Roll No. – 22239

1. Script to implement singly linked list.

```
class Node
{
constructor(value)
{
this.value = value
this.next = null
}
}
class LinkedList
{
constructor(value)
{
const newNode = new Node(value)
this.head = newNode
this.tail = this.head
this.length = 1
}
push(value)
{
const newNode = new Node(value)
if (!this.head)
{
this.head = newNode
this.tail = newNode
}
else
{
this.tail.next = newNode
this.tail = newNode
}
this.length++
return this
}
pop()
{
if(!this.head) return undefined
let temp = this.head
let pre = this.head
while(temp.next)
{
```

```
pre = temp
temp = temp.next
}
this.tail = pre
this.tail.next = null
this.length--
if(this.length === 0)
{
this.head = null
this.tail = null
}
return temp
}
insert(index, value)
{
if(index < 0 || index > this.length) return false
if(index === this.length) return this.push(value)
if(index === 0) return this.unshift(value)
const newNode = new Node(value)
const temp = this.get(index - 1)
newNode.next = temp.next
temp.next = newNode
this.length++
return true
}
remove(index)
{
if(index < 0 || index >= this.length) return undefined
if(index === 0) return this.shift()
if(index === this.length - 1) return this.pop()
const before = this.get(index - 1)
const temp = before.next
before.next = temp.next
temp.next = null
this.length--
return temp
}
reverse()
{
let temp = this.head
this.head  =  this.tail
this.tail = temp
let next = temp.next
let prev = null
for(let i = 0; i < this.length; i++)
```

```
{
next = temp.next
temp.next = prev
prev = temp
temp = next
}
return this
}
}
let myLinkedList = new LinkedList(50)
myLinkedList.push(35)
myLinkedList.push(5)
```

Output:

```
1. LinkedList {head: Node, tail: Node, length: 3}
    1. head: Node
        1. next: Node
            1. next: Node
                1. next: null
                2. value: 5
                3. [[Prototype]]: Object
            2. value: 35
            3. [[Prototype]]: Object
        2. value: 98
        3. [[Prototype]]: Object
    2. length: 3
    3. tail: Node
        1. next: null
        2. value: 18
        3. [[Prototype]]: Object
    4. [[Prototype]]: Object
```

Roll No. – 22239

2.Script to implement doubly linked list.

```
class Node
{
constructor(value)
{
this.value = value
this.next = null
this.prev = null
}
}
class DoublyLinkedList
{
constructor(value)
{
const newNode = new Node(value)
this.head = newNode
this.tail = newNode
this.length = 1
}
push(value)
{
const newNode = new Node(value)
if(this.length === 0)
{
this.head = newNode
this.tail = newNode
}
else
{
this.tail.next = newNode
newNode.prev = this.tail
this.tail = newNode
}
this.length++
return this
}
pop()
{
if(this.length === 0)
return undefined
let temp = this.tail
```

```
if (this.length === 1)
{
this.head = null
this.tail = null
}
else
{
this.tail = this.tail.prev
this.tail.next = null
temp.prev = null
}
this.length--
return temp
}
insert(index, value)
{
if(index < 0 || index > this.length)
return false
if(index === this.length)
return this.push(value)
if(index === 0)
return this.unshift(value)
const newNode = new Node(value)
const before = this.get(index - 1)
const after = before.next
before.next = newNode
newNode.prev = before
newNode.next = after
after.prev = newNode
this.length++
return true
}
remove(index)
{
if(index === 0)
return this.shift()
if(index === this.length - 1)
return this.pop()
if(index < 0 || index >= this.length)
return undefined
const temp = this.get(index)
temp.prev.next = temp.next
temp.next.prev = temp.prev
temp.next = null
temp.prev = null
```

```
this.length--
return temp
}
}
let myDoublyLinkedList = new DoublyLinkedList(35)
myDoublyLinkedList.push(20)
myDoublyLinkedList.push(28)
```

Output:

```
1. DoublyLinkedList {head: Node, tail: Node, length: 3}
    1. head: Node
        1. next: Node {value: 20, next: Node, prev: Node}
        2. prev: null
        3. value: 35
        4. [[Prototype]]: Object
    2. length: 3
    3. tail: Node {value: 28, next: null, prev: Node}
    4. [[Prototype]]: Object
```

