CPSC 221: Algorithms and Data Structures

Assignment #1, due Friday, 2015-Feb-13 at 17:00 (**or** hand-in early by Noon, Feb-10 — see below for explanation)

Submission Instructions

Type or write your assignment on clean sheets of paper with question numbers prominently labeled. Answers that are difficult to read or locate may lose marks. We recommend working problems on a draft copy then writing a separate final copy to submit.

Each submission should include the cover page (see course website) and names and student IDs of the authors at the top of **each** page. (You may work in pairs, but not in groups of three or more.) On the cover page, make sure that you sign the statement "I have read and complied with the CPSC 221 2015W1 academic conduct policy as posted on the CPSC 221 course website." (See: http://www.ugrad.cs.ubc.ca/~cs221/current/syllabus.shtml#conduct.) In keeping with the policy, you should also acknowledge on your first page any collaborators or resources that helped you with the assignment. Finally, **staple** your submission's pages together! We are not responsible for lost pages from unstapled submissions.

Submit your assignment to Box #35 in room ICCS X235. The deadline is 17:00 (5pm) on Friday, 2015-February-13. However, to give you more chance for feedback before the UBC deadline to drop the course, you may optionally submit early by 12:00 noon on Friday, 2015-February-10, and we guarantee that we will mark and return (in class) your assignment before the drop date. (Note that if you submit by Feb 6, you are not allowed to resubmit your work. We will release a sample solution within a few days after February 13. **Late submissions are not accepted.**

The number of marks allocated to a question appears in square brackets before the question number.

Questions

[10] 1. Rank the following functions by order of growth. Partition your list into classes such that functions $f(n)$ and $g(n)$ are in the same class if and only if $f(n) \in \Theta(g(n))$.

$$\lg n! \quad n! \quad \lg\lg n \quad n^2 \quad \ln n \quad 2^{\lg n} \quad \lg n \quad (\tfrac{3}{2})^n \quad n\lg n \quad 2^{2^n} \quad 2^n \quad 2^{n+1} \quad n^{1.13} \quad 4^{\lg n} \quad \sqrt{2}^{\lg n}$$

[27] 2. Simplifying to Asymptotic Bounds.

For each of the following definitions of $T(n)$, give a big-Θ bound, as simplified as you can. You do not need to prove your result. However, to get partial credit for any minor mistakes, you should show your work of how you arrived at your answer.

(a) **Example:** $T(n) = 47$, answer $\Theta(1)$.

(b) $T(n) = 8(n+1)+3$

(c) $T(n) = (n-1)(8(n+1)+3)$

(d) $T(n) = \sum_{i=0}^{n-1}(3i+1)$

(e) $T(n) = \sum_{i=0}^{\sqrt{n}}(i+2)$

(f) $T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{\lfloor n/2 \rfloor} 3$

(g) $T(0) = 1$ and $T(n) = T(n-1)+8$

(h) $T(0) = 1$ and $T(n) = 2T(n-1)+8$

(i) $T(0) = 1$ and $T(n) = T(\lfloor n/2 \rfloor)+8$

(j) $T(0) = 1$ and $T(n) = T(\lfloor n/2 \rfloor)+8n$

[15] 3. Proof Practice.

For these problems, you must do a formal mathematical proof. Each step must be justified based on definitions or basic mathematical identities.

(a) Prove that $3n^2 + 2n\lg n$ is $O(n^2)$.

(b) Prove that $\lg\sqrt{n} \in \Theta(\lg n)$.

(c) Let $T(n)$ be the following recurrence:

$$T(n) = \begin{cases} 1 & ifn = 0 \\ 6 & ifn = 1 \\ 6T(n-1) + 9T(n-2) & ifn \geq 2 \end{cases}$$

Use induction to prove that $T(n) \geq n3^n$ for all $n \geq 0$.

[20] 4. Big-Θ_k versus Big-Θ_h

In their lectures, Dr. Khosravi and Dr. Hu gave what look like different definitions of big-Θ.

Dr. Khosravi's definition, which we'll call big-Θ_k, is as follows:

$$T(n) \in \Theta_k(f(n)) \text{ iff } \exists c > 0, d > 0, \text{ and } n_0 \text{ such that } \forall n \geq n_0, [df(n) \leq T(n) \leq cf(n)].$$

Dr. Hu's definition, which we'll call big-Θ_h, is as follows:

$$T(n) \in \Theta_h(f(n)) \text{ iff } T(n) \in O(f(n)) \text{ and } T(n) \in \Omega(f(n)),$$

where $T(n) \in \Omega(f(n))$ is defined to be equivalent to $f(n) \in O(T(n))$.

