ATSS's
Institute of Industrial and Computer Management and Research, Nigdi Pune
MCA Department
Academic Year : 2022-23

Practical Journal

on

IT11L- Data Structure and Algorithms (SEM-I)

Submitted By:

Student Name : Yash Omkar Yeole

Seat No. :

Roll no : 59

Date :

Course Outcomes:

Student will be able to :

CO1: Demonstrate Collection framework (Apply)

CO2: Develop GUI using awt and swing (Apply)

CO3: Develop Web application using JSP and Servlet, JDBC (Apply)

CO4: Apply Data Structure to solve problems using JavaScript (Apply)

CO5: Demonstrate the concepts of Core Java (Apply)

ATSS's

Institute of Industrial and Computer Management and Research, Nigdi Pune

MCA Department

INDEX

Students Name : Yash Omkar Yeole

Roll No. : 59

Sr. No	Program Title	Course Outcome	Page No.	Teacher's Sign with Date	Remarks
1.	Write a program to implement Singly linked list with required member function(Create, insert, delete, Display)				
2.	Write a program to implement Doubly linked list with required member function(Create, insert, delete, Display)				
3.	Write a program to implement STACK using Array with PUSH, POP operations				
4.	Write a program to implement Stack using Linked List				
5.	Write a application of stack to Check for balanced parentheses.				
6.	Write a program to Reverse a string using stack	n			
7.	Write a program to implement Linear Queue,				
8.	Write a program to Reverse stack using queue				
9.	Write a program to implement binary search tree with its operations				
10.	Write a program to implement Circular Queue,				
11.	Write a Program to print Adjacancy Matrix and Adjacancy List by reading Edges of Graph	n			
12.	Write a Program to find the element in an array using Binary Search	n			
13.	Write a Program to find the element in an array using Linear Search	n			

14.	**Write a Program to implement the following** **1.Print Pascal's triangle for n=5**	**n**			
15.	**Write a Program to implement the following** **GCD of two numbers using Euclidean Algorithm**	**n**			
16.	**Write a program to implement** **1. tower of Hanoi where number of disks=4**	**n**			
17.	**Write a program to implement** **2. Fibonacci series till N**	**n**			

Q.1 Write a program to implement

Singly linked list with required member function(Create, insert, delete, Display)

Solution: Program

```
class Node
{
	constructor(value)
	{
		this.data=value;
		this.next=null;
	}
}
class SLL {
	constructor() {
		this.head=null;
		this.count=0;
	}
	insertAtBegin(value) {
		var temp=new Node(value);if(this.head==null) {
			this.head=temp;
			this.count++;
		}
		else {
			temp.next=this.head;
			this.head=temp;
			this.count++;
		}
```

```
}
insertAtLast(value) {
        var temp=new Node(value);
        if(this.head==null) {
                this.head=temp;
                this.count++;
        }
        else {
                var cur=this.head;
                while(cur.next!=null) {
                        cur=cur.next;
                }
                cur.next=temp;
                this.count++;
        }
}
insertAtPos(value,pos) {
        var temp=new Node(value);
        if(pos<0 || pos>=this.count+2) {
                console.log("invalid position");
        }
        else {
                if(pos==1)
                        this.insertAtBegin(value);
                else if(pos==this.count+1)
                        this.insertAtLast(value);
                else {
                        var cur=this.head;
```

```
                var index=1;
                while(index<pos-1) {
                        cur=cur.next;
                        index++;
                }
                temp.next=cur.next;
                cur.next=temp;
                this.count++;
        }
    }
}
deleteAtFirst() {
    var temp;
    if(this.head==null)
        console.log("List is empty");
    else {
        if(this.head.next==null) {
            temp=this.head;
            this.head=null;
            this.count--;
        }
        else {
            temp=this.head;
            this.head=this.head.next;
            this.count--;
        }
    }
}
```

```
deleteAtLast() {
        var temp;
        if(this.head==null)
                console.log("List is empty");
        else if(this.head.next==null) {
                temp=this.head;
                this.head=null;
                this.count--;
        }
        else {
                var trv=this.head;
                var index=1;
                while(index!=this.count-1) {
                        trv=trv.next;
                        index++;
                }
                trv.next=temp;
                trv.next=null;
                this.count--;
        }
}
deleteAtPos(pos) {
        var temp;
        if(pos<=0 || pos>this.count) {
                console.log("position is invalid");
        }
        else if(pos==1)
                this.deleteAtFirst();
```

```
            else if(pos==this.count)
                this.deleteAtLast();
            else {
                var trv=this.head;
                var index=1;
                while(index<pos-1) {
                    trv=trv.next;
                    index++;
                }
                temp=trv.next;
                trv.next=temp.next;
                this.count--;
            }
        }
        display() {
            var cur=this.head;
            if(this.head==null)
                console.log("List is empty");
            else {
                while(cur!=null) {
                    console.log(cur.data+" ");
                    cur=cur.next;
                }
            }
        }
}
var l1=new SLL();
l1.insertAtBegin(25);
```

