EECE.3220: Data Structures

Homework 1 Solution

1. *(25 points) Assume each expression listed below represents the execution time of a program. Express the order of magnitude for each time using big O notation.*

Solutions: In each case, the fastest growing term (which determines order of magnitude) is in **bold**.

a. $T(n) = \boldsymbol{n^3} + 100n$

2. *(75 points) For each of the code segments below, determine an equation for the worst-case computing time T(n) (expressed as a function of n, i.e. 2n + 4) and the order of magnitude (expressed using big O notation, i.e. O(n)).*

Solutions: In each case, the number of times each line is executed is written to the right in red. Also, for simplicity's sake, a for loop is treated as a single statement, despite the fact that a for loop is really a collection of three statements. *If you analyzed each for loop as a set of three statements, we'll account for that when grading your submissions.*

a.

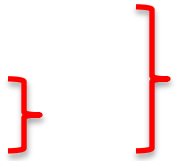
T(n) = 1 + 1 + 1 + (n+1) + n + n + n + 1 = **4n + 5 = O(n)**

Note: While the value of x controls the number of loop iterations, n counts the number of iterations, as it's incremented in every loop iteration. You can therefore express the execution time as a function of n.

b.

T(n) = (n + 1) + n * ((n + 1) + n) = **$2n^2 + 2n + 1 = O(n^2)$**

c.



$T(n) = (n + 1) + n * [\ (n + 1) + n + n * ((n + 1) + n)\ ]$
$= (n + 1) + n * [\ (n + 1) + n + 2n^2 + n\ ]$
$= (n + 1) + 2n^3 + 3n^2 + n =$ **$2n^3 + 3n^2 + 2n + 1 = O(n^3)$**

d.

$T(n) = n + n^2 + (n^2 - n) + n*(n-1) / 2 =$ **$2.5n^2 - 0.5n = O(n^2)$**

Note: The worst case for a bubble sort is that the array is initially sorted from largest to smallest value. So, the body of the if statement executes (n-1) times the first time through the inner loop, (n-2) times the second time, and so on, executing just 1 time in the last inner loop iteration. As shown in our discussion of selection sort, that sum is equal to n*(n-1)/2.

e.

$T(n) = (\log_2 n + 2) + (\log_2 n + 1) =$ **$2 \log_2 n + 3 = O(\log_2 n)$**

Note: The analysis here is similar to the analysis of binary search, in which the loop test executes $(\log_2 n + 2)$ times. A few examples will show this analysis to be true—for example, say $n = 8 = 2^3$. It takes 4 (which is $\log_2 n + 1$) iterations for n /= 2 to produce the value 0, and the loop condition must therefore be tested a 5^{th} time for the loop to end.

f. (extra credit—5 points)

$T(n) = 1 + n + (2^{n-1} + n - 2) + (2^{n-1} - 1) + (n - 1)$
$= 2*2^{n-1} + 3n - 3 = \mathbf{2^n + 3n - 3 = O(2^n)}$

Note: To derive the formula for T(n) shown above, I went through the following analysis:

The number of inner loop iterations is based on the value of x—the for loop condition is always tested (x + 1) times, and the body of the loop executes x times. x doubles every time you go through the outer loop, and the body of that loop executes n-1 times. So, the last time you execute the inner loop, $x = 2^{n-2}$, then x is doubled one last time to 2^{n-1}.

After about 10 minutes of Google searching (I wish that was a joke), I was able to find the following formula that helped me determine a simple value for the sum $1 + 2 + 4 + \ldots + 2^{n-2}$:

$$\sum_{k=0}^{N-1} r^k = \frac{1 \quad r^N}{1 \quad r}$$

Therefore, the body of the inner loop executes $2^n - 1$ times, as shown by evaluating that formula for r = 2 and N = (n-1):

$$\sum_{k=0}^{n-2} 2^k = \frac{1 \quad 2^{n-1}}{1 \quad 2} = 2^{n-1} \quad 1$$

The inner loop condition is tested 1 more time than the loop body executes. The number of terms in the sum $1 + 2 + 4 + \ldots + 2^{n-2}$ is n-1. So, the third line executes $2^{n-1} - 1 + (n-1) = 2^{n-1} + n - 2$ times.