CS/ECE 374 A (Spring 2022)
Midterm Exam 2

Instructions:

- Date/Time: April 11 Monday 7:00pm–9:30pm Central Time.
- Except for your one double-sided handwritten $8.5'' \times 11''$ cheat sheet, exams are **closed-everything**. In particular: No medically unnecessarily electronic devices are allowed in exams, including smart watches and headphones/earbuds.
- You will write your solutions on paper, scan your solutions using a scanning app, convert your scans to PDF, and upload the PDF to Gradescope. (You are **not** allowed to write your solutions on tablets, unlike in some past semesters.) Gradescope will only accept PDF submissions. Please make sure your solution to each numbered problem starts on a new page of your submitted PDF file. Please do not scan your cheat sheets or scratch paper.
- You have **two hours (i.e., 120 minutes)** to write your solutions. We are providing 30 minutes at the end of each exam period for students to scan and upload their exams, and to deal with any unexpected technical difficulties that may arise. Gradescope will stop accepting submissions for each midterm precisely at 9:30pm. We will not accept submissions after the Gradescope deadline. **All work on the exam must stop 30 minutes before the Gradescope deadline, i.e., at 9:00pm.**
- Please make sure that your scans are clear and easy to read. We very strongly recommend that you write with black ink on unlined white paper. Submissions consisting of raw photos or low-quality scans will be penalized.
- If you are ready to scan your solutions before 9:00pm, send a private message to the host of your Zoom call ("Ready to scan") and wait for confirmation before leaving the Zoom call.
- If something goes seriously wrong during the exam, send email to Timothy (tmc@illinois.edu) as soon as possible explaining the situation. If you have already finished the exam but cannot submit to Gradescope for some reason, include a complete scan of your exam as a PDF file in your email. If you are in the middle of the exam, send email to Timothy, continue working until the time limit, and then send a second email with your completed exam as a PDF file. Please do not email raw photos.
- You are reminded about the course's, the department's, and the university's academic integrity policies.
- Avoid the deadly sins. Write complete solutions, not examples. Declare all your variables. Write concisely and precisely.
- Good luck!

---

1. [28 PTS] *Short questions.*

   (a) [6 PTS] Give an asymptotically tight solution to the following recurrence:

   $$T(n) = \begin{cases} 3T(\lfloor n/2 \rfloor) + n + 4n^2 & \text{if } n \geq 374 \\ 2022 & \text{if } n < 374. \end{cases}$$

   Justify your answer.

   (b) [3 PTS] True or False: The median-of-medians-of-5 algorithm given in class is faster than mergesort, when the input size is sufficiently large. Briefly justify your answer.

   (c) [3 PTS] Consider the following pseudocode to compute the number 374^n:

   ```
   1. X = 1
   2. for i = 1 to n do
   3.       X = 374 * X
   4. return X
   ```

   What is the running time as a function of n, assuming that arithmetic operations take constant time?

   (d) [3 PTS] In the same pseudocode as (c), what is the running time as a function of n, when we take into account the bit complexity of the numbers? Justify your answer.

   (e) [3 PTS] Given a directed graph $G = (V, E)$, consider a DFS tree T. True or False: if $(u, v) \in E$, then the finishing time for u must be greater than the finishing time for v. If true, justify your answer; if false, give a counterexample.

   (f) [3 PTS] Answer the same question as (e) but for a DAG G.

   (g) [3 PTS] True or False: In a directed graph G where every vertex has out-degree at least one, G must have a cycle. Justify your answer.

   (h) [4 PTS] Which algorithm from class gives the most efficient solution to the all-pairs shortest paths (APSP) problem in the case when the input graph is sparse and has $m = O(n)$ edges, and all edges have weight 1? (Here, n denotes the number of vertices.) Justify your answer.

2. [20 PTS] We are given an array A of n numbers $\langle A[1], \ldots, A[n] \rangle$, where n is a power of 2. We want to perform a *perfect split*, i.e., change the array to the following:

   $$A[1], A[3], A[5], \ldots, A[n-1], A[2], A[4], A[6], \ldots, A[n].$$

   For example, if the array initially contains $\langle 3, 11, 2, 9, 100, 1, 25, 5 \rangle$, it should contain $\langle 3, 2, 100, 25, 11, 9, 1, 5 \rangle$ at the end.

