QUIZ

Select the correct ordering of runtime complexity, from most efficient to least efficient.

`O(logN), O(1), O(N), O(N^2), O(2^N), O(N!)`

`O(1), O(logN), O(N), O(N^2), O(N!), O(2^N)`

`O(1), O(logN), O(N), O(2^N), O(N^2), O(N!)`

`O(1), O(logN), O(N), O(N^2), O(2^N), O(N!)`

Correct! From left to right: Constant, Logarithmic, Linear, Quadratic, Exponential, Factorial.

Suppose we had a function that performed one step for each element in a collection of input data. What would the Big O time complexity be for this function?

Constant: `O(1)`

Logarithmic: `O(logN)`

Linear: `O(N)`

Correct! If the collection contained 10 elements, the function performs 10 steps, 100 elements would be 100 steps, and so on.

Quadratic: `O(N^2)`

Suppose we had a function that took in a list of data as an input. For each element in the list, the function compared it to **all** of the other elements in the collection. What would the Big O time complexity be for this function?

Linear: `O(N)`

Constant: `O(1)`

Quadratic: `O(N^2)`

👏 Correct! Imagine a list of 5 elements. For each of the 5 elements, it would need to be compared to 4 other elements. `N = 5` and for each element we're comparing to `N - 1` times!

Exponential: `O(2^N)`

Why don't we use time-elapsed (5 minutes, for example) as a measurement for an algorithm's runtime?

Time is a social construct.

Different units of measurement make this infeasible.

Time-elapsed is not relevant in quantum computing.

Algorithms perform under many different circumstances (could be run on various CPUs, different programming languages, etc.), and time-elapsed is not coarse enough to communicate performance across those circumstances.

👏 Correct! Big O notation allows us to communicate the efficiency of an algorithm without worrying about the many different circumstances in which it will be run.

With Big O notation, we evaluate a function's runtime efficiency based on the _____?

Helper function(s).

Number of conditionals.

Size of input argument(s).

👏 Correct! Big O is meant to give a worst case runtime of a function based on its input.

Number of variables.

Suppose we had a function that iterated through a collection of data once to find the minimum value, and then again in a separate iteration to find the maximum value. How would we write the Big O time complexity?

`O(N^2)`

`O(N)`

👏 Correct! Even though we are looping twice, these iterations are not nested. If the input is `N`, we can leave off the `2` which indicates the number of repeated iterations since that would be a constant facto...

Show more

Why are constant factors (such as `2N`) and lower order terms (`N^2 + N` becomes `N^2`) ignored when writing Big O notation?

As the input to the function grows towards infinity, lower order terms and constant factors become insignificant compared to the term with the highest order of magnitude.

👏 Correct! Imagine a runtime complexity of `N + 10`. When `N` is equal to `5`, the constant factor appears significant, but with Big O we're considering inputs (`N`) that approach infinity!

We do incorporate lower order terms and constant factors into Big O notation!

As the input to the function grows towards infinity, lower order terms and constant factors cancel each other out.

We cannot be sure of the significance of lower order terms as they relate to the constant factors.

A function which only performed the operation `6 + 9` would be which Big O time complexity?

Logarithmic: `O(logN)`

Factorial: `O(N!)`

Constant: `O(1)`

Correct! This function would be performing a mathematical operation independent of the input, which makes it `O(1)`.

Linear: `O(N)`

An identical algorithm written in two different programming languages will have the same Big O runtime complexity.

True.

Correct! Big O is meant to be coarse enough to communicate the efficiency of an algorithm across programming language boundaries.

False.