**DSA ASSIGNMENT-1**

**Q1. Write a program to implement singly linked list with all its operations using JavaScript.**

**Ans:-**

class Node{

constructor(element)

{

this.element=element;

this.next=null;

}

}

class LinkedList{

constructor()

{

this.head=null;

this.size=0;

}

AddNode(element)

{

var node = new Node(element);

var current;

if(this.head==null)

this.head = node;

else

{

current = this.head;

while(current.next)

{

current =current.next;

}

current.next=node;

}

this.size++;

}

insertAt(element, index)

{

if (index < 0 || index > this.size)

return console.log("Invalid Index.");

else

{

var node = new Node(element);

var curr, prev;

curr = this.head;

// add the element to the first index

if (index == 0)

{

node.next = this.head;

this.head = node;

}

else

{

curr = this.head;

var ab = 0;

while (ab< index)

{

ab++;

prev = curr;

curr = curr.next;

}

node.next = curr;

prev.next = node;

}

this.size++;

}

}

AddLast(element)

{

var new\_node = new Node(element);

if (this.head == null)

{

this.head = new Node(element);

return;

}

new\_node.next = null;

var last = this.head;

while (last.next != null)

{

last = last.next;

}

last.next = new\_node;

}

// Addding Nodes At Front

AddFront(element)

{

var new\_node = new Node(element);

new\_node.next = this.head;

this.head = new\_node;

}

// Delete Node At Given Position

DelNode(element)

{

if (this.head == null)

return;

var temp =this.head;

if (element == 0)

{

this.head = temp.next;

return;

}

var i;

for(i = 0; temp != null && i < element - 1; i++)

temp = temp.next;

if (temp == null || temp.next == null)

return;

var next = temp.next.next;

temp.next = next;

}

// Delete Node At Last Position

DelLastNode()

{

if (!this.head == null)

return null;

if (this.head.next == null) {

return null;

}

let previous = this.head;

let tail = this.head.next;

while (tail.next !== null){

previous=tail;

tail=tail.next;

}

previous.next = null;

return this.head;

}

// Deleting Nodes At First Position

DelFirstNode()

{

if (this.head == null)

return null;

this.head = this.head.next;

return this.head;

}

// Delete Complete Linked List

DeleteLinkList()

{

this.head = null;

}

//Print function

printList()

{

var curr = this.head;

while(curr)

{

console.log(curr.element);

curr = curr.next;

}

}

}

var l1=new LinkedList();

console.log("ABC");

//Addding Nodes

console.log("\n Addding Nodes :");

l1.AddNode(60);

l1.AddNode(70);

l1.AddNode(80);

l1.printList();

//Addding Nodes At End

console.log("\n Addding Nodes At End 545:");

l1.AddLast(545);

l1.printList();

//Addding Nodes At Front

console.log("\n Addding Nodes At Front 100:");

l1.AddFront(100);

l1.printList();

//Adding Node At Given Position

console.log("\n Addding Nodes At Given Position 3 and element is 50 :");

l1.insertAt(50,3);

l1.printList();

// Deleting Nodes At Position

console.log("\n Deleting Nodes At Position 2 :");

l1.DelNode(2);

l1.printList();

// Deleting Nodes At Last Position

console.log("\n Deleting Nodes At Last Position :");

l1.DelLastNode();

l1.printList();

// Deleting Nodes At First Position

console.log("\n Deleting Nodes At First Position :");

l1.DelFirstNode();

l1.printList();

// Delete Complete Linked List

console.log("\n Delete Complete Linked List :");

l1.DeleteLinkList();

l1.printList();

**Q2. Write a program to implement doubly singly linked list with all its operations using JavaScript.**

**Ans:-**

console.log("\n ABC");

console.log("\n Output :");

//Node

class Node{

constructor(elm, next = null, prev=null)

{

this.element = elm;

this.next = next;

this.prev = prev;

}

}

class doubleLinkedList

{

constructor()

{

this.length = 0;

this.head = null;

this.tail = null;

}

//Add new Element

append(element)

{

let node = new Node(element),current = this.head,previous;

if(!this.head)

{

this.head = node;

this.tail = node;

}

else

{

node.prev = this.tail;

this.tail.next = node;

this.tail = node;

}

this.length++;

}

//Add Element At Given Position

insert(position, element)

{

//Check of out-of-bound values

if(position >= 0 && position <= this.length)

{

let node = new Node(element),

current = this.head,

previous,

index = 0;

if(position === 0)

{

if(!this.head)

{

this.head = node;

this.tail = node;

}

else

{

node.next = current;

current.prev = node;

this.head = node;

}

}

else if(position === this.length)

{

current = this.tail;

current.next = node;

node.prev = current;

this.tail = node;

}

else

{

while(index++ < position)

{

previous = current;

current = current.next;

