CS141, Assignment #1 **Due: 23:59, Sat 1/18, 2026**
Name: **Student ID:**

Deadline. The homework is due **23:59, Sat 1/18, 2026**. You must submit your solutions (in pdf format generated by LaTeX) via GradeScope. The programming assignment has a different deadline and is available on CodeForces.

Late Policy. You have up to 4 grace days (calendar days) for the entire quarter, but you can use at most 2 days for each assignment deadline. You don't lose any point by using grace days and you do not need to provide any reasons for using grace days. The grade days are counted per homework submission deadline (10 deadlines in total). The number of grace days will always be an integer. For example, if you are submitting 25 hours late, that counts as two grace days. We do not accept other requests for late submissions if you do not have grace days.

Academic Integrity. You can discuss with your classmates or use online resources. In particular,

- **Collaboration Policy.** You can discuss your solutions with your classmates. You can get help from the instructors or TAs, but only after you have thought about the problems on your own. **If you discussed with anyone else, you need to acknowledge them in your report**.
- **Other sources.** It is OK to get inspirations (but not solutions) from books or online resources, again after you have carefully thought about the problems on your own. **If you use any reference or webpage, or discussed with anyone, you must cite it fully and completely in your report**.

You must write up your solution independently: close the book and all of your notes when you are writing your final answers. You **CANNOT**:

- share your solutions with anyone else,
- read other's solutions,
- copy anything from other source,
- get help from others without acknowledging them in your submission, or use any relevant resources (e.g., online article) without citing them.

Otherwise it will be considered cheating. We reserve the right to deduct points for using such material beyond reason.

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, pseudocode can be used to help explain your idea, but **please also describe your algorithms in natural language**.

1 A Complex Complexity Problem (2pts)

Yihan recently learned the 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. She wants to know whether she really understands the idea, so she has created a little task. First of all, she found a lot of functions here:

$g_1 : 3^n$	$g_6 : \log n^2$	$g_{11} : \sqrt{n}$	$g_{16} : n$
$g_2 : n!$	$g_7 : \log(n!)$	$g_{12} : \log\log n$	$g_{17} : 2^n$
$g_3 : n^2 + n$	$g_8 : 2^{n+1}$	$g_{13} : (n+1)!$	$g_{18} : n \log n$
$g_4 : n^3$	$g_9 : \ln\ln n$	$g_{14} : \sqrt{\log n}$	$g_{19} : 5n^2 - 13n + 6$
$g_5 : \log^2 n$	$g_{10} : 10000$	$g_{15} : \log\sqrt{n}$	$g_{20} : n/\log n$

Note: $\log n = \log_2 n$; $\ln n = \log_e n$; $\log^2 n = (\log n)^2$; $n!$ is the factorial of n, i.e., $n! = 1 \times 2 \times \cdots \times n$.

Reading the long list, Yihan realized that things were not as simple as she thought. Could you help her with the following problems?

For all the questions in this problem, you only need to show your answer without explaining. The only exception is question (4), where you need to show proofs or explanations.

(1) (0.1pt) What does *polylogarithmic* mean (use $O(\cdot)$, $\Omega(\cdot)$, $\omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$ to present your answer)? (0.1pt) Which of them are polylogarithmic?

(2) (0.1pt) What does *sublinear* mean (use $O(\cdot)$, $\Omega(\cdot)$, $\omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$ to present your answer)? (0.1pt) Which of them are sublinear?

(3) (0.1pt) What does *superlinear* mean (use $O(\cdot)$, $\Omega(\cdot)$, $\omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$ to present your answer)? (0.1pt) Which of them are superlinear?

(4) Then let's compare some them pairwise.

  (a) (0.2pt) Compare $g_{12} : \log\log n$ and $g_{14} : \sqrt{\log n}$. Which one grows faster? Please explain briefly.

  (b) (0.2pt) Compare $g_2 : n!$ and $g_{13} : (n+1)!$. Which one grows faster? Please explain briefly.

  (c) (0.2pt) Compare $g_{20} : n/\log n$ and $g_{11} : \sqrt{n}$. Which one grows faster? Please explain briefly.

  (d) (0.2pt) Yihan finds out that $g_{12} : \log\log n$ and $g_9 : \ln\ln n$ should be asymptotically equivalent (i.e., $\log\log n = \Theta(\ln\ln n)$). Can you prove this?

  (e) (Bonus 0.2pts and 1 candy) Yihan finds out that $g_7 : \log n!$ and $g_{18} : n\log n$ are asymptotic to each other (i.e., $\log n! = \Theta(n\log n)$). Can you prove this?

