Midterm Review Lab/Potential Assignment

March 2, 2014

1 Q1 and Q3 Review

Consider the function $f(n) = n^2(n - 2\lfloor n/2 \rfloor)$. (Note: $\lfloor x \rfloor$ is the "floor" function applied to x.)

1. Hand-calculate $f(5)$, $f(6)$, $f(7)$, and $f(8)$.
2. Give a good, "tight" asymptotic upper-bound on this function. ("Tight" in this case means: as tight as we can get while still using a subjectively "clean and sensible" function. So, this will require a judgment call.)
3. Give a good, "tight" asymptotic lower-bound on this function.
4. Now, give good, "tight" asymptotic upper- and lower-bounds on the similar function $g(n) = n^2(n - \lfloor n/2 \rfloor)$.
5. For both of the statements below: Indicate whether the statement is true or false, briefly (and only informally) explain each answer, write each statement in formal predicate logic, write the negation of each statement in formal predicate logic with the negation moved "inwards" as far as it can go, and then pick one of your answers to **prove**.
   - f is big-O of g.
   - f is big-Ω of g.

2 Q2 Review

1. Give a tight big-Ω bound in terms of n on the number of iterations of the loop that are executed in a call to the following function:

```
void doit(int n) {
  if (n < 100)
    return 0;
  for (int i = 0; i * i < n; i++) {
    dothatotherthing(i);
  }
}
```

2. Why is this not a good description of best-case input to this function: "When n is less than 100."

3. More generally, explain why it's meaningless to ask for a "best-case analysis of the number of iterations of the loop that are executed in a call to the function". (Note: big-Ω does **NOT** mean "best-case analysis", and big-O does **NOT** mean worst-case analysis.)

4. Convert the following function (adapted from the midterm exam) into tail-recursive code with **no** loops.

```
bool prime_tester(int n) {
  for (int i = 2; i*i <= n; i++) {
    if (n%i == 0) return false;
  }
  return true;
}
```

3 Q4 Review

We can describe a social network as a "graph", an abstract data type with a collection of "nodes" or "vertices" some of which are connected one to another by "edges". For a social network, we might decide we care about who has shaken hands with whom. Then, a sensible graph might be one where a particular person is a node and a link from one person to another is an edge meaning "Person A has shaken hands with person B".

(An edge from A to B in this context is the same as an edge from B to A. This is called an "undirected" graph. Compare to a graph where we track who knows whose name. You may know Justin Bieber's name (independent of whether you wish to); he probably doesn't know yours (independent of whether *he* wishes to).)

Let's consider a network consisting of Anika, Brandon, Calla, Dell, Eustace, and Farrad, handy because we can shorten them to A, B, C, D, E, and F. Here is a list of who has shaken hands with everyone else:

- A and B
- B and C
- B and D
- C and E
- D and F
- E and F

Now, let's say we're going to start with Anika and search for a way to reach Farrad via people Anika has shaken hands with. Our algorithm is:

1. Ensure no one is marked as visited and add Anika to a min-priority queue that uses the first letter of each person's name as their priority (where "A" compares less than "B").
2. While the priority queue is not empty, repeat the following:
   (a) Remove the next element i from the priority queue and mark i as "visited".
   (b) If i is Farrad, stop and declare victory!
   (c) Otherwise, for each person j that i has shaken hands with, if j is not marked as "visited", add j to the priority queue.

Simulate running this algorithm to answer the following. Your simulations absolutely, positively **MUST** involve sketching a representation of the graph and of the priority queue data structure. (Why? Because an expert's would.)

1. Write out the sequence of states in the order they are *added* to the priority queue.
2. Write out the sequence of states in the order they are *removed* from the priority queue.
3. Explain why A is not added to the priority queue again when we add the people B has shaken hands with.

