Due date: Thursday, October 10, 11:59 PM.

You will need to submit this via GradeScope. The course access code is MB4434.

Please review the syllabus and course reference for the expectations of assignments in this class. Remember that problem sets are not online treasure hunts. You are welcome to discuss matters with classmates, but remember the Kenny Loggins rule.

Problems 3-5 have a second option: you may submit Python code for the portions that require you to provide an algorithm. If you elect do this, that portion will be graded by the code in the Python script you submit to GradeScope. The starting point, `prob_set1.py`, is available on Piazza under resources.

Unless stated otherwise, all logarithms in CompSci 161 are base-2.

Confirm **soon, don't wait until near the deadline** that you can form a PDF from however you plan to do your homework. If that's LaTeX, make sure you have it installed, or use an online service like Overleaf or CoCalc. If working digitally, make sure that you are able to "save as PDF," and make a PDF. If it's use of a scanner, go scan something with that scanner.

1. Rank the following functions in order from smallest asymptotic running time to largest. That is, if $f_m(n)$ appears before $f_n(n)$ in your list, it should be the case that $f_m(n)$ is $\mathcal{O}(f_n(n))$. Additionally, identify all pairs x, y where $f_x(n)$ is $\Theta(f_y(n))$. Briefly justify your answers. You are not required to use any particular technique to compare two functions.

   - $f_a(n) = n^\pi + n^e$ (where e is the base of the natural log; $e \approx 2.718281828$).
   - $f_b(n) = 3^{n^3}$
   - $f_c(n) = 16^{\log n}$
   - $f_d(n) = 3^{20n}$
   - $f_e(n) = (7^{e-1/e} - 9) \cdot \pi^2$ (where e is the base of the natural log; $e \approx 2.718281828$).
   - $f_f(n) = 1729 \log n$
   - $f_g(n) = n^{\log n}$
   - $f_h(n) = n^4 + 20n^3 + 1729n + 60000$
   - $f_i(n) = \log^{16} n + 1500 \log^8 n + 3 \log^2 n$

2. Recall that we say that $f(n)$ is $\mathcal{O}(g(n))$ if and only if for some constants $c > 0$ and $n_0 \geq 0$, for all $n > n_0$, $f(n) \leq cg(n)$. For each of the following pairs of functions, find values of c and n_0 that demonstrate that $f(n)$ is $\mathcal{O}(g(n))$.

   (a) $f(n) = 3n + 5$, $g(n) = n$
   (b) $f(n) = n^2 + 10n + 21$, $g(n) = n^2$
   (c) $f(n) = 2n + 12$, $g(n) = n$
   (d) $f(n) = 2n + 12$, $g(n) = n^2$
   (e) $f(n) = 3n^3 + 2n^2 + n + 1$, $g(n) = n^3$

3. Suppose we have an array A of n professors; each teacher has two characteristics: difficulty (a number in the range $[0, 10]$, where a higher number indicates a more difficult teacher) and humor (a number in the range $[0, 100]$, where a higher number indicates a funnier teacher). You may assume that each value is distinct (no two professors are exactly as difficult or exactly as funny). Our goal is to determine a set of professors that might satisfy an answer to the question "who is the easiest teacher that is the funniest?" Our goal is to avoid professors with high difficulty and low humor. We would like to find the largest subset of A such that no professor in the chosen subset is **both** less funny **and** more difficult than another professor in the set. Note that if professor A is easier than professor B, it *does not follow* that professor A is also funnier than professor B.

   For example, if our dataset is:

Professor	Difficulty	Humor
Venabili	4	65
Jones Jr	8	15
Jones Sr	8.5	2
Grant	2	35
Moriarty	10	0
Plum	9	85
Walsh	8.2	90

   Then we want to return Venabili, Grant, and Walsh.

   Note that none of these are meant to refer to anyone at UC Irvine and any similarity is unintentional and a coincidence.

   (a) Devise an algorithm which takes as input A and n, and outputs the resulting set. Your algorithm does not need to be particularly efficient, but you may not give an algorithm that enumerates every subset.