Roll No. – 22239

2. Script to implement Stack.

```
class Node

{

constructor(value)

{

this.value = value

this.next = null

}

}

class Stack

{

constructor(value)

{

const newNode = new Node(value)

this.top = newNode

this.length = 1

}

push(value)

{

const newNode = new Node(value)

if(this.length === 0)

{

this.top = newNode

}
else

{

newNode.next = this.top

this.top = newNode

}

this.length++

return this

}

pop()

{

if(this.length === 0)

return undefined

let temp = this.top

this.top = this.top.next

temp.next = null

this.length--

return temp

}

}

let myStack = new Stack(35)

myStack.push(15)

myStack.push(48)

myStack.push(6)
```

Output:

```
1.  Stack {top: Node, length: 4}
    1.  length: 4
    2.  top: Node
        1.  next: Node
            1.  next: Node
                1.  next: Node
                    1.  next: null
                    2.  value: 35
                    3.  [[Prototype]]: Object
                2.  value: 15
                3.  [[Prototype]]: Object
            2.  value: 48
            3.  [[Prototype]]: Object
        2.  value: 6
        3.  [[Prototype]]: Object
    3.  [[Prototype]]: Object
```

4. Script to demonstrate Queue

```
class Node
{
constructor(value)
{
this.value = value
this.next = null
}
}
class Queue
{
constructor(value)
{
const newNode = new Node(value)
this.first = newNode
this.last = newNode
this.length = 1
}
enqueue(value)
{
const newNode = new Node(value)
if (this.length === 0)
{
this.first = newNode
this.last = newNode
}
else
{
this.last.next = newNode
this.last = newNode
}
this.length++
return this
}
dequeue()
{
if(this.length === 0)
return undefined
let temp = this.first
if(this.length === 1)
{
this.last = null
}
else
{
this.first = this.first.next
temp.next = null
}
this.length--
return temp
}
}
let myQueue = new Queue(22)
myQueue.enqueue(6)
myQueue.enqueue(35)
myQueue.enqueue(9)
```

Output:

```
1.  Queue {first: Node, last: Node, length: 4}
        1.  first: Node
                1.  next: Node {value: 35, next: Node}
                2.  value: 6
                3.  [[Prototype]]: Object
                4.  value:22
                5.  [[Prototype]]: Object
        2.  last: Node
                1.  next: null
                2.  value: 9
                3.  [[Prototype]]: Object
        3.  length: 4
        4.  [[Prototype]]: Object
```

5. Script to implement Binary Search Tree.

```
class Node
{
constructor(value)
{
this.value = value
this.left = null
this.right = null
}
}
class BST
{
constructor()
{
this.root = null
}
insert(value)
{
const newNode = new Node(value)
if (this.root === null)
{
this.root = newNode
return this
}
let temp = this.root
while(true)
{
if (newNode.value === temp.value)
return undefined
if (newNode.value < temp.value)
{
if (temp.left === null)
{
temp.left = newNode
return this
}
temp = temp.left
}
else
{
if (temp.right === null)
{
```

```
temp.right = newNode
return this
}
temp = temp.right
}
}
}
contains(value)
{
if (this.root === null)
return false
let temp = this.root
while(temp)
{
if (value < temp.value)
{
temp = temp.left
}
else if (value > temp.value)
{
temp = temp.right
}
else
{
return true
}
}
return false
}
}
let myTree = new BST()
myTree.insert(55)
myTree.insert(30)
myTree.insert(88)
myTree.insert(57)
myTree.insert(60)
myTree.insert(89)
```

Output :

```
BST {root: Node}
root: Node
left: Node
left: Node
left: null
right: null
value: 30
[[Prototype]]: Object
right: Node
left: Node
left: null
right: null
value: 60
[[Prototype]]: Object
right: null
value: 57
[[Prototype]]: Object
value: 89
[[Prototype]]: Object
right: Node
left: null
right: null
value: 88
[[Prototype]]: Object
value: 55
[[Prototype]]: Object
[[Prototype]]: Object
```

Roll No. – 22239

6. Script to implement Graph

```
class Graph
{
constructor()
{
this.adjacencyList = {}
}
addVertex(vertex)
{
if(!this.adjacencyList[vertex])
{
this.adjacencyList[vertex] = []
return true
}
return false
}
addEdge(vertex1, vertex2)
{
if (this.adjacencyList[vertex1] && this.adjacencyList[vertex2])
{
this.adjacencyList[vertex1].push(vertex2)
this.adjacencyList[vertex2].push(vertex1)
return true
}
return false
}
}
let myGraph = new Graph()
myGraph.addVertex("A")
myGraph.addVertex("B")
myGraph.addVertex("C")
myGraph.addVertex("D")
myGraph.addEdge("A", "B")
myGraph.addEdge("A", "C")
myGraph.addEdge("A", "D")
myGraph.addEdge("B", "D")
myGraph.addEdge("C", "D")
myGraph
```

