DSA Assign 1

Q1 Write a program to find all pairs of an integer array whose sum is equal to a given number?

```
function twoSum(nums, target_num) {
  var map = [];
  var indexnum = [];


  for (var x = 0; x < nums.length; x++)
  {
  if (map[nums[x]] != null)
  {
  var index = map[nums[x]];
  indexnum[0] = index;
  indexnum[1] = x;
  break;
  }
  else
  {
  map[target_num - nums[x]] = x;
  }
  }
  return indexnum;
  }
console.log(twoSum([10,20,10,40,50,60,70],50));
```

Output - [2, 3]

Q 2 Write a program to reverse an array in place? In place means you cannot create a new array. You have to update the original array.

```
let reverseArr = arr => {
   newArr = []
   for(let i = arr.length-1 ; i>=0; i--){
      newArr.push(arr[i])
   }
   return newArr
}

console.log(reverseArr([1,2,3,4,5]));
```

Output - [5, 4, 3, 2, 1]

Q3 Write a program to check if two strings are a rotation of each other?

```
function areRotations( str1,  str2)
{
   return (str1.length == str2.length) && ((str1 + str1).indexOf(str2) != -1);
   }
   var str1 = "AACD";
   var str2 = "ACDA";
   if (areRotations(str1, str2))
      document.write("Strings are rotations of each other");
   else
      document.write("Strings are not rotations of each other");
```

Output - Strings are rotations of each other

Q 4 Write a program to print the first non-repeated character from a string?

```
function find_FirstNotRepeatedChar(str) {
  var arra1 = str.split('');
  var result = '';
  var ctr = 0;

  for (var x = 0; x < arra1.length; x++) {
    ctr = 0;

    for (var y = 0; y < arra1.length; y++)
    {
      if (arra1[x] === arra1[y]) {
        ctr+= 1;
      }
    }

    if (ctr < 2) {
      result = arra1[x];
      break;
    }
  }
  return result;
}
console.log(find_FirstNotRepeatedChar('abacddbec'));
```

Output – e

Q 5 Read about the Tower of Hanoi algorithm. Write a program to implement it.

```
// javascript recursive function to solve tower of hanoi puzzle
function towerOfHanoi(n, from_rod, to_rod, aux_rod)
{
	if (n == 1)
	{
		document.write("Move disk 1 from rod " + from_rod +
		" to rod " + to_rod+ " " + "<br/>");
		return;
	}
		towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
		document.write("Move disk " + n + " from rod " + from_rod +
		" to rod " + to_rod+ " " + "<br/>");
		towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
	}

	// Driver code
	var n = 4; // Number of disks
	towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
```

Output –

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
```

Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C

Q 6 Read about infix, prefix, and postfix expressions. Write a program to convert postfix to prefix expression.

```
// function to check if character is operator or not
function isOperator(x)
{

    switch (x) {
    case '+':
    case '-':
    case '/':
    case '*':
        return true;
    }
    return false;
}

// Convert postfix to Prefix expression
function postToPre(post_exp)
{
    let s = [];

    // length of expression
    let length = post_exp.length;
```

```
// reading from right to left
for (let i = 0; i < length; i++) {

        // check if symbol is operator
        if (isOperator(post_exp[i])) {

                // Pop two operands from stack
                let op1 = s[s.length - 1];
                s.pop();
                let op2 = s[s.length - 1];
                s.pop();

                // concat the operands and operator
                let temp = post_exp[i] + op2 + op1;

                // Push String temp back to stack
                s.push(temp);
        }

        // if symbol is an operand
        else {

                // Push the operand to the stack
                s.push(post_exp[i] + "");
        }
}

let ans = "";
```

```
        while (s.length > 0)
            ans += s.pop();
        return ans;
    }

    let post_exp = "ABC/-AK/L-*";

    // Function call
    document.write("Prefix : " + postToPre(post_exp));
```

Output - Prefix : *-A/BC-/AKL

Q 7 Write a program to convert prefix expression to infix expression

