If I call a function that contains a for loop inside a for loop, is that considered O(n^2) time or O(n)?

Asked 1 year, 4 months ago Modified 13 days ago Viewed 338 times

2

for loop inside subArraySum calls sumArray which also contains a for loop. Would this be considered O(n^2) time? Go easy on me as this is my first ever question and as you can see from my code, I'm a beginner. This is my code:

```
const subArraySum = (arr,n) => {
    let i = 0;
    let result = [];
    for (let j = n-1;j<arr.length;j++) {
        let li = arr.slice(i,j+1);
        result.push(sumArray(li));
        i++;
    }
    return maxResult(result);
}

function sumArray(li) {
    counter = 0;
    for (let k of li) {
        counter+=k;
    }
    return counter;
}

function maxResult(result) {
    let highest = 0;
    for (let k of result) {
        if (k>highest) {
            highest = k;
        }
    }
    return highest;
}
```

javascript performance time-complexity

Share Improve this question Follow

edited Nov 1, 2020 at 17:56
Sarun UK
4,883 6 19 42

asked Nov 1, 2020 at 17:48
PRATEEK JAIN
25 4

It's only O(n^2) if the outer loop has the same iterations as the inner. Hence 5 x 5 = 25 or 5 squared. – GetSet Nov 1, 2020 at 17:51

It always depends on how many iterations these loops actually do, but yes, in your example both loops are on average linear to the number array elements so it's quadratic. – Bergi Nov 1, 2020 at 17:56

have a look at functions, asymptotes and limits ... and then plot the function based on the inputs and the ouput (output is num. of actions needed) – Eugen Sunic Nov 1, 2020 at 17:57

n^1 = 1 iteration; n^2 = 1 iteration x 1 iteration; n^3 means 1 iteration x 1 iteration x 1 iteration ... for example n^1 is `for i ... {}` n^2 is `for i... {for i...{}}` , n^3 is `for i...{for i...{for i...{}}}` etc... – Flash Thunder Nov 1, 2020 at 18:55

3 Answers

Active Oldest Votes

1

While `for` and other looping constructs may provide some basic guides on complexity, the analysis must always consider the number of iterations and not rely on counting the number of nested loops. It can be easily shown that any number of nested loops can still result in linear or constant time complexity (i. e. `while (true) { break; }`).

For the functions you mentioned, both `sumArray` and `maxResult` run `n` iterations, where `n` matches the count of input array elements.

For `subArraySum` , however, the complexity is not as trivial. We can easily see, that the function calls both `sumArray` and `maxResult` , however, it is not exactly clear how these invocations relate to the function inputs.

To invoke `subArraySum` for some array `arr` of length `m` and an argument of `n` , we can see that the `for` loop runs from `n - 1` to `m` , that is `m - n + 1` times, or in complexity terms, the loop run count is a function of `O(m - n + 1) = O(m - n)` . The constant 1 is considered insignificant for the purpose of asymptotic analysis (imagine the difference between 10^9 and $10^9 + 1$ operations - probably not that much).

Each loop then makes a sub-array of length `n - 1` . This is based on the `i` and `j` iteration variables - I may be wrong here but I suspect the implementation is. Regardless, that makes an `O(n)` operation.

The slice is then summed via `sumArray` (complexity of `O(n)` as described previously) and inserted to the back of another array (amortized `O(1)`).

Therefore, each run of the loop has complexity of `O(n)` .

As the loop runs `O(m - n)` times and has a complexity of `O(n)` , yielding a total complexity of `O(n * (m - n)) = O(m * n - n^2)` .

Let's now consider this term, $m * n - n^2$. From the semantics of the function, we can assume, that it must always hold that $n < m$ (the complexity is constant for $n = m$: $m * n - n^2 = n^2 - n^2 = 0$ - you can confirm this by mentally going through the algorithm, it would just return 0 without any looping). If n is always smaller than m, then n^2 will always be smaller than $m * n$ and therefore can be ignored.

We arrive at `O(mn)` for the loop.

At the end, `maxResult` is invoked for the array of sums which is `O(m - n)` long (remember that we created the array by running `O(m - n)` iterations, each adding a single number to the array). This gives us `O(m - n)` complexity for the `maxResult` call.

With complexity `O(mn) + O(m - n)` , we can, again apply the $n < m$ and see that the second term always yields smaller value than the first one and therefore can be considered insignificant for the analysis.

So we arrive at the result, the algorithm has a time complexity of `O(mn)` where `m` is the length of the input array and `n` is the length of the sub-arrays.

Share Improve this answer Follow

edited Nov 1, 2020 at 18:54

answered Nov 1, 2020 at 18:47
Zdeněk Jelínek
2,386 15 22

1

Yes, this makes it O(n^2), since complexity applies to the entire algorithm, not matter how it's split up into functions. Moving some code out to a named function doesn't change the overall running time or how it depends on the input.

Sometimes when we're calling built-in functions of the language or OS we may treat them as black boxes whose complexity is unknown, so we may ignore them for the purpose of calculating the complexity of our own algorithm. But you can't usually ignore the complexity of your own functions that form part of your algorithm, since you have the option of redesigning them if there's a more efficient way.

Share Improve this answer Follow

edited Nov 1, 2020 at 17:59

answered Nov 1, 2020 at 17:55

Barmar
654k 50 444 551

since when are built ins treated as o1, are you saying that built in sort functions should be assumed as O(1)? – Eugen Sunic Nov 1, 2020 at 17:58

0

It totally Depends on what kind of approach you're using inside of each for loop.

let us take simple example:-

if you're taking an a for loop which increments linearly (increments constantly till condition works) and inside it (in body) has a calling function which also has another linear for loop in it's body.

so, the time complexity will be O(n^2). The second for loop will iterate through n for each call of the function by each iteration of the first for loop. But it totally depends on what kind of incremental approach and conditions (any) you have for both for loop.

Share Improve this answer Follow

answered Feb 28 at 18:54

Ajay Kachhela
1 1