Stacks & Queues:

Stack:

A stack is a Last In First Out (LIFO) data structure. The last element added to the stack will be the first one to be removed.

Key Operations:

- push(): Add an element to the top of the stack.
- pop(): Remove and return the top element from the stack.
- peek(): View the top element without removing it.
- isEmpty(): Check if the stack is empty.

```
class Stack {
    constructor() {
        this.stack = [];
    }

    // Push element onto the stack
    push(element) {
        this.stack.push(element);
    }

    // Pop the top element from the stack
    pop() {
        if (this.isEmpty()) {
            return "Stack is empty";
        }
        return this.stack.pop();
    }

    // Peek at the top element without removing it
    peek() {
        if (this.isEmpty()) {
            return "Stack is empty";
        }
        return this.stack[this.stack.length - 1];
    }

    // Check if the stack is empty
    isEmpty() {
        return this.stack.length === 0;
    }

    // Get the size of the stack
    size() {
        return this.stack.length;
    }

    // Clear the stack
    clear() {
        this.stack = [];
    }
}

// Example usage:
const stack = new Stack();
stack.push(10);
stack.push(20);
stack.push(30);
console.log(stack.peek());  // Output: 30
console.log(stack.pop());   // Output: 30
console.log(stack.size());  // Output: 2
```

Time Complexity:

- push(): O(1)
- pop(): O(1)
- peek(): O(1)
- isEmpty(): O(1)

Problems Based on Stacks:

Next Greater Element To The right of each element present in the array:

Problem Link:https://www.geeksforgeeks.org/problems/next-larger-element-1587115620/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card

Solution-1:

class Solution

{

```
    //Function to find the next greater element for each element of the array.
    nextLargerElement(arr, n)
    {
        var ansarray=[];
        for(var i=0;i<n;i++){
            var maxi=-1;
            for(var j=i+1;j<n;j++){
                if(arr[j]>arr[i]){
                    maxi=arr[j];
                    break;
                }
            }
            ansarray.push(maxi);
        }
        return ansarray;
    }
}
```

Time Complexity:O(n^2)

Space complexity:O(1) excluding asked space

Solution-2:

```
class Stack {
    constructor() {
```

```
    this.stack = [];
}

// Push element onto the stack
push(element) {
    this.stack.push(element);
}

// Pop the top element from the stack
pop() {
    if (this.isEmpty()) {
        return "Stack is empty";
    }
    return this.stack.pop();
}

// Peek at the top element without removing it
peek() {
    if (this.isEmpty()) {
        return "Stack is empty";
    }
    return this.stack[this.stack.length - 1];
}
```

```
    // Check if the stack is empty
    isEmpty() {
        return this.stack.length === 0;
    }

    // Get the size of the stack
    size() {
        return this.stack.length;
    }

    // Clear the stack
    clear() {
        this.stack = [];
    }
}


class Solution
{

    nextLargerElement(arr, n)
    {
      var s=new Stack();
      var ansarray=[];
```

```
for(var i=n-1;i>=0;i--){
    if(s.isEmpty()){
        ansarray.push(-1);
        s.push(arr[i]);
    }
    else if(!s.isEmpty()&&s.peek()>arr[i]){
         ansarray.push(s.peek());
         s.push(arr[i]);
    }
    else if(!s.isEmpty()&&s.peek()<=arr[i]){
        while(!s.isEmpty()&&s.peek()<=arr[i]){
            s.pop();
        }
         if(s.isEmpty()){
        ansarray.push(-1);
        s.push(arr[i]);
    }
    else if(!s.isEmpty()&&s.peek()>arr[i]){
         ansarray.push(s.peek());
         s.push(arr[i]);
    }
    }
}
ansarray.reverse();
```

```
        return ansarray;
    }
}
```

Time Complexity:O(n)

Space Complexity:O(n)

Problem:Valid Parentheses

Pre- requisite:Discuss Valid Parentheses with round brackets first.

