HW 2 Written Part

Anthony Garcia

September 24, 2019

1. Explain the number of additions to the total (not Big-O) in terms of n for the following program segment:

```
int total = 0;
for(int i = 0; i < n; i += 2)
    total += i;
```

Answer:

i	n	# additions
0	1	1
0	2	1
0,2	3	2
0,2	4	2
.	.	.
.	.	.
.	.	.
0,2,4,..,n-2	n	$\lceil n/2 \rceil$

Therefore

$$f(n) = \begin{cases} \text{n/2}, & \text{if n = 2a where } \ a \in \mathbb{N} \\ \dfrac{n+1}{2}, & \text{if n} \neq \text{2a} \end{cases}$$

2. Explain the number of additions to the total (not Big-O) in terms of n for the following program segment:

```
int total = 0;
for (int i = 0; i < n; i++)
    for (int j = i; j >= 0; j--)
        total += i * j;
```

Answer:

i	j	n	# additions
0	0	1	1
1	1,0	2	2
2	2,1,0	3	3
3	3,2,1,0	4	4
.	.	.	.
.	.	.	.
.	.	.	.
n-1	n-1,n-2,..,1,0	n	n

Therefore

$$f(n) = 1 + 2 + 3 + 4 + ... + n = \frac{n(n+1)}{2}$$

3. Mathematically show that if d(n) is O(f(n)) and f(n) is O(g(n)), d(n) is O(g(n)).

Answer:

Want: if $d(n) \leq k_1 f(n)$ and $f(n) \leq k_2 g(n)$ then $d(n) \leq k_3 g(n)$

d(n) is O(f(n)) means:

$$d(n) \leq k_1 f(n),\ for\ some\ n_0 \leq n$$

f(n) is O(g(n)) means:

$$f(n) \leq k_2 g(n),\ for\ some\ n_0 \leq n$$

then if we multiply k_1 by the second inequality:

$$k_1 f(n) \leq k_1 k_2 g(n)$$

consequently:

$$d(n) \leq k_1 f(n) \leq k_1 k_2 g(n)$$

if we choose $k_3 = k_1 k_2$ then:

$$d(n) \leq k_3 g(n)$$

which means d(n) is O(g(n)).

4. Consider $f(n) = 5n^2+4n-2$, mathematically show that f(n) is O(n^2), $\Omega(n^2)$, and $\Theta(n^2)$.

Answer:

$$f(n)\ is\ O(n^2)$$

$$5n^2 + 4n - 2 \leq k_1 n^2,\ for\ some\ n_{01} \leq n$$

$$5 + \frac{4}{n} - \frac{2}{n^2} \leq k_1,\ let\ n \to \infty$$

$$5 \leq k_1$$

this means that k_1 needs to be bigger or equal than 5

$$5n^2 + 4n - 2 \leq k_1 n^2$$

$$0 \leq k_1 n^2 - (5n^2 + 4n - 2)$$

$$0 \leq (k_1 - 5)n^2 - 4n + 2,\ choose\ k_1 = 7$$

$$0 \leq 2n^2 - 4n + 2$$

$$0 \leq n^2 - 2n + 1$$

$$0 \leq (n-1)^2$$

$$1 \leq n$$

therefore f(n) is O(n^2) for $k_1 = 7$ and $n_{01} = 1$.

$$f(n)\ is\ \Omega(n^2)$$

$$5n^2 + 4n - 2 \geq k_2 n^2,\ for\ some\ n_{02} \leq n$$

$$5 + \frac{4}{n} - \frac{2}{n^2} \geq k_2,\ let\ n \to \infty$$

$$5 \geq k_2$$

this means that k_2 needs to be less or equal to 5

$$5n^2 + 4n - 2 \geq k_2 n^2$$

$$0 \geq 2 - 4n + (k_2 - 5)n^2,\ choose\ k_2 = 5$$

$$0 \geq 2 - 4n$$

$$0 \geq 1/2 - n$$

$$n \geq 1/2$$

Since $n \geq 1/2$ then choose n_{02} be equal to 1 since $1 \geq 1/2$. Therefore f(n) is $\Omega(n^2)$ for $k_2 = 5$ and $n_{02} = 1$.