(5) (0.6pt) Finally, please rank all the functions by order of growth. You can use the result in Question (4)e directly. Present your answer in the form of:

$$g_{i_1} \prec g_{i_2} \prec \ldots \prec g_{i_3} \sim g_{i_4} \prec g_{i_5} \ldots \prec g_{i_{16}}$$

Where $g_i \sim g_j$ means $g_i = \Theta(g_j)$. $g_i \prec g_j$ means $g_i = o(g_j)$. For simplicity, you can also use "$<$" and "$=$" to represent $\prec$ and $\sim$, respectively.

For example, if the functions are $g_1 = n^2, g_2 = n+1, g_3 = n, g_4 = 2^n$. Your answer should be:

$$n \sim n+1 \prec n^2 \prec 2^n, \quad \text{or} \quad n = n+1 < n^2 < 2^n$$

or

$$g_3 \sim g_2 \prec g_1 \prec g_4, \quad \text{or} \quad g_3 = g_2 < g_1 < g_4$$

Note that when you solve this problem, you're also using *divide-and-conquer*: you first classify the algorithms into linear, sublinear, and superlinear. For superlinear, you classify them into polynomial, those larger than polynomial and those smaller than polynomial. For sublinear, you classify them into polylogarithmic, larger/smaller than polylogarithmic. Then for each small class, you sort each of them. Finally you combine them together, to give your final solution.

2 Solve Recurrences (1pt)

Now that you learned Master Theorem that could help you with solving recursions. However, keep in mind that it doesn't apply to all cases: you may need to carefully check the applicability of Master Theorem. Also, it's still important to know how to solve some recurrences directly, e.g., using the expansion tree.

Let's have some exercises. Solve the recurrences below. If you use Master Theorem, please show how you know that Master Theorem is applicable, and which case are you using (case 1, 2, and 3 as shown in class slides). Use $\Theta(\cdot)$ to present your answer. Note that not all of them can be solved using Master Theorem. **Please provide some details about your calculation instead of just giving you final answer.** Of course you can use any other approaches to solve them (e.g., substitution).

For the last question, please first solve it **without** using Master Theorem, and then **check the correctness** of your answer using Master Theorem (in other words, please solve it twice, with or without using Master Theorem).

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.

(1) (0.2pt) $T(n) = T(n-1) + n^2$

(2) (0.2pt) $T(n) = 3T(n/2) + n^2$

(3) (0.2pt) $T(n) = 2T(n/4) + \log\log n$

(4) (0.4pt) $T(n) = 2T(n/2) + n\log n$ [For this question, please provide two solutions (with and without using Master Theorem)]

3 Test the Candies (1pt)

You work at a candy factory. It's not always the case that all the candies produced are perfect. There will be some bad ones (say, not that tasty). Your task is to **identify** these bad candies from n candies and discard them. Fortunately, you have a device to help you with this. If you put a batch of candies in the device, as long as there exists **any bad candy**, the device will return "BAD!" to you. Otherwise (i.e., all the candies are good) it returns "GOOD!". Of course, if you put only one candy into the device, you can directly know if it is good or bad. However, every time you use the device, you need to pay 1 dollar, no matter how many candies you put in the tested batch. You want to use the device fewer times to save money.

Note that the device will not tell you how many bad candies are there in the tested batch or which ones are bad, it only tells you whether there's any bad candy. Also, you don't know in advance how many bad candies are there in the n candies.

You can put all the candies one by one into the device to check which ones are bad (that's n dollars in total), but you want to do better. In fact, you know that the number of bad candies is very small (otherwise the candy factory will go bankrupt).

Again, every test costs 1 dollar, and all the other operations are free. Design a **divide-and-conquer algorithm** for this task, such that, when the number of bad candies is small, you solution is much cheaper than testing all of them one by one. Given n candies in total, when there is only a constant number of bad candies, your algorithm should cost $O(\log n)$ dollars. For simplicity, you can assume the number of candies n is a power of 2.

(1) (0.4pts) Describe a divide-and-conquer algorithm for the candy testing task. You could simply describe it or show pseudocode.

(2) (0.3pts) Prove that when you have a constant number c of bad candies, the total number of tests you need to do is $O(\log n)$.

(3) (0.3pts) If all the candies are bad, how many tests does your algorithm need? Present your answer in big-O.

(4) (bonus, 0.3pts and 1 candy) If there are m bad candies, how many tests does your algorithm need? You should present your bound in big-O using n and m. You should present the tightest bound you could get.

Good luck!