<Character>();

        for (int i = 0; i < expr.length(); i++) {
            char x = expr.charAt(i);

            if (x == '(' || x == '[' || x == '{') {
                stack.push(x);
                continue;
            }
            if (stack.isEmpty())
                return false;
            char check;
            switch (x) {
                case ')':
                    check = stack.pop();
                    if (check == '{' || check == '[')
```

```
                                        return false;
                                    break;

                                case '}':
                                    check = stack.pop();
                                    if (check == '(' || check == '[')
                                        return false;
                                    break;

                                case ']':
                                    check = stack.pop();
                                    if (check == '(' || check == '{')
                                        return false;
                                    break;
                            }
                        }

                        return (stack.isEmpty());
                }
                public static void main(String[] args){
                        String expr = "[( ])" ;
                        if (areBracketsBalanced(expr))
                                System.out.println("\nBalanced \n");
                        else
                                System.out.println("\nNot Balanced \n");
                }
        }
```

Output:

```
Not Balanced
```

Practical 7: Queue

1. **Implement various functionalities of Queue using Arrays. For example: insertion, deletion, front element, rear element etc.**

```
public class Queue {
```

```
int front, rear, size;
int capacity;
int arr[];
public Queue(int capacity){
   this.capacity = capacity;
   front = this.size = 0;
   rear = capacity - 1;
   arr = new int[this.capacity];
}
boolean isFull(Queue queue){
   return (queue.size == queue.capacity);
}
boolean isEmpty(Queue queue){
   return (queue.size == 0);
}
void enqueue(int item){
   if (isFull(this)) {
      return;
   }
   this.rear = (this.rear + 1)
         % this.capacity;
   this.arr[this.rear] = item;
   this.size = this.size + 1;
   System.out.println(item + " enqueued");
}
int dequeue(){
   if (isEmpty(this)) {
      return Integer.MIN_VALUE;
   }
   int item = this.arr[this.front];
   this.front = (this.front + 1) % this.capacity;
   this.size = this.size - 1;
   return item;
}
int front(){
   if (isEmpty(this)) {
      return Integer.MIN_VALUE;
   }
   return this.arr[this.front];
}
int rear(){
   if (isEmpty(this)) {
      return Integer.MIN_VALUE;
   }
   return this.arr[this.rear];
}
public static void main(String[] args){
```

```
        Queue queue = new Queue(10);

        queue.enqueue(37);
        queue.enqueue(28);
        queue.enqueue(65);
        queue.enqueue(12);
        queue.enqueue(78);

        System.out.println(queue.dequeue() + " dequeued\n");
        System.out.println("Front item is " + queue.front());
        System.out.println("Rear item is " + queue.rear());
    }
}
```

Output:

```
37 enqueued
28 enqueued
65 enqueued
12 enqueued
78 enqueued
37 dequeued

Front item is 28
Rear item is 78
```

2. **Implement various functionalities of Queue using Linked Lists. Again, you can implement operation given above.**

```
class Node {
    int key;
    Node next;

    public Node(int key) {
            this.key = key;
            this.next = null;
    }
}
class Queue {
    Node front, rear;
    public Queue() {
            this.front = this.rear = null;
    }
    void enqueue(int key) {
            Node temp = new Node(key);
            if (this.rear == null) {
                    this.front = this.rear = temp;
                    return;
```

```
            }
            this.rear.next = temp;
            this.rear = temp;
      System.out.println(key + " enqueued");
    }
    void dequeue()          {
            if (this.front == null)
                  return;
            Node temp = this.front;
            this.front = this.front.next;
            if (this.front == null) {
        this.rear = null;
      }
      System.out.println("dequeued");
    }
}

public class QueueUsingLL {
    public static void main(String[] args) {
            Queue q = new Queue();
            q.enqueue(10);
            q.enqueue(20);
            q.dequeue();
            q.dequeue();
            q.enqueue(30);
            q.enqueue(40);
            q.enqueue(50);
            q.dequeue();
            System.out.println("Front element: " + q.front.key);
            System.out.println("Rear element: " + q.rear.key);
    }
}
```

Output:

```
20 enqueued
dequeued
dequeued
40 enqueued
50 enqueued
dequeued
Front element: 40
Rear element: 50
```

3. **Implement Priority Queue, where every element has a priority associated with it. Perform operations like Insertion and Deletion in a priority queue.**

```
class Node {
   public int value;
   public int priority;
}
```

```
class PriorityQueue {
    static Node[] pr = new Node[100000];
    static int size = -1;
    static void enqueue(int value, int priority){
        size++;
        pr[size] = new Node();
        pr[size].value = value;
        pr[size].priority = priority;
    }
    static int peek() {
        int highestPriority = Integer.MIN_VALUE;
        int ind = -1;
        for (int i = 0; i <= size; i++) {
            if (highestPriority == pr[i].priority
                    && ind > -1
                    && pr[ind].value < pr[i].value) {
                highestPriority = pr[i].priority;
                ind = i;
            }
            else if (highestPriority < pr[i].priority) {
                highestPriority = pr[i].priority;
                ind = i;
            }
        }
        return ind;
    }
    static void dequeue(){
        int ind = peek();
        for (int i = ind; i < size; i++) {
            pr[i] = pr[i + 1];
        }
        size--;
    }

    public static void main(String[] args){
        enqueue(10, 2);
        enqueue(14, 4);
        enqueue(16, 4);
        enqueue(12, 3);
        int ind = peek();
        System.out.println(pr[ind].value);
        dequeue();
        ind = peek();
        System.out.println(pr[ind].value);
        dequeue();
        ind = peek();
        System.out.println(pr[ind].value);
```

```
    }
}
```

Output:

```
16
14
12
```

4. **Implement Double Ended Queue that supports following operation:**
   a. **insertFront(): Adds an item at the front of Deque.**
   b. **insertLast(): Adds an item at the rear of Deque.**
   c. **deleteFront(): Deletes an item from the front of Deque.**
   d. **deleteLast(): Deletes an item from the rear of Deque.**

```
class Deque {
    static final int MAX = 100;
    int arr[];
    int front;
    int rear;
    int size;

    public Deque(int size) {
        arr = new int[MAX];
        front = -1;
        rear = 0;
        this.size = size;
    }

    boolean isFull() {
        return ((front == 0 && rear == size - 1)
                || front == rear + 1);
    }

    boolean isEmpty() {
        return (front == -1);
    }

    void insertfront(int key) {
        if (isFull()) {
            System.out.println("Overflow");
            return;
        }

        if (front == -1) {
            front = 0;
            rear = 0;
```

```
    }

    else if (front == 0)
        front = size - 1;
    else
        front = front - 1;

    arr[front] = key;
}

void insertrear(int key) {
    if (isFull()) {
        System.out.println(" Overflow ");
        return;
    }
    if (front == -1) {
        front = 0;
        rear = 0;
    }
    else if (rear == size - 1)
        rear = 0;
    else
        rear = rear + 1;
    arr[rear] = key;
}

void deletefront() {
    if (isEmpty()) {
        System.out.println("Queue Underflow\n");
        return;
    }
    if (front == rear) {
        front = -1;
        rear = -1;
    }
    else
        if (front == size - 1)
            front = 0;

        else
            front = front + 1;
}
void deleterear() {
    if (isEmpty()) {
        System.out.println(" Underflow");
        return;
    }
```

```
        if (front == rear) {
            front = -1;
            rear = -1;
        }
        else if (rear == 0)
            rear = size - 1;
        else
            rear = rear - 1;
    }
     int getFront() {
        if (isEmpty()) {
            System.out.println(" Underflow");
            return -1;
        }
        return arr[front];
    }
    int getRear() {
        if (isEmpty() || rear < 0) {
            System.out.println(" Underflow\n");
            return -1;
        }
        return arr[rear];
    }
    public static void main(String[] args) {
        Deque dq = new Deque(5);