4. Change the algorithm so it uses a *max*-priority queue instead and write out the two sequences again.
5. Explain why E is never added to the max-priority queue.
6. During your simulation, do you need to simulate a binary heap? Why or why not?
7. For a priority queue (as opposed to a stack or queue), does it matter what order we add the "neighbors" of a person (the people that person has shaken hands with)?
8. Might your answer to either of the previous two questions change if we add someone named "Curt" to the graph? Explain your answer.
9. If we had just said "Use Best-First Search to explore this graph to find F starting from A", explain why neither the min- nor the max-priority queue would be the unique "right" way to perform the exploration. (*Even* if the Almighty Wikipedia page on Best-First Search seems to say one of them is! :P)

3.1 Q5 Review

1. Consider the following function:

```
int thgieh(Node* root) {
  if (root->right == NULL &&        // Line 1
      root->left == NULL)           // Line 2
    return 0;                       // Line 3
  if (root == NULL)                 // Line 4
    return -1;                      // Line 5
  int tl = thgieh(root->left);      // Line 6
  int tr = thgieh(root->right);     // Line 7
  if (tl < tr)                      // Line 8
    return 1 + tl;                  // Line 9
  return 1 + tr;                    // Line 10
}
```

   (a) Why should lines 4–5 go before line 1 instead?

   (b) Ignoring the issue from the previous question, briefly explain why lines 1–3 are unnecessary, anyway.

(c) Assuming we comment out lines 1–3, give a recurrence relation describing the amount of time it takes to run this function on a tree with n nodes. Hint: in your recursive case, let k be the number of nodes in the left subtree. What would a formula for the number of nodes in the right subtree be in that case?

(d) Solve your recurrence relation in order to produce a big-Θ bound on the runtime of this function it terms of the number of nodes in the input tree n.

2. In this problem, you will prove by structural induction that the mathematical definition of "thgieh"—the thgieh of an empty tree is -1 and the thgieh of any other tree is equal to 1 plus the smaller of the thgiehs of its two subtrees—actually does correspond to the "length of the shortest path from the root to a node with a empty subtree". (This isn't obvious from the definition of "thgieh"!) Note: we will agree by definition that the "shortest path from the root of an **empty tree** to a node with a empty subtree" is of length -1.

   (a) Write down the theorem you're trying to prove defined over a tree t. (If you don't do this, you could end up proving a totally unrelated theorem like "my code to calculate `tghieh` produces the same result as the mathematical definition of `thgieh`".)

   (b) Write down the recursive structure of a binary tree as given in the midterm problem. You *must* (and should!) use this recursive structure to guide the structure of your proof in this problem.

   Note: You may *not* attempt so-called "weak" or "strong" induction on the thgieh number (which does not work well *at all*) or on the number of nodes in the tree (which can actually work OK for this problem... but don't do it anyway!).

   (c) Using your recursive structure, complete this statement of what you're proving in the base case: "The theorem should hold for _______________________."

   (d) Prove the base case. (This may **NOT** just be some equations. It should be a clear explanation of why the theorem holds in what you decided was the base case, including connecting to the structure you're dealing with in the base case, the definition of "thgieh", and what the length of "shortest path from the root to a node with a empty subtree" is.)

(e) Now, use your recursive structure to complete this statement: "Now, consider an arbitrary ________________________ named t." (Note: the answer is not *just* "tree".)

(f) Now, fill in the blank to establish your assumption(s): "For the inductive step, the hypothesis we assume (the "IH") is: The theorem holds for ________________________." (The answer is very much *not* "the $n = k$ case since I'm working on the $n = k + 1$ case.")

(g) Fill in the blank to establish what you need to prove: "We must show that the theorem holds for ______________________."

(h) Finally, complete the proof. (This isn't just some equations either! Look back at the note on the base case.)

(i) Post-Finally, draw a tree with a thgieh of 1 whose right subtree has a thgieh of 0 and whose left subtree has a thgieh of 2. (This is why performing induction on the thgieh of the tree is such bad news. One of the subtrees may have a **LARGER** thgieh than the tree itself! So, assuming your theorem holds for trees with smaller thgieh isn't very useful.)

4 Q6 Review

Consider the following code from Q6 of the midterm:

```
void convert(Node* root) {
  if (root == NULL) {
    return;
  }
  else {
    int thgiehL = thgieh(root->left);
    int thgiehR = thgieh(root->right);
    if (thgiehR > thgiehL) {
      Node* temp = root->right;
      root->right = root->left;
      root->left = temp;
    }
    convert(root->right);
    convert(root->left);
  }
}
```

1. Explain why it would not be possible for $T(0)$ to be either -1 or 0. (Note: $T(n)$ is the amount of time it takes the function to execute on an input of size n. You actually don't need to know what the function is to answer the question!)

2. Give a correct recurrence $T(n)$.

3. Solve your recurrence to produce a big-Θ bound on the runtime of the function in terms of the number of nodes in the tree.