CS 5112 Fall 2023	HW1: Stable Matching, Greedy, Divide and Conquer
Name: Matthew Shen	Due: September 26, 11:59pm ET
NetID: mds377	
Collaborators: Linjing Rao(lr534)	

Problem 1. Your friend is wrapping up medical school and applying for residency programs, but is concerned and confused about how the matching system works. As the local expert on algorithms, your colleague wants your help understanding how matchings work.

(a) (2 points) Your friend tried to implement the textbook Gale-Shapley (G-S) algorithm in Python. We provide this code in the homework materials in problem_1/p1_a.py. Using this implementation, your friend thinks they found a way to propose a ranking that unfairly advantages them in getting the school of their choice.

There is a logical bug in the implementation. Provide a minimal test case demonstrating the bug, i.e., an input with the smallest possible n that, when run with the buggy implementation, outputs a non-stable matching. Write your test case input in problem_1/p1a_test.txt.

(b) (8 points) Now we turn to characterizing the performance of this implementation. Fix the bug from part (a) and conduct an of the implementation (see homework instructions in the first page). Include the fixed code in problem_1/p1_b.py.

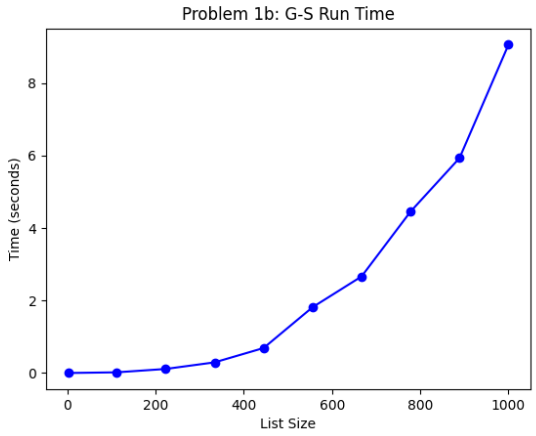
Provide a brief explanation of the performance you observe, and explain why the implementation does not achieve $\mathcal{O}(n^2)$ run time advertised for the G-S algorithm.

Answer: The environment used for the empirical performance analysis can be seen below:

- CPU: Intel(R) Core(TM) i7-8500Y CPU @ 1.50GHz (4 threads, 4.20GHz)
- OS & Version: Debian GNU/Linux 11 (bullseye)
- Memory: 15.53 GB

The graph created from the empirical performance analysis can be seen below:

The following code provided does not reach $\mathcal{O}(n^2)$. For the worst case scenario all hosptials have the same preference order for all doctors, and all doctors have the same preference order for all hospital. Additionally, the hospitals list is in the same order as the preferences that all the doctors have; therefore, a stable match is when the hosptial number and docter number matches (i.e. { 0:0, ... , n:n}).This means, in the first loop through the first hosptial gets matched and all other hospitals are not matched. In the second loop the program goes through all the hospitals again except for the first/matched hosptial. In the second loop on the second hosptial gets matched because all other hosptials want to get the second student, but the second student's top choice is the second hosptial. This means we get a loop has a time complexity of $\mathcal{O}(n)$ and is run $\mathcal{O}(n!)$ times. This means the algorithm is $\mathcal{O}(n!)$.



(c) (10 points) Correct and improve the run time of the G-S implementation and turn it in for auto-grading. It must be correct (always outputting a stable matching) and should run in the expected $\mathcal{O}(n^2)$ time. Provide a description of the optimizations you implemented. Additionally, provide an empirical performance analysis of your implementation in the same environment (same system and configuration) that you performed the earlier analysis. Include your new implementation in problem_1/p1_c.py.

Answer: Please answer here.

Problem 2. In the interval covering problem, one is given a time interval $[0, M]$ and a collection of closed subintervals $\mathcal{I} = \{[a_i, b_i] \mid i = 1, 2, \ldots, n\}$ whose union is $[0, M]$. (Interpret intervals as jobs and, interpret a_i and b_i as the start time and finish time of job $[a_i, b_i]$.) The goal is to find a subcollection $\mathcal{J} \subseteq \mathcal{I}$, containing as few intervals as possible, such that the union of the intervals in $\mathcal{J}$ is still equal to $[0, M]$.

You can assume that the numbers $M, a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n$ are all non-negative integers, and that $a_i < b_i$ for all i. You should assume that the list of intervals given in the problem input could be arbitrarily ordered; in other words, don't make an assumption that the input to the algorithm presents the intervals sorted according to any particular criterion.

(20 points) Below we have listed four greedy algorithms for this problem. At least one of them is correct, and at least one is incorrect. For every incorrect algorithm, provide an example of an input instance such that the algorithm outputs an incorrect answer in your write-up. (Either a set of intervals that fails to cover $[0, M]$, or a set that covers $[0, M]$ but has a greater number of intervals than necessary.) For every correct algorithm, write, "The algorithm is correct." For at least one of the correct algorithms, implement the algorithm to run in $O(n \log n)$; that is $\Theta(n \log n)$ or faster. In the write-up, indicate which algorithm you implemented and briefly describe your implementation strategy. Include your implementation in problem_2/p2.py.

