

CS106X Midterm Examination

This is an open-note, open-reader exam. You can refer to any course handouts, textbooks, handwritten lecture notes, and printouts of any code relevant to any CS106X assignment. You may not use any laptops, cell phones, or handheld devices of any sort.

If you're taking the exam early and no one's around, you can telephone Jerry at 415-205-2242 with questions.

Good luck!

Section Leader: _____

Last Name: _____

First Name: _____

I accept the letter and spirit of the honor code. I've neither given nor received aid on this exam. I pledge to write more neatly than I ever have in my entire life.

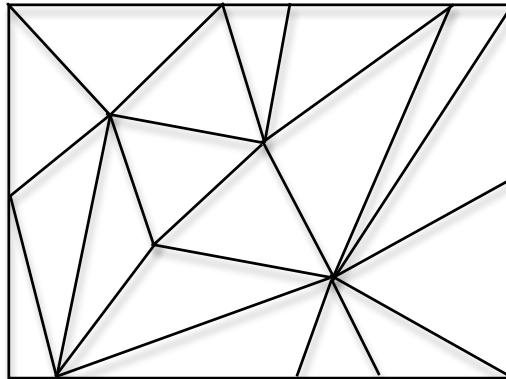
(signed) _____

		Score	Grader
1. Gerrymandering	[10]	_____	_____
2. Campaign Donations and Kickbacks	[10]	_____	_____
3. Letter Rectangles	[10]	_____	_____
4. Cubic Decompositions	[10]	_____	_____
Total	[40]	_____	_____

Problem 1: Gerrymandering [10 points]

Gerrymandering is, in the world of politics, the process by which voting district boundaries are redefined with the intent of shifting the profile of the average voter in a particular direction—perhaps to make a region more liberal, or more conservative, or perhaps to just influence the outcome of some non-partisan voter initiative.

For simplicity, we'll assume that all voter districts are triangular.



A political party is granted the option of removing all internal district lines incident to a single point, opening up a super (not necessarily triangular) district. The political party wants to maximize the geographical reach of the super district, so they're interested in identifying the point that, when all internal district lines incident to it are removed, opens up a polygon with the **greatest area**.

```

struct coord {
    double longitude;
    double latitude;
};

struct district {
    Set<coord> vertices; // all of size 3, pure districts are always triangles
};

double getDistrictArea(district& d) {
    // assume working implementation that computes area is already
    // provided for you, and it just works. Note, however, that this
    // only works for (triangular) districts, and not the super district
}

```

Given a **Vector** of all triangular voting districts, implement the **findBestCoordinate** function that identifies the one coordinate that, when all internal district lines incident to it are removed, opens up the polygon with the largest area. If there are two or more such points that happen to open up polygons of equal maximum area, then you can return any one of them. Use the next page for your implementation.

[Problem 1 continued]

```
coord findBestCoordinate(Vector<district>& districts) {
```

Problem 2: Campaign Donations and Kickbacks [10 points]

When a politician's supporters donate money to help support his or her campaign, it's important to ensure that the supporters aren't illegally compensated for their donations in any way.

Let's assume that a preliminary investigation of, say, a county's tax assessor—an elected official who, among his or her many responsibilities, manages property tax assessments and the property owners' appeals to have those taxes reduced—suggests that a suspiciously large fraction of his financial donors have received tax reductions.

A more complete investigation would entail a comparison of all those who donated money to those whose property tax liabilities were reduced. If the percentage of financial donors receiving a tax reduction is statistically greater than the percentage of the general applicant pool receiving a tax reduction, that might be grounds for an even more thorough investigation.

For the purposes of this problem, assume we have access to two key collections of information:

- a **Set<string>** called **donors**, which stores the names of all those who donated money to help support the election of a county's tax assessor, and
- a **Map<bool>** called **appeals**, which stores the names of all county residents who appealed to have their property taxes reduced, where the keys are the names of those who filed appeals, and the values indicated whether a particular person's appeal was approved (**true**) or denied (**false**). (Note that some donors may not have appealed their property tax assessments and would not be present in this map. Assume that property tax reductions only come about because of an appeal process—that is, no one's property taxes are reduced unless an appeal is filed.)

