CS218, Winter 2024, Assignment #1 [1] **Due: 11:59pm, Friday, 01/19**

Name: **Student ID:**

Deadline. The homework is due **11:59pm, Friday, 01/19**. You must submit your solutions (in pdf format generated by LaTeX) via GradeScope. The training programming assignment is due earlier, see more details below.

Late Policy. You have up to two grace days (calendar day) for the entire quarter. You don't lose any point by using grace days. If you decide to use grace days, please specify how many grace days you would like to use and the reason at the beginning of your submission.

Collaboration Policy. You can discuss the homework solutions with your classmates. You can get help from the instructor, but only after you have thought about the problems on your own. It is OK to get inspiration (but not solutions) from books or online resources, again after you have carefully thought about the problems on your own. However, you **cannot copy anything from other source**. You **cannot** share your solution/code with anyone else. You **cannot** read other's solution/code. If you use any reference or webpage, or discussed with anyone, you must cite it fully and completely (e.g., *I used case 2 in the examples in the Wikipedia page* `https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)` *about Master theorem for Problem 3*, or *I discussed problem 2 with Alice and Bob.*). *If you use ChatGPT or similar AI tools, you have to attach the full conversation to make it clear what help you obtained from them.* **Otherwise it will be considered cheating.** We reserve the right to deduct points for using such material beyond reason. **You must write up your solution independently, and type your answers word by word on your own: close the book/notes/online resources when you are writing your final answers.**

Write-up. Please use LaTeX to prepare your solutions. For all problems, **please explain how you get the answer** instead of directly giving the final answer, except for some special cases (will be specified in the problem). For all algorithm design problems, please describe using **natural language**. You could present pseudocode if you think that helps to illustrate your idea. Please do not only give some code without explanation. **In grading, we will reward not only correctness but also clarity and simplicity. To avoid losing points for hard-to-understand solutions, you should make sure your solutions are either self-explanatory or contains good explanations.** See some more details here `https://www.cs.ucr.edu/~ygu/teaching/218/W24/assignments/index.html`

Programming Problems. You will need to submit your code on CodeForces (a readme file about submitting code is available on the course webpage). You also need to submit a short report through GradeScope along with your solutions of the written assignments. In the report, you need to **specify your submission id**, describe the algorithm you designed, and show cost analysis if necessary. Note that your code will be automatically saved by CodeForces, so you do not need to submit the code again. For each problem, there will be 10-20 test cases. **For the training programming problems, you need to finish before 11:59pm, Friday, 01/19.** With reasonable implementation, using C++ or Java is guaranteed to be able to pass all tests. You can use other languages, but it's not guaranteed that the implementations can be within the time limit.

[1] Some of the problems are adapted from existing problems from online sources. Thanks to the original authors.

1 A Complex Complexity Problem (1.2pts)

Yihan recently learned asymptotical analysis. The key idea is to evaluate the *growth* of a function. For example, she now knows that n^2 grows faster than n, but for more complicated functions, she feels confused. Can you help Yihan compare the functions below? We use $\log n$ to denote $\log_2 n$. $n! = 1 \times 2 \times \cdots \times n$. Explain each of your answers briefly.

For the following questions, use o, Θ or ω to fill in the blanks.

Questions:

1. $(\sqrt{3})^{\log n} = \_\_(n)$
   Explain briefly:

2. $\log \log n = \_\_(\sqrt{\log n})$
   Explain briefly:

3. $\log(n!) = \_\_(n \log n)$
   Explain briefly:

4. $2^n = \_\_(3^n)$
   Explain briefly:

2 Solve Recurrences (0.6pts)

For all of them, you can assume the base case is when n is a constant, $T(n)$ is also a constant. Use $\Theta(\cdot)$ to present your answer.

Questions:

(1) $T(n) = T(n/2) + n \log n$

(2) $T(n) = 2T(n/4) + \sqrt{n}$

(3) $T(n) = 4T(n/4) + n^{1/2}$

3 Test the candies (2.2 pts + 1 candy)

You got job at a candy factory. It's not always the case that all the candies produced are perfect. There will be some bad ones. Your task is to **identify** these bad candies from n candies and discard them. However, you can not tell which ones are bad directly: they look exactly the same. The only thing you know is that, a bad candy has **lighter weight** than standard (good) candies. More precisely, all the good (standard) candies have the same weight w_g, and all the bad candies have the same weight $w_b < w_g$.