        System.out.println("Insert element at rear end  : 5 ");
        dq.insertrear(5);
        System.out.println("insert element at rear end : 10 ");
        dq.insertrear(10);
        System.out.println("Get rear element : " + dq.getRear());
        dq.deleterear();
        System.out.println("After delete rear element new rear become : " + dq.getRear());
        System.out.println("Inserting element at front end");
        dq.insertfront(15);
        System.out.println("Get front element: " + dq.getFront());
        dq.deletefront();
        System.out.println("After delete front element new front become : " + dq.getFront());
    }
}
```

Output:

```
Insert element at rear end  : 5
insert element at rear end : 10
Get rear element : 10
After delete rear element new rear become : 5
Inserting element at front end
Get front element: 15
After delete front element new front become : 5
```

5. Implement Double Ended Queue that supports following operation:

a. getFront(): Gets the front item from the queue.
b. getRear(): Gets the last item from queue.
c. isEmpty(): Checks whether Deque is empty or not.
d. isFull(): Checks whether Deque is full or not.

```
class Deque {
    static final int MAX = 100;
    int arr[];
    int front;
    int rear;
    int size;

    public Deque(int size) {
        arr = new int[MAX];
        front = -1;
        rear = 0;
        this.size = size;
    }

    boolean isFull() {
        return ((front == 0 && rear == size - 1)
                || front == rear + 1);
    }

    boolean isEmpty() {
        return (front == -1);
    }

    void insertfront(int key) {
        if (isFull()) {
            System.out.println("Overflow");
            return;
        }

        if (front == -1) {
            front = 0;
            rear = 0;
        }

        else if (front == 0)
```

```
            front = size - 1;
        else
            front = front - 1;

        arr[front] = key;
    }

    void insertrear(int key) {
        if (isFull()) {
            System.out.println(" Overflow ");
            return;
        }
        if (front == -1) {
            front = 0;
            rear = 0;
        }
        else if (rear == size - 1)
            rear = 0;
        else
            rear = rear + 1;
        arr[rear] = key;
    }

    void deletefront() {
        if (isEmpty()) {
            System.out.println("Queue Underflow\n");
            return;
        }
        if (front == rear) {
            front = -1;
            rear = -1;
        }
        else
            if (front == size - 1)
                front = 0;

            else
                front = front + 1;
    }
    void deleterear() {
        if (isEmpty()) {
            System.out.println(" Underflow");
            return;
        }
        if (front == rear) {
            front = -1;
            rear = -1;
```

```
        }
        else if (rear == 0)
            rear = size - 1;
        else
            rear = rear - 1;
    }
     int getFront() {
        if (isEmpty()) {
            System.out.println(" Underflow");
            return -1;
        }
        return arr[front];
    }
    int getRear() {
        if (isEmpty() || rear < 0) {
            System.out.println(" Underflow\n");
            return -1;
        }
        return arr[rear];
    }
    public static void main(String[] args) {
        Deque dq = new Deque(5);

        System.out.println("Insert element at rear end  : 5 ");
        dq.insertrear(5);
        System.out.println("insert element at rear end : 10 ");
        dq.insertrear(10);
        System.out.println("Get rear element : " + dq.getRear());
        dq.deleterear();
        System.out.println("After delete rear element new rear become : " + dq.getRear());
        System.out.println("Inserting element at front end");
        dq.insertfront(15);
        System.out.println("Get front element: " + dq.getFront());
        dq.deletefront();
        System.out.println("After delete front element new front become : " + dq.getFront());
    }
}
```

Output:

```
Insert element at rear end  : 5
insert element at rear end : 10
Get rear element : 10
After delete rear element new rear become : 5
Inserting element at front end
Get front element: 15
After delete front element new front become : 5
```

Practical 8: Trees

1. **Write a program to insert an element, delete an element and search an element in the Binary Tree.**

