- The assignment is due at Gradescope on Friday, January 27 at 5:00p.
- You can either type your homework using LaTex or scan your handwritten work. We will provide a LaTex template for each homework. If you writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will facilitate the grading.
- You are permitted to study with up to 2 other students in the class (any section) and discuss the problems; however, *you must write up your own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please do list all your collaborators in your submission for each problem.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites. *Consulting problem solutions on the web is not allowed.*
- *Show your work.* Answers without justification will be given little credit.

PROBLEM 1 (25 POINTS) *There is a very simple algorithm, which I'll call the* Laissez-Faire (LF) *algorithm, to attempt to produce a stable matching between sets A and B of individuals, where $|A| = |B| = n$. (To be clear, this is the usual "two-type" or "bipartite" matching: we only allow an $a \in A$ to be matched with a $b \in B$.) Here is the algorithm:*

1. *Let M be an arbitrary initial matching between A and B;*
2. *If there is an unstable pair with respect to M, each of which prefers each other over their current match, let (a, b) be any such pair—chosen arbitrarily, if there's more than one. (If no such pair exists, **halt** with output M.)*
3. *Modify M by having the chosen (a, b) become matched; also, if a', b' are the pair of individuals that were just ditched, then (a', b') are re-matched with each other.*
4. *Goto line 2.*

Your job is to show that this algorithm is capable of running forever, without producing a stable matching—at least, this can happen for some value of n, on some collection of individual preferences, and for some sequence of possible choices made by the algorithm (which, note, is not fully deterministic). Hint: *there are two things you must produce: a description of preferences, and a "story," or sequence of pairing events, describing how the algorithm can run forever. It might be easier to tell the story first, and then design the preferences around the story. Note that this problem does not assert that a certain algorithm behavior is* inevitable*; this problem asserts that for some preference lists, a certain algorithm behavior is* possible.

Solution: Let us have two groups of size 4 with the following preferences.

Group A's preference lists (from most preferred to least preferred):

a_1: b_1, b_2, b_3, b_4

a_2: b_3, b_1, b_2, b_4

a_3: b_3, b_1, b_2, b_4

a_4: b_1, b_4, b_3, b_2

Group B's preference lists (from most preferred to least preferred):

b_1: a_2, a_1, a_4, a_3

b_2: a_2, a_1, a_4, a_3

b_3: a_4, a_2, a_3, a_1

b_4: a_4, a_3, a_1, a_2

Step 0: Starting matches $(a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4)$
Step 1: Preference switch match: (a_2, b_1). Ditched match: (a_1, b_2). Matches: $(a_1, b_2), (a_2, b_1), (a_3, b_3), (a_4, b_4)$
Step 2: Preference switch match: (a_2, b_3). Ditched match: (a_3, b_1). Matches: $(a_1, b_2), (a_2, b_3), (a_3, b_1), (a_4, b_4)$
Step 3: Preference switch match: (a_4, b_1). Ditched match: (a_3, b_4). Matches: $(a_1, b_2), (a_2, b_3), (a_3, b_4), (a_4, b_1)$
Step 4: Preference switch match: (a_1, b_1). Ditched match: (a_4, b_2). Matches: $(a_1, b_1), (a_2, b_3), (a_3, b_4), (a_4, b_2)$
Step 5: Preference switch match: (a_4, b_3). Ditched match: (a_2, b_2). Matches: $(a_1, b_1), (a_2, b_2), (a_3, b_4), (a_4, b_3)$
Step 6: Preference switch match: (a_4, b_4). Ditched match: (a_3, b_3). Matches: $(a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4)$

By Step 6, we have reached the same matching as Step 0. Therefore, given the matching preferences stated above, and following the exact preference switches made in steps 1-6, the algorithm can repeat itself forever.

PROBLEM 2 (25 POINTS) *Solve exercise 3 from Chapter 4 in the Kleinberg-Tardos textbook.*

Solution: Input: A set of packages $i_1, i_2, \cdots i_n$, with a respective weight w_i for each package.

Desired Output: the minimum number of trucks required to ship packages $i_1, i_2, \cdots i_n$ in the order that they arrive.