$$f(n)\ is\ \Theta(n^2)$$

$$k_2n^2 \leq 5n^2 + 4n - 2 \leq k_1n^2,\ for\ some\ n_0 \leq n$$

We already have k_1 from O(n^2) and k_2 from $\Omega(n^2)$. Now choose $n_0 = max\{n_{01}, n_{02}\} = max\{1, 1\} = 1$.
Consequently f(n) is $\Theta(n^2)$ for $k_1 = 7, k_2 = 5$ and $n_0 = 1$

5. For finding an item in a sorted array, consider "ternary search," which is similar to binary search. It compares array elements at two locations and eliminates 2/3 of the array. To analyze the number of comparisons, the recurrence equations are $T(n) = 2 + T(n/3), T(2) = 2$, and $T(1) = 1$, where n is the size of the array. Explain why the equations characterize "tertiary search" and solve for T(n).

Answer:

$$T(n) = 2 + T(\frac{n}{3})$$

$$T(n) = 2 + (2 + T(\frac{n}{3^2}))$$

$$T(n) = 2 + (2 + (2 + T(\frac{n}{3^3})))$$

$$T(n) = 2k + T(\frac{n}{3^k}),\ 3^k \leq n$$

Then:

$$3^k \leq n$$

$$k \leq log_3 n$$

Consequently:

$$T(n) = \lfloor 2log_3 n + T(\frac{n}{3^{log_3 n}}) \rfloor$$

$$T(n) = \lfloor 2log_3 n + T(\frac{n}{n}) \rfloor$$

$$T(n) = \lfloor 2log_3 n + T(1) \rfloor$$

$$T(n) = \lfloor 2log_3 n + 1 \rfloor$$

Since $T(n) = 2 + T(n/3)$ it means that we choose two middle point every time and then we keep 1/3 of the array. Consequently 2/3 of the array will be eliminated. Which means that is a "ternary search".

6. To analyze the time complexity of the "brute-force" algorithm in the programming part of this assignment, we would like to count the number of all possible strings.
(a) Explain the number of all possible strings in terms of n (maximum length of a string).
(b) Consider a computer that can process 1 billion strings per second and n is 100, explain the number of years needed to process all possible strings.
(c) If we don't want the computer to spend more than 1 minute, explain the largest n the computer can process.

Answer:

a) Since n is the maximum length of a string then the length of the string can be from 1 characters to n characters. The alphabet has 26 letters, therefore the number of possible strings of size n will be 26^n, but the sizes of the strings will be from 1 to n. Consequently the amount of possible strings will be

$$\sum_{i=1}^{n} 26^i = 26\frac{1-26^n}{1-26} = 26\frac{26^n-1}{25} = \frac{26^{n+1}-26}{25}$$

b)

$$\sum_{i=1}^{100} 26^i = 26\frac{1-26^{100}}{1-26} = 26\frac{26^{100}-1}{25} = \frac{26^{101}-26}{25} = 3.268647867 * 10^{141}$$

So if $n = 100$ then the amount of strings are $3.268647867 * 10^{141}$. Then the amount of seconds the program will takes are:

$$\frac{3.268647867 * 10^{141}}{10^{-9}} = 3.268647867 * 10^{132}$$

We know that in a year we have $3.154 * 10^7$ seconds. Therefore the amount of years the program will take are:

$$\frac{3.268647867 * 10^{132}}{3.154 * 10^7} = 1.03634999 * 10^{125} \cong 10^{125}$$

c)

$$\frac{1}{10^9}\sum_{i=1}^{n} 26^i \leq 60 \Rightarrow \frac{1}{60 * 10^9}\sum_{i=1}^{n} 26^i \leq 1$$

$$\frac{26}{60 * 10^9}\frac{26^n-1}{25} \leq 1$$

$$\frac{26^n-1}{25} \leq \frac{60 * 10^9}{26}$$

$$26^n \leq \frac{25 * 60 * 10^9}{26} + 1$$

$$n \leq log_{26}(\frac{25 * 60 * 10^9}{26} + 1)$$

$$n \leq 7.60517357201$$

Therefore the largest value of n so it can be processed in no more than 1 minute is $n = 7$, because $n \in \mathbb{N}$