a.

let n = 5 // some number

let sum = 0;

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

sum = sum + 1 // n

}

f(n) = 2n + n + 1

= 3n + 1

= O(n)

------------------------------

b.

let sum = 0;

for(let i = 1; i <= n\*n; i++) { // n2 + 1

sum = sum + 1 // n

}

f(n) = n2 + n + 1

= O(n2)

------------------------------

c.

let sum = 0;

for(let i = 1; i <= n; i++) { // n+1

sum = sum + n // n

}

f(n) = n + 1

= 2n + 1

= O(n)

d.

let sum = 0;

for(let i = 1; i <= n; i++) { // n+1

for(let j = 1; j <= i; j++) { // n\*(n+1)

sum = sum + i // n\*n

}

}

f(n) = n + 1 + n( n + 1) + n\*n

= n + 1 + n2 + n + n2

= 2n2 + 2n + 1

= O(n2)

------------------------------

e.

let sum = 0;

for(let i = 1; i <= 100; i++) { // 100+1

for(let j = 1; j <= n; j++) { // 100\*(n+1)

sum = sum + i // 100\*n

}

}

f(n) = 100 + 1 + 100( n + 1) + 100\*n

= 101 + 100n + 100 + 100n

= 2(100n) + 201

= O(n)

f.

let sum = 0;

let n = 8

for(let i = 1; i <= n; i++) { // n + 1 J C new J

for(let j = 1; j <= n; j\*=2) { // log2(n) ---> 20 x 2 = 21 = 2

sum = sum + 1 // log2(n) // ---> 21 x 2 = 22 = 4

} // ---> 22 x 2 = 23 = 8

} ---> 23 X 2 = 24 = 16 ( j > n)

2k = n

log2(n) = j

f(n) = n+1 + log2(n) + log2(n)

= O(log2n)

**PART 2:**

1. MERGE SORT

const partition = (arr) => { // 1/2 - because the passed array is simply split into 2

const middle = Math.floor(arr.length / 2)

left = arr.slice(0,middle)

right = arr.slice(middle)

return [left, right]

}

const merge = (l,r, result = []) => {

if (l.length > 0 && r.length > 0 ) {

if(l[0] <= r[0]) {

result.push(l.shift())

} else {

result.push(r.shift())

}

return merge(l,r,result)

}

return [...result, ...r, ...l] // n

}

const mergeSort = (arr) => { //O(nlog2n)+1 or O(nlog2n) - extra +1 due to conditional

checking if base case

// let 2k = n; n = 8

// k = log2n

// k = 3

// n = 23 / 2 = 8 / 2 = 4

// the array is exponentially divided into 2 until length === 1

if(arr.length <= 1) {

return arr // 1

}

const [left, right] = partition(arr) // n/2

const l = mergeSort(left)

const r = mergeSort(right)

return merge(l, r) // n

}

1. INSERTION SORT

const swap = (arr, currentIndex, toSwapIndex) => { // n

const current = arr[currentIndex];

arr[currentIndex] = arr[toSwapIndex];

arr[toSwapIndex] = current;

return arr;

};

const insertionSort = (arr) => {

for (let i = 1; i < arr.length; i++) { // n+1

let leftArr = arr.slice(0, i)

for (let j = 0; j < leftArr.length; j++) { // n(n+1)

if (arr[i] < leftArr[j]) {

arr = swap(arr, j, i) // n

}

}

}

return arr; // n

};

f(n) = n+1 + n(n+1) + n + n

= n2+4n+1

= O(n2)

1. SIEVE OF ERATOSTHENES

const createArr = (N) => { // O(n)

let results = []

for(let i = 0; i <= N; i++) { // n + 1

results.push(true)

}

return results // n

}

const trimPrimes = (arr, index, increment) => { // n(1/i) – where i is the index.

The index as pointer is incremented by the original index.

if(index > arr.length) {

return arr

}

arr[index] = false

return trimPrimes(arr, index + increment, increment)

}

const getPrime = (arr) => {

const i = Math.round(Math.sqrt(arr.length - 1))

let trimmed;

let primes = []

for (let index = 0; index < arr.length; index++) { // n + 1

if(index === i) {

trimmed = arr

break

}

if([0,1].includes(index)) {

arr[index] = false

} else {

arr = trimPrimes(arr, index, index) // n/i

}

}

trimmed.forEach((e, num) => { // n

if(e) {

primes.push(num) -

}

})

return primes

}

f(n) = n + 1 + (n/i) + n

= O(n)