The **only** device you have is a **balance scale**, and you **don't have any weights** (so you cannot really know the weight of each candy). As a result, the only thing you can do is to put some candies on the left and some on the right, and the balance will tell you if the left one is heavier, the right one is heavier, or they balance.

Every time you use the balance, you have to pay 1 dollar. All the other cost is free. Your task is to find all bad candies using the lowest cost.

Questions:

For all questions below, please describe your algorithm and briefly explain the cost.

1. (0.2pts) Your boss told you that there is only **one bad candy**. In that case, can you show an algorithm that uses $\lceil \mathbf{\log_2 n} \rceil$ (note: this is not in big-O!) dollars to find this bad candy?

2. (0.2pts) Your boss told you that there is only **one bad candy**. Now let's **improve** the previous cost by a little bit. Can you show an algorithm that uses $\lceil \log_3 n \rceil$ (again, not in big-O!) dollars to find this bad candy?

3. (0.6pts) Prove that $\lceil \log_3 n \rceil$ dollars is the lower-bound of the candy-testing problem in 3.2. In other words, you cannot use fewer than $\lceil \log_3 n \rceil$ dollars to guarantee to find the bad candy.

4. (0.2pts) Your boss told you that there are only **two bad candies**. Can you show an algorithm that uses $O(\log n)$ dollars to find the two bad candies?

5. (0.3pts) If you already know that there are only k **bad candies**, where k is a known constant, can you show an algorithm that uses $O(\log n)$ dollars to find the k bad candies? **You could assume that k is a known value and it could appear in your algorithm.**

6. (0.7pts) In all above questions, we assume you know the bad candy is **lighter** than standard. A more difficult cases is where you only know that the bad candy is of a different weight, but you do not know if it is lighter or heavier. Again assume there is only one bad candy. Prove that, in this case, you need at least $\lceil \log_3 2n \rceil$ dollars to find the bad candy, and also tell whether it is lighter or heavier.

7. (bonus, 0.5pts and 1 candy) Now let's consider the challenging setting where there is only one bad candy, but you don't know if the bad candy is lighter or heavier. Luckily, your boss also gave you **one good candy** as a reference (you'll find this useful). Now you have $n = 13$ candies, and one of them is bad (either lighter or heavier). Plugging this into the lower bound above gives $\lceil \log_3(2 \times 13) \rceil = 3$ dollars. Now, show a solution for $n = 13$ for this case using 3 dollars.

Hint

Maybe divide-and-conquer is a good idea.

4 (Training) Finding the Minimum Value (1.5 pts)

The programming problem can be found on codeforces.

5 (Programming) Sort the Train (1 + 1 pts)

The programming problem can be found on codeforces.

5.1 Programming (1 pt) and Hints

You first need to write a program to determine the least number of "adjacent swaps".

You will find out that, if you run a bubble sort (where you are only allowed to do "adjacent swaps"), and record the number of swaps, that is always the optimal answer (lowest number of swaps).

However, simulating bubble sort takes $O(n^2)$ time. Can you come up with a better algorithm for that? In particular, if you have an algorithm with complexity $O(n \log n)$, you can pass all the test cases.

Note that you need to write a report for your programming assignment - You need to submit your report to a separate entry on GradeScope.

5.2 (Bonus) Sort the Train - Let's prove it! (1 pt bonus + 2 candies)

Well, now, let's prove that your $O(n \log n)$ algorithm for the previous question is correct and efficient.

Questions:

1. (0.3pts + 1 candy) Prove that the number of swaps in the bubble sort algorithm is the optimal solution.
2. (0.3pts) Briefly describe your algorithm that has $O(n \log n)$ running time. Prove the complexity of your algorithm.
3. (0.4pts + 1 candy) Prove that your algorithm computes the number of fewest needed adjacent swaps.

6 Being Unique (1 pt bonus + 2 candies)

You are given an array A of n numbers from the set $\{1, 2, \ldots, n\}$. The array has the "unique-in-range" property if for every range $[i, j]$ there exists an element $A[k]$ (where $i \leq k \leq j$) such that the number $A[k]$ occurs just once in that range.

For example, the sequence: 1 2 1 3 1 2 1 4 1 2 1 3 1 has the unique-in-range property. As does 1 2 3 4 5 6. But the sequence 1 2 3 1 2 3 does not (it fails on the whole range: every number appears twice).

Give an algorithm to determine if a given array has the property. It should have runtime $O(n \log n)$ or better. You must prove your answer.