**1.Create a queue with two stacks**

Program:

public class Queue<E>

{

private Stack<E> queue1 = new Stack<E>();

private Stack<E> queue2= new Stack<E>();

public void queue(E item) {

queue1.push(item);

}

public E dequeue() {

if (queue2.isEmpty()) {

while (!queue1.isEmpty()) {

outbox.push(queue1.pop());

}

}

return queue2.pop();

}

}

**2.Reverse the words of a string.**

Program:

public class StringFormatter {

public static String reverseWord(String str){

    String words[]=str.split("\\s");

    String reverseWord="";

    for(String w:words){

        StringBuilder sb=new StringBuilder(w);

        sb.reverse();

        reverseWord+=sb.toString()+" ";

    }

    return reverseWord.trim();

}

}

public class TestStringFormatter {

public static void main(String[] args) {

    System.out.println(StringFormatter.reverseWord("I am a boy"));

    }

}

**3.Count and show the repeated words in a string without using hashmap.**

Program:

public class DuplicateWord {

    public static void main(String[] args) {

        String string = " abc bca cba abc abc cba ";

        int count;

        string = string.toLowerCase();

        String words[] = string.split(" ");

        System.out.println("Duplicate words in a given string : ");

        for(int i = 0; i < words.length; i++) {

            count = 1;

            for(int j = i+1; j < words.length; j++) {

                if(words[i].equals(words[j])) {

                    count++;

                    words[j] = "0";

                }

            }

                 if(count > 1 && words[i] != "0")

                System.out.println(words[i]);

        }

    }

}

**4. Print the following diagram:**

**\* \* \* \* \* \***

**\* \* \* \***

**\* \***

**\***

**Program:**

public class Pattern

{

public static void main(String[] args)

{

int rows=4;

for (int i= 0; i<= rows-1; i++)

{

for (int j=0; j<=i; j++)

{

System.out.print(" ");

}

for (int k=0; k<=rows-1-i; k++)

{

System.out.print("\*" + " ");

}

System.out.println();

}

}

}

5. **String1: "189, CORNER STREET , HIGHWAY ROAD , HYDERABAD"**

**String2: "189, HIGHWAY ROAD , CORNER STREET , HYDERABAD"**

**Given two address Strings where house number, street and road name and city are separated**

**by commas, but are in a jumbled manner as shown in the example.**

**When the above two addresses are compared they should return "SAME".**

**If address strings don’t match return "NOT SAME"**

Program:

public String compare\_address(String string1, String2){

public static boolean equivalent(String s1, String s2) {

char[] chars1 = s1.toCharArray();

char[] chars2 = s2.toCharArray();

Arrays.sort(chars1);

Arrays.sort(chars2);

return Arrays.equals(chars1, chars2);

}

assert equivalent("abc", "cba");

assert !equivalent("abc", "acc");

if (s1 == null && s2 == null)

return true;

if (s1 == null || s2 == null)

return false;

if (s1.length() != s2.length())

return false;

}

**6. Given an array of numbers. Sort them in the format: Negative Numbers....Positive Numbers (Smallest to Highest)**

**\*\*Don't use any in built sort function. Efficiency of your algorithm will be taken into account.**

**Eg: -5 6 -7 3 -1 3 9**

**Solution: -7 -5 -1 3 3 6 9**

**Program:**

class SmallToHigh {

    static void printArray(int A[], int size)

    {

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

            System.out.print(A[i] + " ");

        System.out.println();

    }

    static void merge(int arr[], int l, int m, int r)

    {

        int i, j, k;

        int n1 = m - l + 1;

        int n2 = r - m;

        int L[] = new int[n1];

        int R[] = new int[n2];

        for (i = 0; i < n1; i++)

            L[i] = arr[l + i];

        for (j = 0; j < n2; j++)

            R[j] = arr[m + 1 + j];

        i = 0;

        j = 0;

        k = l;

        while (i < n1 && L[i] < 0)

            arr[k++] = L[i++];

        while (j < n2 && R[j] < 0)

            arr[k++] = R[j++];

        while (i < n1)

            arr[k++] = L[i++];

        while (j < n2)

            arr[k++] = R[j++];

    }

    static void RearrangePosNeg(int arr[], int l, int r)

    {

        if (l < r) {

            int m = l + (r - l) / 2;

            RearrangePosNeg(arr, l, m);

            RearrangePosNeg(arr, m + 1, r);

            merge(arr, l, m, r);

        }

    }

    public static void main(String[] args)

    {

        int arr[] = { -12, 11, -13, -5, 6, -7, 5, -3, -6 };

        int arr\_size = arr.length;

        RearrangePosNeg(arr, 0, arr\_size - 1);

        printArray(arr, arr\_size);

    }

}

**7. Reverse a stack without any extra space i.e no extra array or Stack**

**Push Order : 1-> 2-> 3-> 4 -> 5(Top)**

Program:

class Test {

    static Stack<Character> st = new Stack<>();

    static void insert\_at\_bottom(char x)

    {

        if(st.isEmpty())

            st.push(x);

        else

        {

            char a = st.peek();

            st.pop();

            insert\_at\_bottom(x);

            st.push(a);

        }

    }

    static void reverse()

    {

        if(st.size() > 0)

        {

            char x = st.peek();

            st.pop();

            reverse();

            insert\_at\_bottom(x);

        }

    }

        public static void main(String[] args)

    {

        st.push('1');

        st.push('2');

        st.push('3');

        st.push('4');

        st.push('5');

        System.out.println("Original Stack");

        System.out.println(st);

        reverse();

        System.out.println("Reversed Stack");

        System.out.println(st);

    }

}