CS515 Lab 8

The purpose of this lab is to experiment with sorting algorithms and to start working with standard string tries.

Download all files from `~cs515/public/8L`

Task 1: Take several mystery sorting algorithms an determine which ones they are by experimentation.

The object file **`mysterysorts.o`** containing five sorting functions, named `mysterySort1`, `mysterySort2`, and so on. Each function is designed to take a vector of numbers and order the elements into ascending order. What's different about them is that each uses a different sorting algorithm. More specifically, the algorithms used are:

Bubble sort
Insertion sort
Selection sort
Heap sort
Quicksort (where first element of the vector is used as the pivot for partitioning.)

Your job as the sorting detective is to figure out which algorithm is implemented by each mystery sort. Below are some techniques you may use.

First, you can interrupt the sort and observe how the values are being moved around the vector. Read the comments of the provided header file that tells you how you can interrupt the sorting operations. You may also want to construct a list of values that are easy to see how values are being moved.

Secondly, you can time the sorting algorithms and get an overall idea of their performance. A way to measure elapsed system time for programs is to use function `gettimeofday()`. If you record the starting and finishing times in the `timeval` structures `start` and `finish`, you can compute the time elapsed in microseconds as shown in the following code:

```
#include <time.h>

int main(){
      clock_t start, finish;
      double elapsed;
      start = clock();
     // … Perform some calculation
      Finish = clock();
      elapsed = double(finish – start) * 1000000 / CLOCKS_PER_SEC;
}
```

In your README file, identify what each sort is and provide an explanation of the evidence you used to make your decision. Demonstrate in writing that you understand the differences in these sorting algorithms and how those differences are reflected when observing the algorithms at work.

Note that the object file **`mysterysorts.o`** is compiled on Agate under Linux, so it may not be compatible with your home machine.

Task 2: Implement a simple Set ADT using a string trie.

The provided header file **`set.h`** declares the public interface of a Set. Your job is to write the implementation in a **`set.cpp`** file. You should assume the Set class supports keys consisting only of lower-case alphabetic characters. Note that the underlying trie implementation is not space efficient. For simplicity, you don't need to include the big five methods, or overload << operator. The provided test driver `settest0.cpp` gives an initial set of tests to test your program.

You are not allowed to use any STL containers in the program. You also should not modify the provided `set.h` header file.

Submission:

- Submit the files **`set.cpp`** and **`README`**.
- To submit the files in Mimir, follow the link for this lab in MyCourses. Here is a list of files you are expected to submit:

```
set.cpp README
```

As always, do not turn in executables or object code, and make sure your submission compiles successfully on Agate. Programs that fail to compile will receive a zero mark (even if it might work perfectly on your home computer.) To check this, use the make command with the provided makefile, and check that your submission passes tests on Mimir.

Important:

You must include the standard comment block in each of your source files, including your name, section, date and collaboration details. You must also finish the README file along with your programs.

You should also include detailed collaboration declaration information in your comments. If you worked in pairs in this lab, each of you must include the partner's name in your program comment. Both you and your partner must complete an individual submission in order to earn a grade. If you work by yourself, you must indicate in the program comments that you have worked on your own independently.

You can resubmit as needed—your last submission will be graded. Here is a tentative grading scheme:

	Correct sort	Good evidence
Sort 1	5	5
Sort 2	5	5
Sort 3	5	5
Sort 4	5	5
Sort 5	5	5
Trie test 1	25	
Trie test 2	25	

Note that only the Trie tests have automated grading in Mimir, for a best-possible score of 50 points for that portion. The sorting analysis will be graded by hand and added to your grade later)