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:

Name : Prasad Ravindra Patil

Seat No : 20302

Roll No : 32(TC)

Date :14-March-2023

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 : Divya Jayakar Yadav

Roll No. 71

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

14.	Write a Program to implement the following 1.Print Pascal's triangle for n=5				
15.	Write a Program to implement the following GCD of two numbers using Euclidean Algorithm				
16.	Write a program to implement 1. tower of Hanoi where number of disks=4				
17.	Write a program to implement 2. Fibonacci series till 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;
      var temp;
   }
}
class SLL
{
   constructor()
   {
      this.head=null;
      this.count=0;
   }
   insertFirst(val)
   {
      let temp = new node(val);
      temp.next=this.head;
      this.head=temp;
      this.count++;
   }
   insertLast(val)
   {
      let temp=new node(val);
      if(this.head==null)
        this.head=temp;
      else
      {
         let ptr=this.head;
         while(ptr.next)
             ptr=ptr.next;
         ptr.next=temp;
      }
      this.count++;
   }
   insertatPos(val,pos)
   {
      if(pos<1||pos>this.count+1)
      {
         console.log("Invalid Position");
```

```
    }
    else
    {
        if(pos==1)
        {
            insertFirst(val);
        }
        else if(pos==count+1)
        {
            insertLast(val);
        }
        else
        {
            temp=new node();
            temp.data=val;
            temp.next=null;
            index=1;
            trv=head;
            while(index<pos-1)
              {
                 trv=trv.next;
                 index++;
              }
            temp.next=trv.next;
            trv.next=temp;
            count++;
        }
    }
}
deleteFirst()
{
    if(this.head==null)
      console.log("List is Empty");
    else
    {
        let val=this.head.data;
        if(this.head.next==null)
            this.head=null;
        else
            this.head=this.head.next;
        this.count--;
        return val;
    }
}
deleteLast()
{
    if(this.head==null)
      console.log("List is Empty");
    else
```

```
{
   if(this.head.next==null)
   {
      let val=this.head.data;
      this.head=null;
      return val;
   }
   else
   {
      let ptr=this.head;
      let ptr1=null;
      while(ptr.next!=null)
      {
        ptr1=ptr;
        ptr=ptr.next;
      }
      ptr1.next=null;
      return ptr.data;
   }
}
deletePos(pos)
{
   if(pos<=0)
   {
      console.log(" Empty ");
   }
   else if(pos==1)
   {
      deleteFirst();
   }
   else if(pos==count)
   {
      deleteLast();
   }
   else
   {
      index=1;
      trv=head;
      while(index<pos-1)
        {
           trv=trv.next;
           index++;
        }
      temp=trv.next;
      trv.next=temp.next;
      free (temp);
      count--;
   }
}
```

```
  }
  display()
  {
    if(this.head==null)
      console.log("List is Empty");
    else
    {
      let ptr=this.head;
      while(ptr)
      {
        console.log(ptr.data);
        ptr=ptr.next;
      }
    }
  }
}
function createList()
{
  let obj=new SLL();
  return obj;
}

let obj=createList();
obj.insertFirst(3);
obj.insertFirst(2);
obj.insertFirst(1);
obj.insertFirst(0);
obj.insertLast(4);
obj.insertLast(5);
obj.display();
console.log("Size of List is:",obj.count);
let val=obj.deleteFirst();
console.log(val);
val=obj.deleteLast();
console.log(val);
obj.display();
```

Output:

Screen Shot

```
0
1
2
3
4
5
Size of List is: 6
0
5
1
2
3
4
undefined
```

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

Solution: Program

