**CSC 332 – Advance Data Structures and Algorithms**

**Midterm Exam (100 pts)**

Assigned: 02/19/2019

Due Date: 02/26/2019

***(Attention: In your solution, do not change the original layout or delete any contents in this file.)***

* **Only one e-submission per group, named “CSC-332-Midterm-Solution-Group-AaaBbbCcc.doc(x)”**

**(Aaa, Bbb, and Ccc stand for the last names of group members and are sorted in an ascending order).**

* **Do not forget to fill in corresponding information in the Header section.**
* **! IMPORTANT! Everything in your solution needs to be typed; otherwise your solution will NOT be graded.**

[Group member contribution]

Robert Allen: 100% (Robert Allen)

William Blackwell: 100% (William Blackwell)

Bradley Sutton: 100% (Bradley Sutton)

Grades:

|  |  |
| --- | --- |
| 1. | 2. |
| 3. | Total: |

1. (11 pts each, 55 pts in total) For each of the following asymptotic notations, give a formal definition (1 pt) and then present one example (3 pts) followed by a strict proof (7 pts).
   1. Big Oh

**Formal Definition**: Let f(n) and g(n) be two functions on positive integers. We say f(n) is O(g(n)) if there exist two positive constants c and k such that f(n) <= cg(n) for all n >= k.

**Example**: let f(n) = 10n + 5 and g(n) = n

We can choose any number for c and k as long as they are positive so I choose c to be 15

Then I need to show that 10n + 5 < 15n

Solve for n: 1 <= n

So f(n) = 10 + 5 <= 15g(n) for all n >= 1 (c=15, k = 1)

This proves f(n) is O(g(n)

* 1. Little Oh

***Formal Definition*** (*c and n0 are fixed positive numbers*):

∀

*Definition* *2*:

***Example***: Unlike Big Oh, o describes an algorithm that is *not* asymptotically “tight”. At some point the growth of g(x) dwarfs f(x) to such an extent that f(x) becomes insignificant. Thus, the only “real” difference (it is a major one) is that Big allows f(x) and g(x) to grow at the same rate.

For the function , its

***Proof by Definition***: using Def 2, where f(x) = 2x + 1 and g(x) = x2

Because the limit is 0, we have demonstrated

* 1. Big Omega

***Formal Definition*** (*c and n0 are fixed positive numbers*):

***Example***: is a description of a functions asymptotic lower bound, i.e. the best-case running time for an algorithm.

For the function , its best case is

***Proof by Definition***:

We will show for some constant c, and

First divide each side of the inequality by :

=

Simplify

For and c = 1, holds true:

=

d) Little Omega

***Formal Definition***:

***Example***: is the inverse to Little Oh, and like Little Oh, it complements Big Omega in that it describes a lower bound that is not asymptotically tight.

For the function , its

***Proof by Definition***: is

Because the limit is , we have shown

e) Big Theta

***Formal Definition*** (*c1, c2 and n0 are fixed positive numbers*):

***Second Definition***:

***Example***: Describes the worst-case scenario for an algorithm. For the function , its worst case is .

***Proof by Definition***: is

We know the above to be true by using the second definition and our results from finding O and for f(x).

Since we’ve shown that for and we can see that for

& g(x)

, where c ≥ )/x

For n0 = x = 1, c = 4 we see that is true.

So, O for f(x) is O()

So holds true and .

We can also prove it by the first definition:

Find constants *c1, c2* using f(x) and g(x), where g(x) = x2

Simplify by dividing by across the inequality.

Simplify

For n0 = x = 1, c2 = 4 and c1 = 1 we see that our definition holds.

1. (15 pts each, 30 pts in total) Use the Master Theorem to solve the following two recurrence equations. DETAILED steps and proof are required.

a = 8

b = 2

f(n) =

nlog28 = n3

n3 > 3n2. f(n) is asymptotically larger than nlogba so case 1 applies.

**Case 1:**

If f(n) is O() for some constant E > 0, then T(n) = ()

Let E = 1

3 = O()

f(x) <= c \* g(x)

3 <= c \*

3 \* x <= c \* x

Let c = 4

3 \* x <= 4 \* x

When True x >= 0, therefore n0 = 0

**Let us now investigate Big Oh:**

Let c = 4, then n0 = 0

When x >= 0, we have 3 \* x <= 4 \* x

3 \* x <= c \* x

3<= c

So 3<= c

According to Big O notation we know that 3 = O()

**So now master theorem:**

T(n) = ()

T(n) = ()

T(n) = ()

**Answer:**

a=3

b=3

f(n) = 2n

nlog33 = n

Since 2n is polynomially similar to n then we have to look into case 2 of master theorem.

**Case 2 States**: if f(n) is () then T(n) = (f(n) \* log(a))

So **Prove**: f(n) = O()

Investigate Big Oh notation:

2x = O(x)

2x <= c1 \* x

2 <= c1

Let c1 = 3

2 <= 3

This is always true, now let n0 = 1

Let c1=3, then n0 = 1

When x >= n0 we have 2x <= 3x

So we have 2x <= c1 \* x

According to Big Oh notation we know that 2x = O(x)

Now we have to investigate Big Omega in order to prove big theta

**Big Omega**: **Prove** f(n) = ()

2x = (x)

2x >= c \* x

2 >= c

Let c = 1

2 >= 1

This is always true, now let n0 = 1

Let c = 1, n0 = 1

When x >= n0 we have 2x >= 1 \* x

That is 2x >= c \* x

According to Big Omega notation we know that 2x = (x)

**Big Theta:**

Because we have proven 2x = O(x) and 2x= (x) we know that 2x= (x)

Since 2x is Big Oh of x and Big Omega of x then 2x is big theta of x

Now that the requirement f(n) is (), we can now say:

T(n) = (f(n) \* log(n))

T(n) = (2x \* log(n))

1. (15 pts) Use the direct unwinding to solve the following recurrence equation. DETAILED steps and proof are required.

\*NOTE: U1 stands for unwind 1, U2 unwind 2, etc.

**U1)** Let and substitute in T(n)

Simplify

Repeat using from **U1** (unwind 1) *.*

**U2)** Begin by letting and substituting into T(n)

Simplify

Pattern

(n/) = 1, = n, k = log3(n)

\* T(1) + log3(n) \* 2n

T(n) = n \* T(1) + log3(n) \* 2n

To understand this equation lets first look into Big Oh

T(n) = n \* T(1) + log3(n) \* 2n

N \* T(1) + log3(n) \* 2n <= n \* c + log3(n) \* 2n <= log3(n) \* n \* c + log3(n) \* 2(n)

T(n) <= log3(n) \* n \* c + log3(n) \* 2(n)

T(n) <= (c + 2)\*n\* log3(n)

T(n) <= c \* 2n \* log3(n)

T(n) <= c \* 2n \* log(n)

This proves Big Oh but now we can look into Big Omega

use c \* 2n \* log(n) since Big Oh was proven above

T(n) >= c \* 2n \* log(n)

n \* T(1) + log3(n) \* 2n >= c \* 2n\* log(n)

Let c = ½

n \* T(1) + log3(n) \* 2n >= n\* log(n)

log3(n) = log(n) / log(3)

n \* T(1) + 2n \* log(n) / log(3) >= n\* log(n)

As 2n \* log(n) / log(3) is always greater than n \* log(n) we know the solution holds.

So similarly to 2b, Since we have shown that T(n) = O(2n log (n)) and T(n) = (2n log(n)) we know that T(n) = (2n log(n))

Since proving Big Oh and Big Omega proves Big Theta