Homework 3

CSC 140 – Advanced Algorithm Design and Analysis

Follow the "homework procedures" found on Piazza to submit files to DBInbox. If anything in this assignment does not make sense, please ask for help.

Quiz: We will have a closed-note 20-30 minute quiz on this homework in class Wed Feb 27.

Non-submitted work:

Read: CLRS Sections 4.4 and 4.5.

Written Problems:

There are three steps to follow for the written problems. Step 1: Do them as if they were homework assigned to be graded (ie, take them seriously and try to do a good job). Step 2: Self-assess your work by comparing your solutions to the solutions provided by me. Step 3: Revise your original attempts as little as possible to make them correct. Submit both your original attempt and your revision as separate files.

Submit two files to DBInbox following the procedure documented on Piazza. The first file should be named exactly **hw3.pdf** and should be your homework solutions before looking at my solutions. The second file should be named exactly **hw3revised.pdf** and should be your homework solutions after looking at my solutions and revising your answers.

NOTE: If you wish to handwrite your solutions you may, but only if your handwriting is easy to read and the file you submit is less than 1MB in size. Also, part of your homework grade may be based on my subjective opinion of how seriously you take this process.

1) Use the master theorem and/or recursion-tree to determine the running time of each of the recurrences listed in Problem 4-1. You will be expected to know how to do both on exams.

2) Let's find a lower bound on a naive fibonacci function. Let's define `fib` in C as `unsigned fib(unsigned n) { if (n<2) return 1; else return fib(n-1) + fib(n-2); }`. Draw a recursion tree for `fib` through enough levels so that you get a feel for how it grows. Now sum all of the rows through the last full row. By "last full row", I mean the first row from the top that has at least one leaf node (rows below this will no longer be doubling in number and will thus not be full). Row sums through this last full row are easy to compute, but does not include *all* of the work of `fib`, making it a lower bound on `fib`.

An upper bound on `fib` can be achieved by pretending that the progression of the row sums continues all the way to the deepest leaf. If that were the case, what would the sum of all the rows be? This is an upper bound because it includes more work than `fib` actually does.

3) Here in your written homework give a divide-and-conquer algorithm that you expect to have $\Theta(n^2)$ running time. Write its recurrence relation and solve it to show that it is $\Theta(n^2)$. The algorithm you choose may be a real algorithm or a toy one that you make up just for this problem.

4) Implement your algorithm from Problem 3 and demonstrate experimentally that its runtime is in $\Theta(n^2)$. Do this by timing it on many different problem sizes and plotting running time versus problem size. Refine your technique until your plot looks like a very nice $\Theta(n^2)$.

5) Write a recurrence relation that expresses the expected worst-case running time of your algorithm in Program B when the problem size n is the total number of integers in the 2D array. What does this recurrence evaluate to using Θ-notation?

Programming:

Due: The program(s) will be first graded sometime – maybe hours, maybe days – after the quiz.

A) Submit to DBInbox the program you wrote to generate your data for Problem 4 as a single file named either **hw3_A.c** or **Hw3_A.java**. The program should be reasonably commented and should include your name and class 4-digit ID number. If using C, the file should compile without emitting warnings when compiled with gcc using the command `gcc -Wextra -Wall -std=c99 hw3_A.c`.

B) Submit to DBInbox a divide-and-conquer two-dimensional `contains` method in Java as a single file named **Hw3_B.java**. Document your file reasonably well, and include your name, 4-digit class ID, and preferred email. In a public class named `Hw3_B` include the following method.

```
// pre: a is r rows, c cols, for some r,c > 0. Each row and column is in non-decreasing order.
// post: returns true iff k is an element of a.
public static boolean contains(int[][] a, int k)
```

Your solution must be structured as a recursive divide and conquer algorithm with running time $T(n) = aT(n/b) + f(n)$ for some positive integers a, b. For efficiency's sake never create a new array as part of your solution.

This interface is merely a convenience to the client and is no good for recursion. You will almost certainly want to write a second version that is good for recursion. For example, if I were writing a binary search algorithm as a recursive divide and conquer version, I would include a public convenience method and write:

```
public static int binary(int[] a, int k) {
    return binary(a, 0, a.length-1, k);
}

public static int binary(int[] a, int l, int r, int k) {
    if (l>r) {
        return -1;
    } else {
        int i = (l+r)/2;
        if (k < a[i]) return binary(a, l, i-1, k);
        if (k > a[i]) return binary(a, i+1, r, k);
        return i;
    }
}
```