CSIE 2136: Algorithm Design and Analysis (Fall 2022)

Midterm

Time: 14:30-17:30 (180 minutes), October 27 2022

Instructions

- This is a 3-hour closed-book exam. There are 7 problems worth a total of 125 points. If your raw score exceeds 100, it will be capped to 100.
- Please write clearly and concisely; avoid giving irrelevant information.
- You are allowed to use basic data structures (limited to array, stack, queue, deque, heap, balanced search tree, doubly linked list) without writing their implementation details. In addition, you can assume that sorting N numbers runs in $O(N \log N)$.
- Please use the assigned answer sheets for each problem. For each page, please write down **your name** and **student ID** on every paper. You will get **3 points** for complying with this policy.
- You **do not** need to prove the correctness and time complexity of your algorithms unless a problem requires you to do so.
- *TL; DR*: For each problem, if the description starts with italic text *TL; DR*, the following lines contains only interesting story with no critical information. Thus, you may rush into the subproblems if you would like to save some time.
- Answer sheet notations
  - 🖨: Your answers in the blanks labeled with scanning machines may be graded in a fancy way like auto judging by AI. So please write down your answer clearly and properly large especially in these cases.
  - 🔑: The words after this symbol are the keywords of the corresponding task. This is simply a friendly design to prevent you from misplacing your answer.

Problem Outline

- Problem 1 - Is it? (10 points)
- Problem 2 - What is it? (12 points)
- Problem 3 - Who is it? (22 points)
- Problem 4 - Why is it? (20 points)
- Problem 5 - Where is it? (25 points)
- Problem 6 - When is it? (30 points)
- Problem 7 - How is it? (3 points)
- Name, ID, and problem number on each page (3 points)

Problem 1 - Is it? (10 points)

TL; DR: **True** (◯) *or* **False** (×). *No explanation required.*

"Is it a great idea to have 10 true-or-false questions as the first problem?"
"No, that's too much. Students might think we are boolean-ing them."

(a) (2 points) If $f(n) = O(g(n))$ and $g(n) = O(f(n))$, then there exists an n_0 such that $f(n) = g(n)$ for all $n > n_0$.

(b) (2 points) In our lecture, 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. A similar algorithm that divides the numbers into groups of 7 also solves the selection problem in linear time.

(c) (2 points) In our lecture, we have demonstrated an algorithm has the optimal substructure for solving the matrix-chain multiplication problem. In the problem of outputting an order of performing $(n-1)$ matrix multiplications in the **maximum** number of operations to obtain the product of $A_1A_2 \cdots A_n$, the optimal substructure still holds in the similar algorithm.

(d) (2 points) If one proposes a greedy algorithm A to solve problem P but it turns out to be rejected by a counterexample, then we can conclude that there does not exist any greedy algorithm able to solve P.

(e) (2 points) When constructing a k-ary prefix tree, in which each node can have at most k children, all non-leaf nodes must have k children in an optimal prefix tree.

Problem 2 - What is it? (12 points)

TL; DR: Explain each term within 3 sentences.

"Do you know what it is?", an annoying ad with Ellen speaking pops up while you're watching YouTube; "though I'm not a mathematician, it sounds great, doesn't it?", you're then extremely irritated upon hearing this sentence – just like most YouTube users.

But now, you get a chance to keep yourself from these toxic words. If you are able to **explain "what it is"** immediately after Ellen's asking "do you know what it is?", you can interrupt Ellen and skip the ad. Though I'm not a trader, it sounds like a good deal, doesn't it?

Since you need to finish your explanation before the ad ends itself, you should write your answer to **each question within 3 sentences**. You will get no point if you write "yes or no" as your answer.

(Ad 1) (3 points) "Do you know what asymptotic notation ω is?"

(Ad 2) (3 points) "Do you know what optimal substructure is?"

(Ad 3) (3 points) "Do you know what memoization technique in dynamic programming is?"

(Ad 4) (3 points) "Do you know what pseudo polynomial is?"

Problem 3 - Who is it? (22 points)

TL; DR: Design an algorithm or do some analysis according to the task requirement. There's no dependency between the tasks in this problem so you can do them in any order.

"There must be a student skipping all ADA lectures this year. They have not even clicked the course page on NTU COOL before."

"Who is it?"

"I don't know either. Maybe we can find them by setting some problems highly related to what is taught in the lectures."

(a) (6 points) Let's briefly analyze the asymptotically tightest bound (with Θ-notation) for the following recurrence relation, assuming $F(x) = 1$ for all $x \le 1$, by completing the following steps.

$$F(n) = F\left(\frac{n}{2}\right) + F\left(\frac{n}{3}\right) + 10^{11} \cdot n^2$$

(i) (3 points) Sketch a recursion tree representing this recurrence relation above.

(ii) (3 points) By observing the recursion tree you've drawn, give a guess. You will either get 3 points for giving a correct guess or 0 point for an incorrect one.

If you argue that this is not rigorous enough as a proof, you're correct. Recursion tree is usually used just for giving a rough estimation on time complexity when an algorithm is proposed.

(b) (10 points) Let's find the asymptotically tightest bound (with Θ-notation) for the following recurrence relation, assuming $G(x) = 1$ for all $x \le 1$, by showing the correctness of the following claims.

$$G(n) = G\left(\frac{5n}{6}\right) + G\left(\frac{2n}{3}\right) + G\left(\frac{n}{2}\right) + n$$

(i) (5 points) Show that $G(n) = \Omega(n^3)$ by giving an induction proof.

(ii) (5 points) Show that $G(n) = O(n^3)$ by giving an induction proof.

Hence, by (i) and (ii), $G(n) = \Theta(n^3)$.

(c) (6 points) In the weighted interval scheduling problem, the input contains n jobs, the i-th ($1 \le i \le n$) of which is specified by its starting time s_i, finishing time f_i and value v_i with $0 < s_i < f_i$. In addition, the tasks are indexed according to their finishing times in non-decreasing order so $f_i \le f_j$ when $i < j$.

In our lecture, we've derived the following recurrence relation which enables us to solve this problem with dynamic programming.

$$M_i = \begin{cases} 0 & \text{if } i = 0 \\ \max\left(v_i + M_{p(i)}, M_{i-1}\right) & \text{otherwise} \end{cases},$$

where $p(i)$ denotes the largest index $j < i$ such that jobs i and j are compatible. If there is no such j, $p(i)$ is defined as 0.

"But how to find $p(i)$? If we do it by iterating over all j's with $j < i$, wouldn't that be $O(n^2)$?" Casper proposed this question on Slido.

"That's a good question," Vivian replied in her live streaming, "maybe we can set it as one of our midterm problems." So here it is – please design an algorithm that fills out the p array in $O(n \log n)$ time.

Problem 4 - Why is it? (20 points)

While you are away from home for college, your family and friends send you boxes of snacks from your hometown to help you overcome homesickness. Today, which is day 1, you receive n boxes, and the i-th box has an expiration date e_i, and eating food in that box will prevent you from homesickness for the following d_i days, where $1 \le i \le n$ and n, e_i, $d_i \in \mathbb{N}$. Also, eating expired food (i.e. eating food with expiration date e at day t but $t > e$) gives you a stomach ache.

For simplicity, let's assume that you eat all food in a box instantly once unpacking it, and you can unpack as many boxes as you want at any time.

There are many ways to schedule when to unpack these boxes, but because they are so precious, you keep questioning yourselves "why is it the best way?" After a while, you finally decide to maximize the duration (in days) you are free from both homesickness and stomach ache. To do so, you have to answer the following questions:

(a) (5 points) Show that the following greedy choice is not optimal: when you are about to feel homesick, unpack the box with the earliest expiration date among those not yet expired.

(b) (8 points) Design an $O(n \log n)$ **greedy** algorithm; clearly state your greedy choice.

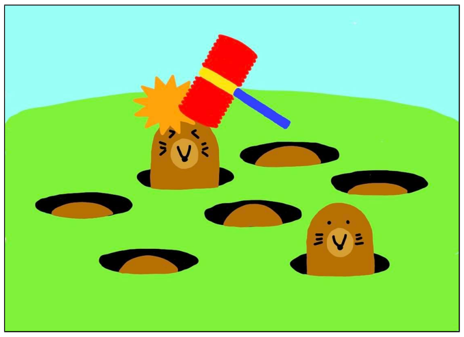
(c) (7 points) Prove that your greedy algorithm is optimal.

Problem 5 - Where is it? (25 points)

BB and his girlfriend are planning to go to Tom's world for tomorrow's date. To show off in front of his girlfriend, BB secretly investigates the **Whack-A-Mole** machine in Tom's world.

A typical Whack-A-Mole machine has a play area with several holes filled with plastic moles that pop up randomly. Points are scored by whacking each mole as it appears.[1]

You can see the following figure for a quick image understanding.



Since BB doesn't like to look flustered by "where is it?", he, in advance, does an in-depth analysis into the Whack-A-Mole machine. He observes that the positions and the times of the popped up moles are fixed. More precisely,

- There are N moles going to appear in sequence.
- The i-th mole appears at P_i with coordinate (x_i, y_i) at t_i seconds after the game starts.
- If the player whacks the i-th mole, the player scores a positive score w_i.
- $t_i < t_{i+1}, \ \forall 1 \leq i < N$.

To whack the i-th mole, the player should grab a mallet to hit it precisely at the coordinate (x_i, y_i) exactly at t_i seconds after the game starts. BB can start at any place and whack the first mole as he wants. However, he can only move a mallet for a unit distance within a unit time due to his speed limit. In other words, BB is able to whack two moles $i < j$ in a row **with the same mallet** if $d(P_i, P_j) \leq t_j - t_i$, where d denotes the Euclidean distance function. In two-dimensional plane, the Euclidean distance between two points (x_a, y_a) and (x_b, y_b) is given by $\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$.

Under the constraints above, BB needs to design an optimal strategy to maximize his score. Help him to solve the tasks on the next page.

Hint: Do not try to investigate the geometry properties of the Euclidean distance function. You are not expected to do so.

[1]Adapted from Wikipedia

For subproblem **(a)**, **(b)** and **(d)**-(i), your grade is mainly based on the state definition and transition function. For the state definition, remember to indicate what the answer will be after filling up your state table. For the transition function, don't forget the base case.

(a) (5 points) Initially, BB has only one mallet and he always grabs it with his right hand. Please design an $O\left(N^2\right)$ **dynamic programming** algorithm to determine the maximum score BB can get.

(b) (6 points) To score higher, BB would like to buy another mallet and grab it in his left hand. For now, he has two mallets, one in each hand, with similar constraints described above. You don't need to worry about BB's hands getting knotted.

Please design an $O(N^3)$ **dynamic programming** algorithm to determine the maximum score BB can get.

However, such an optimal strategy cannot fulfill BB's ambition. He is going to take advantage of his power of money to get a full score by purchasing some **disposable** teleporters to teleport his hands. With the consumption of one teleporter, one hand, along with the mallet it holds, can be instantly (i.e. in 0 unit time) teleported from its current coordinate to any coordinate at BB's own will. In this scenario, BB wants to minimize the number of teleporters he needs to buy.

(c) (3 points) Let's consider BB having only one mallet in his right hand again. Please design an $O(N)$ algorithm to determine the minimum number of disposable teleporters he needs to buy.

(d) (11 points) Let's consider BB having two mallets, one in each hand, again.

(i) (5 points) Please design an $O(N^2)$ **dynamic programming** algorithm to determine the minimum number of disposable teleporters he needs to buy.

(ii) (2 points) Briefly explain why this algorithm runs in $O(N^2)$ time.

(iii) (4 points) Prove its correctness.

Problem 6 - When is it? (30 points)

Grading Policy for Problem 6

For each task (e.g. **(b)** is a **task**) below, if there are multiple subtasks (e.g. **(b)**-(2) is a **subtask**) in a task, you will automatically obtain all the credits from the prior subtasks. For example, there are 3 subtasks in **(b)**, then solving **(b)**-(2) will automatically grants you the score for **(b)**-(1).

When working on a task, you may cite the conclusion of its previous tasks to support your statement, no matter if you have solved it. For example, you can cite the results from **(c)** when you are solving **(e)**.

Difficulty Notations for Problem 6

- *TL; DR*: 👻 $\approx$ 🦇 $<$ 🧙
- 🦇 might not be harder than 👻, but requires some twists. We encourage you to try on these two types of problems.
- 🧙 might be (slightly) difficult.

Words from the Setter of Problem 6

Please keep in mind that you can always skip the tasks if it takes too much time. As mentioned, you can still cite these results in further tasks.

As ZCK woke up, he realized that he was surrounded by some ancient debris. Luckily, he found a time machine. However...

"When is it? Is the firework in Dadaocheng over?"
"It is 5432109020801234 **banana**s since Unix epoch." said an elder.
"Whatttt?" asked ZCK.
"Seems like you come from the Earth, one **banana** is equal to 5432109020834567 seconds."
"So it's 5432109020801234 $\times$ 5432109020834567 seconds." ZCK finally realized.

Multiplying two big integers are nightmares, especially for college students. Can you help him get back and watch fireworks as soon as possible? Assume that **you can only perform single digit addition, subtraction and multiplication in constant time**.

(a) ZCK computes 67×89 using the method he learned in his elementary school, shown in the figure on the right. How many (single digit) additions and multiplications does he use? No explanation is needed.

Note: In the figure, the left column indicates the overall process. The right column indicates the details about how he calculates the final answer from the intermediate results. **Notice that you don't need to do any operation whenever a digit is added with "blank".**

```
   67          63
x  89       +  54
-----       ------
   63          603
  54        +  56
  56        ------
 48           1163
-----       + 48
  5963      ------
              5963
```

(b) Multiplying two integers with N digits takes time, which is denoted as $f(N)$. And $f(N)$ is increasing. Adding/subtracting two integers with N digits takes $O(N)$ time.

(i) Design an algorithm that multiplies two integers with $2N$ digits in $3f(N + c) + O(N)$, where c is a constant.

(ii) Design an algorithm that multiplies two integers with $3N$ digits in $6f(N + c) + O(N)$, where c is a constant.

(iii) Let $2k < b$, where b is the base. Design an algorithm that multiplies two integers with kN digits in $(2k - 1)f(N + c) + c'N$, where c, c' are constants, and c' only depends on k.
You might get partial credits if you prove a weaker version of this subtask where $2k - 1$ is replaced by $O(k)$; however, in this case, you should additionally do subtask (2) correctly to get the scores from (2).

For tasks from here on, if you want to cite the results of any subtask in (b), you can make the following assumptions without a proof.

- $f(N + c) = f(N) + O(N)$.
- The constraint $2k < b$ can be ignored.
- Specially for **(b)**-(3), $c' = O(k^2 \log k)$.

(c) Design a divide-and-conquer algorithm that multiplies two integers with N digits. **Briefly explain your algorithm and analyze the time complexity**. Your score will depend on the time complexity of your algorithm.

Hint: Recall that you can use previous results no matter if you have solved it.

(i) It runs in $O(N^{\log_2 3})$ time.

(ii) For any $\epsilon > 0$, show that there exists an algorithm that runs in $O(N^{1+\epsilon})$.

(d) **Give a proof** to show that

$$N2^{10\sqrt{\log N}} = o(N^{1.01})$$

By some generalizations of **(d)**, we can actually conclude that an algorithm constructed from **(e)** is better than that from **(c)**.

(e) Instead of splitting into a fixed number of groups each time, maybe we can use different k for different N. Design an algorithm to multiply two integers with N digits that runs in

$$O(N2^{c\sqrt{\log N}})$$

and prove its time complexity. The score depends on the constant c:

(i) $c \in \mathbb{R}$

(ii) $c \leq 4$

(iii) $c \leq 1.5$

This task is totally solvable with in-class knowledge and the results from the previous tasks. We discourage you to apply obscure techniques not taught in class. If you insist to do so, you should provide a proof and you should suppose that the base b is unknown. Notice that this does not affect any results from previous tasks.

Problem 7 - How is it? (3 points)

How's your ADA experience so far? We have the tradition of letting students write feedback 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**:

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

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

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

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

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

Master Theorem Let $a \ge 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative 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 $af(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.