l1.insertAtBegin(20);

l1.insertAtBegin(15);

l1.insertAtLast(30);

l1.insertAtLast(40);

l1.insertAtPos(11,3);

l1.deleteAtFirst();

l1.deleteAtLast();

l1.deleteAtPos(4);

l1.display();

Output:

Screen Shot

……

Q.2 Write a program to implement
Doubly linked list with required member function(Create, insert, delete, Display)

Solution: Program

```
class node{
   constructor(val){
      this.data=val;
      this.next=null;
      this.prev=null;
   }
}
class DoubleLinkList{
   constructor(){
      this.head=null;
      this.count=0;
   }
   insertAtFirst(val){
      var temp=new node(val);
      if(this.head == null){
         this.head=temp;
         this.count++;
      }
      else{
         temp.next=this.head;
         this.head=temp;
         temp.prev=null;
         this.count++;
      }
   }
```

```
insertAtLast(val){
   var temp=new node(val);
   if(this.head == null){
      this.head = temp;
      this.count++;
   }
   else{
      var trv=this.head;
      while(trv.next != null){
         trv=trv.next;
      }
      trv.next=temp;
      this.count++;
   }
}
insertAtBetween(val,pos){
   var temp=new node(val);
   if(pos>this.count){
      console.log("index out of bound");
   }
   else if(pos==1){
      this.insertAtFirst(val);
   }
   else if(pos == this.count){
      this.insertAtLast(val);
   }
   else{
      var trv=this.head;
```

```
        var flag=1;
        while(flag != pos-1){
            flag++;
            trv=trv.next;
        }
        temp.next=trv.next;
        temp.prev=trv;
        trv.next=temp;
        this.count++;
    }
}
deleteAtFirst(){
    if(this.head==null){
        console.log("List id empty");
    }
    else if(this.head.next==null){
        this.head=null;
        this.count--;
    }
    else{
        this.head=this.head.next;
        this.count--;
    }
}
deleteAtLast(){
    if(this.head==null){
        console.log("List id empty");
    }
```

```
    else if(this.head.next==null){
        this.head=null;
        this.count--;
    }
    else{
        var trv=this.head;
        var current;
        while(trv.next != null){
            current=trv
            trv=trv.next;
        }
        current.next=null;
        this.count--;
    }
}
deleteAtBetween(pos){
    if(pos == this.count){
        this.deleteAtLast();
    }
    else if(pos == 1){
        this.deleteAtFirst();
    }
    else{
        var trv=this.head;
        var flag=1;
        while(flag != pos){
            trv=trv.next;
            flag++;
```

```
        }
        var temp=trv.next;
        trv.next=temp.next;
        temp=null;
        this.count--;
      }
    }
    display(){
      console.log("Double Link List");
      var trv=this.head;
      if(trv==null){
        console.log("List is empty");
      }
      else{
        while(trv != null){
          console.log(trv.data+" ");
          trv=trv.next;
        }
      }
    }
}
var d=new DoubleLinkList();
d.insertAtFirst(100);
d.insertAtFirst(200);
d.insertAtFirst(300);
d.insertAtLast(1000);
d.insertAtLast(2000);
d.insertAtLast(3000);
```

d.insertAtBetween(10,1);

d.insertAtBetween(20,4);

d.deleteAtFirst();