Implement a function called **analyzeRecords**, which accepts references to **donors**, **appeals**, and two **doubles** called **donorFraction** and **generalFraction**, and populates **donorFraction** and **generalFraction** with the fraction of those receiving tax reductions who donated money to the assessor's campaign and the general fraction of those receiving tax reductions, respectively. Use the next page for your implementation.

[Problem 2 continued]

```
void analyzeRecords(Set<string>& donors, Map<bool>& appeals,  
                   double& donorFraction, double& generalFraction) {
```

Problem 3: Letter Rectangles and Words [10 points]

You are given a large collection of short, fat rectangles, where each half of each rectangle contains a single letter, as with:



Given the option to rearrange, ignore, and rotate pieces, you're charged with the task of identifying all of the even-length words that can be formed by chaining together some subset of the pieces (where some may have been rotated). For the above set of pieces, the list of printed words should surely include "**plum**", since the third-to-last rectangle can be placed after the second-to-last rectangle (rotated so that the '**p**' precede the '**l**') to form "**plum**". Given the above set of rectangles, you should also identify fun words like "**allele**", "**lark**", "**muscle**", "**scales**", and "**umbrella**", in addition to quite a few others. Note that each rectangle can be used at most one time per word, so that words like "**sees**" and "**museum**" can't be formed.

Using this and the next page, implement a recursive function **gatherWords** that accepts references to a **Vector<string>** called **rects** (where each **string** is two characters), a **Lexicon** called **english**, and an initially empty **Lexicon** called **words**, and populates **words** with the collection of those words, and only those words, that can be formed using the rectangles in **rects**. You should implement this using a wrapper function.

```
void gatherWords(Vector<string>& rects,
                 Lexicon& english, Lexicon& words) {
```

[Problem 3 continued]

Problem 4: Cubes as Sums of Smaller, Distinct Cubes [10 points]

All perfect cubes—save for a few relatively small ones—can be expressed as a sum of three or more **distinct, smaller** cubes. Just trust the math as you check out these examples:

$$\begin{aligned}6^3 &= 5^3 + 4^3 + 3^3 \\13^3 &= 12^3 + 7^3 + 5^3 + 1^3 \\69^3 &= 59^3 + 25^3 + 17^3 + 10^3 + 4^3 \\1021^3 &= 1014^3 + 231^3 + 177^3 + 157^3\end{aligned}$$

Most of them can be expressed as a sum of three, four, or five perfect cubes, but some (really big ones) require many, many more.

Write a function call **cubicDecompositionExists**, which takes in an **int** called **n** and a reference to an initially empty **Set<int>** called **cubicRoots**, and returns **true** if and only if **n³** can be decomposed into a sum of **smaller, distinct** cubes. If **true** is returned, then the **Set<int>** referenced by **cubicRoots** should be populated with a collection of distinct integers that, when cubed and added together, produce **n³**.

- **cubicDecompositionExists(5, cubicRoots)** would return **false**, because 5^3 is greater than $4^3 + 3^3 + 2^3 + 1^3$
- **cubicDecompositionExists(6, cubicRoots)** would return **true**, and on return **cubicRoots** would contain 3, 4, and 5. (As it turns out, this is the only such decomposition.)
- **cubicDecompositionExists(11, cubicRoots)** would return **false**, because that's just the way it works out.
- **cubicDecompositionExists(1021, cubicRoots)** would return **true**, and on return **cubicRoots** might contain 1014, 231, 177, and 157. (I say **might**, because there might be some other collection of numbers that also cube and sum together to produce 1021^3 .)

To be clear, when **true** is returned, the collection of **ints** referenced by **cubicRoots** need not be minimal in size, in that a set of size four is fine, even if a set of size three exists, provided the sum of the cubes of the **ints** evaluates to the overall cube of interest.

Using this and the next page, present your implementation of **cubicDecompositionExists**.

```
bool cubicDecompositionExists(int n, Set<int>& cubicRoots) {
```