```
class BinarySearchTree {
   Node root;
   class Node {
      int key;
      Node left, right;
      public Node(int item) {
         key = item;
         left = right = null;
      }
   }
   BinarySearchTree() {
      root = null;
   }

   void insert(int key) {
      root = insertKey(root, key);
   }
   Node insertKey(Node root, int key) {
      // Return a new node if the tree is empty
      if (root == null) {
         root = new Node(key);
         return root;
      }
      if (key < root.key)
         root.left = insertKey(root.left, key);
      else if (key > root.key)
         root.right = insertKey(root.right, key);

      return root;
   }

   void deleteKey(int key) {
      root = deleteRec(root, key);
   }
   Node deleteRec(Node root, int key) {
      // Return if the tree is empty
      if (root == null)
         return root;

      // Find the node to be deleted
      if (key < root.key)
         root.left = deleteRec(root.left, key);
      else if (key > root.key)
         root.right = deleteRec(root.right, key);
      else {
```

```
        // If the node is with only one child or no child
        if (root.left == null) {
            return root.right;
        }
        else if (root.right == null) {
            return root.left;
        }

        // If the node has two children, Place the inorder successor in position of the node to be deleted
        root = inOrderSuc(root.right);

        // Delete the inorder successor
        root.right = deleteRec(root.right, root.key);
    }
    return root;
}

// Find the inorder successor
Node inOrderSuc(Node root) {
    while (root.left != null) {
        root = root.left;
    }
    return root;
}

void inorder() {
    inorderRec(root);
}
void inorderRec(Node root) {
    if (root != null) {
        inorderRec(root.left);
        System.out.print(root.key + " -> ");
        inorderRec(root.right);
    }
}

static Node search(Node root, int key){
    if (root == null || root.key == key){
        return root;
    }
    if (key > root.key){
        return search(root.right, key);
    }
    return search(root.left, key);
}

public static void main(String[] args) {
```

```
        BinarySearchTree tree = new BinarySearchTree();

        tree.insert(8);
        tree.insert(3);
        tree.insert(1);
        tree.insert(6);
        tree.insert(7);
        tree.insert(10);
        tree.insert(14);
        tree.insert(4);

        System.out.print("Inorder traversal: ");
        tree.inorder();

        System.out.println("\n\nAfter deleting 10");
        tree.deleteKey(10);
        System.out.print("Inorder traversal: ");
        tree.inorder();
    }
}
```

Output:

```
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 14 ->

After deleting 10
Inorder traversal: 1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 14 ->
```

2. Study and implement the Binary Tree and perform following three types of traversals in a binary tree:

a. Pre-order Traversal
b. In-order Traversal
c. Post-order Traversal

```
import java.util.*;

class Node {

    int value;

    Node left, right;

    Node(int value) {

        this.value = value;

    }

}

public class TraverseTree {

    static Scanner input = new Scanner(System.in);

    static Node createTree() {
```

```
        Node root = null;
        System.out.print("Enter data: ");
        int item = input.nextInt();
        if(item == -1) {
            return null;
        }
        root = new Node(item);
        System.out.println("\nEnter left item for " + item);
        root.left = createTree();
        System.out.println("\nEnter right item for " + item);
        root.right = createTree();
        return root;
    }
    static void inorder(Node root) {
        if(root == null) {
            return;
        }
        inorder(root.left);
        System.out.print(root.value + " ");
        inorder(root.right);
    }
    static void preorder(Node root) {
        if(root == null) {
            return;
        }
        System.out.print(root.value + " ");
        preorder(root.left);
        preorder(root.right);
    }
    static void postorder(Node root) {
        if(root == null) {
            return;
        }
```

```
        postorder(root.left);
        postorder(root.right);
        System.out.print(root.value + " ");
    }
    public static void main(String[] args) {
        Node root = createTree();
        System.out.print("\nInorder Traversal: ");
        inorder(root);
        System.out.print("\nPreorder Traversal: ");
        preorder(root);
        System.out.print("\nPostorder Traversal: ");
        postorder(root);
    }
}
```

Output:

```
Enter data: 3

Enter left item for 3
Enter data: 55

Enter left item for 55
Enter data: -1
Enter right item for 3
Enter data: 89

Enter left item for 89
Enter data: 81

Enter left item for 81
Enter data: -1

Enter right item for 81
Enter data: -1

Enter right item for 89
Enter data: -1

Inorder Traversal: 55 56 3 81 89
Preorder Traversal: 3 55 56 89 81
Postorder Traversal: 56 55 81 89 3
```

3. **Given a Pre-order traversal sequence of a Binary Search Tree, construct the corresponding Binary Search Tree.**

```
class Node {
    int data;
```

```
    Node left, right;
    Node(int data)
    {
        this.data = data;
        this.left = this.right = null;
    }
}
class CreateBSTFromPreorder {
    private static Node node;

    public static Node createNode(Node node, int data){
        if (node == null)
            node = new Node(data);
        if (node.data > data)
            node.left = createNode(node.left, data);
        if (node.data < data)
            node.right = createNode(node.right, data);
        return node;
    }
    public static void create(int data){
        node = createNode(node, data);
    }
    public static void inorderRec(Node root){
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.data);
            System.out.print(" ");
            inorderRec(root.right);
        }
    }

    public static void main(String[] args){
        int[] nodeData = { 10, 5, 1, 7, 40, 50 };

        for (int i = 0; i < nodeData.length; i++) {
            create(nodeData[i]);
        }
        inorderRec(node);
    }
}
```

Output:

```
1 5 7 10 40 50
```

Practical 9: Graphs

1. **For a given graph $G = (V, E)$, study and implement the Breadth First Search (or traversal) i.e., BFS. Also, perform complexity analysis of this algorithm in-terms of time and space.**

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

```
public class BFS {
   private int node;
   private LinkedList<Integer> adj[];
   private Queue<Integer> que;

   BFS(int v) {
      node = v;
      adj = new LinkedList[node];
      for (int i=0; i<v; i++) {
         adj[i] = new LinkedList<>();
      }
      que = new LinkedList<Integer>();
   }

   void insertEdge(int v,int w) {
      adj[v].add(w);
   }

   void bfsTraversal(int n) {
      boolean nodes[] = new boolean[node];
      int a = 0;
      nodes[n]=true;
      que.add(n);
      while (que.size() != 0) {
         n = que.poll();
         System.out.print(n+" ");
         for (int i = 0; i < adj[n].size(); i++) { // Iterate through the linked list and push all neighbors
into queue
            a = adj[n].get(i);
            if (!nodes[a]) {
               nodes[a] = true;
               que.add(a);
            }
         }
      }
   }
   public static void main(String args[]) {
      BFS graph = new BFS(10);
      graph.insertEdge(0, 1);
      graph.insertEdge(0, 2);
      graph.insertEdge(0, 3);
      graph.insertEdge(1, 3);
      graph.insertEdge(2, 4);
      graph.insertEdge(3, 5);
      graph.insertEdge(3, 6);
      graph.insertEdge(4, 5);
```

```
        graph.insertEdge(4, 7);
        graph.insertEdge(5, 2);
        graph.insertEdge(6, 5);
        graph.insertEdge(7, 5);
        graph.insertEdge(7, 8);

        System.out.print("Breadth First Traversal is: ");
        graph.bfsTraversal(0);
    }
}
```

Output:

```
Breadth First Traversal is: 0 1 2 3 4 5 6 7 8
```

2. **For a given graph $G = (V, E)$, study and implement the Depth First Search (or traversal) i.e., DFS. Also, perform complexity analysis of this algorithm in-terms of time and space.**

```
import java.util.*;

public class DFS {
  private LinkedList<Integer> adj[];
  private boolean visited[];

  DFS(int V)
  {
    adj = new LinkedList[V];
    visited = new boolean[V];

    for (int i = 0; i < V; i++)
      adj[i] = new LinkedList<Integer>();
  }

  void insertEdge(int src, int dest) {
    adj[src].add(dest);
  }

  void dfsTraversal(int vertex) {
    visited[vertex] = true;
    System.out.print(vertex + " ");

    Iterator<Integer> it = adj[vertex].listIterator();
    while (it.hasNext()) {
      int n = it.next();
      if (!visited[n])
        dfsTraversal(n);
    }
  }

  public static void main(String args[]) {
```

```
    DFS graph = new DFS(8);

      graph.insertEdge(0, 1);
      graph.insertEdge(0, 2);
      graph.insertEdge(0, 3);
      graph.insertEdge(1, 3);
      graph.insertEdge(2, 4);
      graph.insertEdge(3, 5);
      graph.insertEdge(3, 6);
      graph.insertEdge(4, 7);
      graph.insertEdge(4, 5);
      graph.insertEdge(5, 2);

      System.out.print("Depth First Traversal is: ");
      graph.dfsTraversal(0);
  }
}
```

Output:

```
Depth First Traversal is: 0 1 3 5 2 4 7 6
```

3. **Given a directed graph, check whether the graph contains a cycle or not. Your function should return true if the given graph contains at least one cycle, else return false. Perform same task for undirected graph as well.**

```
// Detect cycle in a directed graph
import java.util.*

class Graph {
    private final int V;
    private final List<List<Integer>> adj;

    public Graph(int V){
        this.V = V;
        adj = new ArrayList<>(V);

        for (int i = 0; i < V; i++)
            adj.add(new LinkedList<>());
    }

    private boolean isCyclicUtil(int i, boolean[] visited, boolean[] recStack){
        if (recStack[i])
        return true;
        if (visited[i])
            return false;
        visited[i] = true;

        recStack[i] = true;
        List<Integer> children = adj.get(i);
```

```
            for (Integer c: children)
                if (isCyclicUtil(c, visited, recStack))
                    return true;
            recStack[i] = false;
            return false;
    }

    private void addEdge(int source, int dest) {
            adj.get(source).add(dest);
    }

    private boolean isCyclic(){
            boolean[] visited = new boolean[V];
            boolean[] recStack = new boolean[V];

            for (int i = 0; i < V; i++)
                if (isCyclicUtil(i, visited, recStack))
                    return true;

            return false;
    }

    public static void main(String[] args){
            Graph graph = new Graph(4);
            graph.addEdge(0, 1);
            graph.addEdge(0, 2);
            graph.addEdge(1, 2);
            graph.addEdge(2, 0);
            graph.addEdge(2, 3);
            graph.addEdge(3, 3);

            if(graph.isCyclic())
                System.out.println("Graph contains cycle");
            else
                System.out.println("Graph doesn't " + "contain cycle");
    }
}
```

Output:

```
Graph contains cycle
```

// Detect cycle in an undirected graph

```
import java.io.*;
import java.util.*;
class Graph {
   private int V;
```

```
private LinkedList<Integer> adj[];
Graph(int v){
   V = v;
   adj = new LinkedList[v];
   for (int i = 0; i < v; ++i)
      adj[i] = new LinkedList();
}

void addEdge(int v, int w){
   adj[v].add(w);
   adj[w].add(v);
}

Boolean isCyclicUtil(int v, Boolean visited[], int parent){
   visited[v] = true;
   Integer i;
   Iterator<Integer> it = adj[v].iterator();
   while (it.hasNext()) {
      i = it.next();

      if (!visited[i]) {
         if (isCyclicUtil(i, visited, v))
            return true;
      }
      else if (i != parent)
         return true;
   }
   return false;
}
Boolean isCyclic(){
   Boolean visited[] = new Boolean[V];
   for (int i = 0; i < V; i++)
      visited[i] = false;

   for (int u = 0; u < V; u++) {
      if (!visited[u])
         if (isCyclicUtil(u, visited, -1))
            return true;
   }

   return false;
}

public static void main(String args[]){
   Graph g1 = new Graph(5);
   g1.addEdge(1, 0);
   g1.addEdge(0, 2);
```

