CSIE 2136: Algorithm Design and Analysis (Fall 2021)

Midterm

Time: 14:20-17:20 (180 minutes), November 11 2021

Instructions

- This is a 3-hour closed-book exam, and there are seven questions worth a total of 120 points + 15 bonus points. The maximum of your score is 100, so you can allocate your time and select problems based on certain *optimal strategies* of your own.
- Please write clearly and concisely; avoid giving irrelevant information.
- You have access to the appendix on the last page, and you can use basic data structures (i.e., array, stack, queue, deque, heap, balanced search tree, doubly linked list) without writing their implementation details. In addition, you can assume that the time complexity of **sorting** n numbers is $O(n \log n)$.
- Please use a **separate** answer page for each problem. Please write down **your name, student ID, and problem number** in the **upper-right** corner of **every** answer page (see the example below). You will get **2 points** for complying with this policy.

Name: ADA Student ID: B09902xxx Problem K

- **You need to prove the correctness and time complexity of the algorithm you provide in all problems unless the problem does not require**.
- Grading policy: If a problem asks you to design an $O(f(n))$ algorithm, you can get partial points if you design an algorithm worse than $O(f(n))$. However, we will not accept any algorithm running in exponential time or $O(n!)$ unless the problem asks you to do so.

Problem Outline

- Problem 1 - Short Answer Questions (25 points)
- Problem 2 - Alice and Bob (10 points)
- Problem 3 - Squid Game (20 points)
- Problem 4 - Amusement Park (20 points)
- Problem 5 - Island Hopping (20 points + 5 bonus points)
- Problem 6 - $\mathcal{P}$ (20 points + 10 bonus points)
- Problem 7 - Feedback (3 points)
- Name, ID, and problem number on each page (2 points)

Problem 1 - Short Answer Questions (25 points)

1-(a) (3 points) Please explain what it means for a problem to have the greedy-choice property.

1-(b) (3 points) Given that there are N items and a bag of capacity W, explain why the $O(NW)$-time 0/1 knapsack algorithm mentioned in class is pseudo-polynomial.

1-(c) (2 points) Given two non-negative and monotonically increasing functions $f(n)$ and $g(n)$, please provide a counterexample and brief explanation to show that the following statement regarding their "little-o" relationships is **false**:

$$\text{If } e^{f(n)} = o(e^{g(n)}), \text{ then } f(n) = o(g(n))$$

1-(d) (3 points) Given the recurrence relation $dp(i,j) = F(dp(i-2, j+1), dp(i+1, j-2))$, where $F(\cdot,\cdot)$ is a known function, provide a valid traversal order to fill the dynamic-programming table or justify why no valid traversal exists. You can assume $dp(i,j) = 0$ if either $i < 0$ or $j < 0$.

1-(e) (4 points) Suppose there are n students whose student IDs are numbered from 1 to n. At the end of an exam, TAs collect students' answer sheets and sort them by student IDs. However, the TAs soon realize that there are only $n-1$ answer sheets; one is missing. Please describe a $O(\log n)$-time algorithm to identify whose answer sheet is missing, assuming you can access the answer sheet of any index in $O(1)$ time.

1-(f) (2 points) Can the above problem modeling be applied to the soon-will-happen reality (i.e., the ADA TAs will collect your answer sheets and have to make sure everyone's is collected)? Why or why not?

1-(g) (5 points) In class, we have demonstrated a linear-time divide-and-conquer algorithm for solving the selection problem, where the numbers are first divided into groups of 5. Now consider a similar algorithm that divides the numbers into **groups of 4**. In this algorithm, each *odd* group treats its second-largest number as the median, and each *even* group treats its third-largest number as the median. Please derive the time complexity of this algorithm by solving its recurrence relation.

1-(h) (3 points) Please derive the asymptotic tight bound (Θ-notation) for the following recurrence relation, assuming $T(n) = 1$ for all $n \leq 1$.

