EECS 376: Foundations of Computer Science
University of Michigan, Winter 2025

Homework 1

Due 8:00pm, January 15
(accepted until 9:59 pm, no credit after)

This homework has 8 questions, for a total of 100 points and 10 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

(a) Before starting this assignment, carefully read Handout 1 (Running Times and Aysmptotics) and the "Good Writing" section of Handout 2 (the induction section is optional but recommended), and apply them to your work.

(b) If applicable, state the name(s) and uniqname(s) of your collaborator(s).

(10 EC pts) 1. **LaTeX for dummies (optional extra credit).**

You are not required to do this question to receive full credit on this assignment.

We (along with the rest of the civilized math-writing world) use LaTeX for all homeworks and exams. LaTeX is *not* a "what you see is what you get" word-processing software like you may be accustomed to (Word, Google Docs, etc.). Instead, a LaTeX document is best seen as a *program* that *describes* the desired output, and is *compiled* into a human-readable format (e.g., PDF, HTML, Postscript, etc.).

Importantly, the ultimate appearance and contents of the document can vary based on certain options and settings, without making any changes to the main contents. So, when writing in LaTeX, the primary focus should be on the *structure* of the document and the *meaning* of the "code," rather than the layout and appearance. Ensuring a consistent and beautiful appearance is the job of the compiler, not the human author. For more on this perspective and many other useful LaTeX practices and things to avoid, see these good resources:

- Writing Beautifully in LaTeX,
- Short Math Guide for LaTeX,
- DOs and DON'Ts when typesetting a document.

To receive the bonus points, you must typeset this **entire** assignment in LaTeX, and typeset the following two snippets of text in LaTeX. Include the source code in a `Verbatim` environment from the fxectra package.

(a) The following snippet is about some key features of the big-O notation extracted from Handout 1.

For (non-negative) functions $f(n)$ and $g(n)$, we say "$f(n) = O(g(n))$" if there exist positive constants c, n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. The key features here are:

- "there exists a positive constant $c \ldots f(n) \leq c \cdot g(n)$": this says that $f(n)$ is *upper bounded* by some *constant multiple* of $g(n)$. That is, O-notation "hides," or ignores, constant factors. Note also that it captures only an *upper bound*: $f(n)$ might actually be *much smaller* than $c \cdot g(n)$, or not. A big-O bound by itself says nothing about which is the case.
- "there exists a positive constant n_0 ... for all $n \geq n_0$": this says that the upper bound holds for all n *above some constant "threshold."* However, it says nothing about the relationship between $f(n)$ and $g(n)$ *below* the threshold; it could be that $f(n)$ greatly exceeds (even a huge multiple of) $g(n)$ in that range. Moreover, the threshold n_0 can be *any constant*—it could be one, or ten, or a billion, or a googolplex—and the notation hides its exact value.

LaTeX you may need:

- In-line math
- Less-than-or-equal-to symbol
- Quote marks
- Lists

(b) The following snippet is about Fibonnaci numbers.

The Fibonnaci numbers may be defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

with base cases $F_0 = 0$ and $F_1 = 1$. We can compute the n-th Fibonnaci number using the following recursive algorithm:

Input: a natural number n
Output: the n-th Fibonnaci number

```
1: function FIB(n)
2:     if n = 0 then
3:         return 0
4:     if n = 1 then
5:         return 1
6:     return FIB(n − 1) + FIB(n − 2)
```

LaTeX you may need:

- Display math
- The `algorithmic` environment

Hint: Feel free to copy and modify the source code of the algorithms from other problems.

2. **Read the syllabus!**

(5 pts) (a) Suppose you attend 9 out of 14 discussion sessions and submit both course evaluation receipts. Fill in the table below to calculate the weights of each component of your final grade.

Component	**Weight (%)**
Homework	
Exams	
Discussion Attendance	
Course Evaluations	
Total	**100**

Based on your calculation, explain how attending 9 discussion impacts the weight distribution of your final grade compared to attending 10 or more discussions.

(5 pts) (b) Below is a table showing your scores for 12 homework assignments. Assume one homework was not submitted and one homework was submitted late.