}

node.next = current;

previous.next = node;

//New

current.prev = node;

node.prev = previous;

}

this.length++;

return true;

}

else

{

return false;

}

}

//Addding node at last

AddLast(element)

{

var new\_node = new Node(element);

if (this.head == null)

{

this.head = new Node(element);

return;

}

new\_node.next = null;

var last = this.head;

while (last.next != null)

{

last = last.next;

}

last.next = new\_node;

}

// Addding Nodes At Front

AddFront(element)

{

var new\_node = new Node(element);

new\_node.next = this.head;

this.head = new\_node;

}

//Remove element at any position

removeAt(position)

{

//look for out-of-bounds value

if(position > -1 && position < this.length)

{

let current = this.head, previous, index = 0;

//Removing first item

if(position === 0)

{

this.head = current.next;

//if there is only one item, update tail //NEW

if(length === 1)

{

this.tail = null;

}

else

{

this.head.prev = null;

}

}

else if(position === this.length - 1)

{

current = this.tail;

this.tail = current.prev;

this.tail.next = null;

}

else

{

while(index++ < position)

{

previous = current;

current = current.next;

}

//link previous with current's next - skip it

previous.next = current.next;

current.next.prev = previous;

}

this.length--;

return current.element;

}

else

{

return null;

}

}

//Get the indexOf item

indexOf (elm){

let current = this.head,

index = -1;

//If element found then return its position

while(current){

if(elm === current.element){

return ++index;

}

index++;

current = current.next;

}

//Else return -1

return-1;

};

//Find the item in the list

isPresent (elm)

{

return this.indexOf(elm) !== -1;

};

//Delete an item from the list

delete (elm)

{

return this.removeAt(this.indexOf(elm));

};

//Delete first item from the list

deleteHead ()

{

this.removeAt(0);

}

//Delete last item from the list

deleteTail ()

{

this.removeAt(this.length-1);

}

//Check if list is empty

isEmpty ()

{

return this.length === 0;

};

//Get the size of the list

size ()

{

return this.length;

}

//Get the head

getHead ()

{

return this.head;

}

//Get the tail

getTail()

{

return this.tail;

}

printList()

{

var curr = this.head;

while(curr)

{

console.log(curr.element);

curr = curr.next;

}

}

}

var l1=new doubleLinkedList();

//Addding New Element

console.log("\n Addding New Element :");

l1.append(11);

l1.append(22);

l1.append(33);

l1.printList();

//Add Element At Given Position

console.log("\n Add Element(20) At Given Position 1 :");

l1.insert(1,20);

l1.printList();

//Add Element At Last

console.log("\n Add Element(40) At Last :");

l1.AddLast(40);

l1.printList();

//Add Element At First

console.log("\n Add Element(500) At First :");

l1.AddFront(500);

l1.printList();

//Remove Element

console.log("\n Remove Element(20) :");

l1.delete(20);

l1.printList();

//Remove First Element

console.log("\n Remove First Element :");

l1.deleteHead();

l1.printList();

//Remove Last Element

console.log("\n Remove Last Element :");

l1.deleteTail();

l1.printList();

//Remove Element At Given Position

console.log("\n Remove Element At Given Position 1 :");

l1.removeAt(1);

l1.printList();

**Q3.Write a program to implement stack using Array with PUSH, POP operations.**

**Ans:-**

class Node{

constructor(element)

{

this.element=element;

this.next=null;

}

}

class Stack{

constructor()

{

this.head=null;

this.size=0;

}

push(element)

{

var new\_node=new Node(element);

var current;

if(this.head==null)

this.head=new\_node;

else{

current=this.head;

new\_node.next=this.head;

this.head=new\_node;

}

this.size++;

}

pop()

{

if(this.head==null){

console.log("cant pop");

return false;

}

else{

var current=this.head;

var ele=current.element;

current=current.next;

this.head=current;

this.size--;

return ele;

}

}

peek()

{

if(this.head){

return this.head.element;

}

}

size\_of\_stack()

{

console.log("size of stack is" +this.size);

}

printStack()

{

var curr;

curr=this.head;

while(curr){

console.log(curr.element);

curr=curr.next;

}

}

}

var s1=new Stack();

s1.push(60);

s1.push(70);

s1.push(80);

s1.push(90);

s1.printStack();

console.log("after poping");

console.log("poping element is" +s1.pop(60));

s1.printStack();

console.log("top element of stack is" +s1.peek());

s1.size\_of\_stack();

