

CHAPTER 5: Big O (Examples)

Example 1: For the given code, runtime complexity is simply going to be $O(n)$ where n represents the array's length. This is so because although we do have two for loops iterating over the same range 0 to n , these loops aren't nested and their complexities are added. This gives us $O(n+n) = O(2n) = O(n)$.

Example 2: Now, in this case, we do have two for loops, and they are nested, so we multiply their complexities together. The inner for loop works for n iterations, where n is the array's length. So, inner loop complexity = $O(n)$

The outer loop as well iterates n times, giving us $O(n)$ as complexity.

So, for complete code complexity becomes = $O(n*n) = O(n^2)$

Example 3: For this, we have multiple workarounds. We can think like, for first iteration where $i=0$, j works from 1 to $n-1$, for next j does 2 to $n-2$ and so on.

The total number of steps becomes = $(n-1) + (n-2) + \dots + 1$

$$= 1 + 2 + \dots + (n-1)$$

$$= n * (n-1) / 2$$

Total complexity becomes = $O(n^2)$

Example 4: The nested loops have an if statement working inside them which is a constant time working complexity ($O(1)$) as it doesn't change with change in input set dimensions.

The outer loop works for $O(\text{arrayA.length}) = O(a)$

The inner loop works for $O(\text{arrayB.length}) = O(b)$

So, total complexity is $O(ab)$ where a and b are lengths of arrayA and arrayB respectively.

Example 5: Here, we have three nested loops. The outer loop works at $O(a)$ complexity and first inner loop works at $O(b)$ complexity. However, the third inner loop works 100,000 times which is still a constant and doesn't contribute to complexity evaluation.

So, total complexity is $O(ab)$ where a and b are lengths of arrayA and arrayB respectively.

Example 6: The loop here iterates over the half of the input set, so ideally we would say $O(n/2)$, but total complexity is $O(n)$ because we do not consider constants.

Example 7:

- $O(N + P)$, where $P < N/2$: In this, as we are given P is less than $N/2$ in all cases, we can drop it and get complexity as $O(N)$
- $O(2N)$: On dropping constant, we get $O(N)$
- $O(N + \log N)$: Here, $O(N)$ dominates $O(\log N)$ and since we consider worst case, we get $O(N)$

- $O(N + M)$: We aren't given any established relationship between N and M , we keep both terms. This will only become $O(N)$ when, $M = N$.

Example 8: Runtime complexity for an algorithm that takes in an array of strings, sorted each string, and then sorted the full array.

Let the array size = n

And let maximum string size = m

Complexity for sorting each individual string = $O(m \log m)$

Complexity for sorting all array elements = $O(n * m \log m)$

Sorting full array of sorted strings = $O(m * n \log n)$

Total = $O(n * m \log m + m * n \log n) = O(n * m (\log m + \log n))$

Example 9: