# CS 161 (Stanford, Winter 2026) Homework 1

**Style guide and expectations:** We do NOT accept handwritten solutions. Please see the “Homework” part of the “Resources” section on the webpage for guidance on what we look for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material. Please do not distribute this material on any public forum.
**What we expect:** Make sure to look at the “**We are expecting**” blocks below each problem to see what we will be grading for in each problem!

**Exercises.** The following questions are exercises. We suggest you do these on your own. As with any homework question, though, you may ask the course staff for help.

## 1 Course Policies (1 pt.)

Have you read the course policies on the website?

**[We are expecting:** The answer “yes."**]**

Solution

## 2 Prerequisite Quiz (1 pt.)

Have you completed the prerequisite quiz?

**[We are expecting:** The answer “yes."**]**

## 3 Complexity Bounds (6 pt.)

For each row $i$, indicate whether $A_i$ is in O, $\Omega$, or $\Theta$ of $B_i$. Mark all that apply. The first two rows are examples. All logs appearing in the table are in base 2.

| **A** | **B** | $O$ | $\Omega$ | $\Theta$ |
|---|---|---|---|---|
| $5n$ | $n$ | ✓ | ✓ | ✓ |
| $5$ | $n$ | ✓ | | |
| $n^2$ | $2^{10}n$ | | | |
| $(7/8)^n$ | $(8/7)^n$ | | | |
| $8^n$ | $4^n$ | | | |
| $n^5$ | $32^{\log n}$ | | | |
| $n^{2025}$ | $2025^n$ | | | |
| $n^{0.8}$ | $(0.8)^n$ | | | |
| $n^{\log 3}$ | $3^{\log n}$ | | | |
| $\log(n!)$ | $\log(n^n)$ | | | |
| $n^{1/\log n}$ | $1$ | | | |
| $\log^7 n$ | $n^{0.7}$ | | | |
| $\log(\sqrt{n})$ | $\sqrt[4]{n}$ | | | |
| $\log(\sqrt{n})$ | $\sqrt{\log n}$ | | | |

**[We are expecting:** All valid spaces in the table to be marked (checkmark, x, etc.). No explanation is required.**]**

## 4 Big-O Proofs

Using the definition of big-O from class, formally prove the following statements. **[We are expecting:** For each part, a rigorous (but short) proof, using the definition of big-O.**]**

### 4.1 First part (5 pt.)

Show $3n^3 + 4n^2 = O(n^3)$.

### 4.2 Second part (5 pt.)

Show $16^n$ is **not** $O(2^n)$.

**Problems.** The following questions are problems. You may talk with your fellow CS 161-ers about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

## 5 Ocean View

Lucky the Lemur loves the beach, and wants to get the best view of the ocean from the coast. He has a map of the terrain, which is m-pixels tall and n-pixels wide. On the map, land has a pixel value of 1 and water has a pixel value of 0. In any given row, all the land pixels are to the left of all the water pixels. In this problem, you will design algorithms that find the best ocean viewing spot, which are the pixel coordinates of the land closest to the ocean (i.e. the rightmost point of land).

The input of your algorithm is an m x n matrix, where land is represented with 1s and water is represented with 0s. The output is the location coordinate of the rightmost land pixel. We assume that the input matrix has been imported, and the algorithm can access any entry of the matrix in $O(1)$ time.

In this example matrix below, $m = 4$ and $n = 5$. We output (1, 3) since this is the rightmost land coordinate (assuming we use 0-based indexing). In the event that there are multiple best ocean viewpoints, return any valid coordinate.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & \mathbf{1} & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

### 5.1 First part (5 pt.)

Devise an algorithm that runs in $O(m \log n)$ time to find the best ocean viewpoint.

**[We are expecting:** Pseudocode for your algorithm and a clear English description of what your algorithm is doing.**]**

### 5.2 Second part (5 pt.)

Devise an algorithm that runs in $O(m + n)$ time to find the best ocean viewpoint.