(a) Select Latest Finish Time: Initialize $\mathcal{J} = \emptyset$. Until the intervals in $\mathcal{J}$ cover all of $[0, M]$, repeat the following loop: find the maximum $c \in [0, M]$ such that $[0, c]$ is covered by the intervals in $\mathcal{J}$, or if there is no such c then set $c = 0$; of all the intervals containing c select one whose finish time is as late as possible and insert it into J.

Answer: The algorithm is correct. This algorithm was implemented.

(b) Select Longest Interval: Initialize $\mathcal{J} = \emptyset$. Until the intervals in $\mathcal{J}$ cover all of $[0, M]$, repeat the following loop: among all the intervals in $\mathcal{I}$ that are not already contained in the union of the intervals in $\mathcal{J}$, select one that is as long as possible (i.e., that maximizes $b_i - a_i$) and add it to $\mathcal{J}$.

Answer:

$$\mathcal{I} = \{[0_0, 8_0], [8_1, 9_1], [9_2, 10_2], [0_3, 5_3], [5_4, 10_4]\}, [0, 10]$$

$$\mathcal{J}_{ans} = \{[0_0, 8_0], [8_1, 9_1], [9_2, 10_2]\}$$

$$\mathcal{J}_{opt} = \{[0_3, 5_3], [5_4, 10_4]\}$$

(c) Delete Earliest Redundant Interval: Initialize $\mathcal{J} = \mathcal{I}$. While there exists a redundant interval in $\mathcal{J}$ — one that is contained in the union of the other intervals in $\mathcal{J}$ — find the redundant interval in $\mathcal{J}$ with the earliest finish time and remove it from $\mathcal{J}$.

Answer:

$$\mathcal{I} = \{[0_0, 8_0], [8_1, 9_1], [9_2, 10_2], [0_3, 5_3], [5_4, 10_4]\}, [0, 10]$$
$$\mathcal{J}_{ans} = \{[0_0, 8_0], [8_1, 9_1], [9_2, 10_2]\}$$
$$\mathcal{J}_{opt} = \{[0_3, 5_3], [5_4, 10_4]\}$$

(d) Delete Latest Redundant Interval: Initialize $\mathcal{J} = \mathcal{I}$. While there exists a redundant interval in $\mathcal{J}$ — one that is contained in the union of the other intervals in $\mathcal{J}$ — find the redundant interval in $\mathcal{J}$ with the latest finish time and remove it from $\mathcal{J}$.

Answer: This answer is not correct.

$$\mathcal{I} = \{[0_0, 7_0], [7_1, 8_1], [8_2, 9_2], [0_3, 5_3], [5_4, 10_4]\}, [0, 10]$$

$$\mathcal{J}_{ans} = \{[0_0, 7_0], [7_1, 8_1], [8_2, 9_2]\}$$
$$\mathcal{J}_{opt} = \{[0_3, 5_3], [5_4, 10_4]\}$$

A note about tie-breaking: When implementing any of the algorithms above one must choose a tie-breaking rule. In other words, when more than one interval meets the criterion defined in the algorithm specification (such as the latest finishing redundant interval) one must specify which of the eligible intervals is chosen. In the context of this problem, if you are asserting that an algorithm is correct, it should mean that you believe the algorithm gives the correct answer on every input instance no matter how the tie-breaking rule is implemented; that is what your proof of correctness should show. When you assert an algorithm is incorrect, for full credit you should supply an input instance that leads to an incorrect output no matter how the tie-breaking rule is implemented. However, significant partial credit will be awarded for providing an input instance that leads to an incorrect output for some choice of tie-breaking rule, though not necessarily for every tie-breaking rule.

Problem 3. We consider how to count the number of large inversions between two rankings. Consider a set N with size n. A ranking is a 1-1 function rank : $N \to [1..n]$. In other words, rank maps each item in N to a unique integer, and therefore represents an ordering of the N items where lower integer values indicate higher rankings.

As described in the Kleinberg-Tardos textbook, Section 5.3, we can compare two rankings, rank_i and rank_j, by counting the number of inversions. We label each $x \in N$ with $\text{rank}_i(x)$, hence making $\text{rank}_i = [1, \ldots, n]$. Then, we define rank_j in terms of rank_i and count the number of inversions in rank_j.

Suppose now we want to consider large inversions, where we define a threshold $0 < \delta < n$ and let $\textbf{NLI}_{\delta,i,j}$ equal the number of pairs $x, y \in [1..n]$ such that $x < y$ and $\text{rank}_j(x) > \text{rank}_j(y)+\delta$. In other words, we only consider inversions large if their ranking differs by at least δ.

(a) (6 points) Design and implement a $\Theta(n^2)$ algorithm that iterates over every pair of points and determines $\textbf{NLI}_{\delta,i,j}$ given two rankings rank_i and rank_j and a threshold δ. Describe the algorithm in the write-up, and include your implementation in problem_3/p3_a.py.

