

Dates

Assigned: April 3rd, 2016

Due: April 5th, 2016, by end-of-day

Instructions

This is a solo project to teach the basics of searching and sorting.

Make sure to download the **searchfile.txt**, **main.cpp**, and **SimpleTimer.hpp** files to work with.

Table of Contents

| | |
|--|---|
| Timer Class..... | 2 |
| Setting Up..... | 3 |
| Exercise 1: Linear search..... | 3 |
| Exercise 2: Shell Sort..... | 4 |
| Exercise 3: Quick Sort..... | 5 |
| Exercise 4: Write your own search algorithm..... | 6 |
| Screenshot..... | 7 |
| Algorithms..... | 8 |

Timer Class

For this exercise, we will time how long it takes for our searching and sorting functions to run. You will want to create a timer class and write the following code. Note that this code is not supported by C++98, so you will need to make sure your compiler is using C++11 or newer.

Timer class

```
#include <chrono>
using namespace std;

class SimpleTimer
{
public:
    void Start()
    {
        m_beginTime = chrono::system_clock::now();
    }

    double GetElapsed()
    {
        chrono::time_point<chrono::system_clock> endTime;
        endTime = chrono::system_clock::now();
        chrono::duration<double> delta = endTime - m_beginTime;
        return delta.count();
    }

private:
    chrono::time_point<chrono::system_clock> m_beginTime;
};
```

From your program, you can create a SimpleTimer object. Start the timer with `.Start()`, and get the amount of seconds elapsed with `.GetElapsed()`

Setting Up

Create a new project, adding the **main.cpp** and **SimpleTimer.hpp** file provided on D2L. Make sure that **searchfile.txt** is also in the project folder.

For the exercise, you will be filling out the functions above main – FindItem1, FindItem2, QuickSort, ShellSort, and InsertionSort.

In the main function, it calls each of these, and uses the timer to let you know how long it takes to do searching and sorting.

The searchfile.txt file contains 1,000,000 names, so it can take a while to sort.

Exercise 1: Linear search

First, you will write a simple search algorithm that will look at each element of the list in order to try to find our item.

1. Write a for-loop that starts the index at 0 and goes until `index < nameList.size()` . Increment by one each time.
 1. Look at the current element of the list (`nameList[i]`), and compare it to the `searchTerm`. If they are equal, return the current index.
2. After the for loop is over this means no matches have been found (otherwise, the function would have returned already.) At this point, return -1.

Test the program and look at how fast the program finds the value.

Exercise 2: Shell Sort

<https://en.wikipedia.org/wiki/Shellsort>

We will be writing three types of sorting algorithms. In `main()`, it makes a copy of the list of names before it sorts a list, so each sort algorithm is dealing with the same jumbled mess. These sorting algorithms will organize the lists alphabetically.

Within the `ShellSort` function, fill in the following algorithm:

1. Outer for-loop:
 1. `gap` (int), begins at `size / 2`.
 2. loop while `gap` is greater than 0.
 3. each loop, divide `gap` in half. (`gap /= 2`)
2. Inner for-loop:
 1. `i` (int), begins at `gap`.
 2. Loop while `i` is less than `size`
 3. Increment `i` by 1 each time. (`i++`)
3. Inside both loops:
 1. Create a string called **value**, and assign it the value of the array at position **i**.
 2. Create an integer called **j**, and assign it to **i**.
 3. Create a third for-loop:

```
for ( ; j >= gap && value < arr[j-gap]; j -= gap )
```

 1. inside, set the value of the array at position **j** to the value of the array at position **j-gap**.
 4. Outside the 3rd for loop, set the value of array at position **j** to the **value**.

Test the program and look at the time to run.

Exercise 3: Quick Sort

Quick Sort algorithm:

1. Create an integer called **i**, and assign it to the **left** parameter.
2. Create an integer called **j**, and assign it to the **right** parameter.
3. Create a string called **pivot**, and set it to the array at position $(\text{left} + \text{right}) / 2$
4. Create a string called **value**, which doesn't need an initial value.
5. Create a while loop: Loop while **i** is less than or equal to **j**.
 1. Inner while-loop: While the element at position **i** is less than **pivot**, increment **i** by one.
 2. Second inner while-loop: While the element at position **j** is greater than **pivot**, decrement **j** by one.
 3. If **i** is less than or equal to **j**:
 1. Set **value** to the array at position **i**.
 2. Set the element of array at position **i** to the value of element at position **j**.
 3. Set the element of array at position **j** to the **value** variable.
 4. Increment **i** by 1
 5. Decrement **j** by 1
6. Outside the while loop:
 1. If **left** is less than **j**, then call **QuickSort** again – pass in **arr**, **left**, and **j**.
 2. If **right** is greater than **I**, then call **QuickSort** again – pass in **arr**, **i**, and **right**.

Exercise 4: Write your own search algorithm

After both sort algorithms are written, the **sortedList1** and **sortedList2** are both sorted. There is one more function – **FindItem2**.

You have a linear search function already created, but it is up to you to implement FindItem2 however you think would be effective.

