CSE Ph.D. Qualifying Exam, Fall 2024
Algorithms

Instructions:

Please answer three of the following four questions. All questions are graded on a scale of 10. If you answer all four, all answers will be graded and the three lowest scores will be used in computing your total. **This exam is closed-book, closed-notes.**

Questions:

1. **Greedy**

   Consider a sequence of events that occur over time. Purchases at stock exchanges – what's being bought – are one source of data with a natural ordering in time. Given a long sequence S of such events, we want an efficient way to detect certain patterns in them. For example, we may want to know if the four events

   buy Google, buy Amazon, buy Google, buy Nvidia

   occur in the sequence S, in order but not necessarily consecutively (i.e. there may be other events in between).

   Begin with a collection of possible *events* (e.g., the possible transactions) and a sequence S of n of these events. A given event may occur multiple times in S (e.g., Google stock may be bought many times in a single sequence S). We will say that a sequence S' is a subsequence of S if there is a way to delete certain of the events from S so that the remaining events, in order, are equal to the sequence S'. So, for example, the sequence of four events above is a subsequence of the sequence

   buy Apple, buy Google, buy Amazon, buy Google, buy Tesla, buy Nvidia

   Our goal is to be able to take short sequences and detect whether they are subsequences of S. Provide a greedy algorithm that takes two sequences of events – S' of length m and S of length n, each possibly containing an event more than once – and decides whether S' is a subsequence of S. What is the running time of your algorithm? Prove its optimality. (Hint: show that your greedy algorithm is always ahead).

2. **Divide and Conquer, Dynamic Programming: Maximum Subarray**

   Given a one-dimensional array of integers of length n, the *maximum subarray problem* is the task of finding the largest possible sum of a contiguous subarray.

   For example, suppose that we have an an array $A = [-1, 1, -3, 4, -1, 2, 1, -5, 4]$. The contiguous subarray with the largest sum is $[4, -1, 2, 1]$ with sum 6.

   (a) Give a dynamic programming algorithm for the maximum subarray problem. Please define the subproblem and provide the recurrence relations. Additionally, give its asymptotic time and space complexity in theta notation. Pseudocode is not required.

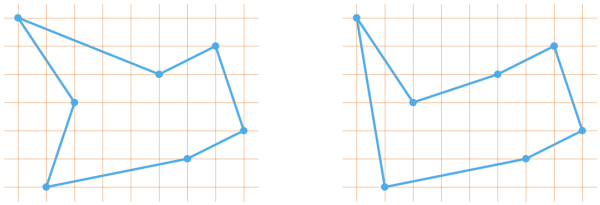

Figure 1: Seven points in the plane, shown on a unit grid. (a) The shortest closed tour, with length approximately 24.89. This tour is not bitonic. (b) The shortest bitonic tour for the same set of points. Its length is approximately 25.58.

(b) Give a divide-and-conquer algorithm for the maximum subarray problem that runs in $O(n \log n)$ time. Provide pseudocode to illustrate your algorithm. Analyze its time complexity.

3. **Dynamic Programming: Euclidean Traveling Salesman**

   In the *euclidean traveling-salesperson problem*, you are given a set of n points in the plane, and your goal is to find the shortest closed tour that connects all n points.

   The general problem is NP-hard, and its solution is therefore believed to require more than polynomial time.

   J. L. Bentley has suggested simplifying the problem by considering only bitonic tours, that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point. Figure 1(b) shows the shortest bitonic tour of the same points. In this case, a polynomial-time algorithm is possible.

   Please provide a dynamic programming algorithm that runs in $O(n^2)$-time for determining an optimal bitonic tour. Define the subproblem and provide the recurrence relations. Provide pseudocode of your implementation and analyze the time complexity. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time. (Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.)

4. **NP-complete: Vertex Cycle Cover**

   Given a *directed* graph $G = \{V, E\}$, a VertexCycleCover problem is: does there exist a subset of vertices S, where $S \subseteq V$ and $|S| \leq k$, such that every simple cycle in G passes through at least one vertex in S?

   Prove that the VertexCycleCover problem is NP-complete using that the VertexCover problem is NP-complete. Remember to include all steps of NP-completeness proof.

   VertexCover(G, k) is the problem of deciding whether in graph G there exists a subset of vertices S of size at most k such that all edges in G have at least one endpoint in the selected vertex set S. A *simple cycle* in a graph is a cycle with no repeated vertices (except for the start and end vertex).