CS 344, Section 5-8, Fall 2022
Homework 1.
100 points + 10 points EC

Note for this HW and all future HW: Unless otherwise specified, you may use any algorithm covered in class as a "black box" – for example you can simply write "sort the array in $O(n\log(n))$ time" without having to describe how to do this.

IMPORTANT NOTE: For all HWs and exams in this class, unless I explicitly say otherwise you can assume that you already have access to a sort function in your library. That is, in your pseudocode you can use a function Sort(A) which takes as input any array A (duplicates allowed) and outputs the array A in sorted order (smallest to largest). The running time of sort(A) is $O(n\log(n))$.

In particular, You will want to use sort(A) for at least one of the problems on this HW.

1 Problem 1 (2 point per part) – 20 total

For each of the following functions, state whether $f(n) = O(g(n))$ or $f = \Omega(g(n))$, or if both are true, then write $f = \Theta(g(n))$. No proofs required for this problem.

1. $f(n) = n^4 - 7n$ and $g(n) = n^3 + 10n^2$
2. $f(n) = (\sqrt{n})^3$ and $g(n) = n^2 - (\sqrt{n})^3$
3. $f(n) = n\log^3(n)$ and $g(n) = n^{log_2(5)}$
4. $f(n) = 2^{\log_2(n)}$ and $g(n) = n$
5. $f(n) = n/\log^2(n)$ and $g(n) = log^6(n)$
6. $f(n) = 4^n$ and $g(n) = 5^n$
7. $f(n) = \log_3(n)$ and $g(n) = \log_5(n)$
8. $f(n) = n^3$ and $g(n) = 2^n$

9. $f(n) = \log^2(n)$ and $g(n) = \sqrt{n}$
10. $f(n) = n\log(n)$ and $g(n) = n^2$

2 Problem 2 – 25 points total

- Part 1 (8 points): Prove by induction that $\sum_{i=0}^{k} i2^i = (k-1)2^{k+1} + 2$
- Part 2 (9 points): Prove that $\sum_{i=1}^{n} \log(i) = \Theta(n\log(n))$
- Part 3 (8 points): What is $\sum_{i=0}^{\log_2(n)} 8^i$ equal to in Θ-notation? (No formal proof necessary, just a brief explanation.)

  HINT: use the formula for geometric sum: $\sum_{i=0}^{k} r^i = (r^{k+1}-1)/(r-1)$. This is generally a very useful formula.

3 Problem 3 (10 points total)

- Part 1 (3 point): Simplify $64^{\log_{16}(n)}$; that is, write it as n to the power of some number.
- Part 2 (3 points): Simplify $5^{\log_7(n)}$ – in particular write it as n to the power of some number.
- Part 3 (4 points): Prove that for any constants c, c', $\log_c(n) = \theta(\log_{c'}(n))$.

4 Problem 4 (10 points total): find the closest pair

Consider the following problem:

- Input: An array A with n distinct (non-equal) elements
- Output: numbers x and y in A that minimize $|x-y|$, where $|x-y|$ denotes absolute-value(x-y). (If there are multiple closest pairs, you only have to return one of them.)

Write pseudocode for an algorithm for the above problem whose running time is $o(n^2)$. Note that this is little-o; in words, your running time must be *better* than $O(n^2)$. So an algorithm with running time $O(n^2)$ will receive very few points.

5 Problem 5 (15 points total): subarray with most 1s

Consider the following problem:

- INPUT: An array $A[0...n-1]$, where each $A[i]$ is either a 0 or a 1. Also, you are guaranteed that the length n of the array A is a multiple of 5.
- OUTPUT: Index $k \leq 4n/5$ such the subarray $A[k], A[k+1], ..., A[k+n/5-1]$ contains as many 1s as possible. If there exist multiple indices k that achieve this maximum, you only have to return one of them.

For example, say that $A = 1,0,0,1,0,0,1,0,1,0,0,0,1,0,0$. Note that $n = 15$ and $n/5 = 3$, so you are looking for the subarray of length 3 with the most number of 1s. The correct output is $k = 6$ because the subarray $A[6], A[7], A[8] = 1,0,1$ contains the maximum possible number of 1s among all subarrays of length 3.

The Problem: Write pseudocode for an algorithm that solves the above problem in $O(n)$ time.

HINT: Say that you already figured out the number of $1s$ in subarray $A[i]...A[i+n/5-1]$ for some i. How can you use this information to very quickly figure out the number of $1s$ in the next subarray $A[i+1]...A[i+n/5]$?

6 Problem 6 (20 points total): find the lurker

Consider the following input problem

- INPUT: a 2-dimensional array $A[0...n-1][0...n-1]$ with n rows and n columns. Note that you can use $A[i][j]$ to refer to the element in row i and column j, and that you can access any particular $A[i][j]$ in constant time. Each entry $A[i][j]$ is either 0 or 1.
- OUTPUT: find an index i such that for *all* $j \neq i$ it is the case that $A[i,j] = 1$ and $A[j,i] = 0$. If no such index exists, return "no solution".

Interpretation in words: The problem might seem more intuitive if you think of at as follows. Say that you have n people $p_0, ..., p_{n-1}$ and think of $A[i][j]$ as representing who follows whom on twitter: $A[i][j] = 1$ means that p_i follows p_j and $A[i][j] = 0$ means that p_i does not follow p_j. Note that it is possible that $A[i][j] = 0$ but that $A[j][i] = 1$. Your goal is to find the person p_i such that they follow everyone but no one follows them. If no such person exists you return "no solution".

6.1 Part 1 (2 points)

Say that $A[i][j] = 1$. From this piece of information alone, which index do you know is definitely NOT the final answer.

6.2 Part 2 (2 points)

Say that $A[i][j] = 0$. From this piece of information alone, which index do you know is definitely NOT the final answer.

6.3 Part 3 (16 points)

Write pseudocode that solves the above problem in $O(n)$ time. Note that the runtime should be $O(n)$, not $O(n^2)$.

HINT: you will want to use parts 1 and 2.

HINT: Similarly to sorted 2 sum from lecture, you will want to keep pointers that track of the set of indexes that are still potential candidates for the final solution.

7 Problem 7 – EXTRA CREDIT – 10 points

Consider the algorithm Foo(n). What is the running time in Θ notation? *For this problem, you must briefly justify your answer.* By briefly justify, I mean that you need to write enough that a knowledgeable reader would understand why your answer is correct.

Foo(n)

- For $i = 1$ to n

- $x = n$.
- While ($x \geq 2$)
  * $x \leftarrow \sqrt{x}$
  * Do placeholder stuff that takes $O(1)$ time.

NOTE: for this problem, you can assume that $\sqrt{x}$ can always be computed in $O(1)$ time.