**Q4. Write a Program to reverse a string using stack.**

**Ans:-**

class Stack {

constructor() {

this.top = null

}

push(ele) {

var node = new newNode(ele)

node.next = this.top

this.top = node

}

pop() {

var temp = this.top

var char = temp.data

this.top = this.top.next

temp = null

return char

}

reverseString(str) {

var i = 0

var reversestr = ""

while (i != str.length) {

this.push(str.charAt(i))

i++

}

var temp = this.top

while (temp != null) {

var char

char = this.pop()

reversestr += char

temp = this.top

}

return reversestr

}

display() {

var temp = this.top

while (temp != null) {

console.log(temp.data)

temp = temp.next

}

}

}

class newNode {

constructor(data, next) {

this.data = data

this.next = null

}

}

const stack = new Stack()

const string = "harshadip"

const reverse = stack.reverseString(string)

console.log(`\nThe String : ${string}\nString in reverse : ${reverse}`);

**5. Write a Program to check for balanced parentheses by using Stacks.**

**Ans:-**

console.log("\n ABC");

console.log("\n Output :");

function areBracketsBalanced(expr)

{

let stack = [];

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

{

let x = expr[i];

if (x == '(' || x == '[' || x == '{')

{

stack.push(x);

continue;

}

if (stack.length == 0)

return false;

let check;

switch (x)

{

case ')':

check = stack.pop();

if (check == '{' || check == '[')

return false;

break;

case '}':

check = stack.pop();

if (check == '(' || check == '[')

return false;

break;

case ']':

check = stack.pop();

if (check == '(' || check == '{')

return false;

break;

}

}

return (stack.length == 0);

}

let expr = "([{}])";

console.log("\n checking balanced brackets : ([{}]) ");

if (areBracketsBalanced(expr))

console.log("\n Brackets are Balanced ");

else

console.log("Not Balanced ");

**Q6. Write a Program to Implement Stack using Linked List.**

**Ans:-**

function stackUsingLL(){

let Node = function(elm){

this.element = elm;

this.next = null;

}

let length = 0;

let head = null;

this.push = function(elm){

let node = new Node(elm),

current;

current = head;

node.next = current;

head = node;

length++;

}

this.pop = function(){

let current = head;

if(current){

let elm = current.element;

current = current.next;

head = current;

length--;

return elm;

}

return null;

}

this.peek = function(){

if(head){

return head.element;

}

return null;

}

this.toArray = function(){

let arr = [];

let current = head;

while(current){

arr.push(current.element);

current = current.next;

}

return arr;

}

this.isEmpty = function(){

return length === 0;

}

this.size = function(){

return length;

}

this.clear = function(){

head = null;

length = 0;

}

}

let stack = new stackUsingLL();

console.log("ABC");

stack.push(1);

stack.push(2);

stack.push(3);

console.log("\n Peek the top item in the stack : \n" +stack.peek());

console.log("Check if stack is empty:\n " +stack.isEmpty());

console.log("the size of the stack: \n" +stack.size());

console.log("Pop the item from the stack :\n " +stack.pop());

console.log(stack.toArray());

console.log("the size of the stack: \n" +stack.size());

stack.clear();

console.log("Check if stack is empty: \n" +stack.isEmpty());

**7. Write a Program to Implement Linear Queue.**

**Ans:-**

class Node

{

constructor(element)

{

this.element=element;

this.next=null;

}

}

class Queue{

constructor()

{

this.front=null;

this.rear=null;

this.size=0;

}

enqueue(element)

{

var new\_node=new Node(element);

if(this.front==null)

{

this.front=new\_node;

this.rear=new\_node;

}

else{

var current=this.rear;

current.next=new\_node;

this.rear=new\_node;

}

this.size++;

}

dequeue()

{

if(this.front==null)

return false;

if(this.front==this.rear)

{

console.log(" element dequeued" +this.front.element)

this.front=null;

this.rear=null;

}

else{

var current=this.front;

var ele=current.element;

current=current.next;

this.front=current;

console.log("element dequeued" +ele);

}

this.size--;

}

printQueue()

{

var curr;

curr=this.front;

while(curr)

{

console.log(curr.element);

curr=curr.next;

}

}

}

console.log("ABC");

var Q1=new Queue();

Q1.enqueue(70);

Q1.enqueue(80);

Q1.enqueue(90);

Q1.printQueue();

console.log(Q1.front);

Q1.dequeue();

Q1.dequeue();

console.log("After Deque");

Q1.printQueue();

Q1.dequeue();

**Q8. Write a Program to Implement Circular Queue.**

**Ans:-**

const CircularQueue = function(k) {

this.size = k

this.queue = []

this.start1 = 0

this.end1 = 0

this.start2 = 0

this.end2 = 0

}

CircularQueue.prototype.enQueue = function(value) {

if(this.isFull()) {

return false

}

if(this.end2 <= this.size - 1) {

this.queue[this.end2++] = value

} else {

this.queue[this.end1++] = value

}

return true

}

CircularQueue.prototype.deQueue = function() {

if(this.isEmpty()) {

return false

}

if(this.queue[this.start2] !== undefined) {

this.queue[this.start2++] = undefined

} else {

this.queue[this.start1++] = undefined

}

return true

}

CircularQueue.prototype.Front = function() {

if(this.isEmpty()) {

return -1

}

return this.queue[this.start2] === undefined ? this.queue[this.start1] : this.queue[this.start2]

