Assignment 16 Solution - Stacks | DSA

Question 1

Given an array arr[ ] of size N having elements, the task is to find the next greater element for each element of the array in order of their appearance in the array.Next greater element of an element in the array is the nearest element on the right which is greater than the current element. If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

Example 1:
Input: N = 4, arr[] = [1 3 2 4]
Output: 3 4 4 -1
Explanation:
In the array, the next larger element to 1 is 3 , 3 is 4 , 2 is 4 and for 4 ? since it doesn't exist, it is -1.

Example 2:
Input: N = 5, arr[] [6 8 0 1 3]
Output: 8 -1 1 3 -1
Explanation:
In the array, the next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1 , for 1 it is 3 and then for 3 there is no larger element on right and hence -1.

Solution Code:

```
package in.ineuron.pptAssignment15;

//Java program to print next
//greater element using stack

public class NextGreaterElement_1 {
	static class stack {
		int top;
		int items[] = new int[100];

		// Stack functions to be used by printNGE
		void push(int x) {
			if (top == 99) {
				System.out.println("Stack full");
			} else {
				items[++top] = x;
			}
		}

		int pop() {
			if (top == -1) {
```

```
                System.out.println("Underflow error");
                return -1;
            } else {
                int element = items[top];
                top--;
                return element;
            }
        }

        boolean isEmpty() {
            return (top == -1) ? true : false;
        }
    }

    static void printNGE(int arr[], int n) {
        int i = 0;
        stack s = new stack();
        s.top = -1;
        int element, next;

        /* push the first element to stack */
        s.push(arr[0]);

        // iterate for rest of the elements
        for (i = 1; i < n; i++) {
            next = arr[i];

            if (s.isEmpty() == false) {

                // if stack is not empty, then
                // pop an element from stack
                element = s.pop();

                while (element < next) {
                    System.out.println(element + " --> " + next);
                    if (s.isEmpty() == true)
                        break;
                    element = s.pop();
                }
                if (element > next)
                    s.push(element);
            }
            s.push(next);
        }
        while (s.isEmpty() == false) {
            element = s.pop();
            next = -1;
```

```
                        System.out.println(element + " -- " + next);
                }
        }

        // Driver Code
        public static void main(String[] args) {
                int arr[] = { 1, 2, 3, 4 };
                int n = arr.length;
                printNGE(arr, n);
        }
}
```

Question 2

Given an array a of integers of length n, find the nearest smaller number for every element such that the smaller element is on left side. If no small element presents on the left print -1.

Example 1:
Input: n = 3
a = {1, 6, 2}
Output: -1 1 1
Explanation : There is no number at the left of 1. Smaller number than 6 and 2 is 1.

Example 2:
Input: n = 6
a = {1, 5, 0, 3, 4, 5}
Output: -1 1 -1 0 3 4
Explanation:
Up to 3 it is easy to see the smaller numbers. But for 4 the smaller numbers are 1, 0 and 3. But among them 3 is closest. Similarly for 5 it is 4.

Solution Code:

```
package in.ineuron.pptAssignment15;

import java.util.Arrays;
import java.util.Stack;

public class NearestSmallerNumber_2 {
        public static int[] findNearestSmaller(int[] a) {
                int n = a.length;
                int[] result = new int[n];
                Arrays.fill(result, -1);

                Stack<Integer> stack = new Stack<>();
                for (int i = 0; i < n; i++) {
                        while (!stack.isEmpty() && a[stack.peek()] >= a[i]) {
                                stack.pop();
```

```
                }

                if (!stack.isEmpty()) {
                        result[i] = a[stack.peek()];
                }

                stack.push(i);
            }

            return result;
        }

        public static void main(String[] args) {
            int[] a = { 1, 6, 2 };
            int[] result = findNearestSmaller(a);
            for (int num : result) {
                System.out.print(num + " ");
            }
        }
    }
```

Question 3

Implement a Stack using two queues q1 and q2.

Example 1:

Input:

push(2)
push(3)
pop()
push(4)
pop()

Output:3 4

Explanation:

push(2) the stack will be {2}
push(3) the stack will be {2 3}
pop() popped element will be 3 the stack will be {2}
push(4) the stack will be {2 4}
pop() popped element will be 4

Example 2:

Input:

push(2)
pop()

pop()
push(3)
Output:2 -1

Solution Code:

```
package in.ineuron.pptAssignment15;

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

public class StackUsingQueues_3 {
    private Queue<Integer> q1;
    private Queue<Integer> q2;

    public StackUsingQueues_3() {
        q1 = new LinkedList<>();
        q2 = new LinkedList<>();
    }

    public void push(int x) {
        q1.add(x);
    }

    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1; // or throw an exception
        }

        while (q1.size() > 1) {
            q2.add(q1.remove());
        }

        int popped = q1.remove();

        Queue<Integer> temp = q1;
        q1 = q2;
        q2 = temp;

        return popped;
    }

    public int top() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1; // or throw an exception
        }
```

```
            while (q1.size() > 1) {
                q2.add(q1.remove());
            }

            int top = q1.peek();
            q2.add(q1.remove());

            Queue<Integer> temp = q1;
            q1 = q2;
            q2 = temp;

            return top;
        }

        public boolean isEmpty() {
            return q1.isEmpty() && q2.isEmpty();
        }

        public static void main(String[] args) {
            StackUsingQueues_3 stack = new StackUsingQueues_3();
            stack.push(2);
            stack.push(3);
            System.out.println(stack.pop()); // Output: 3
            stack.push(4);
            System.out.println(stack.pop()); // Output: 4
        }
    }
```

Question 4

You are given a stack St. You have to reverse the stack using recursion.

