**Assignment15**

1.

function findNextGreater(arr) {

const N = arr.length;

const nextGreater = new Array(N).fill(-1);

const stack = [];

for (let i = N - 1; i >= 0; i--) {

while (stack.length && stack[stack.length - 1] <= arr[i]) {

stack.pop();

}

if (stack.length) {

nextGreater[i] = stack[stack.length - 1];

}

stack.push(arr[i]);

}

return nextGreater;

}

Example:

const arr1 = [1, 3, 2, 4];

const arr2 = [6, 8, 0, 1, 3];

console.log(findNextGreater(arr1));

Output: [3, 4, 4, -1]

console.log(findNextGreater(arr2));

Output: [8, -1, 1, 3, -1]

**2.**

function findNearestSmaller(a) {

const n = a.length;

const nearestSmaller = new Array(n).fill(-1);

const stack = [];

for (let i = 0; i < n; i++) {

while (stack.length && stack[stack.length - 1] >= a[i]) {

stack.pop();

}

if (stack.length) {

nearestSmaller[i] = stack[stack.length - 1];

}

stack.push(a[i]);

}

return nearestSmaller;

}

Example :

const a1 = [1, 6, 2];

const a2 = [1, 5, 0, 3, 4, 5];

console.log(findNearestSmaller(a1));

Output: [-1, 1, 1]

console.log(findNearestSmaller(a2));

Output: [-1, 1, -1, 0, 3, 4]

**3.**

class Stack {

constructor() {

this.q1 = [];

this.q2 = [];

}

push(element) {

this.q1.push(element);

}

pop() {

if (this.q1.length === 0) {

return -1;

}

while (this.q1.length > 1) {

this.q2.push(this.q1.shift());

}

const poppedElement = this.q1.shift();

const temp = this.q1;

this.q1 = this.q2;

this.q2 = temp;

return poppedElement;

}

}

Example:

const stack = new Stack();

stack.push(2);

stack.push(3);

console.log(stack.pop());

Output: 3

stack.push(4);

console.log(stack.pop());

Output: 4

const stack2 = new Stack();

stack2.push(2);

console.log(stack2.pop());

Output: 2

console.log(stack2.pop());

Output: -1

stack2.push(3);

console.log(stack2.pop());

Output: 3

**4.**

class Stack {

constructor() {

this.items = [];

}

push(element) {

this.items.push(element);

}

pop() {

if (this.isEmpty()) {

return -1;

}

return this.items.pop();

}

isEmpty() {

return this.items.length === 0;

}

reverseStack() {

if (this.isEmpty() || this.items.length === 1) {

return;

}

const topElement = this.pop();

this.reverseStack();

this.insertAtBottom(topElement);

}

insertAtBottom(element) {

if (this.isEmpty()) {

this.push(element);

return;

}

const topElement = this.pop();

this.insertAtBottom(element);

this.push(topElement);

}

}

// Example:

const stack1 = new Stack();

stack1.push(3);

stack1.push(2);

stack1.push(1);

stack1.push(7);

stack1.push(6);

stack1.reverseStack();

console.log(stack1.items);

Output: [6, 7, 1, 2, 3]

const stack2 = new Stack();

stack2.push(4);

stack2.push(3);

stack2.push(9);

stack2.push(6);

stack2.reverseStack();

console.log(stack2.items);

Output: [6, 9, 3, 4]

**5.**

function reverseString(S) {

const stack = [];

const reversedString = [];

for (let i = 0; i < S.length; i++) {

stack.push(S[i]);

}

while (stack.length > 0) {

reversedString.push(stack.pop());

}

return reversedString.join('');

}

Example:

const inputString = 'GeeksforGeeks';

console.log(reverseString(inputString));

Output: skeeGrofskeeG

**6.**

function evaluatePostfixExpression(S) {

const stack = [];

for (let i = 0; i < S.length; i++) {

const char = S[i];

if (isOperand(char)) {

stack.push(parseInt(char));

} else {

const operand2 = stack.pop();

const operand1 = stack.pop();

const result = evaluateOperation(operand1, operand2, char);

stack.push(result);

}

}

return stack.pop();

}

function isOperand(char) {

return !isNaN(parseInt(char));

}

function evaluateOperation(operand1, operand2, operator) {

switch (operator) {

case '+':

return operand1 + operand2;

case '-':

return operand1 - operand2;

case '\*':

return operand1 \* operand2;

case '/':

return operand1 / operand2;

default:

return 0;

}

}

Example:

const expression1 = '231\*+9-';

console.log(evaluatePostfixExpression(expression1));

Output: -4

const expression2 = '123+\*8-';

console.log(evaluatePostfixExpression(expression2));

Output: -3

**7.**

class MinStack {

constructor() {

this.stack = [];

this.minStack = [];

}

push(val) {

this.stack.push(val);

if (this.minStack.length === 0 || val <= this.minStack[this.minStack.length - 1]) {

this.minStack.push(val);

}

}

pop() {

const poppedElement = this.stack.pop();

if (poppedElement === this.minStack[this.minStack.length - 1]) {

this.minStack.pop();

}

}

top() {

return this.stack[this.stack.length - 1];

}

getMin() {

return this.minStack[this.minStack.length - 1];

}

}

Example:

const minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

console.log(minStack.getMin());

Output: -3

minStack.pop();

console.log(minStack.top());

Output: 0

console.log(minStack.getMin());

Output: -2

**8.**

function trapWater(height) {

let left = 0;

let right = height.length - 1;

let leftMax = 0;

let rightMax = 0;

let waterTrapped = 0;

while (left < right) {

if (height[left] <= height[right]) {

leftMax = Math.max(leftMax, height[left]);

waterTrapped += leftMax - height[left];

left++;

} else {

rightMax = Math.max(rightMax, height[right]);

waterTrapped += rightMax - height[right];

right--;

}

}

return waterTrapped;

}

Example:

const elevationMap1 = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1];

console.log(trapWater(elevationMap1));

Output: 6

const elevationMap2 = [4, 2, 0, 3, 2, 5];

console.log(trapWater(elevationMap2));

Output: 9