Lab 1: Exception Handling

Information

In-class labs are meant to introduce you to a new topic and provide some practice with that new topic.

Topics: Algorithm Efficiency

Solo work: Labs should be worked on by each individual student, though asking others for help is permitted. Do not copy code from other sources, and do not give your code to other students. **Students who commit or aid in plagiarism will receive a 0% on the assignment and be reported.**

Building and running: If you are using Visual Studio, make sure to run with debugging. **Do not run without debugging!**

Using the debugger will help you find errors.
To prevent a program exit, use this before `return 0;`

```
cin.ignore();      cin.get();
```

Turn in: Once you're ready to turn in your code, prepare the files by doing the following: **(1)** Make a copy of your project folder and name it `LASTNAME-FIRSTNAME-LABNAME`. (Example: `HOPPER-GRACE-LAB-UNIT-TESTS`) **(2)** Make sure that all source files (.cpp, .hpp, and/or .h files) and the `Makefile` files are all present. **(3)** Remove all Visual Studio files - I only want the source files and Makefiles. **(4)** Zip your project folder as `LASTNAME-FIRSTNAME-LABNAME.zip`

Never turn in Visual Studio files!

Starter files: Download from GitHub.

Grading: Grading is based on completion, if the program functions as intended, and absense of errors. **Programs that don't build will receive 0%.** Besides build errors, runtime errors, logic errors, memory leaks, and ugly code will reduce your score.

Contents

1.1 About

In this short lab, you will implement a couple of searching algorithms and test out the time it takes to run them. A Binary Search, a Bubble Sort, and a recursive and iterative Fibonacci number finder are already implemented for you.

1.1.1 Setting up the project

Download the starter zip file, `LAB-ALGORITHM-EFFICIENCY.zip`, off GitHub. This contains:

- `Menu.hpp`
- `Searcher.hpp`
- `Timer.hpp`
- `Timer.cpp`
- `main.cpp`

1.2 Lab specifications

1.2.1 Testing the existing functions

There are several functions already implemented. When you first run the program, you can set a list size and run **Binary Search**, **Sort**, **Recursive Fibonacci**, and **Iterative Fibonacci**.

```
--------------------------------------------
 | Initialize |
 --------------

Enter a list size: 1000
Vector size: 1000



--------------------------------------------
 | Main Menu |
 -------------

 Which function do you want to time?
 1. Search 1
 2. Search 2
 3. Binary Search (for sorted lists)
 4. Sort
 5. Recursive Fibonacci
 6. Iterative Fibonacci

 >>
```

Fibonacci Next, run the Recursive Fibonacci method several times. Find some value that takes a noticable amount of time.

Log its times, and then run the Iterative version for the same value.

n-th term	**value**	**Recursive**	**Iterative**
40	102334155	853 milliseconds	0 milliseconds

Figure 1.1: Example table, yours will be different depending on the computer.

Try out several values for n and log each. Why is the recursive one so much slower than the iterative one? Look up the **Big O** rating for the Recursive Fibonacci algorithm and write it down.

Sorting First start by trying out different list sizes and running **Sort**. Find a list size where it takes a noticable amount of time for the Sort function to complete.

```
Enter a list size: 10000


[...]

Sorting...

Done, time elapsed: 609 milliseconds
```

In a text editor, fill out a table with a list size. Increase the list size by some linear amount each time and compare the time to sort for each one.

List size	**Time to sort**
10,000	776 milliseconds
50,000	16365 milliseconds
100,000	78639 milliseconds

Figure 1.2: Example table, yours will be different depending on the computer.

What kind of efficiency does this seem like?

Look up **Bubble Sort** online and find it's Big-O efficiency and write it down as well.

Searching The Binary Search algorithm is already implemented for you, but you must have a sorted list before running it. There are also two blank Search functions in the `Searcher.hpp` file that you will fill out:

```cpp
int Searcher::Search1( const vector<int>& arr, int findMe )
{
    // Implement a search
}

int Searcher::Search2( const vector<int>& arr, int findMe )
{
    // Implement a search
}
```

The first search can be a simple **Linear search** (from beginning to end, checking each value). Search2 doesn't have to be particularly efficient or elegant, but just experiment with some different ways you can think of to search an unsorted or sorted list.

Log the amount of time it takes to find a given value for several list sizes and for each of the search algorithms. Since the list is filled with random integers, you might get different results each time.

Try to find a list size so that each search takes a noticable amount of time.

```
 >> 1

Enter the INTEGER value to find:
999999
Searching with Search 1...

Done, time elapsed: 3 milliseconds
Result: 999999 found at index 270650
```

List size	**Find...**	**Your Search 1**	**Your Search 2**	**Binary Search**
1,000,000	999999	3 milliseconds	118 milliseconds	0 milliseconds

Figure 1.3: Example table, yours will be different depending on the computer.

Once you're finished, turn in the text document and your `Searcher.hpp` files.