Output:

```
1.  Graph {adjacencyList: {…}}
        1.  adjacencyList:
                1.  A: Array(3)
                        1.  0: "B"
                        2.  1: "C"
                        3.  2: "D"
                        4.  length: 3
                        5.  [[Prototype]]: Array(0)
                2.  B: Array(2)
                        1.  0: "A"
                        2.  1: "D"
                        3.  length: 2
                        4.  [[Prototype]]: Array(0)
                3.  C: Array(2)
                        1.  0: "A"
                        2.  1: "D"
                        3.  length: 2
                        4.  [[Prototype]]: Array(0)
                4.  D: Array(3)
                        1.  0: "A"
                        2.  1: "B"
                        3.  2: "C"
                        4.  length: 3
                        5.  [[Prototype]]: Array(0)
                5.  [[Prototype]]: Object
        2.  [[Prototype]]: Object
```

Roll No. – 22239

7. Script to implement BFS

```
class Node
{
constructor(value)
{
this.value = value
this.left = null
this.right = null
}
}
class BFS
{
constructor()
{
this.root = null
}
insert(value)
{
const newNode = new Node(value)
if (this.root === null)
{
this.root = newNode
return this
}
let temp = this.root
while(true)
{
if (newNode.value === temp.value)
return undefined
if (newNode.value < temp.value)
{
if (temp.left === null)
{
temp.left = newNode
return this
}
temp = temp.left
}
else
{
if (temp.right === null)
{
```

```
temp.right = newNode
return this
}
temp = temp.right
}
}
}
contains(value)
{
if (this.root === null)
return false
let temp = this.root
while(temp)
{
if (value < temp.value)
{
temp = temp.left
}
else if (value > temp.value)
{
temp = temp.right
}
else
{
return true
}
}
return false
}
BFS()
{
let currentNode = this.root
let results = []
let queue = []
queue.push(currentNode)
while(queue.length)
{
currentNode = queue.shift()
results.push(currentNode.value)
if(currentNode.left) queue.push(currentNode.left)
if(currentNode.right) queue.push(currentNode.right)
}
return results
}
}
```

```
let myTree = new BFS()
myTree.insert(55)
myTree.insert(20)
myTree.insert(36)
myTree.insert(14)
myTree.insert(24)
myTree.insert(666)
```

Output:

```
     1. BFS {root: Node}
          1. root: Node
               1. left: Node
                    1. left: Node
                         1. left: null
                         2. right: null
                         3. value: 14
                         4. [[Prototype]]: Object
2.                     right: Node
                         1. left:
                            Node {value: 24, left: null, right: null}
                         2. right: null
                         3. value: 36
                         4. [[Prototype]]: Object
                    3. value: 20
                    4. [[Prototype]]: Object
               2. right: Node
                    1. left: null
                    2. right: null
                    3. value: 666
                    4. [[Prototype]]: Object
               3. value: 55
               4. [[Prototype]]: Object
          2. [[Prototype]]: Object
```

8. Script to implement DFS

```
class Node
{
constructor(value)
{
this.value = value
this.left = null
this.right = null
}
}
class DFS
{
constructor()
{
this.root = null
}
insert(value)
{
const newNode = new Node(value)
if (this.root === null)
{
this.root = newNode
return this
}
let temp = this.root
while(true)
{
if (newNode.value === temp.value)
return undefined
if (newNode.value < temp.value)
{
if (temp.left === null)
{
temp.left = newNode
return this
}
temp = temp.left
}
else
{
if (temp.right === null)
{
```

```
temp.right = newNode
return this
}
temp = temp.right
}
}
}
contains(value)
{
if (this.root === null) return false
let temp = this.root
while(temp)
{
if (value < temp.value)
{
temp = temp.left
}
else if (value > temp.value)
{
temp = temp.right
}
else
{
return true
}
}
return false
}
BFS()
{
let currentNode = this.root
let results = []
let queue = []
queue.push(currentNode)
while(queue.length)
{
currentNode = queue.shift()
results.push(currentNode.value)
if(currentNode.left) queue.push(currentNode.left)
if(currentNode.right) queue.push(currentNode.right)
}
return results
}
DFSPreOrder()
{
```

```
let results = []
function traverse(currentNode)
{
results.push(currentNode.value)
if(currentNode.left) traverse(currentNode.left)
if(currentNode.right) traverse(currentNode.right)
}
traverse(this.root)
return results
}
DFSPostOrder()
{
let results = []
function traverse(currentNode)
{
if(currentNode.left) traverse(currentNode.left)
if(currentNode.right) traverse(currentNode.right)
results.push(currentNode.value)
}
traverse(this.root)
return results
}
DFSInOrder()
{
let results = []
function traverse(currentNode)
{
if(currentNode.left) traverse(currentNode.left)
results.push(currentNode.value)
if(currentNode.right) traverse(currentNode.right)
}
traverse(this.root)
return results
}
}
let myTree = new DFS()
myTree.insert(48)
myTree.insert(13)
myTree.insert(20)
myTree.insert(5)
myTree.insert(8)
myTree.insert(315)
```

