CSC 545 **Homework 2 (version 1)** **Spring 2022**

This homework is due Thursday, February 24, at 3:30pm. Please upload a single `PDF` file containing your submission to `Gradescope` by that time. Do ensure that when you *scan* handwritten work it is legible (using a free app such as `CamScanner` or `Turboscan`), and do remember to mark in `Gradescope` which of your pages *correspond* to which homework problems.

Questions are drawn from the material in class, and in Chapters 9 and 15 of the text, on *finding the kth smallest* and *dynamic programming*.

The homework is worth a total of 100 points. When point breakdowns are not given for the parts of a problem, each part has equal weight.

For general algorithm design questions, (i) present the ideas behind your algorithm in prose and pictures, (ii) argue that your algorithm is correct, and (iii) show your analysis that the algorithm meets the given time bound. Pseudocode is not required, but may aid in your analysis.

When presenting a *dynamic programming* algorithm, your solution must follow the four-part framework:

(1) *characterize* the recursive structure of an optimal solution,
(2) *derive* a recurrence equation for the value of an optimal solution,
(3) *evaluate* the recurrence bottom-up in a table, and
(4) *recover* an optimal solution from the table of solution values.

You will be graded on each part as follows: Part (1) is 30%, Part (2) is 50%, Part (3) is 10%, and Part (4) is 10%. Be sure to analyze the running time for Parts (3) and (4) of your algorithm.

Please 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. Conciseness counts!

(1) **(Finding elements near the median)** (20 points) Given an unsorted array A of n distinct numbers and an integer k where $1 \leq k \leq n$, design an algorithm that finds the k numbers in A that are closest *in value* to the median in $\Theta(n)$ time.

(2) **(Finding quantiles)** (20 points) For a set S of n numbers and an integer k where $1 \leq k \leq n$, the *kth quantiles* of S are $k-1$ elements from S whose ranks in S divide the sorted set into k groups that are of equal size to within one unit.

Given an unsorted array A of n distinct numbers, design a divide-and-conquer algorithm that finds the kth quantiles of A in $O(n \log k)$ time.

(Note: To help explain the problem, the 4-quantiles of a set of scores are the values that define the 25-, 50-, and 75-percentile cutoffs. Similarly, the 10-quantiles of a set are the values that define the 10-, 20-, 30-, ..., and 90-percentile cutoffs.)

(3) **(Longest palindromic subsequence)** (30 points) A *palindrome* is any string that reads the same forwards and backwards. (For example, "`abba`" is a palindrome.) Given a string S, we want to find the *longest subsequence* of S that is a *palindrome*. Using dynamic programming, design an algorithm that finds the longest palindromic subsequence of a string S of length n in $O(n^2)$ time.

(Hint: When characterizing the recursive structure of an optimal solution for the first part of the framework, set up your cases similar to how we did for the longest common subsequence problem: namely, a longest palindromic subsequence of $S[1:n]$ uses both $S[1]$ and $S[n]$, or it does not use $S[1]$, or it does not use $S[n]$.)

(4) **(Restricted traveling salesperson)** (30 points) Given a set S of points in the two-dimensional (x, y) plane, the *traveling salesperson problem* is to find a closed polygon whose vertices are the points in S and that minimizes the perimeter of the polygon, in other words, the sum of the lengths of the edges in the polygon. Such a polygon is a series of line segments between points in S that touches every point in S exactly *once*, and that returns to the point at which it began.

To simplify the problem, we will *restrict* the polygons that we consider. Assume the points in S are indexed $p_1, p_2, \ldots, p_n$, sorted according to increasing x-coordinate, and that their x-coordinates are distinct. (So for $p_i = (x_i, y_i)$, we have $x_1 < \cdots < x_n$.) We only consider restricted polygons that:

(1) starting from p_1, follow a series of line segments to points that are *increasing* in x-coordinate, until reaching p_n, and then
(2) continuing from p_n, follow a series of line segments to points that are *decreasing* in x-coordinate, until returning to p_1.

In other words, in walking around the exterior of such a restricted polygon from vertex p_1, the vertices first form a left-to-right sequence in which x-coordinates are increasing, and then form a right-to-left sequence in which x-coordinates are decreasing.

Design a dynamic programming algorithm for this *restricted form* of the traveling salesperson problem that finds an optimal restricted polygon in $O(n^2)$ time, given a set S with n points.

(Note: It may help you to represent such a polygon as a binary string $s_2 s_3 \cdots s_{n-1}$, where character s_i is either `L` or `R` depending on whether point (x_i, y_i) is on the "left-to-right" or "right-to-left" portions of the polygon.)

(Hint: In working out the structure of an optimal solution, ask how an optimal solution ends. Note that on removing the end of a solution, what remains is no longer a closed polygon. So the form of the recursive subproblem that you solve may be different than the form of the original problem. Also, you may need to develop recurrences for *two* quantities that depend on each other.)

(5) **(bonus) (Increasing subsequence)** (10 points) Given a string S of distinct numbers, an *increasing subsequence* of S is any subsequence $\widetilde{S}$ of S such that the numbers in $\widetilde{S}$ are monotonic increasing.

Using *dynamic programming*, design an algorithm that finds a *longest* increasing subsequence of a string of length n in $O(n \log n)$ time.

(Hint: It may be helpful to first design a dynamic programming algorithm that runs in $\Theta(n^2)$ time, and then figure out how to evaluate its recurrence equation faster.)

Note that Problem (5) is a *bonus* question. It is *not* required, and its points are not added to regular points.