}

CircularQueue.prototype.Rear = function() {

if(this.isEmpty()) {

return -1

}

return this.queue[this.end1 - 1] === undefined ? this.queue[this.end2 - 1] : this.queue[this.end1 - 1]

}

CircularQueue.prototype.isEmpty = function() {

if(this.end2 - this.start2 + this.end1 - this.start1 <= 0) {

return true

}

return false

}

CircularQueue.prototype.isFull = function() {

if(this.end2 - this.start2 + this.end1 - this.start1 >= this.size) {

return true

}

return false

}

const queue = new CircularQueue(2);

console.log(queue.enQueue(1));

console.log(queue.enQueue(2));

console.log(queue.enQueue(3));

console.log(queue.Rear());

console.log(queue.isFull());

console.log(queue.deQueue());

console.log(queue.enQueue(3));

console.log(queue.Rear());

**Q9. Write a Program to Implement Priority Queue.**

**Ans:-**

class QElement {

constructor(element, priority)

{

this.element = element;

this.priority = priority;

}

}

class PriorityQueue {

constructor()

{

this.items = [];

}

enqueue(element, priority)

{

var qElement = new QElement(element, priority);

var contain = false;

for (var i = 0; i < this.items.length; i++) {

if (this.items[i].priority > qElement.priority) {

this.items.splice(i, 0, qElement);

contain = true;

break;

}

}

if (contain!=true) {

this.items.push(qElement);

}

}

dequeue()

{

if (this.isEmpty())

return "Underflow";

return this.items.shift();

}

front()

{

if (this.isEmpty())

return "No elements in Queue";

return this.items[0].element;

}

rear()

{

if (this.isEmpty())

return "No elements in Queue";

return this.items[this.items.length - 1].element;

}

isEmpty()

{

return this.items.length == 0;

}

printPQueue()

{

var str = "";

for (var i = 0; i < this.items.length; i++)

str += this.items[i].element + " ";

return str;

}

}

var priorityQueue = new PriorityQueue();

/\*console.log(priorityQueue.isEmpty()); // testing isEmpty and front on an empty queue

// return true

console.log(priorityQueue.front()); // returns "No elements in Queue" \*/

console.log("harshadip borge\n\n")

// adding elements to the queue

priorityQueue.enqueue("Abhiraj", 3);

priorityQueue.enqueue("ABC", 1);

priorityQueue.enqueue("snehal", 1);

priorityQueue.enqueue("yuvraj", 2);

priorityQueue.enqueue("ankita", 2);

// prints [ABC, snehal,abhiraj,yuvraj,ankita]

console.log(priorityQueue.printPQueue());

/\*// prints ABC

console.log(priorityQueue.front());

// pritns ankita

console.log(priorityQueue.rear());

// removes ABC

// priorityQueue contains

// [snehal, abhiraj yuvraj ankita]

console.log(priorityQueue.dequeue().element);

// Adding another element to the queue

priorityQueue.enqueue("Sunil", 2);

// prints [Piyush Sumit Sunny Suni]

console.log(priorityQueue.printPQueue()); \*/

**Q10. Write a Program to reverse stack using queue.**

**Ans:-**

console.log("\n ABC ");

console.log("\nOutput :");

function printQueue(queue)

{

while (queue.length != 0)

{

console.log(queue[0] + " ");

queue.shift();

}

}

function reverseQueue(q)

{

if (q.length == 0)

return;

let data = q[0];

q.shift();

reverseQueue(q);

q.push(data);

}

let queue = [];

console.log("Stack Elements are : ");

queue.push(52);

queue.push(72);

queue.push(15);

queue.push(54);

queue.push(85);

queue.push(29);

queue.push(85);

queue.push(40);

queue.push(35);

queue.push(50);

console.log(queue);

console.log("Reverse Queue is : ");

reverseQueue(queue);

printQueue(queue);

**ASSIGNMENT-2**

**Q11.Write a program to create a binary search tree, insert, and delete an element from binary search tree.**

**Ans:-**

class Node{

constructor(data)

{

this.data=data;

this.left=null;

this.right=null;

}

}

class Bstree

{

constructor()

{

this.root=null;

}

insert(data)

{

var newnode=new Node(data);

if(this.root==null)

{

this.root=newnode;

}

else

this.insertnode(this.root,newnode);

}

insertnode(node,newnode)

{

if(newnode.data<node.data){

if(node.left==null)

node.left=newnode;

else

this.insertnode(node.left,newnode);

