Practice Problems for the Final Exam

- The following set of practice problems has been compiled from previous semesters to aid you in preparing for the final exam.
- These problems should not be taken as a strict gauge for the difficulty level of the actual exam.
- Where possible we have included the "points" value of a problem, indicating roughly how much time (in minutes) you should spend on the problem (in minutes).
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to *give an algorithm*, you may describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- If you think you have found a bug/errata, please let us know via Piazza.
- ***We encourage you to go through these problems without first looking at the solutions!***

Problem-1: T/F Questions. [3 points each] Answer *true* or *false* to each of the following. *No justification is required for your answer.*

(a) [T/F] Let n and k be integers greater than 10. Consider the following family $\mathcal{H}$ of hash functions that map elements from the set $\{1, \ldots, n\}^k$ to the set $\{1, \ldots, n\}$. $\mathcal{H}$ contains for every $1 \leq i \leq k$ a function h_i that maps $(x_1, \ldots, x_k) \in \{1, \ldots, n\}^k$ to x_i. $\mathcal{H}$ is a universal hash family.

(b) [T/F] Consider the following algorithm for approximating the weight of the minimum spanning tree. Given a weighted graph $G = (V, E)$ as input, pick the first vertex, $s \in V$, from the input, and compute the tree of shortest paths between s and all $v \in V$. Output the sum of the weights of all the edges in the tree. This algorithm is a 2-approximation for the minimum spanning tree problem.

(c) [T/F] Suppose that there is a polynomial time reduction from the general vertex cover problem to bipartite matching. Then, there is a polynomial time algorithm that solves SAT.

(d) [T/F] A randomized algorithm that is always correct and has an expected running time that is linear may be converted into a randomized algorithm that always runs in linear time but is correct only with probability 99%.

(e) [T/F] For every NP-hard minimization problem there is a constant $c > 1$, such that approximating the problem to within a factor c is NP-hard.

(f) [T/F] Finding a maximum clique in a graph with n vertices can be formulated as a linear program whose size is $O(n^3)$.

(g) [T/F] To find the maximum flow in any flow network G with unit capacities, we can perform the following procedure: until there are no more augmenting paths, repeatedly find an augmenting path from the source to the sink, and delete the edges in the path.

(h) [T/F] Given a maximization linear program LP1 and its dual LP2, the objective value of every feasible solution to LP1 is not larger than the objective value of any feasible solution to LP2.

(i) [T/F] We consider an implementation of the ACCESS operation for self-organizing lists which, whenever it accesses an element x of the list, transposes x with its predecessor in the list (if it exists). This heuristic is 2-competitive.

(j) [T/F] Negating all the edge weights in a weighted undirected graph G and then finding the minimum spanning tree gives us the *maximum*-weight spanning tree of the original graph G.

(k) [T/F] Suppose we are given an array A of distinct elements, and we want to find $n/2$ elements in the array whose median is also the median of A. Any algorithm that does this must take $\Omega(n \log n)$ time.

(l) [T/F] In a weighted connected graph $G = (V, E)$, each minimum weight edge belongs to some minimum spanning tree of G.

(m) **[T/F]** Let A and B be two decision problems. If A is polynomial time reducible to B, and $B \in$ NP, then $A \in$ NP.

(n) **[T/F]** Given a set of points $\{x_1, \ldots, x_n\}$, and a degree n bounded polynomial

$$P(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

we can evaluate $P(x)$ at $\{x_1, \ldots, x_n\}$ in $O(n \log n)$ time using the FFT algorithm presented in class.

(o) **[T/F]** The gradient descent algorithm applied to the function:

$$f(x) = (x+1)^2(x-1)(x-2) = x^4 - x^3 - 3x^2 + x + 2$$

is guaranteed to converge to a global minimum of that function.

(p) **[T/F]** Every function that has a single global minimum is convex.

(q) **[T/F]** Consider the function $f(x) = \frac{x^3}{3}$. The gradient descent algorithm, initialized at some non-zero value (i.e. $x^{(0)} \neq 0$), and run with the following iteration scheme:

$$x^{(t+1)} \longleftarrow x^{(t)} - \frac{1}{2} f'(x^{(t)})$$

always converges to $-\infty$.

Problem-2: Short Answer Questions [8 points each]

(a) Consider the *CLIQUE* problem: Given an undirected graph $G = (V, E)$ and a positive integer k, is there a subset C of V of size at least k such that every pair of vertices in C has an edge between them?