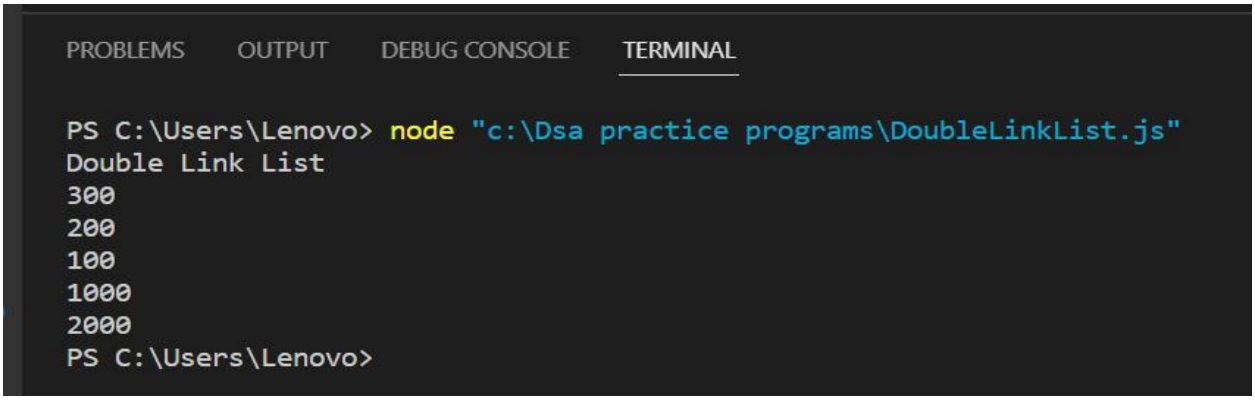
d.deleteAtLast();

d.deleteAtBetween(2);

d.display();

Output:

Screen Shot



……

Q.3 Write a program to implement STACK using Array with PUSH, POP operations

Solution: Program

```
class stack{
   constructor(sz){
      this.data=new Array();
      this.top=-1;
      this.size=sz;
   }
   push(val){
      if(this.top==this.size-1){
         console.log("Stack is full");
      }
      else{
         this.top++;
         this.data[this.top]=val;
      }
   }
   pop(){
      if(this.top == 1){
         console.log("Stack id empty");
      }
      else{
         let temp=this.data[this.top];
         this.top--;
         return temp;
      }
   }
   display(){
```

```
      for(let i=this.top; i>=0; i--){
         console.log(this.data[i]+"--->")
      }
   }
}
var  s=new stack(5);
s.push(10);
s.push(20);
s.push(30);
s.push(40);
s.push(50);
s.pop();
s.pop();
s.display();
```

Output:

Screen Shot

……

Q.4 Write a program to implement Stack using Linked List

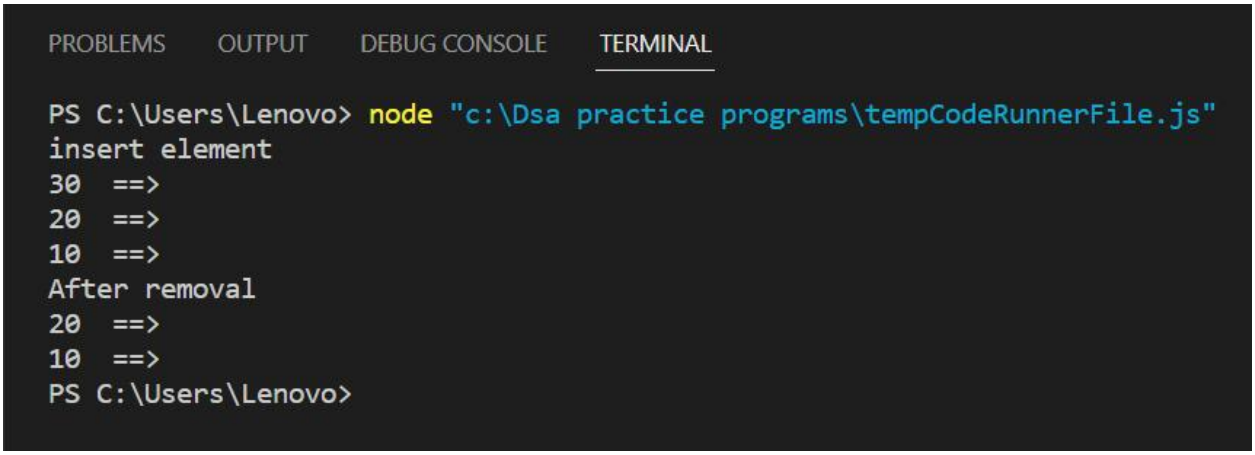
Solution: Program

```
class node{
   constructor(){
      this.data=0;
      this.next=null;
   }
}
class Stack{
   constructor(){
      this.top=null;
   }
   push(value){
      var temp=new node();
      if(temp==null){
         console.log("Stack is full");
         return;
      }
         temp.data=value;
         temp.next=this.top;
         this.top=temp;
   }
   pop(){
      if(this.top==null){
         console.log("Stack is empty pop");
         return;
      }
      this.top=this.top.next;
   }
```

```
  display(){
    if(this.top == null){
      console.log("Stack id empty display");
    }
    else{
      let temp=this.top;
      while(temp != null){
        console.log(temp.data+"  ==>");
        temp=temp.next;
      }
    }}}
var ww=new Stack();
console.log("insert element");
ww.push(10);
ww.push(20);
ww.push(30);
ww.display();
console.log("After removal");
ww.pop();
ww.display();
```

Output:

ScreenShot



Q.5 Write a application of stack to Check for balanced parentheses.

