Assignment Questions 16

Question 1.

Given an array, for each element find the value of the nearest element to the right which is having a frequency greater than that of the current element. If there does not exist an answer for a position, then make the value '-1'.

Examples:

Input: a[] = [1, 1, 2, 3, 4, 2, 1]

Output : [-1, -1, 1, 2, 2, 1, -1]

Explanation:

Given array a[] = [1, 1, 2, 3, 4, 2, 1]

Frequency of each element is: 3, 3, 2, 1, 1, 2, 3

Lets calls Next Greater Frequency element as NGF

1. For element a[0] = 1 which has a frequency = 3,
   As it has frequency of 3 and no other next element
   has frequency more than 3 so '-1'
2. For element a[1] = 1 it will be -1 same logic
   like a[0]
3. For element a[2] = 2 which has frequency = 2,
   NGF element is 1 at position = 6 with frequency
   of 3 > 2
4. For element a[3] = 3 which has frequency = 1,
   NGF element is 2 at position = 5 with frequency
   of 2 > 1
5. For element a[4] = 4 which has frequency = 1,
   NGF element is 2 at position = 5 with frequency
   of 2 > 1
6. For element a[5] = 2 which has frequency = 2,

NGF element is 1 at position = 6 with frequency of 3 > 2

7. For element a[6] = 1 there is no element to its right, hence -1

Input : a[] = [1, 1, 1, 2, 2, 2, 2, 11, 3, 3]

Output : [2, 2, 2, -1, -1, -1, -1, 3, -1, -1]

code:-

```
import java.util.*;

class Greater {
	static void NFG(int a[], int n, int freq[])
	{
		Stack<Integer> s = new Stack<Integer>();
		s.push(0);
		int res[] = new int[n];
		for (int i = 0; i < n; i++)
			res[i] = 0;

		for (int i = 1; i < n; i++)
		{
			if (freq[a[s.peek()]] > freq[a[i]])
				s.push(i);
			else
			{
```

```
                while (freq[a[s.peek()]] < freq[a[i]]
                        && s.size() > 0)
                {
                        res[s.peek()] = a[i];
                        s.pop();
                }
                s.push(i);
        }
    }

    while (s.size() > 0)
    {
        res[s.peek()] = -1;
        s.pop();
    }

    for (int i = 0; i < n; i++)
    {
        System.out.print(res[i] + " ");
    }
}
public static void main(String args[])
{

    int a[] = { 1, 1, 2, 3, 4, 2, 1 };
    int len = 7;
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < len; i++)
    {
```

```
                if (a[i] > max)
                {
                        max = a[i];
                }
        }
        int freq[] = new int[max + 1];

        for (int i = 0; i < max + 1; i++)
                freq[i] = 0;
        for (int i = 0; i < len; i++)
        {
                freq[a[i]]++;
        }
        NFG(a, len, freq);
    }
}
```

Question 2.

Given a stack of integers, sort it in ascending order using another temporary stack.

Examples:

Input : [34, 3, 31, 98, 92, 23]

Output : [3, 23, 31, 34, 92, 98]

Input : [3, 5, 1, 4, 2, 8]

Output : [1, 2, 3, 4, 5, 8]

code:-

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

```
class SortStack
{
	public static Stack<Integer> sortstack(Stack<Integer>
                                                                                    input)
	{
		Stack<Integer> tmpStack = new Stack<Integer>();
		while(!input.isEmpty())
		{
			int tmp = input.pop();
			while(!tmpStack.isEmpty() && tmpStack.peek()
                                                                                    < tmp)
			{
			input.push(tmpStack.pop());
			}
			tmpStack.push(tmp);
		}
		return tmpStack;
	}
	public static void main(String args[])
	{
		Stack<Integer> input = new Stack<Integer>();
		input.add(34);
		input.add(3);
		input.add(31);
		input.add(98);
		input.add(92);
		input.add(23);
		Stack<Integer> tmpStack=sortstack(input);
```

```
                System.out.println("Sorted numbers are:");


                while (!tmpStack.empty())
                {
                        System.out.print(tmpStack.pop()+" ");
                }
        }
}
```

Question 3.

Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

Input : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

Input : Stack[] = [1, 2, 3, 4, 5, 6]

Output : Stack[] = [1, 2, 4, 5, 6]

code:-

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

public class GFG {
        static void deleteMid(Stack<Character> st,int n, int curr)
        {
                if (st.empty() || curr == n)
                        return;
                char x = st.pop();
```

```
            deleteMid(st, n, curr+1);
            if (curr != n/2)
                st.push(x);
        }
        public static void main(String args[])
        {
            Stack<Character> st = new Stack<Character>();
            st.push('1');
            st.push('2');
            st.push('3');
            st.push('4');
            st.push('5');
            st.push('6');
            st.push('7');
            deleteMid(st, st.size(), 0);
            while (!st.empty())
            {
                char p=st.pop();
                System.out.print(p + " ");
            }
        }
}
```

Question 4.

Given a Queue consisting of first n natural numbers (in random order). The task is to check whether the given Queue elements can be arranged in increasing order in another Queue using a stack. The operation allowed are:

1. Push and pop elements from the stack

2. Pop (Or Dequeue) from the given Queue.

3. Push (Or Enqueue) in the another Queue.

Examples :

Input : Queue[] = { 5, 1, 2, 3, 4 }

Output : Yes

Pop the first element of the given Queue i.e 5. Push 5 into the stack.

Now, pop all the elements of the given Queue and push them to second Queue.

Now, pop element 5 in the stack and push it to the second Queue.

Input : Queue[] = { 5, 1, 2, 6, 3, 4 }

Output : No

Push 5 to stack.

Pop 1, 2 from given Queue and push it to another Queue.

Pop 6 from given Queue and push to stack.

Pop 3, 4 from given Queue and push to second Queue.

Now, from using any of above operation, we cannot push 5 into the second Queue because it is below the 6 in the stack.

code:-

```
import java.util.*;

class GFG

{

static class Queue

{

        static Stack<Integer> s1 = new Stack<Integer>();

        static Stack<Integer> s2 = new Stack<Integer>();


        static void enQueue(int x)

        {
```

```
                while (!s1.isEmpty())
                {
                        s2.push(s1.pop());
                }
                s1.push(x);
                while (!s2.isEmpty())
                {
                        s1.push(s2.pop());
                }
        }
        static int deQueue()
        {
                if (s1.isEmpty())
                {
                        System.out.println("Q is Empty");
                        System.exit(0);
                }
                int x = s1.peek();
                s1.pop();
                return x;
        }
};

public static void main(String[] args)
{
        Queue q = new Queue();
        q.enQueue(1);
        q.enQueue(2);
        q.enQueue(3);
```

```
        System.out.println(q.deQueue());
        System.out.println(q.deQueue());
        System.out.println(q.deQueue());
}
}
```

Question 5.

Given a number , write a program to reverse this number using stack.

Examples:

Input : 365

Output : 563

Input : 6899

Output : 9986

code:-

```
import java.util.Stack;

public class GFG
{
        static Stack<Integer> st= new Stack<>();
        static void push_digits(int number)
        {
                while(number != 0)
                {
                        st.push(number % 10);
                        number = number / 10;
```

```
        }
    }
    static int reverse_number(int number)
    {
        push_digits(number);
        int reverse = 0;
        int i = 1;
        while (!st.isEmpty())
        {
            reverse = reverse + (st.peek() * i);
            st.pop();
            i = i * 10;
        }
        return reverse;
    }
    public static void main(String[] args)
    {
        int number = 39997;
        System.out.println(reverse_number(number));
    }
}
```

Question 6.

Given an integer k and a [queue]of integers, The task is to reverse the order of the first k elements of the queue, leaving the other elements in the same relative order.

Only following standard operations are allowed on queue.

- enqueue(x) : Add an item x to rear of queue

- dequeue() : Remove an item from front of queue

- size() : Returns number of elements in queue.

- front() : Finds front item.

code:-

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

public class GFG {
static Queue<Integer> reverseFirstK(Queue<Integer> q, int k) {
        solve(q, k);
        int s = q.size() - k;
        while( s-- > 0){
                int x = q.poll();
                q.add(x);
        }
        return q;
}
static void solve(Queue<Integer> q, int k){
        if(k == 0) return;
        int e = q.poll();
        solve(q, k - 1);
        q.add(e);
}
public static void main (String[] args) {
        Queue<Integer> queue = new LinkedList<Integer>();
        queue.add(10);
        queue.add(20);
        queue.add(30);
        queue.add(40);
```

```
        queue.add(50);
        queue.add(60);
        queue.add(70);
        queue.add(80);
        queue.add(90);
        queue.add(100);
        int k = 5;
        queue = reverseFirstK(queue, k);
        while (!queue.isEmpty()) {
                System.out.print(queue.poll() + " ");
        }
}
}
```

Question 7.