Correctness:Feasibility+ Optimality

Feasibility: As packages are processed as they arrive, all packages will be processed and loaded in the order that they arrive by the greedy algorithm, meaning the greedy algorithm will ship all packages and maintain order. Therefore the solution is feasible.

Optimality: Let $F_O(S_n)$ be the optimal number of trucks required to ship the set $S_n = i_1, i_2, \cdots i_n$ of packages, where $|S_n| = n$. Let $F_A(S_n)$ be the number of trucks required to ship the set $S_n = i_1, i_2, \cdots i_n$ of packages as determined by the greedy algorithm.

Assume that $F_A(S_n)$ is not optimal, or $F_A(S_n) \neq F_O(S_n)$ and $F_A(S_n) \leq F_O(S_n)$. We want to prove $F_A(S_{n+1}) \leq F_O(S_{n+1})$. Let $q_1, q_2, \cdots q_k$ be the set of all trucks produced by $F_A(S_n)$ and let $p_1, p_2, \cdots p_n$ be the set of trucks produced by $F_O(S_n)$.

Base case: consider 1 package. Optimal amount is 1 truck, which will be used by greedy. Therefore, $F_A(S_1) \leq F_O(S_1)$.

Induction: Let there be some package, j such that j is the first package where $F_O(S_n)$ uses a different packing strategy than $F_A(S_n)$. Let us consider the truck that deviates by $F_O(S_n)$ to be m, and let the corresponding truck filled by $F_A(S_n)$ to be l. As we cannot over fill a truck and j causes m to deviate, then m must have less packages, so $m < l$ (in terms of fullness). Therefore the packages of weight $l - m$ that would have been in m if it was full must be in some previous truck, otherwise every truck that deviated from the greedy algorithm would produce more packages which would only cause the same amount or more trucks. However, as j is the first truck to deviate from as the greedy algorithm and every truck $q_1, q_2, \cdots q_j$ is full, then there is no place for the packages in the previous trucks, meaning that these packages must be placed in future trucks. Therefore, for any arbitrary $j + 1$ package that causes a truck to be packed differently, not following the greedy algorithm will only result in the same amount or more packages left after the truck leaves, and

therefore the same amount or more trucks. Therefore packing the truck in a different way cannot decrease the amount of trucks so the algorithm is optimal.

PROBLEM 3 (25 POINTS) *Solve exercise 5 from Chapter 4 in the Kleinberg-Tardos textbook.*

Solution:
Input: A set of sorted integers $i_1, i_2, \cdots i_n$, where integer represents a house with the distance from the starting point.

Desired Output: the minimum number of cell stations required to be built so that every house is within 4 miles of a station.

Run time: Assuming that **the list is sorted in ascending distance** from the starting point and assuming that building a cell station and moving to the next house in the list are $O(1)$ run time functions, so the greedy algorithm should run in $O(N)$ for N houses

Algorithm($i_1, i_2, \cdots i_n$ **)**
$S = \emptyset$ *//Set of coordinates of built stations*
$l = -1$ *//current cell tower range*
For every k in $i_1, i_2, \cdots i_n$ do:
    if $l \ngeq i$ *//check if k is within station range*
        $S = S \cup i$ *//add to ordered set*
        $l = k + 4$ *//build station at house, extend range by 4 miles*
return $|S|, S$ *// return amount of stations, set of station coordinates*
Correctness:
Feasibility: As the algorithm processes houses linearly by increasing distance, for every house in $i_1, i_2, \cdots i_n$, either a station will be built at it's location or the house is within 4 miles of a previously built station. Therefore every house will be within 4 miles of a station.
Optimality: Let $F_O(S_n)$ be the optimal algorithm to determine the minimum number of stations required to cover the set $S_n = i_1, i_2, \cdots i_n$ of houses. Let

$q_1, q_2, \cdots q_h$ be the set of all stations produced by $F_N(S_n)$, the greedy algorithm, and let $p_1, p_2, \cdots p_f$ be the set of all stations produced by $F_O(S_n)$, the optimal algorithm.
We claim through induction that for $i_1, i_2, \cdots i_n$, $F_N(S_n) \leq F_O(S_n)$, or $F_N(S_n)$ will never use more stations to cover the same amount of houses.
Base case: $|S_1| = 1$, for 1 house only 1 station will be built by greedy algorithm, which is optimal as that house requires at least 1 station within 4 miles of it so $F_O(S_1) = F_N(S_1)$, which holds the hypothesis.