Ben Bitdiddle thinks he can solve the clique problem in polynomial time using linear programming as follows:

- Let each variable in the linear program represent whether or not each vertex is a part of our clique. Add constraints stating that each of these variables must be nonnegative and at most one.
- Go through the graph G and consider each pair of vertices. For every pair of vertices where there is not an edge in G, add a constraint stating that the sum of the variables corresponding to the endpoint vertices must be at most one. This ensures that both of them cannot be part of a clique if there is no edge between them.
- The objective function is the sum of the variables corresponding to the vertices. We wish to maximize this function.

Ben argues that the value of the optimum must be the size of the maximum-size clique in G, and we can then simply compare this value to k. *Explain the flaw in Ben's logic.*

Problem-3: Who is missing? [15 points]

In this problem, your input is a stream of m integers, all of which are from the set $\{1, \ldots, n\}$.

(a) Suppose $m = n - 1$. This means that there is at least one element from $\{1, \ldots, n\}$ that does not appear in the stream. Suppose you are also guaranteed that there is *exactly* one element that does not appear (so all other elements appear exactly once). Give a deterministic streaming algorithm to find this element with one pass using only $O(\log n)$ space.

(b) Suppose $m = n - 2$. Then at least two elements do not appear in the stream. Again you are guaranteed that exactly two elements do not appear (so all other elements appear exactly once). Give a deterministic streaming algorithm to find both missing elements with one pass using $O(\log n)$ space.

Problem-4: Set Cover [30 points]

Recall the SET-COVER problem: Given a set U of size n and subsets $S_1, \ldots S_m$ of U such that $\bigcup_{i=1}^{m} S_i = U$, find a set of indices $I \subseteq \{1, \ldots, m\}$ with minimum cardinality ($|I|$) such that $\{S_i\}_{i \in I}$ covers U. I.e. $\bigcup_{i \in I} S_i = U$.

Consider the following $(\ln n + 1)$-approximation algorithm for SET-COVER: if there exists a set of indices $J \subseteq \{1, \ldots, m\}$ of size $|J| = k$ such that $\{S_j\}_{j \in J}$ covers U, then the algorithm outputs a set of indices $I \subseteq \{1, \ldots, m\}$ such that $\{S_i\}_{i \in I}$ covers U and $|I| \leq k(\ln n + 1)$. The algorithm is shown below.

1: $I = \emptyset$
2: **while** U is not empty **do**
3: Pick the largest subset S_i
4: $I = I \cup \{i\}$
5: Remove all elements of S_i from U and from the other subsets
6: **end while**
7: **return** I

In this problem, you may assume a stronger assumption: that there exists a set of indices $J \subseteq \{1, \ldots, m\}$ of size $|J| = k$ such that $\{S_j\}_{j \in J}$ covers U "100 times" - i.e. for each element $u \in U$, there are at least 100 elements j's of J such that $u \in S_j$.

Given this assumption, prove that the algorithm given in class outputs a set of indices $I \subseteq \{1, \ldots, m\}$ such that $\{S_i\}_{i \in I}$ covers U and $|I| \leq \frac{k \ln n}{100} + 1$ by following those two steps:

(a) Let U_l be the value of U after the l-th iteration of the algorithm. (Note that $U_0 = U$) Prove that there exists $j' \in J$ such that $|S_{j'} \cap U_l| \geq \frac{100}{k}|U_l|$.

(b) Now show that the algorithm described above outputs I of size at most $\frac{k \ln n}{100} + 1$.
Hint: the following inequality might be useful: $1 - \alpha \leq e^{-\alpha}$ for all $\alpha > 0$.

Problem-5: Fruitloaf Factory [15 points]

Food Inc has been told by the FDA that they must list more information on the nutrition label, and that customers will refuse to buy the product if it has too much fat or sugar. Some important details about FruitLoaf:

- FruitLoaf is composed entirely of n ingredients: $i_1, i_2, \ldots, i_n$.
- The *original* FruitLoaf had exactly 1 gram of each ingredient i_j.
- The *new* FruitLoaf should be exactly w grams, consisting of g_j grams of ingredient j.
- Customers will not buy a FruitLoaf unless the total amount of sugar is less or equal to S and the total amount of fat is less than or equal to F.
- Each ingredient i_j has f_j grams of fat per gram of ingredient, and s_j grams of sugar per gram of ingredient.