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + \frac{n}{127}$$

Problem 2 - Alice and Bob (10 points)

Alice and Bob are good friends. They shop together in a shopping mall one day and see N items without price labels. Alice guesses the price of the i-th item is a_i, and Bob guesses the price of the i-th item is b_i.

Eve, who is jealous of their friendship, tells them that they are not a good match if there are many *bad pairs* in their price estimation. A pair (i, j) is a bad pair if the following two conditions are met:

- $a_i < a_j$
- $b_i > b_j$

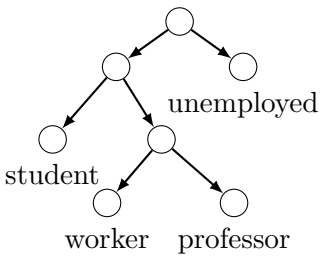
Note that there might be duplicate values in the $\{a_i\}$ and $\{b_i\}$ sequences.

To argue against Eve, please design a **divide-and-conquer** algorithm with time complexity $O(N \log N)$ to find the number of bad pairs. You will get a maximum of 7 points as a partial score if your algorithm can only handle distinct values.

Problem 3 - Squid Game (20 points)

You accidentally participated in the "Squid Game", but it is difficult to escape from the game. Please answer the following questions.

3-(a) (6 points) Before the game starts, you find a notebook that records the winners' occupations of all previous games using Huffman coding illustrated below.



As a current participant, you want to know more about the previous winners' information. Please **explain** whether each of following statements is **True** or **False**:

- The frequency of *unemployed* participants is no less than it of *students*.
- The *unemployed* participants must have a frequency higher than or equal to 1/3.
- The *workers* must have a frequency less than 1/8.

3-(b) (4 points) The first game starts, you are given a machine with two buttons (say 0-button and 1-button), which can be used to send messages to your partner. Your partner is smart enough to crack any new encoding scheme. However, the 1-button is stuck and takes double effort to press than pressing the 0-button. In order to send the message as soon as possible, you want to minimize the effort of pressing buttons. You first think about the following divide-and-conquer approach to design a prefix code:
Given n symbols and their frequencies f_i $(0 \leq i < n)$,

- If $n \leq 1$, return an empty codeword.
- *Divide* Sort the n symbols by their frequencies from left to right, and then divide them into two groups, G_l and G_r, such that the sum of frequencies in the left group is as close to the double of the sum in the right as possible (If there are multiple such divisions, you can select any of them). For example, if the sorted frequencies are $0.3, 0.25, 0.2, 0.1, 0.1, 0.05$, then $G_l = \{0.3, 0.25\}$ and $G_r = \{0.2, 0.1, 0.1, 0.05\}$.
- *Conquer* Solve G_l and G_r recursively, and prepend 0 to the codewords in G_l, and prepend 1 to the codewords in G_r.
- *Combine* Nothing to be done.

Please show that this divide-and-conquer approach cannot produce a prefix code that minimizes the effort of pressing the buttons.

3-(c) (4 points) The 1-button is almost broken. Hence, you would like to limit its usage to at most once per codeword. Please construct an optimal prefix code that minimizes the expected length of codeword given this constraint. Briefly justify the correctness of your construction.

3-(d) (6 points) A staff member updated the button panel, and now the machine has three fully functional buttons that can be used to send different signals. Hence, you extend the Huffman coding to build a 3-ary prefix tree: Given a set of symbols and their frequencies, you first add zero-probability symbols such that the total number of symbols is $n \equiv 1 \pmod 3$. You create one tree node per symbol and sort the nodes by their frequencies. You repeatedly create a new parent node to the three least frequent nodes, similar to the original Huffman coding algorithm. Please extend the proof of the greedy choice property to this case.