Solution: Program

```
class Stack {
   constructor(s) {
      this.data = new Array();
      this.top = -1;
      this.size = s;
   }
   push(val) {
      if (console.log('Stack is full');
      }
      else {
         this.top++;
         this.data.unshift(val);
      }
   }
   pop() {
      if (this.top == -1) {
         console.log('Stack is empty');
      }
      else {
         let val = this.data.shift();
         this.top--;
      }
   }
   peek(){
      return this.data[this.top];
   }
}
```

```
function isBalanced(str) {
   let s = new Stack(6);
   for (let i = 0; i < str.length; i++) {
      let letter = str.charAt(i);
      if (letter === '(' || letter === '[' || letter === '{') {
         s.push(letter);
      }
      else if (letter === ')' || letter == ']' || letter === '}') {
         if (s.peek() === '(') {
         s.pop();
         }
         else if (s.peek() === '[') {
            s.pop();
         }
         else if (s.peek() === '{') {
            s.pop();
         }
         else return 'Not Balanced expression ';
      }
   }
   if (s.top === -1) {
      return 'Balanced expression ';
   }
   else {
      return 'Not Balanced expression ';
   }
}
var exp1 = '{[a+b](b-a)}';
```

console.log(exp1);

console.log(isBalanced(exp1));

var exp2 = '{(a+b)[}';

console.log(exp2);

console.log(isBalanced(exp2));

Output:

Screen Shot

……

Q.6 Write a program to Reverse a string using stack

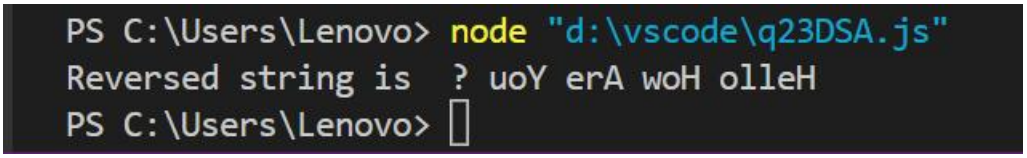
Solution: Program

```
class Stack{
   size;
   top;
   a = [];
   isEmpty(){
      return(this.top < 0);
   }
   constructor(n){
      this.top = -1;
      this.size = n;
      this.a = new Array(this.size);
   }
   push(x){                          // Function to push element in Stack
      if (this.top >= this.size){
         console.log("Stack Overflow<br>");
         return false;
      }
      else {
         this.a[++this.top] = x;
         return true;
      }
   }  // Function to pop element from stack
      if (this.top < 0){
         console.log("Stack Underflow<br>");
         return 0;
      }
      else{
```

```
        let x = this.a[this.top--];
        return x;
      }
   }
}
function reverse(str){                          // Function to reverse the string
   // Create a stack of capacity equal to length of string
   let n = str.length;
   let obj = new Stack(n);
   // Push all characters of string to stack
   let i;
   for(i = 0; i < n; i++)
      obj.push(str[i]);
   // Pop all characters of string  and put them back to str
     for(i = 0; i < n; i++){
      let ch = obj.pop();
      str[i] = ch;
   }
}
let s = "Hello How Are You ? ".split("");
reverse(s);
console.log("Reversed string is " +  s.join(""));
```

Output:

Screen Shot



Q.7 Write a program to implement Linear Queue

Solution: Program

```
class LinearQueue{
   constructor(sz){
      this.data=new Array();
      this.front=-1;
      this.rear=-1;
      this.size=sz;
   }
   enqueue(value){
      if(this.rear==this.size-1){              //for queue is full
         console.log("queue is full");
      }
      else if(this.rear==-1){                  //for 1st adding node
         this.front++;
         this.rear++;
         this.data[this.rear]=value;
      }
      else{                                    // for addding in queue
         this.rear++;
         this.data[this.rear]=value;
      }
   }
   dequeue() {
      if(this.rear==-1){                       //for no node
         console.log("Queue is empty");
      }
      else if(this.rear == this.front){        //for only one node
```

```
        var temp=this.data[this.front];
        this.front=-1;
        this.rear=-1;
        return temp;
      }
      else{
        var temp=this.data[this.front];       //for delete in queue
        this.front++;
        return temp;
      }
    }
    display(){
      if(this.rear == -1)                 //if(this.front==-1)
      {
        console.log("List is empty");
      }
      else{
        for(let i=this.front; i<=this.rear; i++){
          console.log(this.data[i]);
        }
      }
    }
}
var q=new LinearQueue(5);
console.log("Before queue");
q.enqueue(10);
q.enqueue(20);
q.enqueue(30);
```

q.display();

