CSE 1321L: Programming and Problem Solving I Lab
Assignment 6 – 100 points
Methods, Searching and Sorting Information

What students will learn:

1) Searching for information in arrays
2) Sorting arrays to make the search process more efficient
3) Designing and calling methods

Overview: Arrays are great, but just archiving data haphazardly in an array is not enough. For most large applications today, you need to be able to quickly find, update, and use that information. For instance, when you visit our class section in D2L, the server has to find your name, grades, and assignment submissions from a database of all students at Kennesaw State. This assignment will help you practice the foundational skills you'll need to one day develop these large scale systems.

Assignment 6A: *Bogosort.* In lecture, we told you that Bubble Sort, Selection Sort, and Insertion Sort have terrible worst case time complexities of O(n). However, there is an algorithm with an even worse time complexity – Bogosort. Bogosort works by randomly shuffling the elements in the array and then checking to see if they're in non-decreasing order. If they aren't, we repeat the process again. This results in hypothetical worst case time complexity of O(∞); in other words, it might run FOREVER!

To begin, create a 1D integer array of size 19. Fill each index with a random value ranging from 1 to 453 inclusive. You will then design and implement the Bogosort algorithm using the following methods:

- Create a method called **check_if_sorted ()**. It should take in a 1D integer array and return a boolean value. It should return TRUE if the array is sorted in non-descending order, and FALSE otherwise.
  Hint: If you compare elements in the array and a pair is in the wrong order, that would mean the array is not in non-descending order.

- Create a method called **shuffleArray ()**. It should take in a 1D integer array and return a 1D integer array. Shuffle the array so that the values are in random different indexes, and return altered array.
  Hint: There are many approaches to solve this problem – making a second temporary array in the shuffleArray () method might be part of the answer.

- Create a method called **PrintArray ()**. It should take in a 1D integer array and return nothing. Simply print the current values of the array when it's called.

Sample Output:

```
[Bogosort]

Printing array...
2, 9, 232, 1, 89, 74, 34, 122, 344, 19, 1, 1, 2, 78, 43, 12, 19, 3, 92
```

```
Not sorted yet!

Shuffling array...

Printing array...
19, 2, 3, 9, 34, 232, 1, 19, 344, 43, 89, 74, 122, 92, 1, 2, 1, 78, 12,
Not sorted yet!

Shuffling array...
//And so on… (This line is not part of the code)

Printing array...
1,  1, 1, 2, 2, 3, 9, 12, 19, 19, 34, 43, 74, 78, 89, 92, 122, 232, 344,

Hooray, it's sorted! And it only took 687756 attempts!
//You will get a different number each time (This line is not part of the code)
```

Assignment 6B: *Overloaded Sorting.* In class, we have primarily used integer arrays as examples when demonstrating how to sort values. However, we can sort arrays made of other primitive datatypes as well.

In this assignment, you will create three arrays of size 8; one array will be an integer array, one will be a char array, and one will be a float array. You will then ask the user to state what kind of data they want to sort – integers, chars, or floats.
The user will then input 8 values. You will store them in the appropriate array based on what datatype they initially stated they would use.

You will create a function called **arraySort()** that takes in an integer array as a parameter, and two overloaded versions of the same function that take in a char array and float array as parameters respectively. You will use these functions to sort the appropriate array and display the sorted values to the user.
Note: You must make overloaded functions for this assignment – they must all be called **arraySort()**. You can not create unique, non-overloaded functions like sortArrayChars().

Sample Output #1:

```
[Overloaded Sort]
What data type do you want to enter? float
Value 1: 3.4
Value 2: -1.0
Value 3: 2.0
Value 4: 10.3
Value 5: 90.2
Value 6: 8.4
Value 7: 8.6
Value 8: -2.3
Calling arraySort()...
The sorted values are:
-2.3, -1.0, 2.0, 3.4, 8.4, 8.6, 10.3, 90.2,
```

Sample Output #1:

```
[Overloaded Sort]
What data type do you want to enter? char
```

```
Value 1: a
Value 2: c
Value 3: f
Value 4: b
Value 5: e
Value 6: z
Value 7: x
Value 8: y
Calling arraySort()...
The sorted values are:
a, b, c, e, f, x, y, z
```

Assignment 6C: *Minesweeper – Simplified.* For many years, computers sold with the Windows operating system would contain a game called Minesweeper. The player would be presented with a grid, where they would have to click an empty part of the map. If they clicked on a hidden mine, the game would be instantly over. However, if they clicked a safe spot, a hint about nearby mines would be displayed and the player would click another spot. The goal would be to flag all the hidden mines without hitting one. (As an aside, many people did not know these rules and just clicked around randomly until they hit a mine)

We will be developing a simplified version of this game. You will prompt the user for a grid size and then created a 2D array with equal width and height (**C++ students**: Check the Appendix and the lab videos for more information on how to do this). You will initialize the 2D char array with each element equallying a '?' symbol. You will then randomly generate one "mine" value per column in the 2D and store their row locations in a separate 1D array (the 1D array's index will represent the column of the 2D array).

The player will be prompted the user to enter an X and Y coordinate on the grid. If the space is "free", you will replace the value at that index with a '_' symbol. If the space has a hidden mine, you will place an 'X' symbol at that index instead. Afterwards, print the 2D array again.

If the player hit a mine, tell them they lost and that the game is over. Otherwise, conduct a **linear search** of the 2D array and count how many '_' symbols there are (*Hint:* Use a nested FOR loop). If the count equals the grid size, tell the player they won.

Sample Output #1:

```
[Minesweeper – DOS Edition]
What is the grid size? 6
??????
??????
??????
??????
??????
??????
Enter your X coordinate: 2
Enter your Y coordinate: 5

??????
??????
??????
??????
??????
??X???
```

```
Sorry, you hit a mine!
```

Sample Output #2:

```
[Minesweeper - DOS Edition]
What is the grid size? 3
???
???
???
Enter your X coordinate: 0
Enter your Y coordinate: 0

_??
???
???
Enter your X coordinate: 1
Enter your Y coordinate: 1

_??
?_?
???
Enter your X coordinate: 1
Enter your Y coordinate: 2

_??
?_?
?_?
You win!
```

Submission:

1. You will submit 3 separate files
2. File names and class names must be correct.
3. Upload all files (simultaneously) to the assignment submission folder in Gradescope.

Appendex: Dynamic Arrays for C++

There are several popular compilers, but there are a few key differences between what they will and won't allow you to do in C++. In some compilers, the following code is perfectly fine:

```
int i = 12;
int array[i];
```

Other compilers will throw an error because you are not using a constant to define the size of the array. This limitation obviously poses a challenge to completing certain assignments. If your compiler does not allow you to use variables to define arrays, use the following workarounds:

For 1D Arrays

```
int x = 12;
int * array = new int[x];
```

For 2D Arrays

```
int x = 12;
int y = 24;
int ** array = new int * [x];
for(int i = 0; i < x; i++){
   array[i] = new int[y];
}
```

Once you have done so, you can use the arrays as normal (e.g. array[0][0] = 34 will assign the array at index 0, 0 to the value of 34). In our data structures class we discuss more about **why** this works, but for now, it is a useful workaround if your compiler does not allow you to use variables to create arrays from user input.