CIS 320 Problem Set 1

Due: Wednesday 9/22/21

Welcome to Problem Set 1. A few notes before you begin.

- You may collaborate with up to 4 other people. All work must be written up individually. You may collaborate, but you cannot copy. E.g. you can talk to your friends about ideas for a problem together, but you cannot copy their solution and just change some words around. **List your collaborators at the top of your submission**.
- Googling or looking up solutions in any way is not allowed.
- Cheating is not worth it! Your grade in this course does not define your worth as a person, and in 10 years you will not care about your bad grade on a homework assignment. But you will care if you are caught plagiarizing: plagiarizing has serious consequences, including the potential of expulsion.
- These problems are designed to challenge you. Start them early; if we've done our job well writing them, you will have to chew on them for a while before finding the solutions, and that means you will do best if you can sleep on your solutions rather than starting them the night before the due date.
- You do not need to implement anything in code. In fact, please do not. Pseudocode or a clear English explanation of your algorithm are both acceptable: in some cases pseudocode may be clearer than plain English, and in others the plain English might be better.
- If a question asks you to come up with an algorithm with a certain target runtime (say $O(n \log n)$), and you don't know how to achieve this runtime but know how to solve the problem less efficiently (in, say, $\Theta(n^2)$ time), write that down! Depending on how inefficient your solution is compared to the benchmark, it will receive a varying amount of credit.
- All analysis must be mathematically rigorous. Any answer you provide should be proven.
- Remember, we can't evaluate your work if we can't understand it. Communicating mathematical and/or complex ideas is an important skill in computer science; treat your problem sets as practice.
- Have fun!!

Problem 1 (Dishwasher Drama)**.** It's your turn to run the dishwasher. Because your dishwasher sucks, you have to have all of the small dishes together and all of the large dishes together, or else they don't get clean. You only have two sizes of plates: small and large. You also need to keep all of the plastic dishes on the top shelf so they don't melt, and the ceramic ones on the bottom shelf. Your unhelpful roommate Todd already loaded the dishwasher, but put the dishes in willy-nilly. There is one open spot you can temporarily put a dish in, but you cannot put dishes on the counters or sink, which are all covered in dirty dishes (thanks Todd). You also have a broken wrist, so can only move a single dish at a time. Provide an algorithm to implement the sorting and analyze its runtime. Try to be as efficient as possible: you want to be able to spend your time on your algorithms homework not loading the dishwasher after all! Assume that any searches for a particular plate take negligible time; the only operation we care about is the act of moving a single plate, which you should consider a constant-time operation, and that n plates can fit on the top shelf and n plates can fit on the bottom shelf.

Problem 2 (Marvelous Matchings)**.** After being shown a weighted bipartite matching algorithm in CIS 320, you and your friends are so inspired that you begin talking about other algorithms that could accomplish the same task. Three of your friends each propose a different matching algorithm that seems promising.

Following the form discussed in lecture, the algorithms all take as argument a graph $G = (L, R, w)$. L is a set of n *left hand* vertices, R is a set of m *right hand* vertices, and w is a weight function $w : L \times R \to \mathbb{R}_{\geq 0}$, where $w(u, v)$ gives the weight of edge (u, v) for each pair of vertices $u \in L$ and $v \in R$. Each algorithm outputs μ, a list of vertex pairs representing a valid matching.

The three algorithms are as follows:

1. For each vertex L_i, iterate through R to find the unmatched vertex R_j that gives the highest match weight and add (L_i, R_j) to the matching. Ties are broken by selecting the first vertex of that weight seen.
2. Precompute the values of all the edges in the graph using w. Sort the edges by decreasing weight using mergesort. Then, iterate through the list of edges and add each edge to the matching only if both the vertices are currently not in the matching.
3. Iterate through all possible μ and select the μ which maximizes $w(\mu)$. Note that this includes iterating through obviously non-optimal matchings, such as the matching comprised of a single edge in G.

Each friend argues that their algorithm is the fastest and outputs the highest-weight matching $w(\mu)$ on any graph. You are skeptical of their claims and decide to analyze these algorithms for yourself.

For each algorithm A on the list above:

a) Assuming that $|L| = |R| = n$, for algorithms 1 and 2 determine their worst case time complexity, and show that the runtime of algorithm 3 is $\Omega(n! \cdot \texttt{superpolynomial}(n))$.[1]

Hint: The runtime of algorithm 3 scales with the number of possible matchings in a graph, so write an expression counting this and try to lower bound it. You do not need factorial approximations such as Stirling's to solve this problem.

b) Determine how A's matching weight compares to the maximum matching weight OPT in the worst case. Formally: for a particular graph G, let $w(A(G))$ be the weight of the matching which is output by A and let $OPT(G)$ be the weight of the maximum matching. Find the largest σ that satisfies $\sigma \cdot OPT(G) \leq w(A(G))$ over all possible choices of G. Your answer should be in one of the following forms: (i) a constant, (ii) an expression including n, or (iii) the statement that σ is arbitrarily small.