```
class node
{
   constructor(value)
   {
      this.prev=null;
      this.next=null;
      this.data=value;
   }
}
class DLL
{
   constructor()
   {
      this.head=null;
      this.count=0;
   }
   insertFirst(val)
   {
      let obj=new node(val);
      obj.next=this.head;
      this.head=obj;
      this.count++;
   }
   insertLast(val)
   {
      let obj=new node(val);
      if(this.head==null)
         this.head=obj;
      else
      {
         let ptr=this.head;
         while(ptr.next!=null)
            ptr=ptr.next;
         obj.prev=ptr;
         ptr.next=obj;
      }
      this.count++;
   }
   insertatPos(val,pos)
   {
      if(pos<=0||pos>=count+2)
      {
         console.log("Invalid position");
      }
```

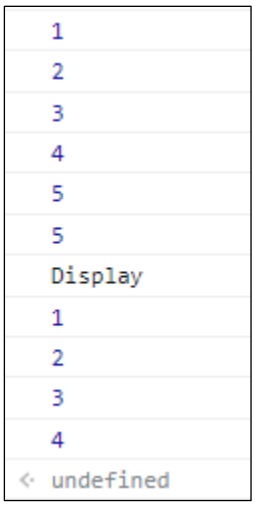
```
    else
    {
      if(pos==1)
      {
        insertFirst(val);
      }
      else if(pos==count+1)
      {
        insertatLast(val);
      }
      else
      {
        var index=1;
        while(index<pos-1)
          {
            trv=trv.next;
            index++;
          }
        temp.prev=trv;
        temp.next=trv.next;
        trv.next=temp;
        temp.next.prev=temp;
      }
    }
}
deleteFirst()
{
    if(this.head==null)
      console.log("List is Empty");
    else
    {
      let val=this.head.data;
      if(this.head.next==null)
        this.head=null;
      else
      {
        this.head.next.prev=null;
        this.head=this.head.next;
      }
      this.count--;
      return val;
    }
}
deleteLast()
{
    if(this.head==null)
      console.log("List is Empty");
    else
    {
```

```
        let val;
        if(this.head.next==null)
        {
          val=this.head.data;
          this.head=null;
        }
        else
        {
           let ptr=this.head;
           let ptr1=null;
           while(ptr.next!=null)
           {
              ptr1=ptr;
              ptr=ptr.next;
           }
           ptr1.next=null;
           val=ptr.data;
        }
        this.count--;
        return val;
    }
}
deletePos()
{
    if(pos<=0||pos>count+1)
    {
       console.log("Position not valid");
    }
    else
    {
       trv=head;
       if(pos==1)
       {
          deleteFirst();
       }
       else if(pos==count)
       {
          deleteLast();
       }
       else
       {
          while(i<pos-1)
             {
                trv=trv.next;
                i++;
             }
          temp=trv.next;
          trv.next=temp.next;
          temp.next.prev=trv;
```

```
            free(temp);
            count--;
          }
        }
    }
    display()
    {
        let ptr=this.head;
        while(ptr!=null)
        {
            console.log(ptr.data);
            ptr=ptr.next;
        }
    }
}
let obj=new DLL()
obj.insertFirst(3);
obj.insertFirst(2);
obj.insertFirst(1);
obj.insertLast(4);
obj.insertLast(5);
obj.display();
val=obj.deleteLast();
console.log(val);
console.log("Display");
obj.display();
```

Output:

Screen Shot



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

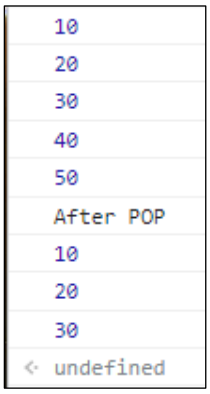
Solution: Program

```
class stack
{
   constructor(size)
   {
     this.arr=Array(size)
     this.top=-1;
     this.size=size;
   }
   push(data)
   {
      if(this.top==this.size-1)
      {
         console.log("Stack is Full");
      }
      else
      {
         this.top++;
         this.arr[this.top]=data;
      }
   }
   pop()
   {
      if(this.top<0)
         console.log("Stack is Empty");
      else
      {
         let val;
         val=this.arr[this.top];
         this.top--;
         return val;
      }
   }
   display()
   {
       let i;
       for(i=0;i<=this.top;i++)
          console.log(this.arr[i]);
   }
}
let obj=new stack(5)
obj.push(10);
obj.push(20);
obj.push(30);
obj.push(40);
obj.push(50);
obj.display();
```

```
obj.pop()
obj.pop()
console.log("After POP")
obj.display();
```

Output:

Screen Shot



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

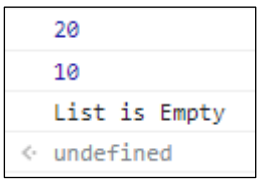
Solution: Program

```
class node
{
   constructor(value)
   {
      this.data=value;
      this.next=null;
   }
}
class stack
{
   constructor()
   {
      this.top=null;
      this.count=0;
   }
   push(val)
   {
      let obj=new node(val);
      obj.next=this.top;
      this.top=obj;
      this.count++;
   }
   pop()
   {
      if(this.top==null)
        console.log("List is Empty");
      else
      {
         let val=this.top.data;
         this.top=this.top.next;
         this.count--;
         return val;
      }
   }
   display()
   {
      let ptr=this.top;
      while(ptr!=null)
      {
         console.log(ptr.data);
         ptr=ptr.next;
      }
   }
}
let obj=new stack();
```