   If you elect to use the programming portion for this, provide your code in your problem set submission also. You still need to answer parts (b) and (c).

   (b) Analyze the worst-case runtime of your algorithm using Θ-notation.

   (c) Do you believe your algorithm has obtained the best possible asymptotic runtime? Explain your reasoning. Note that finding the best possible asymptotic runtime **is not** required to get full credit on this question.

4. Suppose you have two max-heaps, A and B, with a total of n elements between them. You want to discover if A and B have a key in common. Give a solution to this problem that takes time $\mathcal{O}(n \log n)$. For this problem, do not use the fact that heaps are arrays. Rather, use the API of a heap (e.g., calls to `A.max()` or `B.removeMax()`).

   Give a brief explanation for why your algorithm has the required running time.

   If you do this as a programming, put your brief explanation for the algorithm's running time in comments near your code so that the grader can easily find it.

5. Back in your professor's day, some phone calls were more or less expensive to make, depending on the "area code" of the destination call. For example, calling from Irvine CA to Los Angeles CA was probably cheaper than calling from Irvine CA to Boca Raton FL.

   A U.S. phone number consists of a three-digit area code and seven additional digits. Phone numbers whose area code was "800" were free to call (from the point of view of the person making the call).

   Describe how you would solve the following problem. You may write in pseudo-code or in your choice of programming language, but the idea behind your algorithm must be clear.

   The Problem: You are given a list of size at most 10^7 phone numbers, each of which has an area code of "800." You want to do two things:

   (a) Determine if any number appears at least twice in the input. This may be a boolean return value; you do not need to list duplicate(s).

   (b) Output the set of *distinct* numbers in the input in sorted order.

   Your running time for this problem must be strictly better than $\mathcal{O}(n \log n)$. For example, you may not call MergeSort on this data. In fact, there is a particularly clean solution that does not involve calling any algorithms from class.

Not Collected Questions

These questions will not be collected. Please do not submit your solutions for them. However, these are meant to help you to study and understand the course material better. You are encouraged to solve these as if they were normal homework problems.

1. As written in class, SelectionSort and BubbleSort if data is sorted will still run same time. Can we write SelectionSort differently (while preserving the main idea) to make it run faster if the data is nearly sorted? How about Bubble Sort? Does InsertionSort, as written in class, have this issue?

2. In lecture 3, we discussed the methods for inserting a new element into a max heap and for removing the max element from a max heap. From your understanding of the functions, write pseudo-code (or code in your favorite programming language) for these. Try to do this without referencing your textbook or any other sources that would contain the code directly.

3. In lecture 4, we saw (or will see, depending when you read this) that comparison based sorting algorithms have a running time of $\Omega(n \log n)$ to sort n comparable values. Suppose you wish to sort an array (which may have duplicates) of n unsigned (i.e., non-negative) integers, and you know that the maximum value in the array is m. Give an algorithm that takes $\mathcal{O}(m+n)$ to sort the array. You may assume that the keys do not have associated values with them.

Additionally, our textbook has excellent practice problems available. This homework (approximately) covers chapter one and sections 5.3, 5.4, 8.3, and 9.1. Other sections of those chapters may be covered soon.

If you need help deciding which problems to do, consider trying R-1.1, R-1.7, R-1.8 (you may wish to use a spreadsheet application), and any reinforcement problems that deal with $\mathcal{O}$ notation if you struggle with that topic. C-1.8, C-1.9, C-1.10, C-1.13, C-1.19, A-1.4, A-1.5, and A-1.14 are also suggested.

Chapter 5: R-5.4, R-5.5, R-5.8 (also, where could the min be?), R-5.15 (this is tougher than it looks), C-5.5, C-5.7, C-5.9.

Chapter 8: C-8.11, C-8.12, A-8.8

Chapter 9: R-9.3, C-9.1, C-9.2, C-9.3

If you think the other problems look fun too, you're probably right. We will get to some of them before too long.