}

else{

if(node.right==null)

node.right=newnode;

else

this.insertnode(node.right,newnode);

}

}

getrootnode(){

return this.root;

}

inorder(node)

{

if(node!=null)

{

this.inorder(node.left);

console.log(node.data);

this.inorder(node.right);

}

}

remove(data){

this.root=this.removeNode(this.root,data);

}

removeNode(node,key){

if(node==null)

return null;

else if(key < node.data)

{

node.left = this.removeNode(node.left, key);

return node;

}

else if(key > node.data)

{

node.right = this.removeNode(node.right, key);

return node;

}

else

{

if(node.left === null && node.right === null)

{

node = null;

return node;

}

if(node.left === null)

{

node = node.right;

return node;

}

else if(node.right === null)

{

node = node.left;

return node;

}

var aux = this.findMinNode(node.right);

node.data = aux.data;

node.right = this.removeNode(node.right, aux.data);

return node;

}

}

}

var b=new Bstree();

b.insert(15);

b.insert(25);

b.insert(10);

b.insert(7);

b.insert(22);

b.insert(5);

console.log(b.getrootnode().data);

var root=b.getrootnode();

console.log("inorder traversal");

b.inorder(root);

b.remove(5);

console.log("inorder traversal after removing");

b.inorder(root);

**Q12. Write a program to implement binary search tree traversal operations- In order, Preorder and Post order.(use create BST function of program 1)**

**Ans:-**

class Node{

constructor(data)

{

this.data=data;

this.left=null;

this.right=null;

}

}

class Bstree

{

constructor()

{

this.root=null;

}

insert(data)

{

var newnode=new Node(data);

if(this.root==null)

{

this.root=newnode;

}

else

this.insertnode(this.root,newnode);

}

insertnode(node,newnode)

{

if(newnode.data<node.data){

if(node.left==null)

node.left=newnode;

else

this.insertnode(node.left,newnode);

}

else{

if(node.right==null)

node.right=newnode;

else

this.insertnode(node.right,newnode);

}

}

getrootnode(){

return this.root;

}

preorder(node){

if(node!=null){

console.log(node.data);

this.preorder(node.left);

this.preorder(node.right);

}

}

inorder(node){

if(node!=null){

this.inorder(node.left);

console.log(node.data);

this.inorder(node.right);

}

}

postorder(node){

if(node!=null){

this.postorder(node.left);

this.postorder(node.right);

console.log(node.data);

}

}

}

var b=new Bstree();

b.insert(15);

b.insert(25);

b.insert(10);

b.insert(7);

b.insert(2);

console.log(b.getrootnode().data);

console.log("preorder traversal");

var root=b.getrootnode();

b.preorder(root);

console.log("inorder traversal");

b.inorder(root);

console.log("postorder traversal");

b.postorder(root);

**Q13. Write a program to implement Graph using adjacency list and graph traversals-DFS and BFS.**

**Ans:-**

class Graph {

constructor(noOfVertices)

{

this.noOfVertices = noOfVertices;

this.AdjList = new Map();

}

addVertex(v)

{

this.AdjList.set(v, []);

}

addEdge(v, w)

{

this.AdjList.get(v).push(w);

this.AdjList.get(w).push(v);

}

printGraph()

{

var get\_keys = this.AdjList.keys();

for (var i of get\_keys)

{

var get\_values = this.AdjList.get(i);

var conc = "";

for (var j of get\_values)

conc += j + " ";

console.log(i + " -> " + conc);

}

}

dfs(v){

let s=[];

let visited =[];

let keys=this.AdjList.keys();

for(let i of keys){

visited[i]=false;

}

s.push(v);

while(s.length>0){

let element =s.pop();

if(!visited[element]){

console.log(element);

visited[element]=true;

}

else

continue;

let elist=this.AdjList.get(element);

for(var i=elist.length-1;i>=0;i--){

let e=elist[i];

if(!visited[e]){

s.push(e);

}

}

}

}

bfs(v){

let q=[];

let visited=[]

let keys=this.AdjList.keys();

for(let i of keys){

visited[i]=false;

}

q.push(v);

while(q.length>0){

let element=q.shift();

visited[element]=true;

console.log(element);

let elist=this.AdjList.get(element);

for(let i in elist){

let e=elist[i];

if(!visited[e]){

q.push(e);

visited[e]=true;

}

}

}

}

}

var g = new Graph(6);

var vertices = [ 'A', 'B', 'C', 'D', 'E', 'F' ];

for (var i = 0; i < vertices.length; i++) {

g.addVertex(vertices[i]);

}

g.addEdge('A', 'B');

g.addEdge('A', 'D');

g.addEdge('A', 'E');

g.addEdge('B', 'C');

g.addEdge('D', 'E');

g.addEdge('E', 'F');

g.addEdge('E', 'C');

g.addEdge('C', 'F');

g.printGraph();

console.log("DFS");

g.dfs('A');

console.log("BFS");

g.bfs('A');

