15-451 Algorithms, Fall 2010

Homework # 1 **Due: September 9, 2010**

Please hand in each problem on a separate sheet and put your **name** and **recitation** (time or letter) at the top of each page. You will be handing each problem into a separate box in lecture, and we will then give homeworks back in recitation.

Remember: written homeworks are to be done *individually*. Group work is only for the oral-presentation assignments. You are required to typeset your homework.

Problems:

(25 pts) 1. **Asymptotic Relationships.** Each question is worth five points. Please formally justify your answers using the definitions of o, O, Θ given in the lecture notes. Assume that all functions are strictly positive, $f(n) \geq 0, g(n) \geq 0$.

(a) Formally prove that $\binom{n}{k} \in \Theta(n^k)$ for all constants k.

(b) Suppose that $f(n) = o(g(n))$. Prove that $f(n) \notin \Theta(g(n))$.

(c) Suppose that $f(n) \in \Theta(g(n))$, can we conclude that $f(n)^2 \in \Theta(g(n)^2)$? Justify your answer.

(d) Suppose that $f(n) \in \Theta(g(n))$, can we conclude that $2^{f(n)} \in \Theta(2^{g(n)})$? Justify your answer.

(e) Harry Q. Bovik has just found a new matrix multiplication algorithm $\mathcal{A}$ with running time $T(n) = 9T(n/3) + 8T(n/2) + n$ $(T(1) = 1)$, Harry claims that $\mathcal{A}$ runs in time $O(n^2)$ offering the following inductive proof.

> *For small values of k (say $k < 4$) we certainly have $T(k) \leq 100k^2$ so $T(k) \in O(k^2)$. Assume that $T(k) \in O(k^2)$ for $k < n$, now by our assumption we can conclude that $T(n/3) = O(n^2)$ and $T(n/2) = O(n^2)$. Therefore, $9T(n/3) + 8T(n/2) + n = O(n^2)$ and in particular $T(n) = O(n^2)$.*

Is Harry's proof correct? Is his claim correct? Briefly justify your answer.

(f) (Bonus: 5 pts) Suppose that $f(n) \in O(g(n))$ and $f(n) \notin \Omega(g(n))$, can we conclude that $f(n) = o(g(n))$? Justify your answer.

(25 pts) 2. **Is Ms.Perfect correct?** Let $p_d(x)$ denote the d-degree polynomial $c_0x^d + c_1x^{d-1} + \cdots + c_d$ in x. Given x and all the c_is from $i = 0$ through d, Ms.Perfect wants to compute the values of $p_d(x)$ and $\frac{\mathrm{d}}{\mathrm{d}\,x}p_d(x)$ at a single go, where $\frac{\mathrm{d}}{\mathrm{d}\,x}$ denotes the first-order derivative with respect to x. To that effect, she uses the simple looking Algorithm 1 to obtain $p_d(x)$ in v_1 and the derivative in v_2.

(a) (20 pts) An analysis of the above algorithm shows that while v_1 indeed contains the value of $p_d(x)$, v_2 does not contain its derivative. After all, Ms.Perfect is *not perfect*. Your task is to prove that the algorithm above computes $p_d(x)$ in v_1 and $\frac{\mathrm{d}}{\mathrm{d}\,x}p_{d+1}(x)$ in v_2 for some arbitrary c_{d+1}. To prove, come up with expressions for the variables v_1, v_1^{old} and v_2 of the function `compute` which depend on m denoting their respective values after m iterations of the loop for $m \geq 0$ (don't forget to take care of the base cases when coming up with expressions). These are also known as *loop-invariants*. Show the values of these expressions for $m = 0, 1$. Prove by induction with $m = 2$ as the base case.

