CS 106B Section 5 (Week 6)

Recommended problems: 1, 3, 4

1. **Big-O**
   The following snippets of code try to find the number of **'a'** characters in a Grid of characters. What is the Big-O complexity of the following code, where **board** has **N** rows and columns?

```
int numA(Grid<char> board) {
    Vector<int> total;
    for (int row = 0; row < board.numRows(); row++) {
        for (int col = 0; col < board.numCols(); col++) {
            for (int rowT = 0; rowT <= row; rowT++) {
                for (char ch = 'a'; ch <= 'z'; ch++) {
                    if (ch == 'a' && rowT == row && board[row][col] == ch) {
                        total.add(38);
                    }
                }
            }
        }
    }
    return total.size();
}
```

```
int numARecursive(Grid<char> &board) {
    int total = 0;
    for (int i = 0; i < board.numRows(); i++) {
        total += (board[i][0] == 'a');
    }
    if (board.numCols() == 1) {
        return total;
    }
    Grid<char> newBoard(board.numRows(), board.numCols() - 1);
    for (int i = 0; i < board.numRows(); i++) {
        for (int j = 0; j < board.numCols() - 1; j++) {
            newBoard[i][j] = board[i][j + 1];
        }
    }
    return total + numARecursive(newBoard);
}
```

2. **Sorting**
   Bubble sort is a sorting algorithm which loops through a list of elements, compares each pair of adjacent items, and swaps them if they are in the wrong order. This is repeated until swaps are no longer needed. Cocktail sort works similarly to bubble sort but instead sorts in both directions on each pass through the list, alternating between moving up the list and down the list.

   Write a function **cocktailSort** which takes a vector of integers and sorts them in place using the Cocktail Sort algorithm.

```
void cocktailSort(Vector<int> &vec) { . . .
```

CS 106B Section 5 (Week 6)

3. Classes

Consider the **Fraction** class that was introduced in lecture:

```
class Fraction {
public:
    Fraction();
    Fraction(int num,int denom);
    void add(Fraction f);
    void mult(Fraction f);
    float decimal();
    int getNum();
    int getDenom();
    friend ostream& operator<<
        (ostream& out, Fraction &frac);
private:
    int num;   // the numerator
    int denom; // the denominator
    void reduce(); // reduce the fraction
    int gcd(int u, int v);
};
```

Write a **reciprocal** public function to be added to the **Fraction** class which coverts the function to its reciprocal (note that by definition the reciprocal of a number x is a number y such that $xy = 1$ holds).

```
void Fraction::reciprocal() { . . .
```

Write a **divide** public function to be added to the **Fraction** class which divides the **Fraction** by a given **Fraction** by leveraging the existing **mult** function.

```
void Fraction::divide(Fraction f) { . . .
```

---

4. Pointers

What does the following code snippet produce?

```
void NBC(int *liz, int jack, int& tracy) {
    jack += *liz;
    tracy *= jack;
    (*liz) = 1;
    jack = *liz;
}

int main() {
    int jim = -6;
    int pam = 21;
    int dwight = 2;
    NBC(&jim, dwight, pam);
    cout << jim << " " << pam << " " << dwight << " " << endl;
    return 0;
}
```