```
obj.push(10);
obj.push(20);

obj.display();
obj.pop();
obj.pop();
obj.pop();
```

Output:

Screen Shot



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

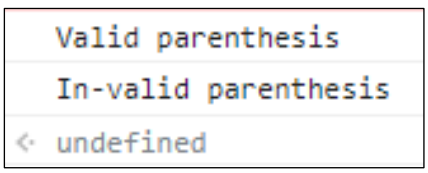
Solution: Program

```
// Stack
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Stack {
  constructor(size) {
    this.data = [];
    this.size = size;
    this.top = -1;
    this.length = 0;
  }

  isEmpty() {
    if (this.length === 0) {
      return true;
    }
    return false;
  }

  isFull() {
    if (this.length === this.size) {
      return true;
    }
    return false;
  }

  // unshift add element at first
  push(value) {
    if (this.isFull()) {
      return 'Stack is full';
    }
    this.top++;
    this.data[this.top] = value;
    this.length++;
    return true;
  }

  // shift //remove element from first
  pop() {
    if (this.isEmpty()) return 'Stack is empty';
    else {
```

```
    let removeElm = this.data[this.top];
    this.data.pop();
    this.top--;
    this.length--;

    return removeElm;
   }
  }

  display() {
   for (let i = 0; i < this.length; i++) {
    console.log(this.data[i]);
   }
  }
}

const parenthesisChecker = (str) => {
  const s = new Stack();

  for (let i of str) {
   if (i == '(' || i == '[' || i == '{') {
    s.push(i);
   }

   if (i == ')' || i == ']' || i == '}') {
    s.pop();
   }
  }
  if (!s.length) {
   console.log(`Valid parenthesis`);
  } else {
   console.log(`In-valid parenthesis`);
  }
};

parenthesisChecker('(a+b)+(a-b)');
parenthesisChecker('(a+b)+(a-b)[+[a/b]{');
```

Output:

Screen Shot



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

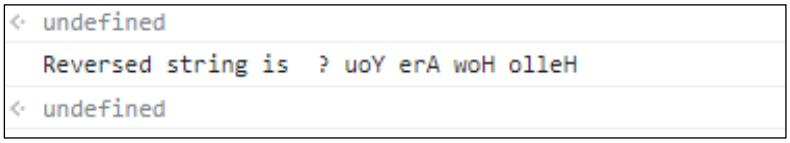
Solution: Program

```
class Stack
{
   size;
   top;
   a = [];
   isEmpty()
   {
      return(this.top < 0);
   }

   constructor(n)
   {
      this.top = -1;
      this.size = n;
      this.a = new Array(this.size);
   }

   // Function to push element in Stack
   push(x)
   {
      if (this.top >= this.size)
      {
         document.write("Stack Overflow<br>");
         return false;
      }
      else
      {
         this.a[++this.top] = x;
         return true;
      }
   }

   // Function to pop element from stack
   pop()
   {
      if (this.top < 0)
      {
         document.write("Stack Underflow<br>");
         return 0;
      }
      else
      {
         let x = this.a[this.top--];
         return x;
      }
```

```
    }
}

// Function to reverse the string
function reverse(str)
{

    // 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

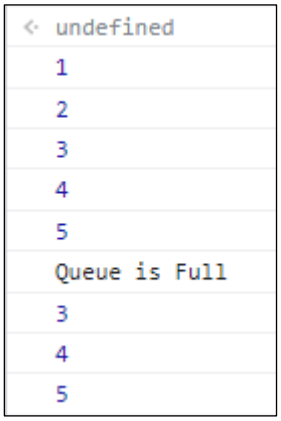
Solution: Program

```
class Queue
{
   constructor(size)
   {
      this.arr=Array(size);
      this.front=-1;
      this.rear=-1;
      this.capacity=size;
   }
   insertion(val)
   {
      if(this.rear==this.capacity-1)
         console.log("Queue is Full");
      else if(this.front==-1)
      {
         this.front=this.rear=0;
         this.arr[this.rear]=val;
      }
      else
      {
         this.rear++;
         this.arr[this.rear]=val;
      }
   }
   deletion()
   {
      if(this.rear==-1)
        console.log("Queue is Empty");
      else
      {
         let val=this.arr[this.front]
         if(this.rear==this.front)
            this.rear=this.front=-1;
         else
            this.front++;
         return val;
      }
   }
   display()
   {
      let i;
      for(i=this.front;i<=this.rear;i++)
        console.log(this.arr[i]);
   }
}
```

```
let obj=new Queue(5)
obj.insertion(1);
obj.insertion(2);
obj.insertion(3);
obj.insertion(4);
obj.insertion(5);
obj.display();
obj.insertion(6);
obj.deletion()
obj.deletion()
obj.display()
```