Given a sequence of n strings, the task is to check if any two similar words come together and then destroy each other then print the number of words left in the sequence after this pairwise destruction.

Examples:

Input : ab aa aa bcd ab

Output : 3

As aa, aa destroys each other so,

ab bcd ab is the new sequence.

Input : tom jerry jerry tom

Output : 0

As first both jerry will destroy each other.Then sequence will be tom, tom they will also destroy each other. So, the final sequence doesn't contain any word.

code:-

```
import java.util.Vector;

class Test
{
	static int removeConsecutiveSame(Vector <String > v)
	{
		int n = v.size();
		for (int i=0; i<n-1; )
		{
			if (v.get(i).equals(v.get(i+1)))
			{
				v.remove(i);
				v.remove(i);
				if (i > 0)
					i--;
				n = n-2;
			}
			else
				i++;
		}
		return v.size();
	}
	public static void main(String[] args)
	{
		Vector<String> v = new Vector<>();

		v.addElement("tom"); v.addElement("jerry");
```

```
                v.addElement("jerry");v.addElement("tom");
                System.out.println(removeConsecutiveSame(v));
        }
}
```

Question 8.

Given an array of integers, the task is to find the maximum absolute difference between the nearest left and the right smaller element of every element in the array.

Note: If there is no smaller element on right side or left side of any element then we take zero as the smaller element. For example for the leftmost element, the nearest smaller element on the left side is considered as 0. Similarly, for rightmost elements, the smaller element on the right side is considered as 0.

Examples:

Input : arr[] = {2, 1, 8}

Output : 1

Left smaller LS[] {0, 0, 1}

Right smaller RS[] {1, 0, 0}

Maximum Diff of abs(LS[i] - RS[i]) = 1

Input : arr[] = {2, 4, 8, 7, 7, 9, 3}

Output : 4

Left smaller LS[] = {0, 2, 4, 4, 4, 7, 2}

Right smaller RS[] = {0, 3, 7, 3, 3, 3, 0}

Maximum Diff of abs(LS[i] - RS[i]) = 7 - 3 = 4

Input : arr[] = {5, 1, 9, 2, 5, 1, 7}

Output : 1

code:-

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

```
class GFG
{
	static void leftSmaller(int arr[], int n, int SE[])
	{
		Stack<Integer> S = new Stack<>();
		for (int i = 0; i < n; i++)
		{
			while (!S.empty() && S.peek() >= arr[i])
			{
				S.pop();
			}
			if (!S.empty())
			{
				SE[i] = S.peek();
			}
			else
			{
				SE[i] = 0;
			}
			S.push(arr[i]);
		}
	}
	static int findMaxDiff(int arr[], int n)
	{
		int[] LS = new int[n]; // To store left smaller elements
		leftSmaller(arr, n, LS);
		int[] RRS = new int[n]; // To store right smaller elements in
		reverse(arr);
		leftSmaller(arr, n, RRS);
```

```
        int result = -1;
        for (int i = 0; i < n; i++)
        {
            result = Math.max(result, Math.abs(LS[i] - RRS[n - 1 - i]));
        }
        return result;
    }

    static void reverse(int a[])
    {
        int i, k, n = a.length;
        int t;
        for (i = 0; i < n / 2; i++)
        {
            t = a[i];
            a[i] = a[n - i - 1];
            a[n - i - 1] = t;
        }
    }
    public static void main(String args[])
    {
        int arr[] = {2, 4, 8, 7, 7, 9, 3};
        int n = arr.length;
        System.out.println("Maximum diff : "+ findMaxDiff(arr, n));
    }
}
```