Customers like the old FruitLoaf flavor (without knowing what it contains), so you're hoping to stay as close to the old recipe as possible. Customers' taste of the difference is the Manhattan distance between the two recipes. In other words, the distance is: $\sum_{j=1}^{n} |1 - g_j|$.

Given the values of w, S, and F as input, *write a linear program in standard form to minimize the difference between the old FruitLoaf flavor profile and the new one subject to the constraints on the weight, fat content and sugar content.* Note that your linear program needs to be in standard form. That is, if it has variables $\{x_1, x_2, \ldots, x_n\}$, it must be written in this form:

$$
\begin{aligned}
\max \quad & c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \\
\text{subject to} \quad & a_{(1,1)} x_1 + a_{(1,2)} x_2 + \ldots + a_{(1,n)} x_n \leq b_1 \\
& a_{(2,1)} x_1 + a_{(2,2)} x_2 + \ldots + a_{(2,n)} x_n \leq b_2 \\
& \ldots \\
& a_{(m,1)} x_1 + a_{(m,2)} x_2 + \ldots + a_{(m,n)} x_n \leq b_m \\
& x_j \geq 0 \text{ for all } 1 \leq j \leq n
\end{aligned}
$$

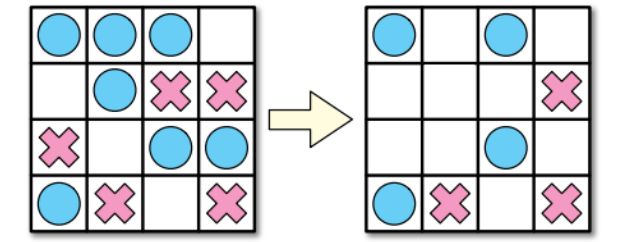
Problem-6: Board Solitaire [20 points]

Consider the following puzzle. The puzzle consists of an $m \times n$ grid of squares, where each square is either empty or occupied by a red or blue stone. The goal of the puzle is to remove some of the given stones so that the remaining stones satisfy two conditions:

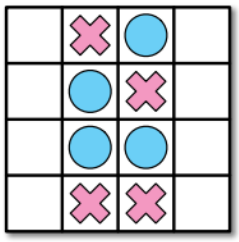
(a) Every row contains at least one stone,

(b) No column contains stones of both colors.

Note that for some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Let $SOLITAIRE$ denote the decision problem of determining, given an initial configuration of red and blue stones, whether the puzzle can be solved. Show that $SOLITAIRE$ is NP-complete. *You may only use a reduction from* $3SAT$ *to* $SOLITAIRE$.

Problem-7: Forgetful Forrest [15 points]

Prof. Forrest Gump is very forgetful, so he uses automatic calendar reminders for his appointments. For each reminder he receives for an event, he has a 50% chance of actually remembering the event (decided by an independent coin flip).

(a) Suppose we send Forrest k reminders for each of n events. What is the expected number of appointments Forrest will remember? Give your answer in terms of k and n.

(b) Suppose we send Forrest k reminders for a *single* event. How should we set k with respect to n so that Forrest will remember the event with high probability, i.e., $1-1/n^{\alpha}$?

(c) Suppose we send Forrest k reminders for each of n events. How should we set k with respect to n so that Forrest will remember *all* the events with high probability, i.e., $1-1/n^{\alpha}$?

Problem-8: Piano Recital [15 points]

Prof. Chopin has a piano recital coming up, and in preparation, he wants to learn as many pieces as possible. There are m possible pieces he could learn. Each piece i takes p_i hours to learn.

Prof. Chopin has a total of T hours that he can study by himself (before getting bored). In addition, he has n piano teachers. Each teacher j will spend up to t_j hours teaching. The teachers are very strict, so they will teach Prof. Chopin only a single piece, and only if no other teacher is teaching him that piece.

Thus, to learn piece i, Prof. Chopin can either (1) learn it by himself by spending p_i of his T self-learning budget; or (2) he can choose a unique teacher j (not chosen for any other piece), learn together for $\min\{p_i, t_j\}$ hours, and if any hours remain ($p_i > t_j$), learn the rest using $p_i - t_j$ hours of his T self-learning budget. (Learning part of a piece is useless.)

(a) Assume that Prof. Chopin decides to learn exactly k pieces. Prove that he needs to consider only the k lowest p_is and the k highest t_js.

(b) Assuming part (a), give an efficient greedy algorithm to determine whether Prof. Chopin can learn exactly k pieces. Argue its correctness.

(c) Using part (b) as a black box, give an efficient algorithm that finds the maximum number of pieces Prof. Chopin can learn. Analyze its running time.