Output:

Screen Shot



Q.8 Write a program to Reverse stack using queue

Solution: Program

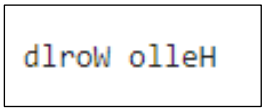
```
class Queue
{
  constructor(size)
  {
    this.arr=Array(size);
    this.front=-1;
    this.rear=-1;
    this.capacity=size;
  }
  insertion(val)
  {
    if(this.front==0 && this.rear==this.capacity-1 || this.rear==this.front-1)
      console.log("Queue is Full");
    else if(this.rear==-1)
    {
      this.front=this.rear=0;
      this.arr[this.rear]=val;
    }
    else if(this.rear==this.capacity-1)
    {
      this.rear=0;
      this.arr[this.rear]=val;
    }
    else
    {
      this.rear++;
      this.arr[this.rear]=val;
    }
  }
  deletion()
  {
    if(this.front==-1)
      console.log("Queue is Empty");
    else
    {
      let val=this.arr[this.front];
      if(this.front==this.rear)
        this.front=this.rear=-1;
      else if(this.front==this.capacity-1)
        this.front=0;
      else
        this.front++;
      return val;
    }
  }
```

```
  display()
  {
     let i;
     if(this.front<this.rear)
        for(i=this.front;i<=this.rear;i++)
            process.stdout.write(String(this.arr[i]))
     else
     {
        for(i=this.front;i<this.capacity;i++)
           process.stdout.write(String(this.arr[i]))
        for(i=0;i<=this.rear;i++)
           process.stdout.write(String(this.arr[i]))
     }

  }
}
// Stack Data Structure
class stack
{
  constructor(size)
  {
    this.arr=Array(size)
    this.top=-1;
    this.size=size;
  }
  push(data)
  {
     if(this.top==this.size-1)
     {
        console.log("Stack is Full");
     }
     else
     {
        this.top++;
        this.arr[this.top]=data;
     }
  }
  pop()
  {
     if(this.top<0)
        console.log("Stack is Empty");
     else
     {
        let val;
        val=this.arr[this.top];
        this.top--;
        return val;
     }
  }
```

```
    display()
    {
        let i;
        for(i=0;i<=this.top;i++)
            console.log(this.arr[i]);
    }
}

let stkObj= new stack(12);
let queueObj = new Queue(12);
let str="Hello World";
for(i=0;i<str.length;i++)
    stkObj.push(str[i]);
for(i=0;i<str.length;i++)
{
    let char = stkObj.pop();
    queueObj.insertion(char);
}

queueObj.display();
```

Output:

Screen Shot



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

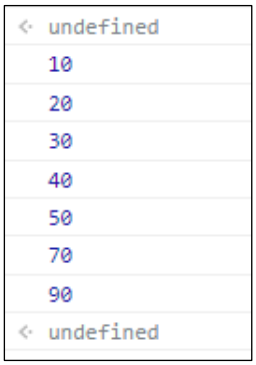
Solution: Program

```
class node
{
   constructor(val)
   {
      this.data=val;
      this.left=null;
      this.right=null;
   }
}
class BST
{
   constructor()
   {
      this.root=null;
   }
   insertion(val)
   {
      let temp=new node(val);
      if(this.root==null)
        this.root=temp;
      else
      {
         let r=this.root;
         while(1)
         {
            if(r.data>val)
            {
               if(r.left==null)
               {
                  r.left=temp;
                  break;
               }
               r=r.left;
            }
            else if(r.data<val)
            {
               if(r.right==null)
               {
                  r.right=temp;
                  break;
               }
               r=r.right;
            }
         }
      }
```

```
    }

    inOrder(r)
    {
        if(r==null)
            return;
        this.inOrder(r.left);
        console.log(r.data);
        this.inOrder(r.right);
    }
    preOrder(r)
    {
        if(r==null)
            return;
        console.log(r.data);
        this.inOrder(r.left);
        this.inOrder(r.right);
    }
    postOrder(r)
    {
        if(r==null)
            return;
        this.inOrder(r.left);
        this.inOrder(r.right);
        console.log(r.data);
    }
}
let obj=new BST();
obj.insertion(10);
obj.insertion(50);
obj.insertion(40);
obj.insertion(30);
obj.insertion(20);
obj.insertion(90);
obj.insertion(70);
obj.inOrder(obj.root);
```