Homework #	**Raw Score (%)**	**Notes**
1	100	
2	80	
3	90	
4	0	Not submitted
5	95	
6	70	
7	85	
8	88	
9	92	
10	50	
11	94	
12	90	Submitted late

Calculate your average homework grade after applying the drop policy (dropping the two lowest scores) and the 5% late penalty.

(5 pts) (c) Name three different ways you can ask a question about course content in this class. How are you more likely to ask questions?

3. **Welcome to EECS 376!**

If you haven't already, please read Handout 1 (Running Times and Aysmptotics) and apply it to the next three problems.

To express our excitement for the start of EECS 376, we present an algorithm that prints `"Welcome to EECS 376!"` multiple times, based on two input parameters n and k. You may assume that the PRINT operation runs in $O(1)$.

Input: positive integers n and $k \in \{1, 2, \ldots, n\}$, both represented in base-2 (binary)

```
1: function WELCOME(n, k)
2:     for i = 1, 2, ..., k do
3:         for j = 1, 2, ..., n − k do
4:             PRINT("Welcome to EECS 376!")
```

(3 pts) (a) Identify the value of k that induces the *most* `"Welcome to EECS 376!"` printed by the algorithm, in terms of n.

(4 pts) (b) Based on your answer in Part a, give the *tightest correct asymptotic* (big-O) bound, as a function of n, on the worst-case number of `"Welcome to EECS 376!"` printed by the algorithm.

(4 pts) (c) Determine whether the algorithm is *efficient*, i.e., runs in at most polynomial time with respect to the input size. Briefly explain your answer.

Reminder: The input size of an integer is defined as the number of bits needed to represent it in binary.

(4 pts) (d) Does your answer in Part c change if n and k are represented in base-10 (decimals) instead? Briefly justify your answer.

Hint: You may want to use the change-of-base formula for logarithms for any positive real numbers a, b, and c, where $b \neq 1$ as well as $c \neq 1$:

$$\log_b a = \frac{\log_c a}{\log_c b}$$

4. **Lumpy array.**

In lecture, we said that you can measure the "size" of an array as the number of elements in it, but we also said the "size" of an input is the number of bits needed to represent it. Consider the following example algorithm on an array:

Input: array A of n integers
Output: sum of all elements in the array

```
1: function SUM(A)
2:     S ← 0
3:     for i = 1, 2, ..., n do
4:         S ← S + A[i]
5:     return S
```

(5 pts) (a) For this part, we will assume that each element of the array is the same fixed size, and the "size" of our input will be considered the number of elements in the array, n. Find the big-O runtime of this algorithm (in terms of n) and state whether the function SUM is efficient, i.e., runs in polynomial time with respect to the input size (n).

Note: Because all elements of the array are the same size, we will treat addition as runnings in constant time, $O(1)$.

(5 pts) (b) Suppose instead, that we measure the "size" of our array as the number of bits used to represent it, call it b. Here, the elements of the array may be of different (arbitrarily large) sizes, so we will treat addition as running in linear time (to sum numbers with k bits, it takes $O(k)$ time; you could think of this as taking finite time to add/carry each place value). Determine whether in this interpretation, our algorithm is still efficient, running in polynomial time with respect to the input size (now b).

Hint: If our input has b bits, find an upper bound on the number of elements that could be in the array and the number of bits in any one element of the array. Note that your big-O bound for the runtime of the algorithm does not need to be a tight bound.

5. **Prof. Upsilon and their oopsie claims.**

Prof. Υ is a brilliant but occasionally absent-minded computer scientist, known for their groundbreaking insights into algorithms—and the occasional "oopsie" moments. Recently, Υ came up with three algorithms with the following running times:

- Algorithm X has running time $T_X(n) = O(n^2)$
- Algorithm Y has running time $T_Y(n) = \Theta(n \log n)$
- Algorithm Z has running time $T_Z(n) = \Theta(n)$

As usual, all running times are stated in terms of the *worst case* for inputs of size n. That is, $T_X(n)$ is the *maximum* number of steps for which X runs, taken over all inputs of size n (and similarly for T_Y, T_Z).

Now, Prof. Υ has made a series of claims about these algorithms. State, with proof, whether Υ's claim is necessarily true, necessarily false, or not necessarily either based on the given information. If it is not necessarily either, give an example when it is true and an example when it is false.

