NAME: ________________________________

Learning Objectives

- Learn how to use vectors and arrays.
- Learn how to write algorithms that process data sequentially in a single loop.
- Learn binary search, a divide and conquer algorithm.

Instructions

Work out the answers to these problems manually without the use of a computer. This will help you develop the ability to read and analyze code. It will also provide you with practice solving the type of problems that will appear on Quiz 3 and the final exam. During these exams, you will not have access to a computer to solve these problems, so you need to develop the ability to read code, analyze it and think critically about it. After coming up with answers manually, you can optionally write and run test code to check whether you have solved the problems correctly.

Submit a single document with your answers either by email or through your remote Git repository.

```
bool areIdentical(const vector<int> & a, const vector<int> & b);
```

Figure 1

1) Write a predicate function that checks whether two vectors are identical (contain exactly the same elements in the same order). A declaration of the function is shown in Figure 1. The function returns true if the two vectors are identical; otherwise it returns false. (5 points)

```
bool isUnlucky(vector<int> & v);
```

Figure 2

2) Implement a function that determines if the unlucky number 13 appears in a vector. The function returns true if 13 appears in the vector at least once; otherwise it returns false. A declaration of the function is shown in Figure 2. (5 points)

3) Write test code that tests every statement in the isUnlucky function you implemented in the previous problem. Express your tests using assertions. (5 points)

```
bool isStrictlyIncreasing(const vector<int> & v);
```

Figure 3

4) Write a predicate function called isStrictlyIncreasing that checks whether a vector of integers

contains values that are in strictly increasing order. A declaration of the function is shown in Figure 3. The function returns true if the elements are in strictly increasing order; otherwise it returns false. For example, it will return true for v = (-2, 4, 5, 6, 8) and it will return false for (3, 4, 6, 6, 9). (5 points)

```
vector<int> flatten(int a[100][200]);
```

Figure 4

5) Write a function named *flatten* that takes a 2-dimensional array of integers with 100 rows and 200 columns and returns a vector that contains all of the array's elements. Copy the values a row at a time. In other words, first copy row 0 into the vector, then row 1, then row 2, and so on. The declaration of *flatten* is given in Figure 4. (5 points)

```
double findMax(double a[ROWS][COLS]);
```

Figure 5

6) Implement a function called *findMax* that determines the maximum value in a 2-dimensional array. A declaration of the function is shown in Figure 5. The variables ROWS and COLS are constants defined elsewhere in the program; you don't need to define them, just use them.
(5 points)

```
int search(const vector<int> & v, int k);
```

Figure 6

7) Implement a function that searches for a given value in a vector of integers. If the value is found, the function returns the index of the value in the vector; otherwise it returns -1. Do not assume the values are in order; do not use binary search. For example, for v = (-2, 4, 18, 6, -10) and k=1, the function returns -1, and for k = 4 it returns 1. A declaration of the function is shown in Figure 6. (5 points)

```
int binarySearch(const vector<int> & v, int k);
```

Figure 7

8) Implement a function that uses binary search to search for a given value in a vector of integers whose elements are in strictly increasing order. If the value is found, the function returns the index of the value in the vector; otherwise, it returns -1. You can assume that the values passed into the function are in strictly increasing order. For example, for v = (-2, 4, 5, 6, 8) and k=1, the function returns -1, and for k = 4 it returns 1. A declaration of the function is shown in Figure 7. (5 points)

```
vector<int> flatten(int a[100][200]);
```

Figure 8

9) Write a function named *flatten* that takes a 2-dimensional array of integers with 100 rows and 200 columns and returns a vector that contains all of the array's elements. Copy the values a row at a time. In other words, first copy row 0 into the vector, then row 1, then row 2, and so on. The declaration of *flatten* is given in Figure 8. (5 points)

```
double findMax(double a[ROWS][COLS]);
```

Figure 9

10) Implement a function called *findMax* that determines the maximum value in a 2-dimensional array. A declaration of the function is shown in Figure 9. The variables ROWS and COLS are constants defined elsewhere in the program; you don't need to define them, just use them.
(5 points)