CPSC 221: Written Assignment 1

Last Updated: September 19, 2016 Due: 21.00 Monday October 3, 2016.

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. **Be sure to include the names and ugrad login IDs of both partners on all solution pages, but only one partner should handin the assignment.**

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. You cannot hand in more than 30 minutes late.

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 where it is used as is usual for citations. We don't need a formal citations list, although that's not too hard to produce with LaTeX.

Questions

1. **Pointer manipulation** (5 points) Write code (pseudo-code or C++) that given a pointer `a` to the head (front) of one linked list and a pointer `b` to a node in a second linked list, inserts all of the nodes in the first linked list (pointed to by `a`), in order, between `b` and `b->next` in the second linked list.

2. The city of Vancouver is having trouble keeping track of an ever-increasing number of houses. They want you to construct a data structure that given a street name and a house number, outputs the house numbers of the two houses next-door to it. The trouble is, Vancouver has changed the construction laws and a new (perhaps very, very narrow) house can be built between any two existing houses. Since it's Vancouver, this new construction happens frequently. Therefore, your data structure should also permit the addition of a new house: Given a street name and an existing house number, add a new house with a larger house number next to the given house and return the number of the new house. House numbers may no longer be integers since someone might want to add a house after house number 9 on Apple Street when there already exists a house number 10 on Apple Street. That's o.k. You can give the new house the number 9.5 (or any number strictly between 9 and 10). (Don't worry about streets having even numbered houses on one side and odd numbered houses on the other; assume the numbers only need to be increasing along a street.) Of course, for either operation, if the given house number and street name don't exist, the structure should report this error.

   The city already has a great implementation of a Dictionary ADT for street names. Given a street name, it will return your structure for the house numbers on the street. But what is "your structure"?

(a) (4 points) Suppose your structure is an array. How would it work? You don't need to go into too much detail, just enough to explain how much time it would take to output the next-door house numbers and to add a new house after the given house.

(b) (4 points) How would your structure work if it were a singly-linked list? Again, sketch just enough detail to explain how much time it would take for the two operations.

(c) (2 points) What data structure would you choose if you could pick one and why?

3. (5 points) Simplify the following expressions to a number or an expression without logarithms. Recall that lg is $\log_2$. Assume m, n, and p are positive integers.

$$\text{(a) } \lg 32^n \qquad \text{(b) } 2^{\lg(n^2m^2)-\lg(m^2)} \qquad \text{(c) } -\lg\frac{1}{8} \qquad \text{(d) } \log_p(1/p) \qquad \text{(e) } 64^{\lg(n^2)}$$

4. (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 $\log(f(n)) < \log(g(n))$.

$$n\log n \qquad n^{2\lg n} \qquad \log n \qquad 2^{2^n} \qquad \sqrt{n} \qquad n^n \qquad n \qquad 2^n \qquad \sum_{i=1}^{n} i^3 \qquad \lg(n!)$$

5. (10 points) For each of the following definitions of $T(n)$, give a big-Θ bound that is as simple as possible. You do not need to prove your results but you might want to show your work to get partial credit for minor mistakes.

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

(b) $T(n) = (4n + 12)(6n + 2)$

(c) $T(n) = \sum_{i=0}^{n} 2^{i+c}$ where c is a constant.

(d) $T(n) = \sum_{i=1}^{n} \sum_{j=i^2}^{n^2} c$ where c is a constant.

(e) $T(0) = 1$, $T(1) = 1$, and $T(n) = 2T(n-2) + 4$ for $n \geq 2$.

(f) $T(1) = 1$ and $T(n) = T(\lceil n/2 \rceil) + 3$ for $n \geq 2$.

6. **Proofs** Use the definitions of O-, Ω-, and Θ-notation to:

(a) (5 points) Prove that $54n^3 + 17$ is $\Theta(n^3)$.

(b) (5 points) Prove that $54n^3 + 17$ is **not** $\Theta(n^2)$.

(c) (5 points) Let $T(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0$ where a_i is constant for all $i = 0, \ldots, d$ and $a_d > 0$. ($T(n)$ is a degree-d polynomial.) Prove that $T(n) \in \Theta(n^d)$.

7. **Runtime Analysis** Give tight asymptotic worst-case bounds for the time complexity of each of the following pieces of pseudocode. Show your work. First calculate the running time, $T(n)$, (which you can assume is the number of lines of pseudocode executed as a function of n) then express its order of growth using Θ-notation.

(a) (5 points) Let the input size n be the numerical value of the parameter `n`.

```
// Calculcate x^n
double pow(double x, unsigned int n) {
  if( n == 0 ) return 1;
  double result = pow(x*x, n/2);
  if( n & 1 ) result *= x;       // if n is odd...
  return result;
}
```

(b) (5 points) Let n be the number of elements in array `A` (i.e., $n =$ `A.length`).

```
// Returns the length of the longest subsequence of strictly increasing
// elements in the array A.  A subsequence of array A is not necessarily
// contiguous.
longestIncreasing(A)
  ResultForPrefix = new Array[A.length]
  for i = 0 to A.length - 1 {
    r = 1
    for j = 0 to i - 1 {
      if A[j] < A[i] and r < ResultForPrefix[j] + 1 then
        r = ResultForPrefix[j] + 1
    }
    ResultForPrefix[i] = r
    if bestOverall < r then bestOverall = r
  }
  return bestOverall
```