Problem-9: Startups are Hard [20 points]

For your new startup company, *Uber for Algorithms*, you are trying to assign projects to employees. You have a set P of n projects and a set E of m employees. Each employee e can only work on one project, and each project $p \in P$ has a subset $E_p \subseteq E$ of employees that must be assigned to p to complete p. The decision problem we want to solve is whether we can assign the employees to projects such that we can complete (at least) k projects.

(a) Give a straightforward algorithm that checks whether any subset of k projects can be completed to solve the decisional problem. Analyze its time complexity in terms of m, n, and k.

(b) Is your algorithm in part (a) fixed-parameter tractable (FPT)? Explain briefly.

(c) Recall the *3D-MATCHING Problem*: You are given three sets X, Y, Z, each of size m; a set $T \subseteq X \times Y \times Z$ of triples; and an integer k. Determine whether or not there is a subset $S \subseteq T$ of (at least) k disjoint triples.

Show that the problem is NP-hard via a reduction from *3D-MATCHING* matching.

Problem-10: Load Balancing [15 points]

Suppose you need to complete n jobs, and the time it takes to complete job i is t_i. You are given m identical machines $M_1, M_2, \ldots, M_m$ to run the jobs on. Each machine can run only one job at a time, and each job must be completely run on a single machine. If you assign a set $J_j \subseteq \{1, 2, \ldots, n\}$ of jobs to machine M_j, then it will need $T_j = \sum_{i \in J_j} t_i$ time. Your goal is to partition the n jobs among the m machines to minimize $\max_i T_i$.

(a) Describe a greedy approximation algorithm for this problem.

(b) Show that your algorithm from part (a) is a 2-approximation algorithm.

Hint: First determine an ideal bound on the optimal solution OPT and then consider the machine M_ℓ with the longest T_ℓ, and the last job i^* that was added to it.

Problem-11: Testing Polynomial Products [15 points]

An instance of the *Testing Polynomial Products (TPC)* decision problem is a quadruple of univariate polynomials (A, B, C, D). We say that (A, B, C, D) is a yes-instance if $AB = CD$, and a no-instance otherwise.

Consider the following probabilistic algorithm for solving the problem. On input polynomials (A, B, C, D): Choose uniformly at random an integer y from a set of integers $\{1, \ldots, m\}$. Compute the product $L = A(y)B(y)$ and the product $R = C(y)D(y)$ If $R = L$ outputs "PASS", otherwise output "FAIL".

(a) Analyze the probability that the algorithm is correct as a function of m and n (where the degree of any of the polynomials is at most n). Namely, it passes yes-instances and fails no-instances.

(b) How large should we make m to make the probability of correctness at least $1 - \frac{1}{\log n}$.

Problem-12: Simplex Subs [20 points]

Sandwich shop Simplex Subs sells sandwiches which are made up of one type of meat, one type of vegetable, and one type of bread. There are n different types of each ingredient: the meats are $m_1, m_2, \ldots, m_n$, the vegetables are $v_1, v_2, \ldots, v_n$, and the breads are $b_1, b_2, \ldots, b_n$. After years of market research, they have developed a compatability mapping C, where each meat is mapped to a list of compatable vegetables and each vegetable is mapped to a list of compatable breads. Simplex subs only makes sandwiches of the form (m, v, b), where $b \in C(v)$ and $v \in C(m)$.

Each ingredient has a current supply, which is a non-negative integer. One unit of supply is enough for one sandwich, so a sandwich (m, v, b) uses one unit of meat m, one unit of vegetable v, and one unit of bread b.

Here is an example for $n = 2$:

ingredient	supply	compatability
m_1	10	v_1, v_2
m_2	10	v_2
v_1	15	b_1
v_2	15	b_2
b_1	5	N/A
b_2	20	N/A

Simplex Subs has just received a large order for k sandwiches. **The order only specifies which meats the sandwiches should have** and can be represented as a list of n numbers $[x_1, x_2, \ldots, x_n]$, where $\sum x_i = k$ and each x_i is a positive integer representing the number of sandwiches that should have meat m_i.

In the above example with $n = 2$, an order of $[10, 10]$ corresponds to 10 sandwiches with meat of type m_1 and 10 with meat of type m_2, and this order can be fulfilled by making 5 sandwiches (m_1, v_1, b_1), 5 of (m_1, v_2, b_2) and 10 of (m_2, v_2, b_2).