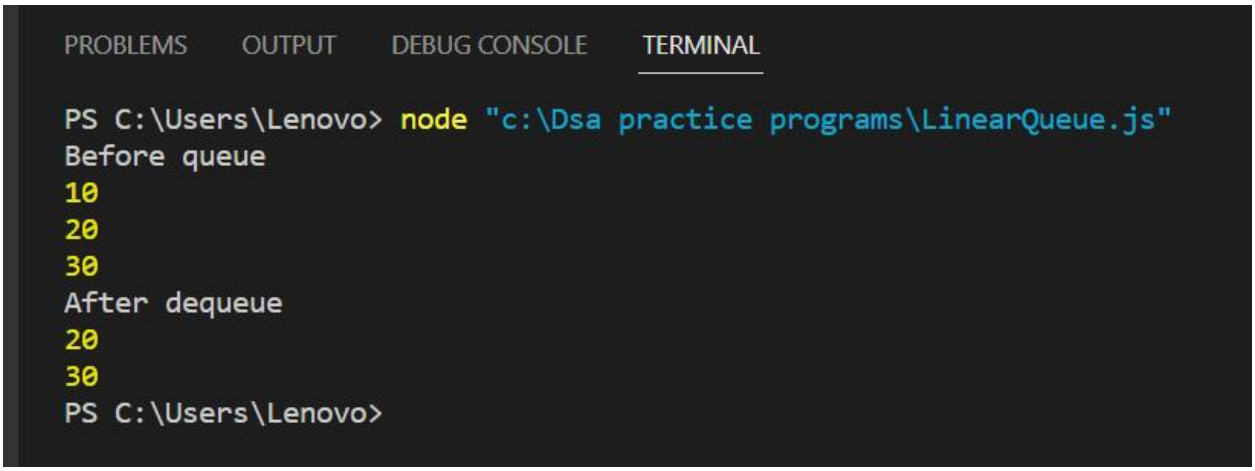
console.log("After dequeue");

q.dequeue();

q.display();

Output:

Screen Shot



……

Q.8 Write a program to Reverse stack using queue

Solution: Program

```
class Stack {
   constructor() {
      this.elements = [];
   }
   push(element) {
      this.elements.push(element);
   }
   pop() {
      if (this.isEmpty())
         return 'Underflow situation';
         else return this.elements.pop();
      }
      isEmpty(){
         return this.elements.length == 0;
      }
      print() {
         return this.elements;
      }
  }
   class Queue {
      constructor() {
         this.elements = [];
      }
      enqueue(element) {
         this.elements.push(element);
```

```
    }
    dequeue() {
        if (!this.isEmpty()) {
            return this.elements.shift();
        }
        else {
            return 'Underflow situation';
        }
    }
    isEmpty() {
        return this.elements.length == 0;
    }
}
function reverse(stack) {
    const queue = new Queue();
    while (!stack.isEmpty()) {
        queue.enqueue(stack.pop());
    }
    while (!queue.isEmpty()) {
        stack.push(queue.dequeue());
    }
}
const stack = new Stack();
stack.push('javaprogram');
stack.push('dsaprogram');
stack.push('Hellow');
console.log('Printing stack before reversal: ', stack.print());
reverse(stack);
```

console.log('Printing stack after reversal: ', stack.print());

Output:

Screen Shot

……

Q.9 Write a program to implement binary search tree with its operations

Solution: Program

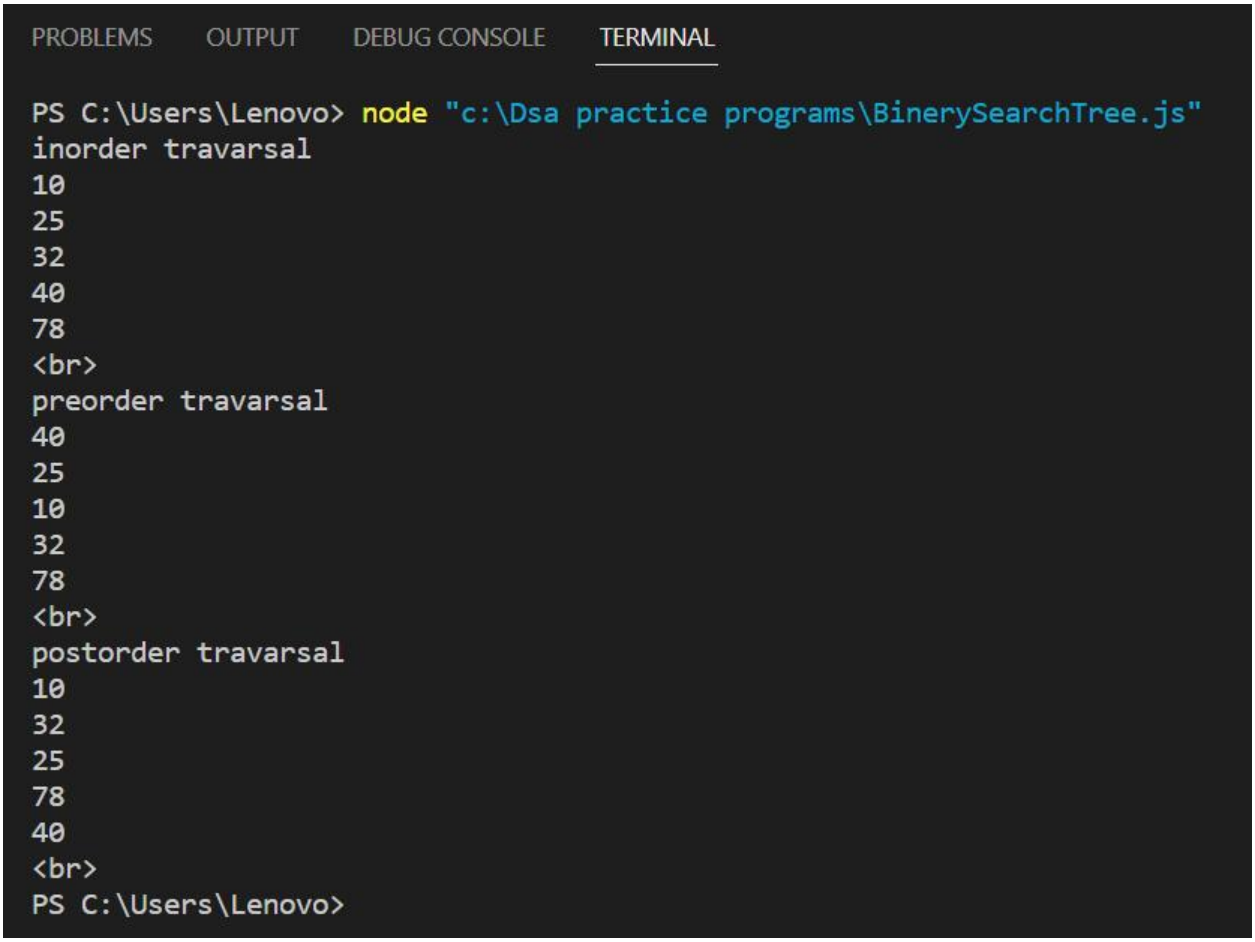
```
class node {
   constructor(value) {
      this.data = value;
      this.left = null;
      this.right = null;
   }
}
  class bst {
      constructor() {
         this.root = null;
      }
      insert(value){
         var newnode = new node(value);
         if (this.root == null) {
            this.root = newnode;
         }
         else {
            var temproot = this.root;
            while (1) {
               if (value < temproot.data) {
                  if (temproot.left == null) {
                     temproot.left = newnode;
                     break;
                  }
                  temproot = temproot.left;
               }
```

```
            if (value > temproot.data) {
                if (value > temproot.data) {
                    if (temproot.right == null) {
                        temproot.right = newnode;
                        break;
                    }
                    temproot = temproot.right;
                }
            }
        }
    }
}
inorder(root) {
    if (root == null) {
        return;
    }
    this.inorder(root.left);
    console.log(root.data+" ");
    this.inorder(root.right);
}
preorder(root) {
    if (root == null) {
        return;
    }
    console.log(root.data+" ");
    this.preorder(root.left);
    this.inorder(root.right);
}
```

```
    postorder(root) {
        if (root == null) {
            return;
        }
        this.postorder(root.left);
        this.postorder(root.right);
        console.log(root.data+" ");
    }
}
const obj = new bst();
obj.insert(40);
obj.insert(25);
obj.insert(78);
obj.insert(10);
obj.insert(32);
console.log("inorder travarsal ");
obj.inorder(obj.root);
console.log("<br>");
console.log("preorder travarsal ");
obj.preorder(obj.root);
console.log("<br>");
console.log("postorder travarsal ");
obj.postorder(obj.root);
console.log("<br>")
```