   Now, the problem can easily be solved in $O(n)$ time by copying to an extra array of $O(n)$ size, and then copying back. However, in this question, we want an algorithm that is both time-efficient and space-efficient.

   Describe a divide-and-conquer algorithm to solve this problem using just $O(1)$ (or $O(\log n)$) extra space. Give pseudocode using recursion, and analyze its running time using a recurrence. The running time should be strictly better than $O(n^2)$ to receive full credit.

3. [28 PTS] For a sequence $\langle b_1, \ldots, b_m \rangle$, we say that position i is an *up-turn* iff $i \geq 2$ and $b_i > b_{i-1}$, and position i is a *down-turn* iff $i \geq 2$ and $b_i < b_{i-1}$.

Consider the following problem:

> Given a sequence $\langle a_1, \ldots, a_n \rangle$ and a number $K \leq n$, find a longest subsequence that has at most K up-turns. (You may assume that the a_i's are all distinct.)

Consider the following dynamic programming approach. Define the following subproblems: for each $i \in \{1, \ldots, n\}$ and $k \in \{0, \ldots, K\}$, let $C(i, k)$ be the length of the longest subsequence of $\langle a_i, \ldots, a_n \rangle$ such that the first element is a_i and the number of up-turns is at most k.

We have the following recursive formula: for each $i \in \{1, \ldots, n-1\}$ and $k \in \{0, \ldots, K\}$,

$$C(i,k) = \max \begin{cases} 1 \\ \max\limits_{j \in \{i+1,\ldots,n\} \text{ such that } a_j > a_i} (C(j, k-1) + 1) \\ \max\limits_{j \in \{i+1,\ldots,n\} \text{ such that } a_j < a_i} (C(j, k) + 1) \end{cases}$$

For the base case, $C(i, -1) = -\infty$ for all $i \in \{1, \ldots, n\}$. (You may assume correctness of the above formula without proof.)

(a) [15 PTS] Write pseudocode to solve the problem using the above formula. Your pseudocode should output the length of the longest subsequence of the original problem (you are not required to output the optimal subsequence itself). Analyze the running time of your pseudocode as a function of n and K (it must run in polynomial time).

(b) [13 PTS] Consider the following modified problem:

> Given a sequence $\langle a_1, \ldots, a_n \rangle$ and a number $L \leq n$, find a subsequence of exactly length L such that the number of up-turns is equal to the number of down-turns, while maximizing the sum of the elements in the subsequence. (You may assume that the a_i's are all distinct.)

Describe a dynamic programming algorithm to solve this modified problem, i.e., give a new definition of subproblems, give recursive formulas (including base cases, and how to obtain the optimal value), and specify the evaluation order. You do not need to give pseudocode for this part of the question (and you do not need to give justifications of your formula if it is clear enough). Analyze the running time of your algorithm as a function of n and L.

[Hint: you may want to add one more parameter to $C(\cdot)$, i.e., use a three-dimensional table.]

4. [24 PTS]

(a) [9 PTS] We are given a directed graph $G = V, E)$ with n vertices and m edges (with $m \geq n$). We want to decide whether there exists a closed walk that visits a majority of the vertices, i.e., at least $n/2$ distinct vertices of V would be visited by the walk. Describe an efficient solution to this problem by applying a known algorithm. Briefly justify correctness, and bound the running time.

(b) [15 PTS] We are given a weighted directed graph G with n vertices and m edges (with $m \geq n$). We are also given two vertices s and t.

> We want to compute a walk from s to t that alternates between positive-weight edges and negative-weight edges and also has positive total weight. (Note that the walk could start with either a positive-weight edge or negative-weight edge, i.e., the pattern could be $+ - + - + - \cdots$ or $- + - + - + \cdots$.)

Describe an efficient solution to this problem by constructing a new graph and applying a known algorithm. Remember to clearly define the vertices and edges of your new graph, bound the number of vertices and edges, and analyze the total running time.