(a) Design and analyze an algorithm which determines whether or not Simplex Subs has the required supply to fulfill a given order.

(b) Now suppose you are given a function P which maps each ingredient to a non-negative cost. For each sandwich (m, v, b), the cost to Simplex Subs is $P(m)+P(v)+P(b)$. Define a linear program which finds the minimum total cost of the k sandwiches required to fulfill the order. More points are awarded to linear programs which use asymptotically fewer variables and constraints.

Problem-13: Multi-operation Queue [15 points]

Recall that a standard (FIFO) queue maintains a sequence of items subject to the following operations.

- PUSH(x): Add item x to the end of the sequence.
- PULL(): Remove and return the item at the beginning of the sequence.

It is possible to implement a queue using a doubly-linked list and a counter, so that the worst-case time per operation is $O(1)$.

(a) Now suppose we want to support the following operation instead of PULL:

- MULTIPULL(k): Remove the first k items from the front of the queue, and return the k^{th} item removed.

Suppose we use the following algorithm to implement MULTIPULL:

```
MULTIPULL(k)
1  for i = 0 to k − 1:
2       x = PULL()
3       if i == k − 1:
4            return x
```

Use the *accouting* method to prove that in any intermixed sequence of PUSH and MULTIPULL operations, the amortized cost of each operation is $O(1)$.

(b) Now suppose we also want to support the following operation instead of PUSH:

- MULTIPUSH(k, x): Insert $x, x + 1, \ldots, x + k - 1$ into the back of the queue.

Suppose we use the following algorithm to implement MULTIPUSH:

```
MULTIPUSH(k, x)
1  for i = 0 to k − 1:
2       PUSH(x + i)
```

Use the *aggregate* method to prove that for any integers l and n, there is a sequence of l MULTIPUSH and MULTIPULL operations that require $\Omega(nl)$ time, where n is the maximum number of items in the queue at any time. Such a sequence implies that the amortized cost of each operation is $\Omega(n)$.

(c) Describe a data structure that supports arbitrary intermixed sequences MULTIPUSH and MULTIPULL operations in $O(1)$ amortized cost per operation. In this question, you are asked to use the *potential function* method to prove that the amotrized cost of MULTIPUSH and MULTIPUSH is $O(1)$ for your data structure.

Problem-14: Largest Weight Cycle [25 points]

You are given a weighted tree on n vertices with possibly negative weights. You are allowed to add a single edge of weight 0 to this tree. Your goal is to maximize the weight of the cycle thus formed. Find an efficient way to do this.

i. Give an algorithm with running time $\mathcal{O}(n^3)$.

ii. Give an algorithm with running time $\mathcal{O}(n^2)$.

iii. Give an algorithm with running time $\mathcal{O}(n)$.

Problem-15: k-Center Problem [27 points]

Consider a simple undirected complete weighted graph $G = (V, E, d)$ where $d(u, v)$ represents the distance between u and v. We assume that $d(u, v)$ is symmetric and satisfies the triangle inequality. More formally for any $u, v, w \in V$ we have that $d(u, v) = d(v, u)$ and $d(u, v) \leq d(u, w) + d(w, v)$. We define the distance of a point $v \in V$ from a set of points $P \subseteq V$ as follows $d(v, P) = \min_{u \in P} d(u, v)$.

Given an integer number $k \geq 1$ our goal is to find a set $P \subseteq V$ of k points such that $\max_{v \in V} d(v, P)$ is minimized. This problem is known as the *k-center problem*. For simplicity we will assume that no two pairs of points have equal distances, i.e. $d(u, v) \neq d(w, z)$ whenever $\{u, v\} \neq \{w, z\}$. The goal of this problem is to find a 2-approximation algorithm for the k-center problem.

(a) Prove that for $k = 1$, the solution $P = \{u\}$ for any $u \in V$ is a 2-approximation to the 1-center problem.

Given a solution $P = \{p_1, \ldots, p_k\}$, define its i-th cluster C_i to be the set of vertices that are closer to p_i than to any other vertex in P. More formally:

$$C_i = \{u \in V \mid d(u, p_i) < d(u, p_j) \ \ \forall j \neq i\}$$

Let OPT be the value of the optimal solution to the k-center problem and let P^* be a set of k points that achieves this optimum, namely, $OPT = \max_{v \in V} d(v, P^*)$.

(b) Consider any solution $P = \{p_1, \ldots, p_k\}$. Argue that, if p_i and p_j are contained in the same cluster of P^*, then $d(p_i, p_j) \leq 2 \cdot OPT$.