Output:

Screen Shot


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS C:\Users\Lenovo> node "c:\Dsa practice programs\BinerySearchTree.js"
inorder travarsal
10
25
32
40
78

preorder travarsal
40
25
10
32
78

postorder travarsal
10
32
25
78
40

PS C:\Users\Lenovo>

……

Q.10 Write a program to implement Circular Queue.

Solution: Program

```
class CircularQueue {

    constructor(size) {

        this.front = -1;

        this.rear = -1;

        this.data = [];

        this.size = size;

        this.length = 0;

    }

    enqueue(value) {

        if (this.length === this.size) return 'Queue is full';

            if (this.length === 0) {

                this.front++;

                this.rear++;

                this.data[this.front] = value;

            } else if (this.rear === this.size - 1 && this.front !== 0) {

                this.rear = 0;

                this.data[this.rear] = value;

            } else {

                this.rear++;

                this.data[this.rear] = value;

            }

            this.length++;

            return true;

        }

        dequeue() {

            if (this.length === 0) return 'Queue is empty';
```

```
                this.data[this.front] = null;
                if (this.length === 1) {
                        this.front = -1;
                        this.rear = -1;
                } else if (this.front === this.size - 1) {
                        this.front = 0;
                } else {
                        this.front++;
                }
                this.length--;
                return true;
        }
display() {
        if (this.length === 0) return 'Queue is empty';
                if (this.front <= this.rear) {
                        for (let i = this.front; i <= this.rear; i++) {
                                console.log(this.data[i]);
                        }
                } else {
                        for (let i = this.front; i < this.size; i++) {
                                console.log(this.data[i]);
                        }
                        for (let i = 0; i < this.rear; i++) {
                                console.log(this.data[i]);
                        }
                }
        }
}
```

let cq = new CircularQueue();

cq.enqueue(14);

cq.enqueue(22);

cq.enqueue(13);

cq.enqueue(7);

cq.dequeue();

cq.dequeue();

cq.enqueue(9);

cq.enqueue(99);

cq.dequeue();

cq.display();

Output:

Screen Shot

……

Q.11 Write a Program to print Adjacancy Matrix and Adjacancy List by reading Edges of Graph

Solution: Program

```
class Graph {
        constructor(edges) {
                this.adjMatrix = this._generateAdjacencyMatrix(edges);
                this.adjList = this._generateAdjacencyList(edges);
        }s
        _generateAdjacencyMatrix(edges) {
                const nodes = [...new Set(edges.flat())];
                const numNodes = nodes.length;
                const adjMatrix = Array(numNodes)
        .fill()
        .map(() => Array(numNodes).fill(0));
                const nodeToIndex = {};
                nodes.forEach((node, index) => {
                        nodeToIndex[node] = index;
        });

        edges.forEach(([src, dest]) => {
        const srcIndex = nodeToIndex[src];
        const destIndex = nodeToIndex[dest];
        adjMatrix[srcIndex][destIndex] = 1;
        adjMatrix[destIndex][srcIndex] = 1;
        });

        return adjMatrix;
}
```

```
    _generateAdjacencyList(edges) {
        const adjList = {};

        edges.forEach(([src, dest]) => {
        if (!adjList[src]) {
            adjList[src] = [];
        }
        if (!adjList[dest]) {
            adjList[dest] = [];
        }
        adjList[src].push(dest);
        adjList[dest].push(src);
        });
        return adjList;
    }
    printAdjMatrix() {
        console.log("Adjacency Matrix:");
        console.log(this.adjMatrix);
    }
    printAdjList() {
        console.log("Adjacency List:");
        console.log(this.adjList);
    }
}
    const edges = [  [0, 1], [0, 2], [1, 3], [2, 3] ];
    const graph = new Graph(edges);
    graph.printAdjMatrix();
    graph.printAdjList();
```

Output:

Screen Shot

……

Q.12 Write a Program to find the element in an array using Binary Search

Solution: Program

```
function binarySearch(arr, x) {
        let left = 0;
        let right = arr.length - 1;
        while (left <= right) {
                const mid = Math.floor((left + right) / 2);


                if (arr[mid] === x) {
                return mid;
                }
                else if (arr[mid] < x) {
                left = mid + 1;
                }
                else {
                right = mid - 1;


const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const x = 6;
const index = binarySearch(arr, x);
if (index === -1) {
                console.log(`Element ${x} not found in the array`);
} else {
                console.log(`Element ${x} found at index ${index}`);
}
                }
}
return -1;
```

}

Output:

Screen Shot

……

Q.13 Write a Program to find the element in an array using Linear Search

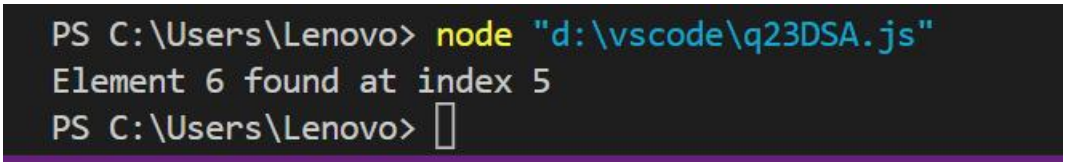
Solution: Program

```
function linearSearch(arr, x) {
        for (let i = 0; i < arr.length; i++) {
                if (arr[i] === x) {
                return i;
                }
        }
        return -1;
}
        const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
        const x = 6;
        const index = linearSearch(arr, x);

        if (index === -1) {
                console.log(`Element ${x} not found in the array`);
        }
        else {
                console.log(`Element ${x} found at index ${index}`);
        }
```

Output:

Screen Shot



……

Q.14 Write a Program to implement the following

1.Print Pascal's triangle for n=5

Solution: Program

```
function pascalTriangle(n) {
        const triangle = [];


        for (let i = 0; i < n; i++) {
                const row = [];


                for (let j = 0; j <= i; j++) {
                if (j === 0 || j === i) {
                        row.push(1);
                }
                        else {
        row.push(triangle[i - 1][j - 1] + triangle[i - 1][j]);
        }
        }
                triangle.push(row);
        }
        return triangle;
}
const n = 5;
const triangle = pascalTriangle(n);
for (let i = 0; i < triangle.length; i++) {
        console.log(triangle[i].join(" "));
}
```

Output:

Screen Shot

```
PS C:\Users\Lenovo> node "d:\vscode\q23DSA.js"
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
PS C:\Users\Lenovo>
```

……

Q.15 Write a Program to implement the following GCD of two numbers using Euclidean Algorithm

Solution: Program

```
function gcd(a, b) {
        while (b !== 0) {
                const temp = b;
                b = a % b;
                a = temp;
        }
        return a;
}

const num1 = 24;
const num2 = 36;
const result = gcd(num1, num2);
console.log(`GCD of ${num1} and ${num2} is ${result}`);
```

Output:

Screen Shot

```
PS C:\Users\Lenovo> node "d:\vscode\q23DSA.js"
GCD of 24 and 36 is 12
PS C:\Users\Lenovo>
```

……

Q.16 Write a program to implement

1. tower of Hanoi where number of disks=4

Solution: Program

```
function towerOfHanoi(n, source, destination, auxiliary) {
        if (n === 1) {
        console.log(`Move disk 1 from ${source} to ${destination}`);
        return;
   }
   towerOfHanoi(n - 1, source, auxiliary, destination);
   console.log(`Move disk ${n} from ${source} to ${destination}`);
   towerOfHanoi(n - 1, auxiliary, destination, source);
}
const numDisks = 4;
towerOfHanoi(numDisks, 'A', 'C', 'B');
```

Output:

Screen Shot

```
PS C:\Users\Lenovo> node "d:\vscode\q23DSA.js"
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
PS C:\Users\Lenovo>
```

Q.17 Write a program to implement 2. Fibonacci series till N

Solution: Program

```
function fibonacciSeries(n) {
        if (n === 0) {
                return [];
        }
        if (n === 1) {
                return [0];
        }
        const series = [0, 1];
        while (series[series.length - 1] < n) {
                const nextNumber = series[series.length - 1] + series[series.length - 2];
                if (nextNumber > n) {
                        break;
                }
        series.push(nextNumber);
        }
        return series;
        }
        const N = 100;
        const series = fibonacciSeries(N);
        console.log(`Fibonacci series up to ${N}: ${series.join(', ')
}`);
```

Output:

Screen Shot

```
PS C:\Users\Lenovo> node "d:\vscode\q23DSA.js"
Fibonacci series up to 100: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89
PS C:\Users\Lenovo>
```