Output:

```
1. DFS {root: Node}
    1. root: Node
        1. left: Node
            1. left: Node
                1. left: null
                2. right: null
                3. value: 5
                4. [[Prototype]]: Object
            2. right: Node
                1. left: Node
                    1. left: null
                    2. right: null
                    3. value: 8
                    4. [[Prototype]]: Object
                2. right: null
                3. value: 20
                4. [[Prototype]]: Object
            3. value: 13
            4. [[Prototype]]: Object
        2. right: Node
            1. left: null
            2. right: null
            3. value: 315
            4. [[Prototype]]: Object
        3. value: 48
        4. [[Prototype]]: Object
    2. [[Prototype]]: Object
```

Roll No. – 22239

9. Script to implement Hash Table.

```
class HashTable
{
constructor(size = 7)
{
this.dataMap = new Array(size)
}
_hash(key)
{
let hash = 0
for (let i = 0; i < key.length; i++)
{
hash = (hash + key.charCodeAt(i) * 23) % this.dataMap.length
}
return hash
}
set(key,value)
{
let index = this._hash(key)
if(!this.dataMap[index]) this.dataMap[index] = []
this.dataMap[index].push([key, value])
return this
}
get(key)
{
let index = this._hash(key)
if(this.dataMap[index])
{
for(let i = 0; i < this.dataMap[index].length; i++)
{
if(this.dataMap[index][i][0] === key)
{
return this.dataMap[index][i][1]
}
}
}
return undefined
}
keys()
{
let allKeys = []
for(let i = 0; i < this.dataMap.length; i++)
{
if(this.dataMap[i])
{
```

```
for(let j = 0; j < this.dataMap[i].length; j++)
{
allKeys.push(this.dataMap[i][j][0])
}
}
}
return allKeys
}
}
let myHashTable = new HashTable()
myHashTable.set('bolts', 1200)
myHashTable.set('washers', 80)
```

Output:

```
1. HashTable {dataMap: Array(7)}
        1. dataMap: Array(7)
            1. 4: Array(2)
                1. 0: Array(2)
                    1. 0: "bolts"
                    2. 1: 1200
                    3. length: 2
                    4. [[Prototype]]: Array(0)
                 2. 1: Array(2)
                   1. 0: "washers"
                    2. 1: 80
                    3. length: 2
                    4. [[Prototype]]: Array(0)
                3. length: 2
                4. [[Prototype]]: Array(0)
          2. length: 7
          3. [[Prototype]]: Array(0)
2. [[Prototype]]: Object
```

Roll No. – 22239

10. Script to implement Linear Search

```
function linearSearch()
{
var arr = []
size= prompt("Enter the size of array :")
for(var i=0; i<size; i++)
{
arr[i] = prompt('Enter Element ' + (i+1));
}
var Key = prompt('Enter the Key:');
console.log(arr); //display array
for (var i=0; i<size; i++)
{
if (arr[i] === Key)
return alert(+Key+" present on " +i+ " index")
}
return alert(+Key+ " is not present in List")
}
linearSearch();
```

Output :

```
(6) ['7', '5', '6', '2', '0', '3']
```

Roll No. – 22239

11. Script to implement Binary Search

```
function BinarySearch()
{
var arr = []
size= prompt("Enter the size of array :")
for(var i=0; i<size; i++)
{
arr[i] = prompt('Enter Element ' + (i+1));
}
var Key = prompt('Enter the Key:');
console.log("UNSORTED ARRAY");
console.log(arr); //display array
arr.sort();
console.log("SORTED ARRAY");
console.log(arr);
let start=0, end=arr.length-1;
while (start<=end)
{
let mid=Math.floor((start + end)/2);
if (arr[mid]===Key)
return alert(+Key+" present "+ mid + " index");
else if (arr[mid] < Key)
start = mid + 1;
else
end = mid - 1;
}
return alert(+Key+" not present");
}
        BinarySearch();
```