(6 pts) (a) "We can upper bound the runtime of Y by $O(n^2)$."

(8 pts) (b) "On every input, Y runs faster than X."

(10 pts) (c) "For all large enough n, there is an input of size n for which Y runs faster than X."

(10 pts) (d) "For all large enough n, there is an input of size n for which Z runs faster than Y."

6. **Set of proofs by contra-.**

Recall that a *set* is a collection of distinct elements. Below are some important set notations:

- **Elements of a set:** $x \in A$ means x os an element of set A.
- **Complement of a set:** $\overline{A}$ [1] refers to the set of all elements in the universal set U that are not in A.

[1] In some texts, you may encounter notations like A^C, $A^\complement$, A', etc. to refer to the complement of A.

- **Subset:** $A \subseteq B$ means every element of A is also an element of B.
- **Intersection of sets:** $A \cap B$ is the set of elements that are in both A and B.
- **Union of sets:** $A \cup B$ is the set of elements that are in A, in B, or in both.
- **Set difference:** $A \setminus B$ [2] is the set of elements that are in A but not in B. We can also write it as $A \cap \overline{B}$.

(5 pts) (a) Recall the transitive property of the subset relation:

If $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$.

Prove the above property by contradiction.

(5 pts) (b) Recall the DeMorgan's Law, which says:

If $x \in \overline{A \cap B}$, then $x \in \overline{A} \cup \overline{B}$.

State the contrapositive of the statement. Then, prove the contrapositive.

7. **Euclid's algorithm, extended.**

Given two integers x, y (that are not both zero), one may wonder what values can be obtained by summing integer multiples of x and y, i.e., expressed as $ax + by$ for some (not necessarily positive) integers a, b. It is not too hard to see that it is impossible to obtain any positive integer *smaller* than their GCD $g = \gcd(x, y)$, because $ax + by$ must be divisible by g. A less obvious fact is that the GCD *can* be obtained in this way; this theorem is known as *Bézout's identity* (pronounced "BAY-zoo").

In this problem, you will modify the standard Euclidean algorithm to output not only $g = \gcd(x, y)$ itself, but also a pair (a, b) of integers for which $ax + by = g$. (As we will see later, this is a very important tool for cryptography.) Such integers are called *Bézout coefficients* for x, y. We give most of the algorithm below:

Input: integers $x \geq y \geq 0$, not both zero
Output: a triple (g, a, b) of integers where $g = \gcd(x, y) = ax + by$

```
1: function ExtendedEuclid(x, y)
2:     if y = 0 then
3:         return (x, 1, 0) // Base case: 1x + 0y = x = gcd(x, 0)
4:     else
5:         Divide x by y, writing x = qy + r for integer quotient q and remainder 0 ≤ r < y
6:         (g, a', b') ← ExtendedEuclid(y, r)
7:         [FOR YOU TO DETERMINE: compute appropriate a, b]
8:         return (g, a, b)
```

(8 pts) (a) State what a and b should be on Line 7, and prove that the output is correct, i.e., that (1) $g = \gcd(x, y)$, and (2) $ax + by = g$.

Hint: by recursion/induction, we know that $g = \gcd(y, r)$ and $a'y + b'r = g$.

(8 pts) (b) Run the Extended Euclid algorithm by hand to find Bézout coefficients for the input $(x, y) = (376, 281)$, and show that the output is correct. (You may use a calculator/computer only for the division steps.) Fill in the table below with a 'trace' of the execution, i.e., all

[2] Some texts may use $A - B$ to denote set difference.

the variables' values in all the iterations. Also include the potential values $s = x + y$, the ratios (as fractions) of potentials s_{i+1}/s_i for adjacent iterations, and Y/N indications of whether $s_{i+1}/s_i \leq 2/3$.

	input		division		rec ans		output				
i	x	y	q	r	a'	b'	a	b	s	s_{i+1}/s_i	$\leq 2/3$?
0	376	281									

The entries labeled 'input', 'division', s, and s_{i+1}/s_i should be filled from top to bottom, corresponding to the recursive calls; the 'recursive answer' and 'output' entries will need to be filled from bottom to top, corresponding to the 'post-processing' (Line 7) of each recursive call's results. The entries for a, b should match those of a', b' one row above. Put '–' for any entries that are not defined due to the base case.