Sunday 8/22/2020

Saturday, August 22, 2020 12:12 PM

Study Notes

PreReq Asymptotic Complexity:

O notation expresses an asymptotic upper bound

Ω notation is for asymptotic lower bound

Θ notation is for asymptotic tight bound

1. $3n^2 = O(n^2)$
2. $3n^2 = \Omega(n^2)$
3. $3n^2 = \Theta(n^2)$
4. $3n^2 \neq O(n \lg n)$
5. $3n^2 = O(n^3)$
6. $3n^2 \neq \Omega(n^3)$
7. $\lg n = O(n^{0.1})$

Intro to Algorithms Book

Page 22: Sorting Examples problem

Chapter 3: Aymptotic Notation, Functions and Running Times, pg 44

Worst case running time of insertion sort if $T(n) = \Theta(n^2)$

For any two functions f(n) and g(n), we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Example, a search function

```
For x in len(arrray):
        If x == targetvalue:
                Return Print("Found it")
```

The size of the array is n. The max number of times the for loop can run is n, and the worst cas
being in the array.

Each time the for-loop iterates, it has to do several things:

- compare guess with array.length
- compare array[guess] with targetValue
- possibly return the value of guess
- increment guess.

Each of these little computations takes a constant amount of time each time it executes. If the
then the time for all n iterations is c1·n where c1 is the sum of the times for the computation
Now, we cannot say here what the value of c1 is, because it depends on the speed of the comp
language used, the compiler or interpreter that translates the source program into runnable co

This code has a little bit of extra overhead, for setting up the for-loop (including initializing gue
returning -1 at the end. Let's call the time for this overhead c2, which is also a constant. Theref
linear search in the worst case is c1·n+c2.

We suddenly change our minds about what to call the constants, c, to calling them k instead w

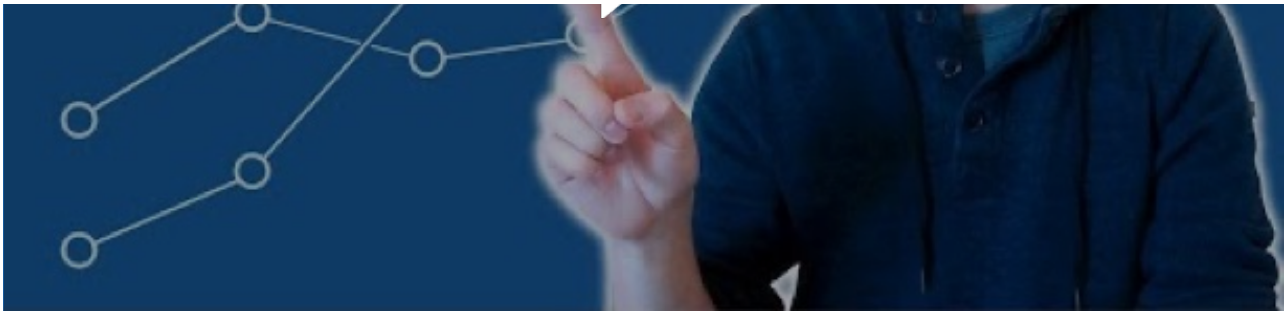
Intro to Big O Notation and Time Complexity
[Introduction to Big O Notation and Time Complexity (Data Structures & Algorithms #7)]

e scenario is the value not

for-loop iterates n times,
s in one loop iteration.
puter, the programming
ode, and other factors.

ss to 0) and possibly
fore, the total time for

hen we refer to Θ



To find the Big O for a problem,
If your problem is Linear, O(n)
If constant, O(1)
If quadratic, O(n^2)

When you're finding the tie complexity, you want to take the highest variable in your problem,
coefficient.
For example,

5n^2 + 3n + 1
T = 5n^2 = n^2 = O(n^2)

If you have no variables, for example, if your problem is only a constant, c, then you can look a

c = 0.115
T = c = 0.115 = 0.115 * 1 = O(1)

If you're finding the time complexity of a function, it's important to take into account the time
step of the function. For example,

```
def function(given_array):
        total = 0
        return total
```

The first step, total = 0, does not rely on any variables, since it will be the same every time. The
The second step, return total, performs the same action every time without relying on any cha
therefore, O(1)

, and remove the

t it like this

it takes to perform each

erefore, O(1)
anging variables as well,

The time complexity will be
T = O(1) + O(1) = c1 + c2

You can call c1 + c2 a new variable of itself, c3
c1 + c2 = c3 = c3 * 1 = O(1)

This proves that any function in which each step is 0(1) will ultimately have a time complexity c

New example

```
def find_sum(given_array):
        total=0
        for i in given_array:
                total += i
        return total
```

If you assume that given_array will always be a reasonable length, then each step here is also C
*I don't understand why you would assume that, cuz it seems like this relies on a variable, but
*Never mind, he went back and said it's actually n*O(1)
*Ok ok I get it now. He's saying the addition of a number to the variable total takes the same a
repeated n times

All of em are O(1) except for the line total += i, because it relies on the length of the given_arra
n*O(1)

c1 n c2 c3
T = O(1) + n*O(1) + O(1) = c4 + n * c5 = O(n)

New example, say you're finding the time complexity for a for loop that goes into a 2d array

```
def find_sum_2d(array_2d):
        total = 0
        for row in array_2d:
                for i in row:
                        total += i
        return total
```

This is similar to the previous problem in that each line takes O(1) amount of time, but whatle

of O(1)

O(1)
whatev

mount of time. It's just

ay, so it will be denoted as

different is this is a nested

This is similar to the previous problem in that each line takes O(1) amount of time, but what's d
for loop.
Since we know that each for loop is depending on a variable, array_2d and row, and we don't k
will be n

T = O(1) + n^2 * O(1) + O(1)
T = c3 + n^2 * c4
T = n^2
T = O(n^2)

different is this is a nested

know what those are, they