```
/* Program that reverses array in less number of swaps*/

public class arrayReverse {
   static void reverse(int a[], int n)
   {
      int i, k, t;
      for (i = 0; i < n / 2; i++) {
         t = a[i];
         a[i] = a[n - i - 1];
         a[n - i - 1] = t;
      }
      System.out.println("Reversed array is: \n");
      for (k = 0; k < n; k++) {
         System.out.println(a[k]);
      }
   }
   public static void main(String[] args)
   {
      int [] arr = {10, 20, 30, 40, 50};
      reverse(arr, arr.length);
   }
}
```

```
/*JAVA program to check whether two strings  are anagrams of each other */
import java.io.*;
import java.util.Arrays;
import java.util.Collections;

class GFG {
   static boolean areAnagram(char[] str1, char[] str2)
   {
      int n1 = str1.length;
      int n2 = str2.length;
      if (n1 != n2)
         return false;
      Arrays.sort(str1);
      Arrays.sort(str2);
      for (int i = 0; i < n1; i++)
         if (str1[i] != str2[i])
            return false;
```

```
        return true;
    }
    public static void main(String args[])
    {
        char str1[] = { 't', 'e', 's', 't' };
        char str2[] = { 't', 't', 'e', 'w' };
        if (areAnagram(str1, str2))
            System.out.println("The two strings are"
                    + " anagram of each other");
        else
            System.out.println("The two strings are not"
                    + " anagram of each other");
    }
}
```

```
/*Java code to find  duplicates in O(n) time */

class FindDuplicate
{
    void printRepeating(int arr[], int size)
    {
        int i;
        System.out.println("The repeating elements are : ");

        for (i = 0; i < size; i++)
        {
            if (arr[ Math.abs(arr[i]) ] >= 0)
                arr[ Math.abs(arr[i]) ] = -arr[ Math.abs(arr[i]) ];
            else
        System.out.print(Math.abs(arr[i]) + " ");
        }
    }
    public static void main(String[] args)
    {
        FindDuplicate duplicate = new FindDuplicate();
        int arr[] = {1, 2, 3, 1, 3, 6, 6};
        int arr_size = arr.length;
        duplicate.printRepeating(arr, arr_size);
    }
```

```
}
```

```
/* An efficient Java program to randomly select k items from a stream of items */

import java.util.Arrays;
import java.util.Random;
public class ReservoirSampling
{
   static void selectKItems(int stream[], int n, int k)
   {
      int i;
      int reservoir[] = new int[k];
      for (i = 0; i < k; i++)
         reservoir[i] = stream[i];
      Random r = new Random();
      for (; i < n; i++)
      {
         int j = r.nextInt(i + 1);
         if (j < k)
            reservoir[j] = stream[i];
      }
      System.out.println("Following are k randomly selected items");
System.out.println(Arrays.toString(reservoir));
   }
   public static void main(String[] args) {
      int stream[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
      int n = stream.length;
      int k = 5;
      selectKItems(stream, n, k);
   }
}
```

```
/*Java program to rearrange link list in place */

class LinkedList {
```

```
static Node head;
static class Node {
    int data;
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    }
}
void printlist(Node node)
{
    if (node == null) {
        return;
    }
    while (node != null) {
        System.out.print(node.data + " -> ");
        node = node.next;
    }
}
Node reverselist(Node node)
{
    Node prev = null, curr = node, next;
    while (curr != null) {
        next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    node = prev;
    return node;
}
void rearrange(Node node)
{
    Node slow = node, fast = slow.next;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    Node node1 = node;
    Node node2 = slow.next;
    slow.next = null;
    node2 = reverselist(node2);
```

```
        node = new Node(0);
        Node curr = node;
        while (node1 != null || node2 != null)
{
if (node1 != null) {
                curr.next = node1;
                curr = curr.next;
                node1 = node1.next;
            }
            if (node2 != null) {
                curr.next = node2;
                curr = curr.next;
                node2 = node2.next;
            }
        }
        node = node.next;
    }
    public static void main(String[] args)
    {
        LinkedList list = new LinkedList();
        list.head = new Node(1);
        list.head.next = new Node(2);
        list.head.next.next = new Node(3);
        list.head.next.next.next = new Node(4);
        list.head.next.next.next.next = new Node(5);
        list.printlist(head);
        list.rearrange(head);
        System.out.println("");
        list.printlist(head);
    }
}
```

```
/*Java program for reversing the linked list*/

class LinkedList {
    static Node head;
    static class Node {
        int data;
```

```
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    }
}
Node reverse(Node node)
{
    Node prev = null;
    Node current = node;
    Node next = null;
    while (current != null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    node = prev;
    return node;
}
void printList(Node node)
{
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
}
public static void main(String[] args)
{
    LinkedList list = new LinkedList();
    list.head = new Node(85);
    list.head.next = new Node(15);
    list.head.next.next = new Node(4);
    list.head.next.next.next = new Node(20);
    System.out.println("Given Linked list");
    list.printList(head);
    head = list.reverse(head);
    System.out.println("");
    System.out.println("Reversed linked list ");
    list.printList(head);
}
}
```

```
/* Java program to implement a stack that supports */

import java.util.*;
class GFG
{
static class MyStack
{
   Stack<Integer> s = new Stack<Integer>();
   int maxEle;
   void getMax()
   {
      if (s.empty())
         System.out.print("Stack is empty\n");
      else
         System.out.print("Maximum Element in" +
                  "the stack is: "+maxEle + "\n");
   }
   void peek()
   {
      if (s.empty())
      {
         System.out.print("Stack is empty ");
         return;
      }
      int t = s.peek();
      System.out.print("Top Most Element is: ");
      if(t > maxEle)
         System.out.print(maxEle);
      else
         System.out.print(t);
   }
   void pop()
   {
      if (s.empty())
      {
         System.out.print("Stack is empty\n");
         return;
      }
      System.out.print("Top Most Element Removed: ");
      int t = s.peek();
```

```
        s.pop();
        if (t > maxEle)
        {
            System.out.print(maxEle + "\n");
            maxEle = 2 * maxEle - t;
        }
        else
            System.out.print(t + "\n");
    }
    void push(int x)
    {
        if (s.empty())
        {
            maxEle = x;
            s.push(x);
            System.out.print("Number Inserted: " + x + "\n");
            return;
        }
        if (x > maxEle)
        {
            s.push(2 * x - maxEle);
            maxEle = x;
        }
        else
            s.push(x);
        System.out.print("Number Inserted: " + x + "\n");
    }
};
public static void main(String[] args)
{
    MyStack s = new MyStack();
    s.push(3);
    s.push(5);
    s.getMax();
    s.push(7);
    s.push(19);
    s.getMax();
    s.pop();
    s.getMax();
    s.pop();
    s.peek();
    }
}
```