CPSC 221 Written Homework #1

Due: Friday, February 5, 2016 at 9pm

Submission Instructions

Handin your solutions using `handin`. You can write your solutions by hand and scan the pages or take pictures of them with your phone; or use a word processing package to typeset your solutions and produce a `.pdf` file.

To handin: Copy the files that contain your solutions to the directory `~/cs221/assign1` in your home directory on an undergraduate machine. (You may have to create this directory using `mkdir ~/cs221/assign1`.) Then run `handin cs221 assign1` from your home directory.

We encourage you to work in pairs. Please: If you are working with a partner, only one partner should handin the assignment. **Be sure to include names and ugrad login IDs of both partners on all solution pages!**

Late submissions are accepted subject to the following penalty: You lose $100 \times (2^{\lfloor m/5 \rfloor})/64$ percent of your mark, where m is the number of minutes late your assignment is. For example, if you hand in 10 minutes late, you'll lose $100 \times (2^2)/64 = 6.25\%$ of the mark, but if you hand in 25 minutes late, you'll lose 50% of the mark. (Handing in more than 30 minutes late is very dangerous to your entire course mark!)

Acknowledgments: Acknowledge all collaborators or sources of assistance (besides the course staff, handouts, and required textbooks) on the first page of your assignment by name. (If you quote from or derive work from any source, you *must* acknowledge that source at that point as is usual for citations. We don't need a formal citations list, although that's not too hard to produce with LaTeX. See the course website for details on the collaboration policy.)

Questions

1. Pointers. (5 points)

   A commonly used data structure is a doubly-linked list, where each node has a pointer to the next node, as well as to the node that precedes it in the list. Here is how the nodes would be declared:

```
struct Node {
    int data; // In real life, this would be whatever data you want to
       store.
    Node * prev; // Pointer to the previous node in the list.
    Node * next; // Pointer to the next node in the list.
};
```

Given a pointer `Node * foo` that points to a node in such a doubly-linked list, write code to delete that node from the list (maintaining that the remaining nodes are still in a doubly-linked list). Be sure to handle the cases where the node foo points to has no previous node, or no next node. Also, remember not to leak memory! (A correct solution only needs a few lines of code. BTW, this is a common interview question, so you'll easily find solutions online. But you'll learn more if you actually try it yourself!)

2. Pointers in C++.

   Unlike Java, the C++ and C languages let you do some unusual things with pointers. Consider the following program.

```
int a[10];
int *b = a;
while(b) {
    *b = 1;
    cout << b << endl;
    b++;
}
```

   (a) What does the `while` condition mean? (1 point)

   (b) How often does the loop run? (1 point)
   (We don't want a specific number, but instead, explain what the loop does and when it will stop.)

   (c) What does the assignment inside the loop do? (1 point)

   (d) What does the code print? (1 point)
   (We don't want the specific output on your machine, but explain what the numbers that the program prints out mean.)

   (e) When we run this program, it aborts with "segmentation fault". What does that mean? (1 point)

3. Function growth. (10 points)

   List the following functions of n in non-decreasing order of growth rate. In particular, if $f(n)$ precedes $g(n)$ in your list then $f(n) \in O(g(n))$. Remember that for functions $f(n)$ and $g(n)$, and any

increasing function $h(x)$, $f(n) < g(n)$ if and only if $h(f(n)) < h(g(n))$. So, for example, for positive functions f and g, $f(n) < g(n)$ if and only if $log(f(n)) < log(g(n))$.

$n^3 \log n$ $\sum_{i=1}^{n} i^4$ $\log n$ 2^{n^3} n n^n 2^{3^n} $n^{\frac{1}{2}}$ 42 $\log n!$

4. Simplifying to Asymptotic Bounds. (10 points)

   For each of the following definitions of $T(n)$, give the 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:** $T(n) = 42$, answer $\Theta(1)$.
   (b) $T(n) = (n + 42)c + d$, where c and d are constants.
   (c) $T(n) = \frac{n(n+1)}{2}$.
   (d) $T(n) = \sum_{i=0}^{n-1}(100i^2 + 2)$.
   (e) $T(n) = \sum_{i=0}^{\sqrt{n}} \sum_{j=i}^{\sqrt{n}} c$, where c is a constant.
   (f) $T(0) = 1$ and $T(n) = T(\lfloor \frac{2n}{3} \rfloor) + c$, where c is a constant.

5. Proof Practice.

   (a) Prove that $100\frac{n^2}{2} + 12.5$ is $\Theta(n^2)$. (5 points)
   (b) Prove that $100\frac{n^2}{2} + 12.5$ is **not** $\Theta(n^3)$. (5 points)
   (c) In their lectures, Dr. Kotthoff and Dr. Hu gave what look like different definitions of big-Ω.

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

   $T(n) \in \Omega_k(f(n))$ iff $\exists d > 0$, and n_0 such that $\forall n \geq n_0, [df(n) \leq T(n)]$.

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

   $T(n) \in \Omega_h(f(n))$ iff $f(n) \in O(T(n))$.

   (Both Drs. Kotthoff 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. Kotthoff and Dr. Hu — these two definitions are exactly the same. In other words, you will **prove that** $f(n) \in \Omega_k(g(n))$ **if and only if** $f(n) \in \Omega_h(g(n))$**.**

   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 $f(n) \in \Omega_k(g(n))$, and prove that $f(n) \in \Omega_h(g(n))$.
- Then, assume $f(n) \in \Omega_h(g(n))$, and prove that $f(n) \in \Omega_k(g(n))$.

6. Analyzing Runtime. (10 points)

   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 mystery(int n) {
    if(n < 2)
        return;
    else {
        int i = 0;
        for(i = 1; i <= 8; i += 2) {
            mystery(n/3);
        }

        int count = 0;
        for(i = 1; i < n*n; i++) {
            count = count + 1;
        }
    }
}
```

7. Heaps.

   (a) Convert the sequence $\boxed{5}\boxed{9}\boxed{1}\boxed{8}\boxed{11}\boxed{4}\boxed{6}\boxed{7}\boxed{2}\boxed{13}\boxed{12}\boxed{10}$ into a minimum heap using Floyd's *heapify* algorithm shown in the lectures. Show the binary tree at each point that a key swaps down as far as it can go. (5 points)

   (b) Insert the new element $\boxed{3}$ into the heap. Show the binary tree after each time a key moves/swaps with another key. (5 points)

   (c) Given a minimum heap H and a value q, suppose that you want to print out all keys in H that are greater than q. For example, if H is $\boxed{2}\boxed{6}\boxed{4}\boxed{8}\boxed{10}\boxed{20}\boxed{30}$ and q is 9, the algorithm should output 10, 20, 30.

      i. Give an algorithm (in English or pseudocode) that solves this problem in worst-case runtime $O(n)$, where n is the size of the heap. (Hint: Keep this simple.) (1 point)

ii. Prove that any algorithm for this problem must take worst-case runtime lower-bounded by $\Omega(n)$. (Hint: Consider a heap of size n with 1 big element, and $n-1$ identical small elements.) (4 points)