Note: You can assume all the above algorithms have data structures for constant-time lookups and updates in order to check whether a particular element is already in the matching. Beyond this, when analyzing the runtimes of these algorithms, consider them exactly as presented, without any extra techniques that might optimize the approach. The intention of this problem is algorithm analysis, not algorithm design.

[1] A function $f(n)$ is *superpolynomial* in n if $f(n) = \omega(n^c)$ for any constant $c > 0$.

Problem 3 (Synergistic Selection)**.** Consider $n \geq 1$ roller derby players, with skill levels $a_1, ..., a_n$. You are a coach and are in charge of selecting a team *of any size* from among these players. Above all, you want your team to be close-knit, that is, to consist of players with comparable skills.

A team is close-knit if for any player on the team, their skill level does not exceed the combined skill of any two players on the team. Consider the following examples:

- Bonnie has skill 100, Mia has skill 80, and Tara has skill 77. Together they form a close-knit team.
- Helix has skill 40, Tart has skill 40, Bart has skill 60 and Eagle has skill 100. Together they do not form a close-knit team, since Helix and Tart together have worse skill than Eagle alone: $40 + 40 < 100$.

Teams of size at most 2 are considered close-knit.

a) You are given a list of n teammates' skill levels in sorted order. Implement an $O(1)$ algorithm that checks whether they form a close-knit team.

b) Now you are given $n \geq 1$ players and a sorted list of their skill levels: $a_1 \leq a_2 \leq \ldots \leq a_n$. That is, player 1 has the lowest skill level and player n has the highest. You want to find a close-knit team with the maximum total skill level. The team can be any size. Design an algorithm that does so in $O(n)$ time. *Note:* even if you can't get an $O(n)$ solution, please provide the fastest algorithm you can come up with and its run-time. Any $O(n \log n)$ solution will get most of the credit.

Problem 4 (Ads Apocalypse)**.** Pierogi Supplies Incorporated (PSI), a specialty dumpling-making supply store in Philadelphia, is looking to raise their revenue. The manager of PSI is convinced that broadcasting in-store advertisements is the best way towards financial prosperity. To determine when to play the ads, she splits a business day into $N \geq 1$ small chunks of time numbered $1, \ldots, N$ (these chunks might be seconds or minutes), and faithfully records customer visit times on a sunny Monday: for each of the $n \geq 1$ customers on that day, she writes down a_i and b_i $(1 \leq i \leq n)$, the chunk indices corresponding to customer i's entry and exit time. E.g. if the business day were split into minute-long intervals, AJ is customer i and they enter 2 hours after the store was opened and leave 10 minutes later, the manager would write down $a_i = 120, b_i = 130$.

Absent further data, the manager extrapolates that on all future business days, customers will arrive and leave at the exact same times as today. Given this assumption, she tasks you with finding an optimal ad schedule. She clarifies that each ad takes up a single chunk of time, and for any customer i to hear that ad, the chunk number must be in $[a_i, b_i]$. The manager wants to expose every customer to at least 2 ads. Assume that the customers stay long enough that this is possible. Additionally, to reduce psychological damage to the store employees, the manager asks you to minimize the total number of ads played. Design a $O(n \log n)$ runtime algorithm that finds an optimal schedule.

Problem 5 (Gregarious Gourmands)**.** Filibucha, an elite kombucha lovers' club in Center City, keeps extensive records of past memberships. For each of its $n \geq 1$ past members, Filibucha keeps the start date s_i and end date t_i of their membership. In an attempt to reconstruct the club's illustrious history, Filibucha hires you to help them find all maximal Kombucha Contemporaries Cliques (KCC). A KCC is any subset of the n past members who could have plausibly enjoyed Filibucha's kombucha together on at least one date (we assume each member i is still around on the last day t_i of their membership). A KCC is maximal if no other KCC strictly contains it.

Let us denote by w the total size of all maximal KCC (for each member of the club, w counts them as many times as there are maximal KCC they were ever part of). Design an $O(n \log n + w)$-time algorithm that takes as input Filibucha's membership records, and outputs all maximal KCC's. For each maximal KCC, all of its members need to be listed. *Note:* even if you can't achieve the above runtime, please provide the fastest algorithm you can come up with and its run-time. Any $O(n^2)$ solution will get significant partial credit.