Calculate time complexity of following code snippets:   
  
function print(){   
console.log("Hello World")   
}

=>O(1)

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

=>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;   
}

=>O(n2)  
  
  
function getFirstTwoElements(arr) {   
if (arr.length < 2) {   
return null;   
}   
const first = arr[0];   
const second = arr[1];   
return [first, second]; }

=>0(1)

function processTwoArrays(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;   
}   
  
=>O(n+m)  
  
function countF(n) {   
let count = 0;   
for (let i = 1; i < n; i = i \* 2) {   
count++;   
}   
return count;   
}

=>O(logn)

**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;   
}   
  
**Best case:** **O(1)**

**Average case: O(n)**

**Worst case:** **O(n)**

function recursiveSum(n) {   
if (n <= 0) {   
return 0;   
}   
return n + recursiveSum(n - 1);   
}   
  
**Best = Average = Worst case:** O(n)  
  
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;   
}   
  
 **Best case:** **O(1)**

**Average case:** **O(n)**

**Worst case:** **O(n)**

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');   
}   
}   
}   
  
**Best case:** **O(n)**

**Average case:** **O(n + sum)**

**Worst case:** **O(n^2)**

Implement a queue with the following operations:   
- enqueue   
- dequeue   
- search

=>class Queue<T> {

private items: T[] = [];

enqueue(item: T): void {

this.items.push(item);

}

dequeue(): T | undefined {

if (this.isEmpty()) {

console.log("Queue is empty!");

return undefined;

}

return this.items.shift();

}

search(item: T): boolean {

return this.items.includes(item);

}

isEmpty(): boolean {

return this.items.length === 0;

}

print(): void {

console.log(this.items.join(" <- "));

}

}