Problem Link:https://leetcode.com/problems/valid-parentheses/

Solution:

```
class Stack {
    constructor() {
        this.stack = [];
    }

    // Push element onto the stack
    push(element) {
        this.stack.push(element);
    }

    // Pop the top element from the stack
    pop() {
        if (this.isEmpty()) {
            return "Stack is empty";
        }
        return this.stack.pop();
    }

    // Peek at the top element without removing it
    peek() {
        if (this.isEmpty()) {
            return "Stack is empty";
        }
        return this.stack[this.stack.length - 1];
    }

    // Check if the stack is empty
    isEmpty() {
        return this.stack.length === 0;
    }

    // Get the size of the stack
    size() {
        return this.stack.length;
    }

    // Clear the stack
    clear() {
        this.stack = [];
    }
}


var isValid = function(s) {
    var st=new Stack();
    for(var i=0;i<s.length;i++){
        if(s[i]=='('||s[i]=='{'||s[i]=='['){
            st.push(s[i]);
        }
        else if(s[i]==')'){
            if(st.isEmpty()){
                return false;
            }
            else if(st.peek()!='('){
                return false;
            }
            else{
                return false;
                st.pop();
            }
        }
        else if(s[i]=='}'){
            if(st.isEmpty()){
```

```
            return false;
        }
        else if(st.peek()!='{'){
            return false;
        }
        else{
            st.pop();
        }
    }
    else if(s[i]==']'){
        if(st.isEmpty()){
            return false;
        }
        else if(st.peek()!='['){
            return false;
        }
        else{
            st.pop();
        }
    }
}
if(st.isEmpty()){
    return true;
}
```

```
    else{
        return false;
    }
};
```

Time Complexity:O(n)

Space Complexity:O(n)

Queues:

A queue is a linear data structure that follows the FIFO (First In, First Out) principle, meaning the first element added to the queue will be the first one to be removed.

In JavaScript, we can implement a queue using an array by:

- Using `push()` to add elements to the end of the array (enqueue operation).
- Using `shift()` to remove elements from the front of the array (dequeue operation).

Queue Operations:

1. Enqueue: Add an element to the end of the queue.
2. Dequeue: Remove an element from the front of the queue.
3. Peek/Front: View the element at the front of the queue without removing it.
4. isEmpty: Check if the queue is empty.
5. Size: Get the number of elements in the queue.

Queue Implementation Using Array

```
class Queue {
  constructor() {
    this.queue = [];
  }

  // Enqueue operation (Add element to the end of the queue)
  enqueue(element) {
```

```
  this.queue.push(element);
  console.log(`${element} added to the queue`);
}

// Dequeue operation (Remove element from the front of the queue)
dequeue() {
  if (this.isEmpty()) {
    console.log('Queue is empty, cannot dequeue');
    return;
  }
  const removedElement = this.queue.shift();
  console.log(`${removedElement} removed from the queue`);
  return removedElement;
}

// Peek operation (View the element at the front of the queue)
peek() {
  if (this.isEmpty()) {
    console.log('Queue is empty');
    return;
  }
  return this.queue[0];
}
```

```
  // Check if the queue is empty
  isEmpty() {
    return this.queue.length === 0;
  }

  // Get the size of the queue
  size() {
    return this.queue.length;
  }

  // Print the queue
  printQueue() {
    console.log('Queue:', this.queue.join(', '));
  }
}

// Example usage
const myQueue = new Queue();

// Enqueue elements
myQueue.enqueue(10);
myQueue.enqueue(20);
myQueue.enqueue(30);
```

```
// Print queue

myQueue.printQueue();


// Peek at the front of the queue

console.log('Front of queue:', myQueue.peek());


// Dequeue elements

myQueue.dequeue();

myQueue.dequeue();


// Check the size of the queue

console.log('Queue size:', myQueue.size());


// Print the queue again

myQueue.printQueue();


// Check if the queue is empty

console.log('Is queue empty?', myQueue.isEmpty());
```

Output:

```
10 added to the queue

20 added to the queue

30 added to the queue

Queue: 10, 20, 30
```

```
Front of queue: 10
10 removed from the queue
20 removed from the queue
Queue size: 1
Queue: 30
Is queue empty? False
```

Rotate an array k Times:

Problem-Link:https://leetcode.com/problems/rotate-array/description/

Solution:

```
var rotate = function(nums, k) {
    k=k%nums.length;
    k=nums.length-k;
    var q=new MyQueue();
    for(var i=0;i<nums.length;i++){
        q.push(nums[i]);
    }
    while(k>0){
        var elementPopped=q.pop();
        q.push(elementPopped);
        k--;
    }
    var i=0;
    while(!q.isEmpty()){
        var elementPopped=q.pop();
        nums[i]=elementPopped;
        i++;
    }
};
```

Time Complexity:O(n*k)

Space Complexity:(n)