**Q14. Write a program to implement Hash table using array by simple hashing function.**

**Ans:-**

class HashTable {

constructor() {

this.values = {};

this.length = 0;

this.size = 0;

}

//hashing Funtion

calculateHash(key) {

return key.toString().length % this.size;

}

//adding key and value pair in hash table

add(key, value) {

const hash = this.calculateHash(key);

if(!this.values.hasOwnProperty(hash)) {

this.values[hash] = {};

}

if (!this.values[hash].hasOwnProperty(key)) {

this.length++;

}

this.values[hash][key] = value;

}

search(key) {

const hash = this.calculateHash(key);

if (this.values.hasOwnProperty(hash) && this.values[hash].hasOwnProperty(key)) {

return this.values[hash][key];

} else {

return null;

}

}

}

//create object of type hash table

const ht = new HashTable();

//add data to the hash table ht

ht.add("Pune", "1");

ht.add("Mumbai", "10");

ht.add("Shimla", "20");

//search

console.log();

console.log("Value of Shimla is: "+ht.search("Shimla"));

**Q15. Write a program to implement Hash table using bucket chaining method.**

**Ans:-**

class HashTable {

constructor(size = 42) {

this.buckets = new Array(size)

this.size = size

}

hash(key) {

return key.toString().length % this.size;

}

set(key, value) {

let index = this.hash(key);

if(!this.buckets[index]){

this.buckets[index] = [];

}

this.buckets[index].push([key,value])

return index

}

get(key) {

let index = this.hash(key);

if(!this.buckets[index])

return null

for(let bucket of this.buckets[index]) {

if(bucket [0] === key) {

return bucket [1]

}

}

}

}

const has = new HashTable(10);

has.set("userId3","pop")

has.set("userId4","king")

console.log(has.get("userId3"))

console.log(has.get("userId4"))

**Q16. Write a program to implement Linear Search algorithm(Brute Force approach)**

**Ans:-**

function linearsearch(arr,key){

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

if(arr[i]==key){

return i;

}

}

return -1;

}

var a=[80,45,70,40,35,50];

i=linearsearch(a,35)

console.log("linear search using brute force approach");

if(i==-1)

console.log("element is not in the list");

else

console.log("element is present at position",i);

**Q17. Write a program to implement Binary Search algorithm(Divide and conquer approach)**

**Ans:-**

function binaryser(arr,x,l,r)

{

while(l<=r)

{

let mid=Math.floor((l+r)/2);

if(arr[mid]===x)

return mid;

else if(arr[mid]<x)

l=mid+1;

else

r=mid-1;

}

return false;

}

let arr=[1,3,5,7,8,9];

let x=5;

console.log("binary search using divide and conquer");

var index=binaryser(arr,x,0,arr.length-1);

if(index)

console.log("element",x,"found at index",index);

else

console.log("element",x,"not found");

**Q18. Write a program to solve trapping rain water problem using brute force approach.**

**Ans:-**

function maxwater(arr,n){

var res=0;

for(var i=1;i<n-1;i++){

var left=arr[i];

for(var j=0;j<i;j++){

left=Math.max(left,arr[j])

}

var right=arr[i];

for(j=i+1;j<n;j++){

right=Math.max(right,arr[j])

}

res+=Math.min(left,right)-arr[i]

}

return res

}

var arr=[0,1,0,2,1,0,1,3,2,1,2,1];

var n=arr.length

console.log("trapping rain water problem using brute force approach");

console.log("answer:",maxwater(arr,n))

**Q19. Write a program to solve recursive staircases problem using brute force approach.**

**Ans:-**

function countWays (n)

{

if(n<0)

return 0;

if(n==0)

return 1;

return(countWays(n-1) + countWays(n-2));

}

let s=8;

console.log("Number of ways" + countWays(s));

**Q20. Write a program to find maximum sum of a subarray among all subarrays of given array using brute force approach**

**Ans:-**

function max\_Subarray\_sum(A,n)

{

let max\_sum= A[0];

let sum\_so\_far=0

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

{

sum\_so\_far = sum\_so\_far + a[i];

if(max\_sum < sum\_so\_far)

max\_sum = sum\_so\_far;

if (sum\_so\_far <0)

sum\_so\_far =0;

}

return max\_sum;

}

let a=[-5,8,9,-6,10,-15,3];

console.log("max Sum of =" + max\_Subarray\_sum(a,a.length));

**ASSIGNMENT-3**

**Q21. Write a program to solve jump game problem using greedy method.**

**Ans:-**

function jump(nums)

{

var n = nums.length; //n=5

if (n < 2)

return 0;

var maxReachable = nums[0];

var maxSteps = nums[0];

var jumps = 1;

for (let i = 1; i < n; ++i)