(c) Consider any solution $P = \{p_1, \ldots, p_k\}$ and assume that, for every cluster of P^*, there is a $p_i \in P$ contained in this cluster. Prove that P is a 2-approximation to the k-center problem.

(d) Using ideas from the previous parts, propose a greedy 2-approximation algorithm for the k-center problem. Analyze the running time of your algorithm and prove its correctness.

Problem-16: Dynamic Programming. [15 points]

Prof. Child is cooking from her garden, which is arranged in grid with n rows and m columns. Each cell (i, j), with $1 \leq i \leq n$ and $1 \leq j \leq m$, has an ingredient growing in it, with ***tastiness*** given by a positive value $T_{i,j}$.

Prof. Child doesn't like cooking "by the book". To prepare dinner, she will stand at a cell (i, j) and pick one ingredient from each quadrant relative to that cell. The tastiness of her dish is the product of the tastiness of the four ingredients she chooses. Help Prof. Child find an $O(nm)$ dynamic programming algorithm to maximize the tastiness of her dish.

Here the four ***quadrants*** relative to a cell (i, j) are defined as follows:

$$\begin{aligned} \text{top-left} &= \{\text{all cells } (a, b) \mid a < i, b < j\}, \\ \text{bottom-left} &= \{\text{all cells } (a, b) \mid a > i, b < j\}, \\ \text{top-right} &= \{\text{all cells } (a, b) \mid a < i, b > j\}, \\ \text{bottom-right} &= \{\text{all cells } (a, b) \mid a > i, b > j\}. \end{aligned}$$

Because Prof. Child needs all four quadrants to be non-empty, she can only stand on cells (i, j) where $1 < i < n$ and $1 < j < m$.

(a) Define $TL_{i,j}$ to be maximum tastiness value in the top-left quadrant of cell (i, j):

$$TL_{i,j} = \max\{T_{a,b} \mid 1 \leq a \leq i, 1 \leq b \leq j\}$$

Find a dynamic programming algorithm to compute $TL_{i,j}$, for all $1 < i < n$ and $1 < j < m$, in $O(nm)$ time.

(b) Use the idea in part (a) to obtain an $O(nm)$ algorithm to find the tastiest dish.

Problem-17: Sushi Delivery [20 points]

After having all of their cattle repurposed as space-objects in a galactic maze, John and Bob have moved on to running a sushi resturant in a big city. The streets of this city are well planned: they are either North-South or East-West, and are evenly spaced in each direction. In other words, it can be viewed as a grid. The resturant has two delivery drivers, A and B, available to run deliveries.

One evening, n customers $1 \dots n$ called requesting sushi deliveries. These requests are given in order in which the customers called. A and B wish to split these requests so that each delivery is made by exactly one of them, and they return to the restaurant at the end. In order to be fair, if customers $i < j$ are assigned to the same driver then the delivery must be made to the customer i before customer j, even if that makes the trip longer. Note that if i and j are assigned to different drivers, deliveries can be made to them in any order.

A and B would like to keep the total distance that they travel small. However, each of them also know some of the customers better than others. Customer i will give a tip of value a_i if A delivers, or will give a tip of value b_i if B delivers. As a result, A and B would like to maximize their total happiness, which is the amount of tip that they receive minus the total distance that they travel.

Each location is given as the intersection of two streets, and is parameterized by two values, (x_i, y_i), the id of the North-South street and the East-West street that the intersection is on respectively. The resturant is located at $p_0 = (x_0, y_0)$, and customer i is located at $p_i = (x_i, y_i)$. Since A and B drive along the streets, the distance between points i and j is:

$$d(i, j) = |x_i - x_j| + |y_i - y_j|$$

(a) Describe and analyze an algorithm that takes n, the locations of customers,

$$p_0 = (x_0, y_0), p_1 = (x_1, y_1), \dots, p_n = (x_n, y_n)$$

and the tip values $a_1 \dots a_n$ and $b_1 \dots b_n$ as input and outputs the maximum happiness for A and B. Prove that your algorithm is correct. Your algorithm should run in $O(n^{10})$ time or faster.

(b) Give an algorithm for finding the maximum happiness that uses $O(n)$ space and runs in $O(n^2)$ time or faster .

(c) Give an algorithm for finding the maximum happiness that runs in $O(n \log^3 n)$ time or faster.

Hint: Parts (a) and (b) can be solved for any distance function $d(i, j)$.