```
// Function to check if characteris operator or not
    function isOperator(x)
    {
        switch(x)
        {
            case '+':
            case '-':
            case '*':
            case '/':
                return true;
        }
        return false;
    }
    // Convert prefix to Infix expression
    function convert(str)
    {
```

```
let stack = [];

// Length of expression
let l = str.length;

// Reading from right to left
for(let i = l - 1; i >= 0; i--)
{
	let c = str[i];

	if (isOperator(c))
	{
		let op1 = stack[stack.length - 1];
		stack.pop()
		let op2 = stack[stack.length - 1];
		stack.pop()

		// Concat the operands and operator
		let temp = "(" + op1 + c + op2 + ")";
		stack.push(temp);
	}
	else
	{
		// To make character to string
		stack.push(c + "");
	}
}
return stack[stack.length - 1];
```

```
        }

let exp = "*-A/BC-/AKL";
document.write("Infix : " + convert(exp));
```

Output - Infix : ((A-(B/C))*((A/K)-L))

Q 8 Write a program to check if all the brackets are closed in a given code snippet.

```
console.clear();
class Stack {
    constructor() {
        this.items = []
        this.length = 0
        this.push = function(elements) {
            this.items.push(elements)
            this.length++
        }
        this.pop = function() {
          this.length--
          return this.items.pop()

        }
        this.peek = function() {
            return this.items[this.length - 1]
        }
        this.reverse = function() {
            var reverseArray = []
            for (var i = this.length - 1; i >= 0; i--) {
```

```
            reverseArray.push(this.items[i])
        }
        this.items = reverseArray
        return this.items
    }
  }
}
// var stack = new Stack();
// stack.push(1)
// stack.push(5)
// stack.push(3)
// stack.push(2) //push 3 element
// stack.pop() //remove or pop 1 elements
// console.log(stack.length);
// //console.log(stack.peek())

var expression = "({})"

var stack = new Stack()

for (var i = 0; i < expression.length; i++) {
  if (expression.charAt(i) == "[" || expression.charAt(i) == "(" || expression.charAt(i) == "{") {
    stack.push(expression.charAt(i))
  } else {
    if (stack.length == 0) {
      console.log("Not balanced1")
      break;
    } else {
```

```
            var lastElement = stack.pop()
            if (lastElement == "{" && expression.charAt(i) == "}" ||
                lastElement == "[" && expression.charAt(i) == "]" ||
                lastElement == "(" && expression.charAt(i) == ")") {


            } else {
                console.log("Not balanced2")
                break;
            }


        }
        if (i == expression.length - 1) {
            if (stack.length == 0) {
                console.log("Balanced")
            } else {
                console.log("Not balanced3")
            }
        }
    }
}
```

Output – Balanced

Q 9 Write a program to reverse a stack.

```
function Stack() {
  let data = [];
  let length = 0;
  return {
    push: (item) => {
```

```
      length++;
      return data.push(item);
    },
    pop: () => {
      if (length <= 0) {
        return null;
      } else {
        length--;
        return data.pop();
      }
    },
    peek: () => {
      if (length <= 0) {
        return null;
      } else {
        return data[length - 1];
      }
    },
    isEmpty: () => {
      return !length;
    },
  };
}
function reverseString(str) {
  let result = "";
  let stack = new Stack();
  let strArr = str.split("");
  strArr.forEach((element) => {
```

```
    stack.push(element);
  });
  while (!stack.isEmpty()) {
    result += stack.pop();
  }
  return result;
}
let str = "ABCDEFG";
console.log(reverseString(str));
```

Output - GFEDCBA

Q 10 Write a program to find the smallest number using a stack.

```
const numbers = [2, 4, 9, 2, 0, 16, 24];

const smallest_number = Math.min(...numbers);
const largest_number = Math.max(...numbers);

console.log('Smallest Value:', smallest_number);
console.log('Largest Value:', largest_number);
```

Output –

Smallest Value: 0

Largest Value: 24