Assume that for some arbitrary j, $F_N(S_j) \leq F_O(S_j)$ and we want $F_N(S_{j+1}) \leq F_O(S_{j+1})$. Let us consider $j+1$, or the following house after j. If $j+1$ is in the range of a previous station $F_N(S_j)$, then we don't have to build a station and have $F_N(S_{j+1}) \leq F_O(S_{j+1})$ and we are done. Therefore, assume that $j+1$ is not in the range of $F_N(S_j)$. $j+1$ must also be in the range of $F_O(S_j)$, or else we would have $F_N(S_{j+1}) \leq F_O(S_{j+1})$ (both algorithms build a station). Therefore, there must be some position l where $F_O(S_j)$ places a station that is within 4 miles of $j+1$, that $F_N(S_j)$ doesn't place. Therefore there must be one fewer station placed in $F_N(S_j) \leq F_O(S_j)$, because other wise we have $F_N(S_j) \leq F_O(S_j) + 1$ (where $+1$ is adding an additional station at l), and we can grow at most by 1 or 0 stations per house, meaning that we would subsequently have $F_N(S_{j+1}) \leq F_O(S_{j+1})$ and we are done. Therefore, let q_r be some arbitrary station placed in $q_1, q_2, \cdots q_h$ that is not in $p_1, p_2, \cdots p_f$. However, as $F_N(S_j)$ only builds stations when encountering a house and when 4 miles from the previous station, if q_r is not in $p_1, p_2, \cdots p_f$, then there must be some house h which not within 4 miles of a station and therefore is not covered by $F_O(S_j)$. Therefore this is a contradiction, meaning that it is impossible to have one less station in $p_1, p_2, \cdots p_f$, meaning that if a station is at l we have $F_N(S_j) \leq F_O(S_j) + 1$ which implies $F_N(S_{j+1}) \leq F_O(S_{j+1})$, or if it is not built, both algorithms would have to build the station, meaning we have $F_N(S_{j+1}) \leq F_O(S_{j+1})$. Therefore, for all arbitrary j in $i_1, i_2, \cdots i_n$ greedy will stay ahead, meaning the algorithm is optimal.

PROBLEM 4 (25 POINTS) *Solve exercise 6 from Chapter 4 in the Kleinberg-Tardos textbook.*

Solution: Input: A set of triathlon contestants $S_n = i_1, i_2, \cdots i_n$, with a respective swimming, w_n biking b_n, and running time r_n for each n, where each contestant cannot swim concurrently but can bike and run concurrently.

Desired Output: A sorted set of starting the triathlon contestants so that overall completion time is minimized.

Run time: Time needed to sort list, is $O(N \cdot log(N))$. Time to query through list is $O(N)$. Therefore we have $O(N \cdot log(N) + N) = O(N \cdot log(N))$ run time.
Algorithm($i_1, i_2, \cdots i_n$)

Process+relabel $i_1, i_2, \cdots i_n$ by largest to smallest $b_n + r_n$, and renumber contestants such that $b_1 + r_1 \geq b_2 + r_2 \geq \cdots \geq b_n + r_n$

$A = 0$ // *Overall run time*
$k = 0$ // *Current max lateness*
$S = \emptyset$ // *Building ordered set of contestants*
For j in $i_1, i_2, \cdots i_n$ do:
    $A = A + w_j$ // *Increment overall run time by current j swim time*
    $k = max(0, b_j + r_j, k - w_j)$// *Update maximum lateness*
    $S = S \cup j$ // *build ordered set*
$A = A + k$ // *Sum of swim times + maximum lateness equals overall run time*
return A, S //*return overall run time A, ordered set S*