```
        g1.addEdge(2, 1);
        g1.addEdge(0, 3);
        g1.addEdge(3, 4);
        if (g1.isCyclic())
            System.out.println("Graph contains cycle");
        else
            System.out.println("Graph doesn't contain cycle");

        Graph g2 = new Graph(3);
        g2.addEdge(0, 1);
        g2.addEdge(1, 2);
        if (g2.isCyclic())
            System.out.println("Graph contains cycle");
        else
            System.out.println("Graph doesn't contain cycle");
    }
}
```

Output:

```
Graph contains cycle
Graph doesn't contain cycle
```

4. **Implement Minimum Spanning Tree (MST) using the greedy Kruskal's algorithm. The MST or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.**

```
import java.util.*;

class Graph {
  class Edge implements Comparable<Edge> {
    int src, dest, weight;

    public int compareTo(Edge compareEdge) {
      return this.weight - compareEdge.weight;
    }
  };

  // Union
  class subset {
    int parent, rank;
  };

  int vertices, edges;
  Edge edge[];

  // Graph creation
  Graph(int v, int e) {
    vertices = v;
```

```
  edges = e;
  edge = new Edge[edges];
  for (int i = 0; i < e; ++i)
    edge[i] = new Edge();
 }

 int find(subset subsets[], int i) {
  if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);
  return subsets[i].parent;
 }

 void Union(subset subsets[], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);

  if (subsets[xroot].rank < subsets[yroot].rank)
    subsets[xroot].parent = yroot;
  else if (subsets[xroot].rank > subsets[yroot].rank)
    subsets[yroot].parent = xroot;
  else {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;
  }
 }

 // Applying Krushkal Algorithm
 void KruskalAlgo() {
  Edge result[] = new Edge[vertices];
  int e = 0;
  int i = 0;
  for (i = 0; i < vertices; ++i)
    result[i] = new Edge();

  // Sorting the edges
  Arrays.sort(edge);
  subset subsets[] = new subset[vertices];
  for (i = 0; i < vertices; ++i)
    subsets[i] = new subset();

  for (int v = 0; v < vertices; ++v) {
    subsets[v].parent = v;
    subsets[v].rank = 0;
  }
  i = 0;
  while (e < vertices - 1) {
    Edge next_edge = new Edge();
```

```
    next_edge = edge[i++];
    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);
    if (x != y) {
      result[e++] = next_edge;
      Union(subsets, x, y);
    }
  }
  for (i = 0; i < e; ++i)
    System.out.println(result[i].src + " - " + result[i].dest + ": " + result[i].weight);
}

public static void main(String[] args) {
  int vertices = 6; // Number of vertices
  int edges = 8; // Number of edges
  Graph G = new Graph(vertices, edges);

  G.edge[0].src = 0;
  G.edge[0].dest = 1;
  G.edge[0].weight = 4;

  G.edge[1].src = 0;
  G.edge[1].dest = 2;
  G.edge[1].weight = 4;

  G.edge[2].src = 1;
  G.edge[2].dest = 2;
  G.edge[2].weight = 2;

  G.edge[3].src = 2;
  G.edge[3].dest = 3;
  G.edge[3].weight = 3;

  G.edge[4].src = 2;
  G.edge[4].dest = 5;
  G.edge[4].weight = 2;

  G.edge[5].src = 2;
  G.edge[5].dest = 4;
  G.edge[5].weight = 4;

  G.edge[6].src = 3;
  G.edge[6].dest = 4;
  G.edge[6].weight = 3;

  G.edge[7].src = 5;
  G.edge[7].dest = 4;
```

```
    G.edge[7].weight = 3;
    G.KruskalAlgo();
  }
}
```

Output:

```
1 - 2: 2
2 - 5: 2
2 - 3: 3
3 - 4: 3
0 - 1: 4
```