CSC 445 **Homework 2 (version 1)** **Fall 2021**

This homework is due Thursday, September 30, at 3:30pm. Please upload a single `PDF` file containing your submission (ensuring scans of handwritten work are legible) on `Gradescope` by that time.

The questions are drawn from the material in Chapters 3, 4, and Appendix A of the text, and the lectures in class on *asymptotics*, *sums*, *recurrences*, and *divide-and-conquer*. The homework is worth a total of 100 points.

For algorithm design questions, present the ideas behind your algorithm in prose and pictures, be sure to argue that your algorithm is correct, and give your analysis to show the algorithm meets a given time bound. Pseudocode is not required, but may aid your time analysis.

Remember to (a) start each problem on a *new page*, and (b) put your answers in the *correct order*. If you can't solve a problem, state this, and write only what you know to be correct. Neatness and conciseness counts.

(1) **(Analyzing nested loops)** (10 points) Prove a Θ-bound on the running time of the following procedure as a function of n. In your analysis, do *not* use an exact formula for $\sum_i i$ or $\sum_i i^2$.

```
function F($n$) begin
    array $A[1:n, 1:n]$
    for $i := 1$ to $n$ do
        $A[i,i] := 0$
    for $\ell := 2$ to $n$ do
        for $i := 1$ to $n - \ell + 1$ do begin
            $j := i + \ell - 1$
            $A[i,j] := \infty$
            for $k := i$ to $j - 1$ do
                $A[i,j] := \min\{A[i,j],\ A[i,k] + A[k+1,j] + ijk\}$
        end
    return $A[1,n]$
end
```

(2) **(Hybrid sorting)** (30 points) Insertion sort is one of the fastest algorithms in practice for sorting an array of length n when n is small. Its worst-case running time is $\Theta(n^2)$, however, so for large n merge sort will be faster in the worst case as it runs in $\Theta(n \log n)$ time.

Consider the following *hybrid* sorting algorithm, which tries to combine the best features of insertion sort and merge sort. Suppose we divide the sorting problem into subproblems as in merge sort, but use insertion sort to solve a subproblem once it is small enough. More precisely, suppose we divide the input array into $\lceil n/k \rceil$ lists of length at most k, sort each list using insertion sort, and then merge them into one sorted list, where k is a parameter.

The following questions analyze the running time of this hybrid algorithm.

(a) (10 points) Show that $\lceil n/k \rceil$ lists, each of length at most k, can be sorted by insertion sort in $\Theta(nk)$ worst-case time.

(b) (10 points) Show that the sorted sublists from Part (a) can be merged into one sorted list in $\Theta(n \log(n/k))$ worst-case time.

(c) (10 points) By Parts (a) and (b), the hybrid sorting algorithm runs in worst-case time $\Theta(nk + n\log(n/k))$. We would like to choose k as a function of n so that the worst-case order of growth for this hybrid algorithm is no worse than the order of growth for merge sort. What is the fastest rate of growth of k for which this holds?

Be sure to prove your answer.

(Hint: Answering this requires two steps: (1) coming up with a function $f(n)$ such that if $k = \Theta(f(n))$, the hybrid algorithm runs in $O(n \log n)$ time; and (2) showing that if $k = \omega(f(n))$, the hybrid algorithm runs in $\omega(n \log n)$ time.)

(3) **(Understanding asymptotics)** (25 points)

(a) (15 points) Order the following functions according to their rate of growth. Specifically, group the functions into equivalence classes such that functions f and g are in the same class iff $f \in \Theta(g)$, and then order the equivalence classes from slowest to fastest growing.

For each successive pair of functions (f, g) in your order, state the relationship between f and g, namely either $f \in \Theta(g)$ or $f \in o(g)$.

$$\begin{array}{ccccc} n! & \lfloor \ln n \rfloor ! & n^2 & (\ln n)^2 & \ln(n!) \\ 2^{2^n} & \ln \ln n & n^{\ln \ln n} & \sqrt{\ln n} & 2^{\ln n} \\ (\ln n)^{\ln n} & 4^n & n & 2^n & n \ln n \end{array}$$

(Hint: First form a rough initial order of the functions, and then use insertion sort on this list to obtain the final order. To determine the relationship between two functions, take a limit of their ratio, or use the list of asymptotic properties given in class. The exponential property in the handout from class may be especially useful.)

(b) (10 points) Using the definition of Θ, prove the following.

Theorem *For any two functions $f(n)$ and $g(n)$ that are asymptotically non-negative,*

$$f(n) + g(n) = \Theta\Big(\max\{f(n), g(n)\}\Big).$$

(c) **(bonus)** (10 points) Find a function that grows faster than any polynomial, but slower than any exponential. More precisely, find a function $f(n)$ such that both $f \in \omega(n^a)$ for all a, and $f \in o(b^n)$ for all $b > 1$, where a and b are constants. Prove that your function satisfies these properties.

(4) **(Solving recurrences)** (20 points) Derive a Θ-bound on the solution to each of the following recurrences. Do not worry about taking floors or ceilings of arguments.

(a) $T(n) = 4\,T(n/2) + n^2\sqrt{n}$.

(b) $T(n) = 32\,T(n/4) + n^2\sqrt{n}$.

(c) $T(n) = 3\,T(n/2) + n \lg n$.

(d) $T(n) = 3\,T(n/3) + n \lg n$.

(e) **(bonus)** (10 points) $T(n) = T(\sqrt{n}) + 1$.

(5) **(Reducing kth-smallest to median-finding)** (15 points) If we have an algorithm that finds the kth-smallest element of an n element set, we can obtain an algorithm for finding

the median element simply by calling the kth-smallest algorithm with $k = \lceil (n+1)/2 \rceil$. This question asks you to do the converse.

Given an algorithm that finds the median element of an n element set in $\Theta(n)$ time, design an algorithm that finds the kth-smallest element for arbitrary k in $\Theta(n)$ time using the median-finding algorithm. Be sure to analyze the running time of your algorithm.

(Note: Do *not* invoke the linear-time kth-smallest algorithm presented in class and the book. You must reduce the problem of finding the kth-smallest element to the problem of finding the median.)

Note that Problems (3)(c) and (4)(e) are *bonus* questions, and are *not* required.