Output:

Screen Shot



Q.10 Write a program to implement

Circular Queue

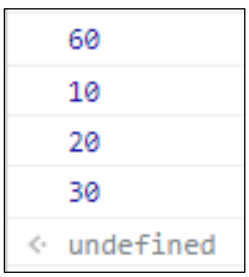
Solution: Program

```
class Queue
{
   constructor(size)
   {
     this.arr=Array(size);
     this.front=-1;
     this.rear=-1;
     this.capacity=size;
   }
   insertion(val)
   {
      if(this.front==0 && this.rear==this.capacity-1 || this.rear==this.front-1)
         console.log("Queue is Full");
      else if(this.rear==-1)
      {
         this.front=this.rear=0;
         this.arr[this.rear]=val;
      }
      else if(this.rear==this.capacity-1)
      {
         this.rear=0;
         this.arr[this.rear]=val;
      }
      else
      {
         this.rear++;
         this.arr[this.rear]=val;
      }
   }
   deletion()
   {
      if(this.front==-1)
         console.log("Queue is Empty");
      else
      {
         let val=this.arr[this.front];
         if(this.front==this.rear)
            this.front=this.rear=-1;
         else if(this.front==this.capacity-1)
            this.front=0;
         else
            this.front++;
         return val;
```

```
    }
  }
  display()
  {
    let i;
    if(this.front<this.rear)
      for(i=this.front;i<=this.rear;i++)
          console.log(this.arr[i]);
    else
    {
      for(i=this.front;i<this.capacity;i++)
        console.log(this.arr[i]);
      for(i=0;i<=this.rear;i++)
        console.log(this.arr[i]);
    }

  }
}
let obj=new Queue(6);
obj.insertion(10);
obj.insertion(20);
obj.insertion(30);
obj.insertion(40);
obj.insertion(50);
obj.insertion(60);
obj.deletion()
obj.deletion()
obj.deletion()
obj.deletion()
obj.deletion()
obj.insertion(10);
obj.insertion(20);
obj.insertion(30);
obj.display();
```

Output:

Screen Shot



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

Solution: Program

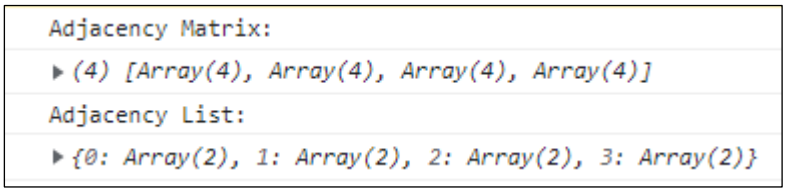
```
class Graph {
  constructor(edges) {
    this.adjMatrix = this._generateAdjacencyMatrix(edges);
    this.adjList = this._generateAdjacencyList(edges);
  }

  _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;
     }
   }

   return -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}`);
}
```

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

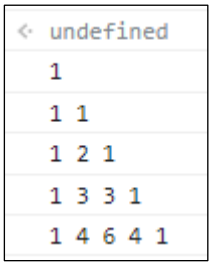
Q.14 Write a Program to implement the following 1.Print Pascal's triangle for n=5

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



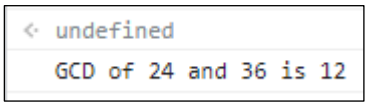
Q.15 Write a Program to implement the following GCD of two numbers using Euclidean Algorithm

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



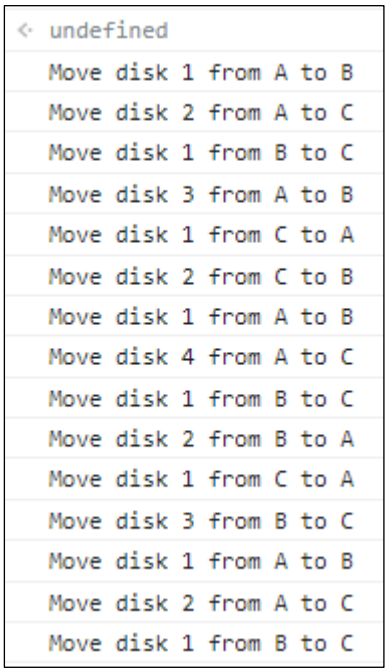
Q.16 Write a program to implement 1. tower of Hanoi where number of disks=4

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



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