Assignment 16 Solution - Stacks | DSA

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

Let's 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]

Solution Code:

```
package in.ineuron.pptAssignment16;

import java.util.*;

public class NearestGreaterFrequency_1 {
	public static int[] findNearestGreaterFrequency(int[] arr) {
		int[] result = new int[arr.length];
		Map<Integer, Integer> frequencyMap = new HashMap<>();
		Map<Integer, Integer> nextGreaterFrequency = new HashMap<>();
		TreeMap<Integer, Integer> sortedMap = new TreeMap<>();

		for (int i = arr.length - 1; i >= 0; i--) {
			frequencyMap.put(arr[i], frequencyMap.getOrDefault(arr[i], 0) + 1);
```

```
                sortedMap.put(arr[i], i);
                Integer higherKey = sortedMap.higherKey(arr[i]);

                if (higherKey != null && frequencyMap.get(higherKey) >
                frequencyMap.get(arr[i])) {
                        nextGreaterFrequency.put(arr[i], higherKey);
                } else {
                        nextGreaterFrequency.put(arr[i], -1);
                }
        }

        for (int i = 0; i < arr.length; i++) {
                result[i] = nextGreaterFrequency.get(arr[i]);
        }

        return result;
    }

    public static void main(String[] args) {
        int[] arr = { 1, 1, 1, 2, 2, 2, 2, 11, 3, 3 };
        int[] result = findNearestGreaterFrequency(arr);

        System.out.println(Arrays.toString(result));
    }
}
```

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]

Solution Code :

```
package in.ineuron.pptAssignment16;

import java.util.Stack;

public class StackSort_2 {
	public static Stack<Integer> sortStack(Stack<Integer> stack) {
		Stack<Integer> tempStack = new Stack<>();

		while (!stack.isEmpty()) {
			int temp = stack.pop();

			while (!tempStack.isEmpty() && tempStack.peek() > temp) {
				stack.push(tempStack.pop());
			}

			tempStack.push(temp);
		}

		return tempStack;
	}
	public static void main(String[] args) {
		Stack<Integer> stack = new Stack<>();
		stack.push(34);
		stack.push(3);
		stack.push(31);
		stack.push(98);
		stack.push(92);
		stack.push(23);
s		Stack<Integer> sortedStack = sortStack(stack);

		System.out.println("Sorted Stack:");
		while (!sortedStack.isEmpty()) {
			System.out.print(sortedStack.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]

Solution Code:

```
package in.ineuron.pptAssignment16;

import java.util.Stack;

public class DeleteMiddleElement_3 {
	public static void deleteMiddle(Stack<Integer> stack, int k) {
		// Base case: if the stack is empty or k becomes 0, return
		if (stack.isEmpty() || k == 0) {
			stack.pop();
			return;
		}

		// Pop the top element
		int temp = stack.pop();

		// Recursive call to delete the middle element
		deleteMiddle(stack, k - 1);

		// Push the popped element back if it's not the middle element
		if (k != stack.size() / 2 + 1) {
			stack.push(temp);
		}
	}

	public static void main(String[] args) {
		Stack<Integer> stack = new Stack<>();
		stack.push(1);
		stack.push(2);
		stack.push(3);
		stack.push(4);
		stack.push(5);
		stack.push(5);

		int middleIndex = stack.size() / 2 + 1;
		deleteMiddle(stack, middleIndex);
```

```
                System.out.println("Modified Stack:");
                while (!stack.isEmpty()) {
                        System.out.print(stack.pop() + " ");
                }
        }
}
```

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 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

Solution Code:

```
package in.ineuron.pptAssignment16;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;
public class QueueArrangeInIncreasingOrder_4 {
	public static boolean checkIncreasingOrder(Queue<Integer> queue) {
		int n = queue.size();
		Stack<Integer> stack = new Stack<>();
		Queue<Integer> sortedQueue = new LinkedList<>();

		for (int i = 1; i <= n; i++) {
			if (!queue.isEmpty() && queue.peek() == i) {
				sortedQueue.add(queue.poll());
			} else if (!stack.isEmpty() && stack.peek() == i) {
				sortedQueue.add(stack.pop());
				i--;
			} else {
				if (queue.isEmpty()) {
					return false;
				}
				stack.push(queue.poll());
				i--;
			}
		}

		return sortedQueue.equals(queue);
	}

	public static void main(String[] args) {
		Queue<Integer> queue = new LinkedList<>();
		queue.add(5);
		queue.add(1);
		queue.add(2);
		queue.add(3);
		queue.add(4);

		boolean canArrange = checkIncreasingOrder(queue);

		if (canArrange) {
		System.out.println("Yes, the elements can be arranged in increasing order.");
		} else {
		System.out.println("No, the elements cannot be arranged in increasing order.");
		}
	}
}
```

Question 5

Given a number , write a program to reverse this number using stack.
Examples:
Input : 365
Output : 563

Input : 6899
Output : 9986

Solution Code:

```
package in.ineuron.pptAssignment16;

import java.util.Stack;

public class NumberReverseUsingStack_5 {
	public static int reverseNumber(int number) {
		Stack<Integer> stack = new Stack<>();

		// Push each digit of the number into the stack
		while (number > 0) {
			int digit = number % 10;
			stack.push(digit);
			number /= 10;
		}

		int reversedNumber = 0;
		int placeValue = 1;

		// Pop digits from the stack and construct the reversed number
		while (!stack.isEmpty()) {
			int digit = stack.pop();
			reversedNumber += digit * placeValue;
			placeValue *= 10;
		}

		return reversedNumber;
	}

	public static void main(String[] args) {
		int number = 365;s
		int reversedNumber = reverseNumber(number);

		System.out.println("Reversed Number: " + reversedNumber);
	}
}
```

Question 6

Given an integer k and a [queue](https://www.geeksforgeeks.org/queue-data-structure/) 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.

Solution Code:

```
package in.ineuron.pptAssignment16;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class ReverseKElementsInQueue_6 {
	public static void reverseFirstKElements(Queue<Integer> queue, int k) {
		if (queue.isEmpty() || k <= 0 || k > queue.size()) {
			return; // Invalid input, no reversal needed
		}

		Stack<Integer> stack = new Stack<>();

		// Push the first k elements into the stack
		for (int i = 0; i < k; i++) {
			stack.push(queue.poll());
		}

		// Enqueue the elements from the stack back into the queue
		while (!stack.isEmpty()) {
			queue.add(stack.pop());
		}

		// Rotate the remaining elements by k positions
		for (int i = 0; i < queue.size() - k; i++) {
			queue.add(queue.poll());
		}
	}
```

```
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(1);
        queue.add(2);
        queue.add(3);
        queue.add(4);
        queue.add(5);
        queue.add(6);

        int k = 4;
        reverseFirstKElements(queue, k);

        System.out.println("Reversed Queue:");
        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 than 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.

Solution Code:

```
package in.ineuron.pptAssignment16;

import java.util.*;

public class PairwiseDestruction_7 {
	public static int countWordsLeft(String[] words) {
		Stack<String> stack = new Stack<>();

		for (String word : words) {
			if (!stack.isEmpty() && stack.peek().equals(word)) {
				stack.pop();
			} else {
				stack.push(word);
			}
		}

		return stack.size();
	}

	public static void main(String[] args) {
		String[] words = { "ab", "aa", "aa", "bcd", "ab" };
		int wordsLeft = countWordsLeft(words);

		System.out.println("Number of Words Left: " + wordsLeft);
	}
}
```

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

Solution Code:

```
package in.ineuron.pptAssignment16;

import java.util.Stack;
public class MaxAbsoluteDifference_8 {
	public static int maxAbsoluteDifference(int[] arr) {
		int n = arr.length;
		int[] leftSmaller = new int[n];
			// Stores the nearest smaller element on the left side
		int[] rightSmaller = new int[n];
			// Stores the nearest smaller element on the right side

		// Find the nearest smaller element on the left side for each element
		Stack<Integer> stack = new Stack<>();
		for (int i = 0; i < n; i++) {
			while (!stack.isEmpty() && stack.peek() >= arr[i]) {
				stack.pop();
			}
			leftSmaller[i] = stack.isEmpty() ? 0 : stack.peek();
			stack.push(arr[i]);
		}
		stack.clear();
```

```
            // Find the nearest smaller element on the right side for each element
            for (int i = n - 1; i >= 0; i--) {
                while (!stack.isEmpty() && stack.peek() >= arr[i]) {
                    stack.pop();
                }
                rightSmaller[i] = stack.isEmpty() ? 0 : stack.peek();
                stack.push(arr[i]);
            }

            int maxDiff = 0;

            // Find the maximum absolute difference between the nearest left and right
            // smaller elements
            for (int i = 0; i < n; i++) {
                int diff = Math.abs(leftSmaller[i] - rightSmaller[i]);
                maxDiff = Math.max(maxDiff, diff);
            }

            return maxDiff;
        }

        public static void main(String[] args) {
            int[] arr = { 2, 1, 8 };
            int maxDiff = maxAbsoluteDifference(arr);

            System.out.println("Maximum Absolute Difference: " + maxDiff);
        }
    }
```