{

if (maxSteps < i)

{

++jumps;

maxSteps = maxReachable;

}

maxReachable = Math.max(maxReachable, nums[i] + i);

}

return jumps;

}

var a=[1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9];

console.log(jump(a));

**Q22. Write a program to implement Prim’s algorithm to find minimum spanning tree using greedy method.**

**Ans:-**

matrix = [[0,4,0,2,3], [4,0,6,0,5], [0,6,0,0,2], [2,0,0,0,1], [3,5,2,1,0]]

var total = 0, u, v, min;

var visited = new Array(5)

visited[0] = 1

for(let c = 0; c < 4; c++) {

min = 999

for(let i = 0; i < 5; i++) {

if(visited[i] == 1){

for(let j = 0; j < 5; j++){

if(i == j || matrix[i][j] == 0)

continue;

if(visited[j] != 1) {

if(min > matrix[i][j]) {

min = matrix[i][j]

u = i

v = j

}

}

}

}

}

total += min

visited[v] = 1

console.log("Edge found " ,u, " -> " + v + " : weight : " + min)

}

**Q23. Write a program to solve Pascal’s triangle problem using divide and conquer method.**

**Ans:-**

function pascals(numRows) {

if (numRows === 0) return [];

if (numRows === 1) return [[1]];

let result = [];

for (let row = 1; row <= numRows; row++) {

let arr = [];

for (let col = 0; col < row; col++) {

if (col === 0 || col === row - 1) {

arr.push(1);

} else {

arr.push((result[row-2][col-1] + result[row-2][col]))

}

}

result.push(arr);

}

return result;

}

console.log(pascals(5))

**Q24. Write a program to implement Euclidean Algorithm using divide and conquer method.**

**Ans:-**

function gcd( a, b)

{

if (a == 0)

return b;

return gcd(b % a, a);

}

// Driver Code

let a = 10, b = 15;

console.log( "GCD(" + a + ", "

+ b + ") = " + gcd(a, b) +"<br/>");

a = 35, b = 10;

console.log( "GCD(" + a + ", "

+ b + ") = " + gcd(a, b) +"<br/>");

a = 31, b = 2;

console.log( "GCD(" + a + ", "

+ b + ") = " + gcd(a, b) +"<br/>");

**Q25. Write a program to sort an array using merge sort. (divide and conquer method)**

**Ans:-**

function a(arr) {

if(arr.length < 2) return arr

const half = arr.length/2

let left = arr.splice(0,half)

return merge(a(left), a(arr))

}

function merge(left, right) {

let ar = []

while(left.length && right.length) {

if(left[0] < right[0]) {

ar.push(left.shift())

}

else {

ar.push(right.shift())

}

}

return [...ar, ...left, ...right]

}

console.log(a([41, 18, 9, 7, 2, 11, 1, 3]))

**Q26. Write a program to solve Tower of Hanoi problem using divide and conquer method.**

**Ans:-**

function Hanoi(n, source, dest , aux) {

if (n == 0) return;

Hanoi(n - 1, source, aux , dest);

console.log("Move Disk " + n ,"FROM " + source, "TO " + dest);

Hanoi(n - 1, aux, dest , source);

}

Hanoi(3, 'source', 'dest', 'aux')

**Q27. Write a program to find Fibonacci number using Dynamic programming.**

**Ans:-**

function fib(n) {

let f = new Array(n);

let i;

f[0] = 0;

f[1] = 1;

for (i = 2; i <= n; i++) {

f[i] = f[i-1] + f[i-2];

}

return f

}

console.log(fib(5))

**Q28. Write a program to find unique path using Dynamic programming.**

**Ans:-**

function uniquepath(m,n)

{

const dp= new Array(m);

for (var i=0;i<m;i++)

dp[i]= new Array(n).fill(1);

//console.log(dp);

for(i=1;i<m;i++){

for(j=1;j<n;j++)

{

dp[i][j]= dp[i-1][j] + dp[i][j-1];

}

}

return dp[m-1][n-1];

}

//driver code

console.log("unique pathe for given matrix is "+uniquepath(3,4));

**Q29. Write a program to find Longest Common Substring from given strings using Dynamic programming.**

**Ans:-**

function LCS\_dynamic(s1,s2,m,n)

{

const LCS\_table=new Array(m);

for(let i=0;i<=m;i++)

LCS\_table[i]=new Array(n);

for(var i=0;i<=m;i++)

{

for(j=0;j<=n;j++)

{

if(i==0||j==0)

LCS\_table[i][j]=0;

else if(s1.charAt(i-1)==s2.charAt(j-1))

LCS\_table[i][j] = LCS\_table[i-1][j-1]+1;

else

LCS\_table[i][j]=Math.max(LCS\_table[i-1][j],LCS\_table[i][j-1])