Example 1:

Input:St = {3,2,1,7,6}

Output:{6,7,1,2,3}

Example 2:

Input:St = {4,3,9,6}

Output:{6,9,3,4}

Solution Code:

```
package in.ineuron.pptAssignment15;

import java.util.Stack;

public class ReverseStackUsingRecursion_4 {
	public static void reverseStack(Stack<Integer> St) {
		if (St.isEmpty() || St.size() == 1) {
			return;
		}

		int topElement = St.pop();
		reverseStack(St);
		insertAtBottom(St, topElement);
	}

	public static void insertAtBottom(Stack<Integer> St, int element) {
		if (St.isEmpty()) {
			St.push(element);
			return;
		}

		int top = St.pop();
		insertAtBottom(St, element);
		St.push(top);
	}

	public static void main(String[] args) {
		Stack<Integer> St = new Stack<>();
		St.push(3);
		St.push(2);
		St.push(1);
		St.push(7);
		St.push(6);

		System.out.println("Original Stack: " + St);
```

```
                reverseStack(St);

                System.out.println("Reversed Stack: " + St);
        }
    }
```

Question 5

You are given a string S, the task is to reverse the string using stack.
Example 1:
Input: S="GeeksforGeeks"
Output: skeeGrofskeeG

Solution Code:

```
package in.ineuron.pptAssignment15;

import java.util.Stack;

public class ReverseStringUsingStack_5 {
    public static String reverseString(String S) {
        Stack<Character> stack = new Stack<>();
        int n = S.length();

        // Push characters onto the stack
        for (int i = 0; i < n; i++) {
            stack.push(S.charAt(i));
        }

        StringBuilder reversedStr = new StringBuilder();

        // Pop characters from the stack and append to reversedStr
        while (!stack.isEmpty()) {
            reversedStr.append(stack.pop());
        }

        return reversedStr.toString();
    }

    public static void main(String[] args) {
        String S = "GeeksforGeeks";
        String reversedString = reverseString(S);
        System.out.println(reversedString);
    }
}
```

Question 6

Given string S representing a postfix expression, the task is to evaluate the expression and find the final value. Operators will only include the basic arithmetic operators like , /, + and -.

Example 1:

Input: S = "231+9-"

Output: -4

Explanation:

After solving the given expression, we have -4 as result.

Example 2:

Input: S = "123+8-"

Output: -3

Explanation:

After solving the given postfix expression, we have -3 as result.

Solution Code:

```
package in.ineuron.pptAssignment15;

import java.util.Stack;

public class EvaluatePostfixExpression_6 {
	public static int evaluatePostfix(String S) {
		Stack<Integer> stack = new Stack<>();
		int n = S.length();

		for (int i = 0; i < n; i++) {
			char ch = S.charAt(i);

			if (Character.isDigit(ch)) {
				stack.push(ch - '0'); // Convert char to int and push onto stack
				} else {
					int operand2 = stack.pop();
					int operand1 = stack.pop();
					int result = performOperation(operand1, operand2, ch);
					stack.push(result);
			}
		}

		return stack.pop();
	}

	public static int performOperation(int operand1, int operand2, char operator) {
		switch (operator) {
		case '+':
			return operand1 + operand2;
		case '-':
```

```
                    return operand1 - operand2;
                case '*':
                    return operand1 * operand2;
                case '/':
                    return operand1 / operand2;
                default:
                    return 0; // Handle invalid operator
                }
        }

        public static void main(String[] args) {
            String S = "123+8-";
            int result = evaluatePostfix(S);
            System.out.println(result);
        }
    }
```

Question 7

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

Input : ["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output

[null,null,null,null,-3,null,0,-2]

Explanation

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
```

```
minStack.pop();
minStack.top();    // return 0
minStack.getMin(); // return -2
```

Solution Code:

```
package in.ineuron.pptAssignment15;

import java.util.Stack;

public class MinStack_7 {
    private Stack<Integer> stack;
    private Stack<Integer> minStack;

    public MinStack_7() {
        stack = new Stack<>();
        minStack = new Stack<>();
    }

    public void push(int val) {
        stack.push(val);
        if (minStack.isEmpty() || val <= minStack.peek()) {
            minStack.push(val);
        }
    }

    public void pop() {
        if (stack.isEmpty()) {
            return;
        }
        int popped = stack.pop();
        if (popped == minStack.peek()) {
            minStack.pop();
        }
    }

    public int top() {
        return stack.peek();
    }

    public int getMin() {
        return minStack.peek();
    }
```

```
    public static void main(String[] args) {
            MinStack_7 minStack = new MinStack_7();
            minStack.push(-2);
            minStack.push(0);
            minStack.push(-3);
            System.out.println(minStack.getMin()); // Output: -3
            minStack.pop();
            System.out.println(minStack.top()); // Output: 0
            System.out.println(minStack.getMin()); // Output: -2
        }
    }
```

Question 8

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:


```
```

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:
Input: height = [4,2,0,3,2,5]
Output: 9

Solution Code:

```
package in.ineuron.pptAssignment15;

public class TrappingRainWater_8 {
    public static int trap(int[] height) {
            int left = 0;
            int right = height.length - 1;
            int leftMax = 0;
            int rightMax = 0;
            int water = 0;
```

```
                while (left <= right) {
                        if (height[left] <= height[right]) {
                                if (height[left] > leftMax) {
                                        leftMax = height[left];
                                } else {
                                        water += leftMax - height[left];
                                }
                                left++;
                        } else {
                                if (height[right] > rightMax) {
                                        rightMax = height[right];
                                } else {
                                        water += rightMax - height[right];
                                }
                                right--;
                        }
                }

                return water;
        }

        public static void main(String[] args) {
                int[] height = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
                int trappedWater = trap(height);
                System.out.println("Trapped water: " + trappedWater);
        }
}
```