Correctness:
Feasibility: Every contestant in $i_1, i_2, \cdots i_n$ is processed and their run time is added up such that no contestants are swimming concurrently ($A = A + w_j$ for each iteration) and k updates the max $b_j + r_j$ for each contestant while subtracting the swimming time, therefore the algorithm will give enough time for the i_n person to finish the race. Therefore the run time A is feasible.
Optimality: In order to design a schedule which minimizes the overall run time of the triathlon, we must first determine what composes this run time. Let k be some non-negative integer. Therefore, the completion time is:

$$A = w_1 + w_2 + w_3 + \cdots + w_n + k$$

Proof: No two people can be at the pool at once, therefore the optimal completion time will have to be $w_1 + w_2 + w_3 + \cdots + w_n$, plus some other non-negative integer k.
In order to determine k, consider that after w_1 time, i_1 will immediately start running and then biking while the other competitors are swimming. Because we want to minimize the overall completion time of the algorithm, we want to subtract all the following swimming times from a contestant's biking and running time to determine how that contestant impacts overall performance. We will define this operation below as the maximum lateness, $T(j)$ for some j

in S_n.

$$T(j) = (b_j + r_j) - (w_{j+1} + w_{j+2} + w_{j+3} + \cdots + w_n)$$

In other words, $T(j)$ is the amount that j could potentially extend the race based on the swimmers after him. Because $(j+1), (j+2), + \cdots + n$ people still have to swim, we subtract their swim times from $t(j)$ to get the lateness amount.
Now we return to k with the following formula, for $i_1, i_2, \cdots i_n$

$$k = max(0, t(1), t(2), t(3) \cdots t(n))$$

Proof: We include 0 in k, as we can never have a race shorter than the sum of all swimming times, so $k \geq 0$. We take maximum lateness, as we can only finish the triathlon when every n in S_n is done with the race, meaning that they have completed their biking and running relative to the swimmers after them.
Therefore in order to minimize run time A, we must minimize k.

Optimality proof: Assume $F_N(S_n)$ is non optimal. Therefore, have the optimal solution $F_O(S_n)$ adopt some schedule where a contestant with a larger $b_j + r_j$, some arbitrary j in $i_1, i_2, \cdots i_n$ is scheduled before a contestant with a smaller $b_l + r_l$, some arbitrary l in $i_1, i_2, \cdots i_n$ and where $t(j) = k$, or j has maximum lateness. Therefore, there is some competitor l scheduled to swim before j where $(b_l + r_l) < (b_j + r_j)$. In other terms, there is an inversion between l and j. Let t be maximum lateness caused by j before switching j and l and t' be maximum lateness after the switch. Therefore we have,
$t = (b_j + r_j) - (w_{j+1} + w_{j+2} + w_{j+3} + \cdots + w_n)$
$t' = (b_j + r_j) - (w_2 + w_3 + w_4 + \cdots + w_n)$.
Note that as j is scheduled after l, $j > 1$
Through these we achieve:

$$\begin{aligned}
t &= (b_j + r_j) - (w_{j+1} + w_{j+2} + w_{j+3} + \cdots + w_n) \\
t - (w_2 + w_3 \cdots w_j) &= (b_j + r_j) - (w_2 + w_3 + w_4 + \cdots + w_n) \\
t - (w_2 + w_3 \cdots w_j) &= t' \\
t &= t' + ((w_2 + w_3 \cdots w_j)) \\
// \text{ As } 0 &\leq (w_2 + w_3 \cdots w_j) \\
t &\leq t'
\end{aligned}$$

Therefore, we have proved that exchanging an inversion will not increase maximum lateness but can decrease t or overall maximum lateness. Therefore, we will exchange competitors until $b_1 + r_1 \geq b_2 + r_2 \geq \cdots \geq b_n + r_n$, meaning that the algorithm that schedules the fewest inversions will always stay ahead in comparison to any other algorithm. As proved in class, the schedule with

the no inversions is optimal if switching competitors in an inversion reduces inversions and does not increase maximum lateness. As shown by $t \leq t'$, switching inversions cannot increase run time for the triathlon run time problem. As greedy schedules no inversions, greedy stays ahead in comparison to any other schedule. As $F_N(S_n)$ picked a schedule with no inversions, it must be optimal.