(Both Drs. Khosravi and Hu define big-O the same way, that a function $T(n) \in O(f(n))$ iff $\exists c > 0$ and n_0 such that $\forall n \geq n_0, [T(n) \leq cf(n)]$.)

In this problem, you must prove that there's no conflict between Dr. Khosravi and Dr. Hu — these two definitions are exactly the same. In other words, you will **prove that** $f(n) \in \Theta_k(g(n))$ **if and only if** $f(n) \in \Theta_h(g(n))$**.**

Sometimes students are confused and try to do a proof for specific functions $f(n)$ and $g(n)$. For example, they might say "Suppose $f(n) = n^2$ and $g(n) = 2n^2$...." That's not how a proof should be! Your proof must be general and handle all possible functions $f(n)$ and $g(n)$. (You may assume that these functions are always positive, if that's helpful.)

Because you are proving an "if and only if" property, it's easiest to prove it in two separate parts. Please write your answer following this structure:

- First, assume $f(n) \in \Theta_k(g(n))$, and prove that $f(n) \in \Theta_h(g(n))$.
- Then, assume $f(n) \in \Theta_h(g(n))$, and prove that $f(n) \in \Theta_k(g(n))$.

[10] 5. Analyzing Runtime (1).

Analyze the running time of the following recursive procedure as a function of n and find a tight big-O bound on the runtime for the function. You may assume that each assignment or division takes unit time. You do not need to provide a formal proof, but you should show your work: at a minimum, show the recurrence you derive for the runtime of the code, and then how you solved the recurrence.

```
void DC(int n){
  if (n < 2)
    return;
  else{
    int i=0;
    int count = 0;
    for(i =1; i<=8; i++)
      DC(n/2);

    for(i=1; i<n*n*n; i++)
      count = count + 1;
  }
}
```

[18] 6. Analyzing Runtime (2).

Download the file `written1_matrix.cpp` from the course webpage. This is a small C++ program to raise an $n \times n$ matrix to the mth power. (You do not need to understand matrices to do this problem!)

(a) Find a tight big-O bound on the runtime for the function `matrix_mult`. Write your result in terms of n, the dimension of the matrix, rather than in terms of the size of the matrix (which is n^2). You do not need to prove your result, but you should show your work if you'd like to receive partial credit, in case you make any mistakes.

(b) Find a tight big-O bound on the runtime for the function `matrix_exp1`. Note that your bound will be a function of both n and m, e.g., it will look something like $O(mn)$ (which is NOT the correct answer).

(c) Find a tight big-O bound on the runtime for the function `matrix_exp2`. Again, your bound will be a function of both n and m.

(d) The code you downloaded is set to use `matrix_exp1`. Make sure that is still the case. Then, compile and run the program for various values of n and m, until you find values of n and m for which the program takes about 10 seconds to run. For example, on Dr. Hu's office computer, $n = 60$ and $m = 5$ takes about 12 seconds. Write down the values you found, how long it took to run, and what kind of computer you are using.

(e) Based on your asymptotic analysis of `matrix_exp1`, if you kept m at the same value, roughly what value of n would make the program take twice as long? For example, if your analysis said the runtime was $O(mn)$, then doubling n should make the program run twice as long. Run the program with the value of n you estimate. What was the run time? Was your estimate a good one?

(f) Similarly, based on your asymptotic analysis of `matrix_exp1`, if you kept n at the original value from part 6d, roughly what value of m would make the program take twice as long? Run the program with the value of m you estimate. What was the run time? Was your estimate a good one?

(g) Now, comment out the call to `matrix_exp1` and uncomment the call to `matrix_exp2`, and recompile the program. Again, find values of n and m that will make the program take around 10 seconds to run. On Dr. Hu's office computer, $n = 40$ and $m = 1000$ takes about 11 seconds. Write down the values you found, and how long it took to run.

(h) Based on your asymptotic analysis of `matrix_exp2`, if you kept m at the same value as in part 6g, roughly what value of n would make the program take twice as long? Run the program with the value of n you estimate. What was the run time? Was your estimate a good one?

(i) Based on your asymptotic analysis of `matrix_exp2`, if you kept n at the original value from part 6g, roughly what value of m would make the program take twice as long? Run the program with the value of m you estimate. What was the run time? Was your estimate a good one?