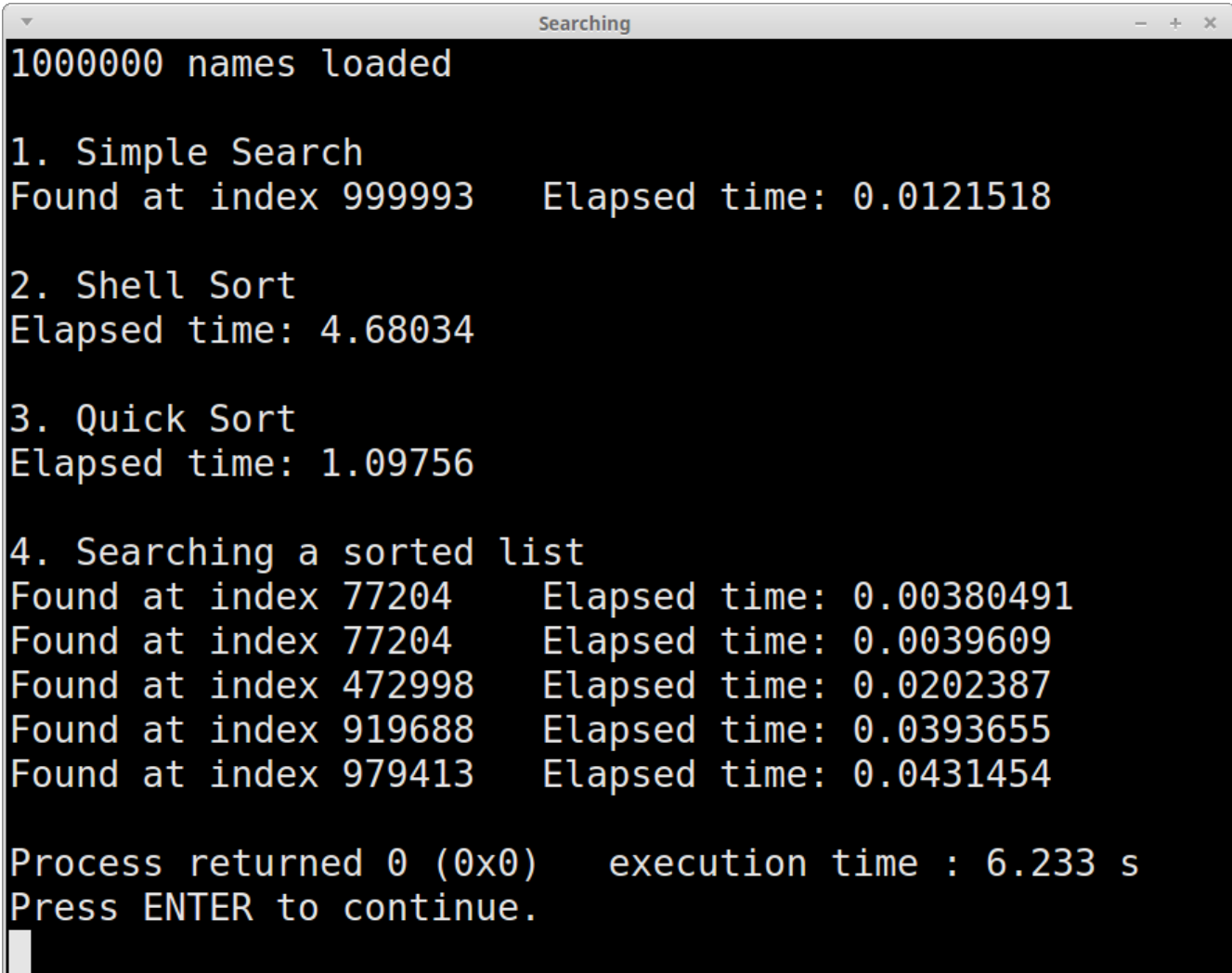
There is no “right” or “wrong” way to do this, just have it return the index of the element at some point.

Some ideas:

- Only search half the list – look at the center element, compare it to the searchTerm, and either search the top half or bottom half of the list.
- Search every 100 items to figure out the range that the searchTerm would be in ($\text{lower} < \text{searchTerm} < \text{higher}$), then just search that range of elements.

Implement any search technique you can think of, and compare it against the linear search.

Screenshot



```
Searching
1000000 names loaded

1. Simple Search
Found at index 999993      Elapsed time: 0.0121518

2. Shell Sort
Elapsed time: 4.68034

3. Quick Sort
Elapsed time: 1.09756

4. Searching a sorted list
Found at index 77204      Elapsed time: 0.00380491
Found at index 77204      Elapsed time: 0.0039609
Found at index 472998     Elapsed time: 0.0202387
Found at index 919688     Elapsed time: 0.0393655
Found at index 979413     Elapsed time: 0.0431454

Process returned 0 (0x0)    execution time : 6.233 s
Press ENTER to continue.
```

Algorithms

Linear Search

```
int FindItem1( const string& searchTerm, vector<string>&
nameList )
{
    for ( unsigned int i = 0; i < nameList.size(); i++ )
    {
        if ( nameList[i] == searchTerm )
        {
            return i;
        }
    }
    return -1;
}
```

Shell Sort

```
void ShellSort( vector<string>& arr, int size )
{
    for ( int gap = size / 2; gap > 0; gap /= 2 )
    {
        for ( int i = gap; i < size; i++ )
        {
            string value = arr[i];
            int j = i;

            for ( ; j >= gap && value < arr[j-gap]; j -= gap )
            {
                arr[j] = arr[j-gap];
            }
            arr[j] = value;
        }
    }
}
```


Quick Sort

```
void QuickSort( vector<string>& arr, int left, int right )
{
    int i = left, j = right;
    string pivot = arr[ (left + right) / 2 ];
    string value;

    while ( i <= j )
    {
        while ( arr[i] < pivot ) { i++; }
        while ( arr[j] > pivot ) { j--; }

        if ( i <= j )
        {
            value = arr[i];
            arr[i] = arr[j];
            arr[j] = value;
            i++;
            j--;
        }
    }

    if ( left < j ) { QuickSort( arr, left, j ); }
    if ( right > i ) { QuickSort( arr, i, right ); }
}
```