}

}

console.log(LCS\_table);

var index= LCS\_table[m][n];

var temp=index; //3 //[.H .L .o .'/0']

var result=new Array(index+1);

result[index]='\0';

var i=m, j=n;

while(i>0 && j>0){

if(s1.charAt(i-1)===s2.charAt(j-1)){

result[index-1]=s1.charAt(i-1);

i--;

j--;

index--;

}

else if(LCS\_table[i-1][j]>LCS\_table[i][j-1])

i--;

else

j--;

}

console.log("Longest common Subsequence is :")

for (var k=0;k<=temp;k++)

console.log(result[k]);

}

//driver code

s1="HELO";

s2="HLO";

m=s1.length;

n=s2.length;

console.log("s1 :" +s1, "\ns2 :" +s2);

LCS\_dynamic(s1,s2,m,n);

**Q30. Write a program to implement 0/1 knapsack problem using Dynamic programming.**

**Ans:-**

function knapsack(v, w, n, M) {

const V\_table = new Array(n);

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

V\_table[i] = new Array(M);

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

for (var j = 0; j <= M; j++) {

if (i == 0 || j == 0)

V\_table[i][j] = 0;

else if (w[i-1] > j)

V\_table[i][j] = V\_table[i-1][j];

else

V\_table[i][j] = Math.max(V\_table[i - 1][j], V\_table[i - 1][j - w[i - 1]] + v[i - 1]);

}

}

console.log(V\_table);

i = n, j = M;

while (i > 0 && j > 0){

if(V\_table[i][j] == V\_table[i - 1][j]) {

i--;

}

else {

console.log("item selected " + i);

i--;

j = j - w[i];

}

}

}

var n=3;

var v=[10,15,40] ;//[1,2,5,6];;

var w=[1, 2, 3];//[2,3,4,5];

var M=6;

console.log(knapsack(v,w,n,M))

**Q31. Write a program to solve Integer Partition problem using Dynamic Programming.**

**Ans:-**

function isSubsetSum(arr,n,sum)

{

// Base Cases

if (sum == 0)

return true;

if (n == 0 && sum != 0)

return false;

// If last element is greater than sum, then ignore

// it

if (arr[n - 1] > sum)

return isSubsetSum(arr, n - 1, sum);

/\* else, check if sum can be obtained by any of

the following

(a) including the last element

(b) excluding the last element

\*/

return isSubsetSum(arr, n - 1, sum)

|| isSubsetSum(arr, n - 1, sum - arr[n - 1]);

}

// Returns true if arr[] can be partitioned in two

// subsets of equal sum, otherwise false

function findPartition(arr,n)

{

// Calculate sum of the elements in array

let sum = 0;

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

sum += arr[i];

// If sum is odd, there cannot be two subsets

// with equal sum

if (sum % 2 != 0)

return false;

// Find if there is subset with sum equal to half

// of total sum

return isSubsetSum(arr, n, Math.floor(sum / 2));

}

// Driver code

let arr=[3, 1, 5, 9, 12 ];

let n = arr.length;

// Function call

if (findPartition(arr, n) == true)

console.log("Can be divided into two "

+ "subsets of equal sum");

else

console.log("Can not be divided into "

+ "two subsets of equal sum");

**Q32. Write a program to implement N-Queens Problem using Backtracking method.**

**Ans:-**

function isSafe(board,row,col){

var i,j;

// Check this row on left side

for (j = 0; j < col; j++){

if (board[row][j])

return false;

}

// Check upper diagonal on left side

for (i = row, j = col; i >= 0 && j >= 0; i--, j--){

if (board[i][j])

return false;

}

// Check lower diagonal on left side

for (i = row, j = col; j >= 0 && i < N; i++, j--){

if (board[i][j])

return false;

}

return true;

}

/\* A recursive utility function to solve N

Queen problem \*/

function solveNQUtil(board, col)

{

if (col >= N) //base case: If all queens are placed then return true

return true;

/\* Consider this column and try placing

this queen in all rows one by one \*/

for (var i = 0; i < N; i++) {

if (isSafe(board, i, col)){ /\*Check if the queen can be placed on board[i][col]\*/

board[i][col] = 1; /\* Place this queen in board[i][col] \*/

if (solveNQUtil(board, col + 1)) /\* recur to place rest of the queens \*/

return true;

board[i][col] = 0; // BACKTRACK

}

}

/\* If the queen cannot be placed in any row in

this colum col then return false \*/

return false;

}

function solveNQ(N)

{

const board=new Array(N).fill(0).map(\_=>Array(N).fill(0));

if (solveNQUtil(board, 0) == false) {

console.log("Solution does not exist");

return false;

}

console.log(board);

return true;

}

// driver program to test above function

const N=4;

solveNQ(N);