Lab 13

Comp 11 - Summer Session — Search Performance

13.1 Description

In this lab we are going to implement bubble sort. Bubble sort is nice sorting algorithm for sorting small sets of data. In addition to sorting, we are going to time how long it takes and compare it with the sort function that C++ gives us. Now, a lot of smart C++ programmers have worked on sorting, so it should be much faster! Never fear though, we will look at more sorting algorithms to catch up to C++'s performance later on!

Our objectives are the following:

- Implement bubble sort
- Start by sorting just a small set of numbers (say 10) to make sure it works. Then try 10000, or even larger numbers.
- You may uncomment some of the std::cout statements to help you debug (but if you try to sort more than 100 items, it may be helpful to comment them back out).

13.2 Files

You may use the following code to help get you started. If you find it easier, you may (and are encouraged) to break this project into separate files, just remember to submit them all!

```
class SortInt{
    int* m_elements; // prefixed with m_, so we know it is a member
     variable.
    int size;

public:
    // Default constructor which sets the size
    // of how many items it holds in its array
    SortInt(int _size){
        m_elements = new int[_size];
        size = _size;
    }

    ~SortInt(){
        delete[] m_elements;
    }

    void set(int index, int value){
        m_elements[index] = value;
    }

    int at(int index){
        return m_elements[index];
    }


    // Implement our sorting algorithm
    // The sorting algorithm we will implement
    // is bubble_sort.
    void bubble_sort(){

    }

};
```

Listing 13.1: The interface

```cpp
// Remember to compile this lab with std=c++11
//
#include <algorithm> // For sort
#include <vector>
#include <iostream>
#include <cstring>
// Chrono library for measuring time in C++ 11
#include <chrono>

// Include for random integers
#include <stdlib.h>
#include <time.h>

#include "SortInt.h"

int main(){

    int items;
    std::cout << "How many items would you like to sort(Try 10000)
    ?\n";
    std::cin >> items;

    // Create our list and a c++ vector
    SortInt ourList(items);
    std::vector<int> cppVector;

    // Add identical items to each data structure at the same index
    srand(time(NULL));
    for(int i =0; i < items; ++i){
        int randomNum = rand() % items + 1;
        ourList.set(i,randomNum);
        cppVector.push_back(randomNum);
    }

    // Great reference here to what is going on with vector sort.
    // http://www.cplusplus.com/reference/algorithm/sort/
    auto cppStartTimer = std::chrono::high_resolution_clock::now();
    std::sort(cppVector.begin(),cppVector.end());
    auto cppStopTimer = std::chrono::duration_cast<std::chrono::
    milliseconds>(std::chrono::high_resolution_clock::now()-
    cppStartTimer).count();

    // Now we call our sort function
    auto ourStartTimer = std::chrono::high_resolution_clock::now();
    ourList.bubble_sort();
    auto ourStopTimer = std::chrono::duration_cast<std::chrono::
    milliseconds>(std::chrono::high_resolution_clock::now()-
    ourStartTimer).count();


    // Check to see if all of the elemements are sorted
    // You may uncomment the lines below if you would like to
    // see which items do not match. Start with a small list~
    int matches = 0;
    for(int i =0; i < items; ++i){
        if(ourList.at(i) == cppVector[i]){
//            std::cout << "[" << i << "]" << ourList.at(i) << " =
```

```
    " << cppVector[i] << "\n";
53              matches++;
54          }else{
55 //              std::cout << "[" << i << "]" <<'\t' << ourList.at(i)
     << " != " << cppVector[i] << "\n";
56          }
57      }
58
59      // Output some statistics
60      std::cout << "======== The results are in! ========\n";
61      std::cout << matches << " of " << items << " match in your sort
     .\n";
62      std::cout << "our sort takes: " << ourStopTimer << " in ms\n";
63      std::cout << "cpp sort takes: " << cppStopTimer << " in ms\n";
64
65
66      return 0;
67  }
```

Listing 13.2: main.cpp

13.3 Output

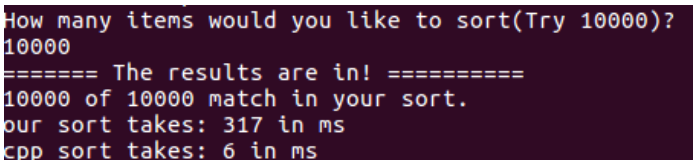

Figure 13.1: Sample output showing different runtimes on my machine

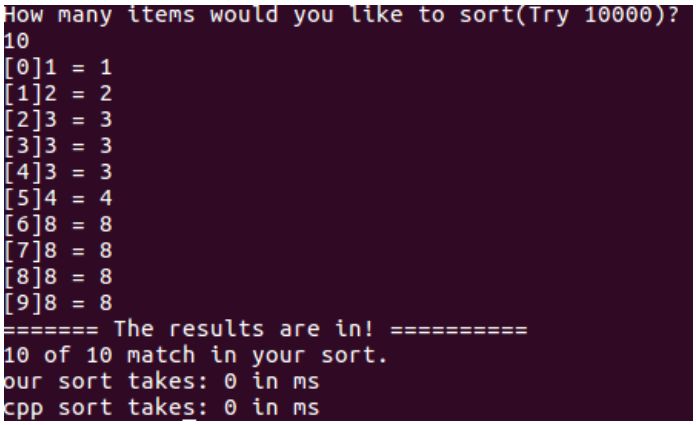

Figure 13.2: Sample output showing each sorted element matching up

13.4 Refresher

Revisit the lecture slides for the bubble sort algorithm.

13.5 Submission

```
provide comp11 lab13 your_cpp+files README
```

Listing 13.3: Submit Assignment

13.6 Going Further

Did you enjoy this lab? Want to try out some additional commands to go further?

- Try adding another sort function in our class.