Output :

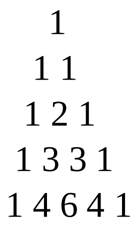
```
UNSORTED ARRAY
(5) ['8', '5', '6', '1', '2']
SORTED ARRAY
 (5) ['1', '2', '5', '6', '8']
```

Roll No. – 22239

12. Script to implement Pascal's Triangle

```
function printPascal(n)
{
   for(line = 1; line <= n; line++)
   {
      for(index=1; index<=n-line;index++)
      {
         document.write("&nbsp");
      }
   var C=1;
      for(i = 1; i <= line; i++)
      {
         document.write(" "+C+" ");
         C = C * (line - i) / i;
      }
   document.write("<br>");
   }
}
var n = 5;
printPascal(n);
```

Output :



```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

Roll No. – 22239

13. Script to implement Merge Sort

```
 function merge(array1, array2)
 {
 let combined = []
 let i = 0
 let j = 0
 while(i < array1.length && j < array2.length)
 {
 if(array1[i] < array2[j]) {
 combined.push(array1[i])
 i++
 }
 else
 {
 combined.push(array2[j])
 j++
 }
 }
 while(i < array1.length)
 {
 combined.push(array1[i])
 i++
 }
 while(j < array2.length)
 {
 combined.push(array2[j])
 j++
 }
 return combined
 }
 function mergeSort(array)
 {
 if(array.length === 1) return array
 let mid = Math.floor(array.length/2)
 let left = array.slice(0,mid)
 let right = array.slice(mid)
 return merge(mergeSort(left), mergeSort(right))
 }
mergeSort([5,3,1,6,2])
```

Output :

```
1.  (5) [1, 2, 3, 5, 6]
        1.   0: 1
        2.   1: 2
        3.   2: 3
        4.   3: 5
        5.   4: 6
        6.   length: 5
        7.  [[Prototype]]: Array(0)
```

Roll No. – 22239

14. Script to implement Quicksort

```
var items = [6,4,8,7,3,1];
function swap(items, leftIndex, rightIndex){
var temp = items[leftIndex];
items[leftIndex] = items[rightIndex];
items[rightIndex] = temp;
}
function partition(items, left, right) {
var pivot = items[Math.floor((right + left) / 2)], //middle element
i = left, //left pointer
j = right; //right pointer
while (i <= j) {
while (items[i] < pivot) {
i++;
}
while (items[j] > pivot) {
j--;
}
if (i <= j) {
swap(items, i, j); //swapping two elements
i++;
j--;
}
}
return i;
}
function quickSort(items, left, right) {
var index;
if (items.length > 1) {
index = partition(items, left, right); //index returned from partition
if (left < index - 1) {
quickSort(items, left, index - 1);
}
if (index < right) {
quickSort(items, index, right);
}
}
return items;
}
var sortedArray = quickSort(items, 0, items.length - 1);
        document.write(sortedArray);
```

Output :

```
1,3,4,6,7,8
```

Roll No. – 22239

15. Script to implement Euclidean Algorithm

```
Function gcdExtended(a, b, x, y)
{
if (a==0)
{
x = 0;
y = 0;
return b;
}
Let gcd = gcdExtended(b%a,a,x,y);
x=y-(b/a)*x;
y=x;
return gcd;
}
Let x=0;
let y = 0;
let a = 98;
let b = 56;
let g = gcdExtended(a, b, x, y);
document.write("gcd(" + a);
document.write(", " + b + ")");
document.write(" = " + g);
```

Output :

```
gcd(98, 56) = 14
```

Roll No. – 22239

16. Script to print Fibonacci series

```
function fibonacci(num) {
var answer = [];
var x = 0;
var y = 1;
var z;
answer.push(x);
answer.push(y);
var i = 2;
while (i < num) {
z = x + y;
x = y;
y = z;
answer.push(z);
i = i + 1;
}
return answer;
}
var num = 15;
answer = fibonacci(num);
console.log("The Fibonacci series is: ", answer);
```

Output :

```
The Fibonacci series is:  (15) [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```