Problem 4 - Amusement Park (20 points)

As the government lifts the COVID-19 restrictions, many people rush to a newly opened amusement park. Suppose there are N groups of people lining up to ride a roller coaster. The i-th group has p_i people, and groups are ordered by their arrival time. The roller coaster has R seats and thus can take at most R people in each ride, and $1 \leq p_i \leq R$ for all i. Your job is to put people onto the roller coaster while satisfying the following three rules:

1. Everyone should be served.
2. People in the same group should take the same ride.
3. First come, first served. That is, $ride(i) \leq ride(j)$ for all $i < j$, where $ride(i)$ denotes when the i-th group is served.

4-(a) (7 points) To maximize the profit, your manager asks you to minimize the total number of empty seats. That is, suppose your algorithm divides the groups into K rides, your algorithm should minimize $\sum_{k=1}^{K} \left(R - \sum_{ride(i)=k} p_i \right)$. Please design a greedy algorithm that runs in $O(N)$ time and prove its correctness.

4-(b) (3 points) Your manager found out that the cost of operating each ride is proportional to the cube of the number of empty seats, and thus asks you to minimize the total cost, which can be formulated as $\sum_{k=1}^{K} \left(R - \sum_{ride(i)=k} p_i \right)^3$.

Please show that minimizing this total cost is NOT equivalent to minimizing the number of rides. Providing a counterexample is sufficient.

4-(c) (10 points) Now suppose there are two lines: VIP and regular. Groups in the VIP line paid more and may be served faster than expected (see the 4th rule below). For ease of description, we denote that the i-th group in the VIP line has p_i^{vip} people, arrives at t_i^{vip} and is served by the $ride^{vip}(i)$-th ride; the i-th group in the regular line has p_i^{re} people, arrives at t_i^{re} and is served by the $ride^{re}(i)$-th ride.

Your job is to put people onto the roller coaster while satisfying the following four rules; notice that the 3rd one is modified and the 4th is new:

1. Everyone should be served.
2. People in the same group should take the same ride.
3. First come, first served, *within the same line*. That is, for all $i < j$, $ride^{vip}(i) \leq ride^{vip}(j)$ and $ride^{re}(i) \leq ride^{re}(j)$.
4. A VIP group should *never be served after* a regular group that arrives later; a VIP group *may be served before* a regular group that arrives earlier. That is, $ride^{vip}(i) \leq ride^{re}(j)$ for all $t_i^{vip} \leq t_j^{re}$.

Please design a $O(mn)$ dynamic programming algorithm to minimize the number of rides, where m and n are the number of VIP and regular groups, respectively.

Problem 5 - Island Hopping (20 points + 5 bonus points)

As a travel lover, Vivian is going on an island hopping on an $n \times m$ (you may assume that $n \times m \geq 2$ in all the following subproblems) matrix on the coming winter vacation. The trip will start from $(1, 1)$ and end at (n, m). To put her plan into practice, she has prepared a map and a malfunctioned cruise ship.

The map is a binary matrix $B_{n\times m} = \big(b_{ij}\big)$, where $b_{ij} = 1$ if and only if there is a travelable island located on (i, j). By convention, $b_{11} = b_{nm} = 1$.

The malfunctioned cruise trip brings Vivian to hop from one island to another. However, since the ship is malfunctioned, there are several restrictions when hopping between islands. More precisely, a hop from (x, y) to (x', y') is valid if all of the following conditions hold:

- $b_{x'y'} = 1$
- $x \leq x' \leq n$
- $y \leq y' \leq m$
- $1 \leq (x' - x) + (y' - y) \leq k$

Given n, m, B, k, Vivian wonders the number H of the possible island-hopping plans.

For example, if $n = 1, m = 5, k = 3, B = \begin{pmatrix}1 & 0 & 1 & 1 & 1\end{pmatrix}$, then $H = 3$ since there are 3 possible plans:

- $(1, 1) \to (1, 3) \to (1, 4) \to (1, 5)$
- $(1, 1) \to (1, 3) \to (1, 5)$
- $(1, 1) \to (1, 4) \to (1, 5)$

5-(a) (3 points) Show that if $n = 1, k = 2$ and $b_{11} = b_{12} = \cdots = b_{1m} = 1$, then $H = F_m$, where F is the Fibonacci sequence defined as follow: $\begin{cases} F_1 = F_2 = 1 \\ F_{k+2} = F_{k+1} + F_k, \forall k \in \mathbb{N} \end{cases}$.

5-(b) (2 points) Please find H for $n = 3, m = 3, k = 2, B = \begin{pmatrix}1 & 1 & 0\\ 1 & 1 & 1\\ 1 & 0 & 1\end{pmatrix}$. We will only grade the result so you will receive either 2 or 0 points from this subproblem.

5-(c) (8 points + bonus 5 points) = (D-score: 0.8 + bonus 0.5 points) × (E-score: 0 ∼ 10) Please come up with algorithms that could find H.

- For the detailed grading policy to this subproblem, please read the last section of this problem (i.e. Problem 5). Briefly speaking,
  - You will get (D-score, E-score) = $(0.8, 10)$ if you come up with a completely correct algorithm running in $O(nmk)$ time.
  - You will get a higher D-score if your algorithm runs in $o(nmk)$ time.

Vivian and HC are good friends. Knowing Vivian is going island-hopping this winter vacation, HC decides to do the similar thing. However, HC wants to come up with an extraordinary island-hopping plan so that for any island I except the starting and ending island in her plan, I doesn't appear in Vivian's traveling plan. Vivian and HC wonder the number J of such pair of plans.

For example, in the example where $n = 1, m = 5, k = 3, B = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \end{pmatrix}$, $J = 2$ because there are 2 possible pairs of plans:

- $\begin{cases} \text{Vivian's plan: } (1,1) \to (1,3) \to (1,5) \\ \text{HC's plan: } (1,1) \to (1,4) \to (1,5) \end{cases}$
- $\begin{cases} \text{Vivian's plan: } (1,1) \to (1,4) \to (1,5) \\ \text{HC's plan: } (1,1) \to (1,3) \to (1,5) \end{cases}$

5-(d) (7 points) For $\boldsymbol{k = 1}$, please come up with a correct algorithm that could find J in $O\left((n+m)\min(n,m)^2\right)$ time.

.....

Grading Policy of Subproblem 5-(c)

Read this only if you're interested in the grading details.

We define a **solution** S as a combination of (an algorithm A, a time complexity $O(f)$), where the algorithm A is supposed to run in $O(f)$ time.

- Determined by two metrics, the grading of one solution is just a generalized scoring system in gymnastics in the Olympics.
  - **D-score**, as known as the difficulty score, depends on the time complexity you claim that your algorithm could achieve.
    - The better its time complexity is, the more bonus points you could obtain.
    - As a concrete hint, if your solution is claimed to run in $O(nmk)$ time, you would get 0.8 points from this metric.
    - You would receive bonus points by coming up with an algorithm running in $o(nmk)$ time.
  - **E-score**, as known as the execution score, depends on the correctness and completeness of your proof.
    - You would receive an integer value from 0 to 10 from this metric.
    - The correspondence between "the time complexity you've claimed" and "the actual time complexity of your algorithm" is considered a part of correctness. Warning: if they are not correspondent, you may receive a really poor E-score.

- You are allowed to write some (possibly zero, one or multiple) solutions in this subproblem.
  - Please clearly enumerate your solutions with 1-based indexing.
  - Please clearly claim the time complexity and show the correctness of your algorithm for **each** of your solution.
  - If you've come up with z solution(s) which are $S_1, S_2, \ldots, S_z$ and the D-score and E-score on each solution are $D_1, D_2, \ldots, D_z$ and $E_1, E_2, \ldots, E_z$, respectively, then you would get $\begin{cases} 0, & \text{if } z = 0 \\ \max\limits_{i=1}^{z} (D_i \times E_i), & \text{if } z \in \mathbb{N} \end{cases}$ points from this problem.

Problem 6 - $\mathcal{P}$ (20 points + 10 bonus points)

Part 1

$\mathcal{P}$ is a fan of sequence problems, especially the "maximum subarray problem". Now, $\mathcal{P}$ gives you a sequence $(a_1, a_2, \ldots, a_N)$ of length N and a constant K. $\mathcal{P}$ wants to remove **at most** K **elements** from the sequence such that the maximum subarray sum of the result sequence is maximized. Specifically, the sum of an empty sequence is 0.

6-(a) (2 points) Please solve a simplifed problem with an additional constraint $K = 0$ using dynamic programming in $O(N)$ time. The proof of correctness and time complexity is not required.

6-(b) (5 points) Please solve the problem with dynamic programming in $O(NK)$ time.

Part 2

However, $\mathcal{P}$ thinks $O(NK)$ is too slow, so he finds something interesting.

He defines $f(i, j)$ as the **maximum sum** of $a_i, a_{i+1}, \ldots, a_j$ where at most K elements can be deleted if $i \leq j$. Specifically, the sum of an empty sequence is 0. If $i > j$, he defined $f(i, j)$ as $\min(0, a_1, a_2, \ldots, a_N) \times N - (i - j)$.

6-(c) (1 point) Given $i < j$ and let $m = j - i + 1$, calculate $f(i, j)$ in $O(m \log m)$ time. The proof of correctness and time complexity is not required.

Part 3

Also, $\mathcal{P}$ finds "totally monotone matrix" interesting. An $N \times M$ matrix A is called a totally monotone matrix if every submatrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ of A satisfies the following two conditions:

1. if $c > d$, then $a > b$.
2. if $c = d$, then $a \geq b$

Note that a submatrix does not have to consist of adjacent rows or columns. That is, $\begin{pmatrix} 1 & 3 \\ 7 & 9 \end{pmatrix}$ is a submatrix of $\begin{pmatrix} 1 & 8 & 3 \\ 4 & 10 & 6 \\ 7 & 8 & 9 \end{pmatrix}$, also, the previous matrix is a totally monotone matrix.

6-(d) (bonus 10 points) Let B be an $N \times M$ matrix where $B_{i,j} = f(i,j)$, prove that B is a totally monotone matrix.

6-(e) (3 points) Let $h(i)$ be the index of the column containing the leftmost maximum element of row i in an $n \times m$ matrix. Prove that $h(1) \leq h(2) \leq \cdots \leq h(n)$ for any $n \times m$ totally monotone matrix.

6-(f) (9 points) Assume that you can calculate an entry in an $n \times m$ totally monotone matrix in $g(n,m)$ time. Please design a divide and conquer algorithm to find $h(1), h(2), \ldots, h(n)$ in $O((n + m \log n)g(n,m))$ time.

(Hint: Consider divide the rows into even-indexed rows and odd-indexed rows. Try to get the answers of odd-indexed rows by even-indexed rows.)

After solving the above problems, $\mathcal{P}$ is able to solve the original in $O(n \log n g(n,n))$ time. If $\mathcal{P}$ can calculate $g(n,n)$ efficiently, he can solve the problem better then $O(NK)$.

Problem 7 - Feedback (3 points)

We have the tradition of letting students write feedbacks about the course in the exam. Please write down 3 things you like about this course and 3 things that you would like to see some changes (and your suggestion about how we should change them).

Appendix

Asymptotic Notations

1. **Θ-notation**:

$$\Theta(g(n)) = \{f(n) \ : \ \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$$

2. ***O*-notation**:

$$O(g(n)) = \{f(n) \ : \ \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

3. ***o*-notation**:

$$o(g(n)) = \{f(n) \ : \ \text{for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}$$

Master Theorem Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.