This is an oral presentation assignment. Again, there are three regular problems (#1-#3) and one programming problem (#4). You should work in groups of three. The sign-up sheet will be online soon (details on Piazza) and your group should sign up for a 1-hour slot. Each person in the group must be able to present every regular problem. The TA/Professor will select who presents which regular problem. You are not required to hand anything in at your presentation, but you may if you choose.

The programming problem will be submitted to autolab, due May 4th, 11:59pm. You will not have to present anything orally for this. You can discuss the problem with your group-mates, but must write the program by yourself. Please do not copy.

(25 pts) 1. **Counting Chord Crossings.** You’re given a unit circle $C$, and $n$ chords on the circle. A chord is a line segment $(a, b)$, where $a \neq b$ and both $a$ and $b$ are on the circle. Give an $O(n \log n)$ algorithm to count the number of pairs of chords that intersect. For simplicity you can assume that no two chords have any endpoints in common.

(25 pts) 2. **Algorithms from another Planet.** Your spaceship has landed on LineWorld, which as the name suggests, looks like the real line $\mathbb{R}$. You start at the origin $x_0 = 0$. At each day $t \geq 1$, the adversary puts a linear force field which is centered at some location $c_t \in \mathbb{R}$ and has slope $s_t \geq 0$. This force field means that if you are standing at location $x \in \mathbb{R}$, you spend $f_t(x) := s_t \cdot |x - c_t|$ of fuel for this day. Moreover, you spend 1 unit of fuel for each unit of distance you move along the line. So on day $t$, you observe the force field for this timestep (i.e., you get to know $s_t, c_t$). Then you may move from the previous location $x_{t-1}$ to some location $x_t$ (it takes you zero time to travel), and then spend day $t$ at this location $x_t$, hence burning $|x_t - x_{t-1}| + f_t(x_t)$ amount of fuel on this day. The total fuel usage until time $T$ would hence be

$$\sum_{t=1}^{T} (|x_t - x_{t-1}| + f_t(x_t)).$$

You cannot see the future, so you make decisions on each day $t$ without knowing the force fields on days $t + 1$ or later. You want to give an online algorithm that is 4-competitive. I.e., for each $T$, the fuel usage of the algorithm until day $T$ is at most 4 times the fuel usage of any other algorithm $B$ until that same day.

Here’s an algorithm: on day $t$, move from $x_{t-1}$ towards the center $c_t$, until either (a) you reach $c_t$, or (b) the distance you have traveled (i.e., $|x_{t-1} - x|$) equals $f_t(x)$ at the point $x$ you reach. This is your new point $x_t$. Show this algorithm is 4-competitive.

*Hint: What would be a good potential to use?*
(25 pts) 3. **Points, Lines, and Triangles.** You’re given a set $S$ of $n$ points in the plane with integer coordinates.

(a) Give an $O(n^2)$ algorithm that finds the largest subset of $S$ all of whose points are colinear.

(b) Give an $O(n^2)$ algorithm to find the triangle among the points of $S$ with the largest area.

(25 pts) 4. **Square.** Write a program that takes as input $n$ (not necessarily distinct) points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, along with a side length $s$ of a square. All these values are integers. Your program should output the location of the lower-left corner of an axis-aligned square of side length $s$ which contains the largest number of the given points. (A square includes all the points on its boundary.) It should also output the number of points contained in that square. The time limit is 10 seconds (60 seconds for Python).

**Input:**

The first line contains two space-separated integers $s$ and $n$. $0 \leq s \leq 10^6$, $1 \leq n \leq 5 \times 10^5$. The following $n$ lines each contain a pair of integers, which are the $x$ and $y$ coordinates of one of the points. $0 \leq x, y \leq 10^6$.

**Output:** The output consists of two lines. The first tells the location of the lower-left corner of a square which contains the most points. This is a pair of integers ($x$ first then $y$). The next line tells how many of the input points that square contains. In case there are several equally good solutions, output any of them. See the examples below for more details.

**Input1:**

```
2 7
0 0
2 0
4 0
1 1
0 2
2 2
4 2
```

**Output1:**

```
square lower-left corner = (0, 0)
count = 5
```

**Input2:**

```
3 5
5 5
5 5
```
Extending a Classic On-Line Algorithm. A classic homework problem in on-line algorithms is as follows: There’s a graph of two vertices (1 and 2) connected by an edge. A McGuffin is located at one of the vertices. There is a sequence of requests from vertices 1 and 2 to access the McGuffin. For one request, if it is from vertex $i$ and the McGuffin is at $i$ the cost is 0. If the request is from vertex $i$ and the McGuffin is at vertex $3 - i$, then the cost is 1. After any request the McGuffin can be moved to the other vertex at an integral cost of $d \geq 1$. An on-line algorithm for this problem must make this decision without knowing the future requests.

The problem is to give an optimally competitive algorithm to decide when to move the McGuffin to the other vertex. It turns out that the optimal competitive factor is 3. Here’s the algorithm that achieves this. Maintain a counter $c$, which is the number of costly requests that occurred since the McGuffin was last moved. When this counter reaches $2d$, move the McGuffin, and reset the counter.

For example, suppose the sequence consisted of the following repeated pattern: 2$d$ 1s, followed by 2$d$ 0s. The cost of each cycle for the above algorithm is 6$d$, whereas the cost of the algorithm that never moves is 2$d$.

So your first task is to prove that the above algorithm is 3-competitive.

A natural extension to this problem is what happens on other graphs with more than two vertices. So specifically what happens to a complete graph on $n$ vertices? The rules are the same. Accessing the McGuffin remotely costs 1. Moving the McGuffin costs $d$. Design a 3-competitive algorithm for this more general problem.