Name: ______________________________

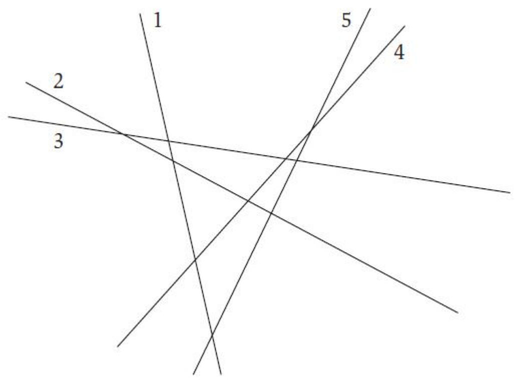
1. Time allowed: It is estimated that this portion of the exam will take no more than four hours to complete.
2. Submit your take home solutions as a portable document format (PDF) to Canvas before the in class portion of the exam.
3. In addition to this take home portion there will be a in class portion of the exam on Canvas and problems in a printed document.
4. Complete all sections, all questions. Do not leave any question unanswered.

Question:	I	II	III	Total
Points:	40	20	40	100
Score:				

I. *Hidden surface removal* is a problem in computer graphics that scarcely needs an introduction — when Woody is standing in front of Buzz you should be able to see Woody but not Buzz; when Buzz is standing in front of Woody, . . . well, you get the idea.

The magic of hidden surface removal is that you can often compute things faster than your intuition suggests. Here's a clean geometric example to illustrate a basic speed-up that can be achieved. You are given n non-vertical lines in the plane, labeled $L_1, \ldots, L_n$, with the i^{th} line specified by the equation $y = a_i x + b_i$. We will make the assumption that no three of the lines all meet at a single point. We say line L_i is *uppermost* at a given x-coordinate x_0 if its y-coordinate at x_0 is greater than the y-coordinates of all the other lines at x_0: $a_i x_0 + b_i > a_j x_0 + b_j$ for all $j \neq i$. We say line L_i is *visible* if there is some x-coordinate at which it is uppermost — intuitively, some portion of it can be seen if you look down from "$y = \infty$."

An example with five lines (labeled $1 - 5$ is shown here with all lines except for 2 are visible.



(1) (2 points) Which algorithmic paradigm will you use to solve this problem?
- A. Brute force
- B. Greedy
- C. Divide and Conquer
- D. Dynamic Programming
- E. other: ____________________

(2) (3 points) Why did you choose the algorithmic paradigm selected above to solve this problem?

(3) (15 points) Give an algorithm that takes n lines as input, and in $O(n \log n)$ time returns all of the lines that are visible. Provide a clear description of the algorithm (an English description or pseudo-code is fine)

(4) (10 points) Perform asymptotic analysis of your algorithms running time. Also, consider the runtime performance of a best case, worst case, and average case input model scenario.

(5) (10 points) Provide a proof that your algorithm works correctly.

II. In a bipartite graph $G = (X \cup Y, E)$ a matching M is a subset of the edges from E chosen in such a way that such that $\forall e \in M$ one end is in X and the other in Y. A *maximum matching* is a matching with the maximum number of edges – if any edge can be added then it is no longer a matching. A given bipartite graph can have only one maximum matching. Consider the following algorithm for finding a matching in a bipartite graph:

> As long as there is an edge whose endpoints are unmatched, add it to the current matching M.

(1) (2 points) Which algorithmic paradigm best describes this algorithm?

- A. Brute force
- B. Greedy
- C. Divide and Conquer
- D. Dynamic Programming
- E. other: ____________________

(2) (3 points) Why did you choose the algorithmic paradigm selected above?

(3) (5 points) Give an example of a bipartite graph G for which this algorithm does not return the maximum matching

(4) (5 points) Let M and M' be matchings in a bipartite graph G. Suppose that $|M'| > 2|M|$. Show that there is an edge $e' \in M'$ such that $M \cup \{e'\}$ is a matching in G.

(5) (5 points) Using the previous claim (and your supporting proof) to further prove that the algorithm is optimal or that the algorithm is ρ-optimal approximate (in this case be sure to derive the value of ρ as part of your proof).

III. (40 points) You and a friend have been trekking through various far-off parts of the world, and have accumulated a big pile of souvenirs. At the time you weren't really thinking about which of these you were planning to keep, and which your friend was going to keep, but now the time has come to divide everything up.

Here's a way you could go about doing this. Suppose there are n objects, labeled $1, 2, \ldots, n$, and object i has an agreed-upon *value* x_i. (We could think of this, for example, as a monetary re-sale value; the case in which you and your friend don't agree on the value is something we won't pursue here.) One reasonable way to divide things would be to look for a *partition* of the objects into two sets, so that the total value of the objects in each set is the same.

This suggests solving the following *Number Partitioning* problem. You are given positive integers $x_1, \ldots, x_n$; you want to decide whether the numbers can be partitioned into two sets S_1 and S_2 with the same sum:

$$\sum_{x_i \in S_1} x_i = \sum_{x_j \in S_2} x_j.$$

Show that *Number Partitioning* is NP-complete using the *Subset Sum* problem.