HW 2 - Skyler Benjamin

Tuesday, September 14, 2021 5:37 PM

Collaborated with: Clark, Connor, Jack, Andrew

1. $T(n) = 3(n+5)^2 + n$
   a. WTS: $T(n)$ is $\Theta(N^2)$
   b. def: WTS there exists c_1 , c_2 , $n_0 > 0$ such that
      i. $C_1n^2 <= 3(n+5)^2 + n <= c_2n^2$ for all $n >= n_0$
   c. Work:
      i. $C_1\cancel{n^2} <= (3(n+5)^2 + n) / n^2 <= c_2\cancel{n^2}$
      ii. $(3(n+5)^2 + n) / n^2$
      iii. $(3(n+5)(n+5) + n) / n^2$
      iv. $(3(n^2 + 10n + 25) + n) / n^2$
      v. $(3n^2 + 31n + 75) / n^2$
      vi. $\underline{3 + (31/n) + (75/n^2)}$

N	$3 + (31/n) + (75/n^2)$
1	109
2	37.5
3	21.66
4	15.43
…	3

   d. $C_1 <= 3 + (31/n) + (75/n^2) <= c2$ for all $n >= n_0$
      i. Where $c_1 = 3$, $c_2 = 109$, $n_0 = 1$
      ii. 1 is the lowest $n_0 > 0$, where this function reaches its peak at 109, so 109 is an upperbound
      iii. By inspection it is clear that as n increases, T(n) will decrease as it asymptotically approaches 3. It cannot go lower than 3 because 3 is a constant in T(n), so no matter how low the two second terms go, the function will always be >= 3 for n >= 1.

2. $T(n) = lg(n) + n$
   a. WTS: $T(n)$ is $\Theta(n)$
   b. Def: WTS there exists c_1 , c_2 , $n_0 > 0$ such that
      i. $C_1n <= lg(n) + n <= c_2n$ for all $n >= n_0$
   c. Work:
      i. $C_1\cancel{n} <= (lg(n) + n) / n <= c_2\cancel{n}$
      ii. $C_1 <= (Lg(n) / n) + 1 <= c_2$

n	$(Lg(n) / n) + 1$
1	1
2	1.5
3	1.528
4	1.5
5	1.46
6	1.43
7	1.40
…	1

   d. $C_1 <= (Lg(n) / n) + 1 <= c_2$ for all $n >= n_0$
      i. Where $c_1 = 1$, $c_2 = 1.53$, $n_0 = 3$
      ii. 1 is the lower bound because of the constant 1 which prevent the function value from ever dipping below 1.
      iii. 1.53 is the upper bound because there is a small tick up over 1.5 around n=3, but after this point the value only decreases. Since the smallest of $n_0 > 0$ appears to be an anomaly from the rest of the function, $n_0 = 3$ so we only look at the most relevant parts of the function.

3. Suppose that $T_1(n)$ is O(f(n)) and $T_2(n)$ is also O(f(n)) for some function f(n). Is it true that $T_1(n)$ is $O(T_2(n))$?
   a. It is NOT always true that a $T_1(n)$ which is O(f(n)) will be $O(T_2(n))$ if $T_2(n)$ is also O(f(n)) because the function f(n) which upper bounds both $T_1(n)$ and $T_2(n)$ does not have to be a "tight bound". As such, the order of both functions is not necessary captured by only a one-sided bound, in this case the upper bound represented by O(f(n)) only really captures the order of the higher order T(n), and the other is just ~ somewhere below. Because of this defining characteristic of an upper bound, many counter examples could be presented to disprove this claim, one of which is as follows:

      i. $T_1(n) = n^3$, $T_2(n) = n$, $f(n) = n^3$

   b. In the above case, both $T_1(n^3)$ and $T_2(n)$ are $O(n^3)$ with many constants, c that could be found to apply to f(n) to upper bound both functions. However, it is not true that $T_1(n^3)$ is $O(T_1(n))$, so the original claim is false.

4. Suppose that $T_1(n)$ is Θ(f(n)) and $T_2(n)$ is also Θ(f(n)) for some function f(n). Is it true that $T_1(n)$ is $\Theta(T_2(n))$?
   a. Assuming $T_1(n)$ and $T_2(n)$ are both Θ(f(n)) then the follows holds true
      i. $C_1f(n) <= T_1(n) <= c_2f(n)$
      ii. $C_3f(n) <= T_2(n) <= c_4f(n)$
   b. Since both $T_1(n)$ and $T_2(n)$ are upper and lower bounded by a single function fn which is only modified via a constant per the definition of Θ, they must be of the same degree (both n^2 for example) because as n approaches infinity no constant would modify f(n) enough to be a proper bounding. For example if $T_1(n)$ was $\Theta(n^2)$ then approaching infinity it would cross any Θ(n) or $\Theta(n^3)$. If $T_1(n)$ and $T_2(n)$ are the same order, then they must be Θ of each other because a constant could always be found for a function to bound on either side of its own "pure form" ($\underline{n^2}$ vs. $3n^2$)

5.

   A.
   a. T(n) = n(n(n + s))
      i. S is a constant equal to the number of ops needed to store into array B
   b. WTS: T(n) is $O(n^3)$
   c. Def: WTS there exists c , $n_0 > 0$ such that:
      i. $n(n(n + s) <= cn^3$
      ii. $n(n^2 + ns)$
      iii. $N^3 + sn^2 <= c\cancel{n^3}$
      iv. $(\cancel{n^3 / n^3}) + (s\cancel{n^2 / n^3})$
      v. (s / n) <= c
   d. Since n is in the denominator, it is clear that as n approaches infinity (s/n) will approach 0 and the numerator is ultimately insignificant. Assuming this insignificance holds S could be replaced by any number, such as 1. In this case, that constant number would be the upper bound reached when n = 1. So c would be 1 and n_0 would also be 1.

C.

a. The current function operates under a wasteful paradigm that iterates the list a third time to find the sum, when the sum could be taken along the way. My redesigned paradigm for this problem is to simply take the prior entry in B and add the new j element from A to it, forming the sum so far for whatever A[i] through A[j] is at that time. This paradigm works so effectively specifically because the sum stored in B[i,j] is the sum from A[i] - A[j] and not the entire array A, so you only really need the prior sum to find the new sum.

```
For(i=1; i<=n; i++){
        For(j=i+1; j<=n; j++)
                B[i][j] = B[i][j-1] + A[j]
```

The data structures in this are an array (n) and a two-dimensional array ($n_x n$), and the two dimensional array is only ever accessed via a constant time operation so the functions main bounding comes from the nested looping used to aggregate the sums of array A. By changing the function to use a singly nested for loop instead of a doubly nested for loop I eliminated wasted operations and shaved the time complexity of this function from n^3 to n^2.