**[We are expecting:** Pseudocode for your algorithm and a clear English description of what your algorithm is doing.**]**

## 5.3 Third part (3 pt.)

Lucky, upon happily using these algorithms at different coastal locations, notices that sometimes it's more efficient to use the first algorithm, but sometimes it's more efficient to use the second algorithm. For each of the values of $n$ below, determine which algorithm is the winner in terms of efficiency (or state that they are equally efficient). The case $n = m$ is filled in as an example.

| $n = ?$ | 500 | $m^{0.2}$ | $\frac{m}{\log m}$ | $m$ | $m \log m$ | $m^{10}$ | $8^m$ |
|---|---|---|---|---|---|---|---|
| Runtime for (a) | | | | $O(m \log m)$ | | | |
| Runtime for (b) | | | | $O(m)$ | | | |
| Which is better? | | | | (b) | | | |

**[We are expecting:** For each value of $n$ in terms of $m$, write the best big-O runtimes you can guarantee for both algorithms (a) and (b), and a conclusion on which is the more efficient choice, asymptotically. You do not need to write any formal proofs, but your answers should be **in the simplest terms possible**.**]**

## 5.4 Fourth part (2 pt.)

Based on your results from Question 5.3, how could Lucky combine algorithms (a) and (b) to create a more efficient algorithm?

**[We are expecting:** A 1-2 sentence summary of your results from the previous part, and a conclusion on how Lucky could create a new algorithm that combines the efficiencies of both algorithms. No runtime analysis is needed.**]**

## 6 Alien Arithmetic (10 pt.)

Plucky, an inquisitive penguin mathematician, has been exploring ancient ice caves in Antarctica, where he stumbles upon a mysterious tablet depicting a long lost alien civilization. The tablet illustrates the aliens' mathematical prowess. Interestingly, he discovers that they use the same base 10 numerical system, addition, and subtraction as we do, but their system of multiplication is completely different. Their special multiplication operation is denoted as $\star$. Plucky notices a 1-digit multiplication table carved into the icy surface. Here are some examples:

$$1 \star 1 = 3 \qquad 2 \star 3 = 19 \qquad 3 \star 5 = 49 \qquad 9 \star 9 = 243$$

and so on. After months of pecking away at the problem, Plucky deduces the underlying rule behind this operation:

$$x \star y = x^2 + xy + y^2.$$

Plucky is thrilled by this discovery and wonders whether this multiplication operation could revolutionize mathematical techniques. To start, Plucky decides to test whether the $\star$ operation is at least as efficient as the methods devised by his own penguin ancestors.

Plucky wants to design an algorithm to compute the $\star$ product of two $n$-digit numbers that runs in $O(n^{\log_2 3})$ time. For example, if given $x = 4321$ and $y = 1234$, the algorithm should output $x \star y = 25525911$. However, Plucky must strictly follow the rules of the tablet and cannot refer to his own form of multiplication (even implicitly as a sum). For example, terms like $x^2$, $xy$, or $y^2$ cannot be explicitly computed.

Help Plucky figure out an efficient way to compute the $\star$ product and determine whether this ancient math could hold the key to a faster form of computation! Assume that $n$ is always a power of 2, and Pluckys penguin-built computers are capable of performing basic arithmetic operations like addition, subtraction, and digit-shifting. You may also assume that the computers have easy access to the $\star$-multiplication table found in the tablet.

**[Hint:** Let $x = a \cdot 10^{n/2} + b$ and $y = c \cdot 10^{n/2} + d$ for some $n/2$-digit numbers $a$, $b$, $c$, $d$. Can you find a way to rewrite $x \star y$ in terms of $(a + b) \star (c + d)$, $(a \star c)$, and $(b \star d)$?**]**

**[We are expecting:** An expansion of $x \star y$ in terms of $(a + b) \star (c + d)$, $(a \star c)$, and $(b \star d)$, pseudocode for your algorithm, and a clear English description of what your algorithm is doing. You do not need to prove that your algorithm is correct.**]**