CSE 241 Algorithms and Data Structures Fall Semester 2015

Homework 1

Assigned: 8/31/2015 *Due Date: 9/16/2015*

This homework must be completed and submitted electronically. Formatting standards, submission procedures, and (optional) document templates for homeworks may be found at

`http://classes.engineering.wustl.edu/cse241/ehomework/ehomework-guide.html`

Advice on how to compose homeworks electronically, with links to relevant documentation for several different composition tools, may be found at

`http://classes.engineering.wustl.edu/cse241/ehomework/composing-tips.html`

Please remember to

- **typeset (do not hand-write) your homework's text;**
- **create a separate PDF file for each problem;**
- **include a header with your name, WUSTL key, and the homework number at the top of each page of each solution;**
- **include any figures (typewritten or hand-drawn) inline or as floats;**
- **upload and submit your PDFs to Blackboard before class time on the due date.**

Always show your work.

The approximate weights of each problem (out of 100%) are given in parentheses, but things that come up in the course of grading may cause these weights to be adjusted. Note that you need only do any four problems out of five to get full credit for this homework, though you may do all five for extra credit.

Unless otherwise noted, all occurrences of “log” refer to the **base-2 logarithm**.

1. (25%) Professor Powerball is an obsessive Lotto player who is always looking for an edge. He has collected all the “Pick Six” results for the past forty years and is trying to group them into clusters of similar results to look “hot” patterns of numbers that come up unusually often.

   The professor has a list of n results $S_1 \dots S_n$. Each result contains six 2-digit integer values. The distance $d(S_i, S_j)$ between two results is the number of values present in one set but not both. The professor's clustering scheme requires computing the matrix of all pairwise distances $d(S_i, S_j)$ for $1 \leq i \leq j \leq n$.

   The professor has three algorithms at his disposal to compute all the distances. The first algorithm, A, simply computes all the distances directly and has a running time $0.03n^2$ seconds on an input of size n. The second algorithm, B, uses a fancy sorting scheme to speed up the computation and runs in time $0.15n \log n + 0.00001n^2$ seconds. The third algorithm, C, uses an even fancier hashing scheme and runs in time $n + 0.00001n^2$ seconds.

   (a) What is the smallest problem size n_0 such that algorithm B is (strictly) faster than algorithm A for all $n \geq n_0$? (*Hint:* I don't know of an analytical solution to this problem, so try solving it numerically.)

   (b) What is the smallest problem size n_1 such that algorithm C is (strictly) faster than algorithm B for all $n \geq n_1$?

   (c) Describe how to construct a distance computing algorithm that always achieves the best running time of any of algorithms A, B, and C on its input.

   (d) Professor Nikrasch suggests processing the Lotto results using a different clustering algorithm altogether, one which avoids computing distances between results. This new algorithm runs in $n^{1.2}$ seconds on an input of size n. Is this algorithm ever faster than the fastest of A, B, or C? If so, for what value of n does it start to win?

2. (25%) Compute precise statement counts for the following code fragments. Show your work, including for each loop how many times its body executes per iteration of the next outermost loop. S_1 and S_2 below represent single constant-time statements.

   (a)

```
$j \leftarrow 1$
while $j < n$ do
    $S_1$
    $k \leftarrow n$
    while $k > j$ do
        $S_2$
        $k--$
    $j++$
```

(b)

```
$j \leftarrow 0$
while $j \leq n$ do
    $k \leftarrow 0$
    while $k < j$ do
        $S_1$
        $k$++
    $j$++
```

(c)

```
$j \leftarrow 1$
while $j \leq n$ do
    $k \leftarrow 1$
    while $k < j \times j$ do
        $S_1$
        $k$++
    $j$++
```

3. (25%) Consider the following generic divide-and-conquer algorithm:

```
DivAndConq($n$)
    if $n \leq 1$
        DoBaseCase
    else
        DivAndConq($n/2$)
        MungeBytes($n$)
        DivAndConq($n/2$)
```

MungeBytes includes both the divide and combine steps and is assumed to run in time cn. Assume that DoBaseCase takes constant time c_0. This algorithm behaves very much like the fast closest-pair algorithm we saw in class, except that the base case is only for $n = 1$, not for $n = 2$.

Suppose we modify this algorithm so that, instead of recurring on two subproblems of size $n/2$, it recurs on **five** subproblems of size $n/5$. We saw that dividing the problem in half can speed up the computation, so perhaps dividing into it more pieces is even better? If it makes you feel better, you may assume in this problem that n is an exact power of five.

(a) Write a recurrence for the running time $T(n)$ of the modified algorithm.

(b) Sketch the recursion tree for your recurrence and compute an exact expression for its solution.

(c) Dash our hopes for a faster algorithm by showing that the expression in part **(b)** is still $\Theta(n \log n)$.

(d) In general, what happens if you try to recur on m subproblems of size n/m when dividing and combining take total time cn? You don't need to work out all the details again – a brief argument in words is fine.

4. (25%) Answer each of the following questions. Justify your answers using either the definitions of O, Ω, and Θ or the techniques shown in class (along with basic math).

(a) Does $(n+1)^2 = \Omega(n \log n)$?

(b) Does $3^{9 \log n + \log \log n} = \Omega(n^3)$?

(c) Does $n \log_5 n = \Theta(n \ln n)$?

(d) Does $(n-2)^2 = \Theta(n \log n)$?

(e) Does $n^{61/60} = O(n \log n)$?

(f) Let $f(n)$ and $g(n)$ be non-negative functions of n.
If $f(n) = O(g(n))$, does $f(n) + g(n) = \Theta(g(n))$?

(g) Let $f(n)$ and $g(n)$ be non-negative functions of n.
If $f(n) = \Theta(g(n))$, does $f(n)/g(n) = \Theta(1)$?

5. (25%) An unusual feature of the fast closest-pair algorithm discussed in class is that it requires supplying the input points sorted in *two* different orders: by x and by y. Professor Uitsmijter proposes to simplify the algorithm as follows.

The input points are sorted *only* by x-coordinate; that is, there is a `ptsByX` array, but no `ptsByY` array. To create the `yStrip` array in the combine stage, the algorithm makes a single linear-time pass over `ptsByX` to select the points in `yStrip`, then sorts these points by y-coordinate before continuing with the combining algorithm. Since there can be $\Theta(n)$ selected points in the worse case, and the point coordinates are arbitrary real numbers, this sort requires worst-case time $\Theta(n \log n)$.

(a) Write a recurrence for the running time of the modified closest-pair algorithm, in terms of the number of input points n.

(b) Sketch the recursion tree for this recurrence and derive a non-recursive (but not necessarily closed-form) exact expression for its solution.

(c) Show that the expression you got in part **(b)** is $\Theta(n \log^2 n)$. *Hint*: recall that

$$\log(n/m) = \log n - \log m.$$

(d) Professor Strammermax claims that the running time of the new algorithm can be reduced to $\Theta(n \log n)$. The key idea is to *reconstruct* the sorted `ptsByY` array dynamically inside the algorithm.

Suppose that the two recursive calls in the algorithm are modified to return both the closest pairs on left and right *and* two arrays containing *all* the left and right points, respectively, each sorted by y-coordinate. (Clearly, we can compute such a sorted array in constant time when $n \leq 2$.) Describe in pseudocode how to combine these two arrays in time $\Theta(n)$ to produce an array of *all* input points sorted by y. Justify the correctness and running time of your solution.