Answer: Please answer here.

(b) (14 points) Design and implement a $\Theta(n \log n)$ algorithm to calculate $\textbf{NLI}_{\delta,i,j}$. Include your implementation in problem_3/p3_b.py and explain your algorithm in the write-up. Using your local experimental system perform an empirical performance analysis of both implementations (see homework instructions in the first page) and determine for what n does the asymptotically faster implementation dominates.

Answer: Please answer here.

Problem 4. Given a sequence of integers $a_1, \ldots, a_n$ we want to find the number of times the most frequent pairwise difference appears. Let $\delta_{i,j} = a_i - a_j$ for $i \neq j$ in the list. Suppose you know one of the modes (i.e. the most frequent value) of the list of all the $\delta_{i,j}$'s is δ_{mode}. We want to count for how many values $\delta_{i,j} = \delta_{mode}$.

(a) (8 points) Design and implement a $\Theta(n \log n)$ algorithm that given a sequence of integers $a_1, \ldots, a_n$ and δ_{mode} as described above, outputs the frequency of δ_{mode}. Include your implementation in problem_4/p4_a.py.

Answer: Please answer here.

(b) (7 points) Design and implement a $\Theta(n)$ algorithm that given a sequence of integers $a_1, \ldots, a_n$ and δ_{mode} as described above, outputs the frequency of δ_{mode}. Include your implementation in problem_4/p4_b.py.

Answer: Please answer here.

(c) (5 points) Provide a comparison of the empirical performance analysis of your algorithms from parts (a) and (b). There is a clear performance advantage of using the algorithm from part (b) over the one from part (a). Is there any trade-off? Hint: Consider the space complexity of the algorithms, which is the space an algorithm takes to execute.

Answer: Please answer here.

Note that the mode can be negative. If $\delta_{i,j}$ is a mode of the list, then $-\delta_{i,j} = \delta_{j,i}$ will be a mode too. The input to the function will be one of these modes, which can be positive, negative, or 0.

Problem 5. Eman baked some cookies for her friends, and asked Avital to help by distributing them among their friends. Their group of friends is very close-knit, so they all live in a building together, called "The Building".

The architect of "The Building" designed the property such that every floor is a very long corridor. To one side of the corridor are all the apartments in that floor, with the distance between every pair of consecutive doors being always the same. The architect is not very fond of elevators, so to the other side of the corridor, there is a pair of staircases in front of every apartment door, leading to all the floors in the building. Apartments are labeled first by floor and then by index in the floor, hence $apt_{(12,21)}$ is the 21st apartment in the 12 floor.

Avital is not a big fan of walking, so as she delivers the cookies, she may give some extra cookies to her friends, and ask them to distribute them to some other friends. All the friends may subsequently divide their cookies and ask other friends to help with the deliveries too.

Avital wants to minimize everyone's total number of steps. Assume that the distance between an apartment $apt_{(f,i)}$ and $apt_{(f,i+1)}$ is the same as that between $apt_{(f,i)}$ and $apt_{(f+1,i)}$.

Assume the friends are distributed across n apartments, where each apartment, including Eman's, has $p > n$ friends currently hanging out. Finally, assume that Avital starts at apartment $apt_{(1,1)}$, where Eman lives, and she can recruit some of the friends currently there.

(20 points) Design and implement an algorithm that given a list of apartments where the friends live, returns pairs (a, b), where friends from apartment a take the cookies to the friends from apartment b, that minimize the total number of steps. Your algorithm should run in $O(n^2 \log n)$. Describe your algorithm in the write-up and include your implementation in problem_5/p5.py.

Answer: Please answer here.

CS 5112 Fall 2023	HW1: Challenge Problem
Name: Matthew Shen	Due: September 26, 11:59pm ET
NetID: mds377	
Collaborators: Linjing Rao(lr534)	

A closer look at Integer Multiplication

Given two n-bit integers (represented as a list of binary values, i.e. base 2), $x, y \in \{0,1\}^n$, implement three approaches to computing their product $x \cdot y$ and perform an empirical performance analysis of the algorithms.

(2 points) The first approach will be a reference implementation. Compute the product by converting from their binary representation to an int, then multiplying. Please provide a brief description of your implementation here.

Answer: Please answer here.

(5 points) For the second approach, we would like you to implement the Karatsuba algorithm, described in section 5.5 of the Algorithm Design textbook. Please provide a brief description of your implementation here.

Answer: Please answer here.

(10 points) The final algorithm to implement is the Fast Fourier Transform (FFT), described in section 5.6 of the Algorithm Design textbook. Please provide a brief description of your implementation here.

Answer: Please answer here.

(3 points) Finally, provide an empirical performance analysis of the algorithms. Ensure that the analysis samples enough problem sizes (i.e. integer sizes n) such that the runtimes can be clearly differentiated.

Answer: Please answer here.