**Calculate time complexity of following code snippets:**

function print(){

console.log("Hello World")

}

Ans: Time complexity : O(1)

function sumArray(arr) { let sum = 0;

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

}

return sum;

}

Ans: Time complexity : O(n)

function findX(arr) {

let x= []; for (let i = 0; i < arr.length; i++) { for (let j = 0; j < arr.length; j++) { if (arr[i] + arr[j] === 10) { x.push([arr[i], arr[j]]);

}

}

}

return x;

}

Ans: Time complexity : O(n)^2

Here we can see two nested loops where each loop running n no of times.

function getFirstTwoElements(arr) {

if (arr.length < 2) {

return null;

}

const first = arr[0]; const second = arr[1]; return [first, second]; }

ans: Time complexity : O(1)

FunctionprocessTwoArrays(arr1, arr2) {

let sum1 = 0; for (const item of arr1) {

sum1 += item;

}

let sum2 = 0; for (const item of arr2) {

sum2 += item;

}

return sum1 + sum2;

}

Ans:

Time Complexity: O(n + m) because here as first loop run n times and second loop runs m times .

function countF(n) { let count = 0;

for (let i = 1; i < n; i = i \* 2) { count++;

}

return count;

}

Ans: Time complexity : O(log n) because we can see i is doubling each time which is an logarithmix growth .

**Find worst, average and best cases:**

function findElement(sortedArr, target) { for (let i = 0; i < sortedArr.length; i++) {

if (sortedArr[i] === target) {

return i;

}

}

return -1;

}

Ans :

Best case :O(1) because here target is an first element .

Average case: O(n) because target is in the middle of somewhere .

Worst case : O(n) because target get lost or can be said not found .

function recursiveSum(n) {

if (n <= 0) {

return 0;

}

return n + recursiveSum(n - 1);

}

Ans: Time complexity : O(n) because here it is called n times itself .

function dFunction(arr) { const seen = {}; for (let i = 0; i < arr.length; i++) { if (seen[arr[i]]) {

return true;

}

seen[arr[i]] = true;

}

return false;

}

Ans : Best case : O(1) cause duplicate are found early .

Average case : O(n) duplicates are in the middle.

Worst case : O(n) there is no duplicates

function repeatLog(arr) {

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

let repetitions = arr[i]; for (let j = 0; j < repetitions; j++) {

console.log('hello');

}

}

}

Ans: Time complexity : O(sum of arr[1])

Because here we can see outer loop is running n times where inner loops is depending on the value of each arr[i]

**Implement a queue with the following operations:**

* enqueue
* dequeue
* search



