**COSC 3304 – Algorithms Design and Analysis**

**Assignment 2**

**Due: 23:59:00pm, Jan. 30, 2024 (Tuesday)**

1. If *f(n)=2n2+n+10* and *g(n)=n2*, please find *no* to make

*c1g(n) ≤ f(n) ≤ c2g(n)*)

when *c1*=1 and *c2*=3

(30 points)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | c1(g(n) | fn() | c2(g(n)) | Correct? |
| 2 | 4 | 20 | 12 | X, 20>12 |
| 3 | 9 | 31 | 27 | X, 31>27 |
| 4 | 16 | 46 | 48 | ✔️ |
| 5 | 25 | 65 | 75 | ✔️ |
| 6 | 36 | 88 | 108 | ✔️ |

Since all the other n’s following n=4 correctly followed the lower and upper bounds, we can say that n­0 = 4.

Alternatively (for a more mathematical definition), we can set the inequality as such:

* n2 <= 2n2 + n + 10 <= 3n2
* If we are to solve for n0, let us say n0 = k, such that
* k2 <= 2k2 + k + 10 <= 3k2
* To prove this is n0, this means that all k2 values will be below and all 3k2 values will be above
* We can check this as setting n = k+1:
* (k+1)2 <= 2(k+1)2 + (k+1) + 10 <= 3(k+1)2 ==>
* k2 + 2k + 1 <= 2k2 + 5k + 13 <= 3k2 + 6k + 3
* We can also expand this out for n = k+2, but that can be done separately.

Solving into this equation for both k and k+1, we will see that k must be 4 (so k+1 =5).

1. Please prove if *f*(*n*) = Θ(*g*(*n*)) and *g*(*n*) = Θ(*h*(*n*)),

then *f*(*n*) = Θ(*h*(*n*)) (35 points)

1. This can be explained by transitivity.
2. Expanding this out (and using the literal language), we can deconstruct this logically.
   1. Expansion: f(n) = Θ(g(n)) => f(n) = Θ(Θ(h(n)))
   2. Since Θ is denoted as the average case, the enclosed Θ do not hold any value as the average of an average is still the same.
   3. Eliminating: f(n) = Θ(Θ(h(n))) => *f*(*n*) = Θ(*h*(*n*))
3. What is the big O notation of the following program by assuming n=2k

(please show detailed steps for full credits):

for (int i=1; i<=n; i\*=2)

for (int j=n; j>=1; j--)

{//other constant time statements ;}

(35 points)

1. Plugging in 2k for n, we get the outer loop to run k times
   1. This is because if we start of with i=1, it will run again for i=2, then i=4, i=8, … till k times (the last run will be k due to the equal portion).
2. The inner loop will need to reach every element, so replacing n for 2k, we have to run 2k steps for the inner loop
3. Combining the loops, we get T(2k)=k2k
4. Replacing 2k with n, we get T(n) = O(nlogn)
5. Alternatively
6. The outer loop will be O(log(n)) due to the i\*2, which will go cause the search to go up logarithmically.
7. The inner loop will be O(n) due to j--, which will go through each element (hence relying on the number of elements, n).
8. Taking both the inner loop and outer loop into account, the expected big O would be:
   1. O(log(n)) \* O(n) = O(nlogn)