Algorithm 1 Ms.Perfect's algorithm

`compute` (given x and c_is from $i = 0$ through d)

1: $v_1 \leftarrow 0$, $v_1^{old} \leftarrow 0$, $v_2 \leftarrow 0$
2: **for** $i = 0$ through d **do**
3: $\quad v_1^{old} \leftarrow v_1$
4: $\quad v_1 \leftarrow v_1.x + c_i$
5: $\quad v_2 \leftarrow (v_2 + v_1^{old}).x + c_i$
6: **end for**
7: **return** (v_1,v_2)

(b) (5 pts) Can you help Ms.Perfect compute her values by slightly modifying the algorithm? You are allowed to perform at most two modifications which can only involve a reordering of the statements inside the *for* loop or using a different coefficient c_j while keeping the *structure* of the statements unchanged. Assume that c_{-1} is equal to 0. At the end of the algorithm, v_1 should contain the value of $p_d(x)$ and v_2 should contain the value of $\frac{\mathrm{d}}{\mathrm{d}\,x}p_d(x)$. You just need to mention the modifications. You need not prove it correct.

(25 pts) 3. **Recurrences.** Solve the following recurrences, giving your answer in Θ notation. For each of them, assume the base case $T(x) = 1$ for $x \leq 5$ and lg is log base 2. Show your work.

(a) (7 pts) $T(n) = 2T(n-1) + 2^n$.

(b) (5 pts) $T(n) = 4T(n/5) + n^{(\frac{\lg 4}{\lg 5})}$.

(c) (5 pts) $T(n) = n \lg n + 9T(n/3)$.

(d) (8 pts) $T(n) = n^{3/4}T(n^{1/4}) + n$.
(Hint: writing out the recursion tree might make it easier to analyze)

(25 pts) 4. **Jonsort**

Jon has come up with an amazing new algorithm for sorting numbers he's calling jonsort.

Jon's new sorting algorithm is as follows:

```
Input: An unsorted list of numbers
Output: A sorted list of numbers in ascending order

1. FUNCTION jonsort(list A):
2.    sorted_list = []
3.    while length(A) > 0:
4.        subsection = []
5.        subsection[0] = A[0]
6.        remove A[0] from A
7.
8.        for i in 0 to length(A):
```

```
9.            if A[i] >= subsection[last]:
10.               append A[i] to the end of subsection
11.               remove A[i] from A
12.       merge(subsection, sorted_list)
13.   return sorted_list
```

For the merge function, if m is the size of the list before the subsection is removed from it, assume that the merge function always takes m comparisons to merge the subsection and the sorted list.
In other words, assume that the merge function takes the same number of comparisons as the length of the list A as it was in line 3 of the pseudocode.

For example, consider the unsorted list $[5, 2, 1, 7]$. Below is a chart designating the state of the list, the subsection, and the sorted list after each iteration:

list	subsection	sorted_list
[5,2,1,7]	[]	[]
[2,1]	[5,7]	[]
[2,1]	[]	[5,7]
[1]	[2]	[5,7]
[1]	[]	[2,5,7]
[]	[1]	[2,5,7]
[]	[]	[1,2,5,7]

(a) (9 pts) Jon claims the algorithm is asymptotically faster than quicksort in the worst case *in terms of the number of comparisons*. Is he correct? Provide an ordering of elements that constitutes jonsort's worst case input and prove that it is the worst case. Additionally, generate a recurrence for the number of comparison's jonsort makes in the worst case and solve the recurrence for its asymptotic bound.

(b) (15 pts) Jon is interested in the recurrence for the expected *number of comparisons* of his algorithm on randomized input. Find a recurrence representing the expected *number of comparisons* that jonsort makes. Present this recurrence in the simplest form possible. If you can simplify summations, do so (you are not expected to solve the recurrence, just to present it). Explain how you derived your recurrence.

(c) (1 pts) Thus far, we have only been analyzing this algorithm in terms of the number of comparisons made. Note that this algorithm, however, contains a lot of insertions and deletions from a list. In what real world situations may this algorithm be more practical than quicksort?