University of Washington Bothell

CSS 342: Data Structures, Algorithms, and Discrete Mathematics

Autumn 2015

Program 4: Sorting*

*this lab was inspired/influenced by Munehiro Fukuda's lab

Purpose

This lab will serve two purposes. First, it will provide hands-on experience for utilizing many of the sorting algorithms we will introduce in the class. Second, it will viscerally demonstrate the cost of O(n^2) v. O(n *log*n) algorithms. It will also clearly show that algorithms with the same complexity may have different running times.

Problem:

You will build a program which implements the following sorts and compares the performance for operations on arrays of integers of size 10, 100, 1000, 5000, 10000, 25000.

1) BubbleSort
2) InsertSort
3) MergeSort
4) Non-Recursive, One Extra Array MergeSort (We'll call this improved version, IterativeMergeSort from here on out in this homework)
5) QuickSort

Details:

IterativeMergeSort

In-place sorting refers to sorting that does not require extra memory arrays. For example, QuickSort performs partitioning operations by repeating a swapping operation on two data items in a given array. This requires no extra arrays.

MergeSort as defined in Carrano allocates a temporary array at each recursive call. Due to this MergeSort is slower than quicksort even though their running time is upper-bounded to O(n *log* n).

We can improve the performance of MergeSort by utilizing a non-recursive method and using only one additional array (instead of one array on each recursive call). In this improved version of MergeSort, *IterativeMergeSort*, one would merge data from the original array into the additional array and **alternatively copy back and forth between the original and the additional temporary array.** Please re-read the last sentence as it is critical to the grading of the lab.

For this improved version of MergeSort we still need to allow data items to be copied between the original and this additional arrays as many times as O(*log* n). However, given the iterative nature we are building up large stack usage.

Other Sorts

BubbleSort, InsertionSort, MergeSort, and QuickSort are well documented and you should implement them with the aid of examples in the Carrano book.

Runtime Details

Your program, called Sorter, will take in two parameters: 1) sort type as a string of characters and 2) an array size as an integer. Your program will create and sort an int array of the size with the specified sort: MergeSort, BubbleSort, InsertSort, QuickSort, or IterativeMergeSort. The driver functions below will help with the creation.

Examples:

Sorter MergeSort 100 (creates and sorts an array of 100 using MergeSort)
Sorter QuickSort 1000 (creates and sorts an array of 1000 using QuickSort)
Sorter IterativeMergeSort 10000 (creates and sorts an array of 10000 using the newly implemented non-recursive semi-in-place MergeSort)

What to turn in:

Turn in, in a .zip (not gz, etc..):

(1) Your SortImpls.h file which has implementations of all of the sorts
(2) Your .exe file or a.out
(3) A separate report in word or pdf which includes: Graphs that compares the performance among the different sorting algorithms with increasing data size

Driver Code

Use the code below as driver code to test and time the different Sort functions. Notice the sort definitions are in the SortImpls.h file that is included at the top of the driver file. This is the file you will turn in. Please make sure to spell each of the sorts by the exact name signature below.

```
#include <iostream>
#include <vector>
#include <string>
// windows.h is specific for windows.  For Linux please use
// #include <stdlib.h>
// #include <sys/time.h>
#include <windows.h>
#include “SortImpls.h”
using namespace std;

void PrinArray(const vector<int> &array, string name);
void InitArray(vector<int> &array, int randMax);

int main(int argc, char* argv[])
{
        if (argc != 3)
        {
                cout << "Usage: Sorter SORT_TYPE ARRAY_SIZE";
                return -1;
        }

        string sort_name = string(argv[1]);
        int size = atoi(argv[2]);

        if (size <= 0) {
                cerr << "array size must be positive" << endl;
                return -1;
        }

        srand(1);
        vector<int> items(size);
        InitArray(items, size);
        cout << "Sort employed:  " << sort_name << endl;
        cout << "initial:" << endl;
        PrinArray(items, string("items"));

//      GetTickCount is windows specific.
//      For linux use gettimeofday.  As shown::
//              struct timeval startTime, endTime;
//              gettimeofday(&startTime, 0);
        int begin = GetTickCount();

// Place logic to determine and call function of interest here.
// Do not call all of them
```

```
	MergeSort(items, 0, size - 1);
	IterativeMergeSort(items, 0, size - 1);
	QuickSort(items, 0, size - 1);
	BubbleSort(items, 0, size - 1);
	InsertSort(items, 0, size - 1);

	int end = GetTickCount();
//	Linux timer:
//	gettimeofday(&endTime, 0);

	cout << "sorted:" << endl;
	PrinArray(items, string("item"));
	int elapsed_secs = end - begin;
//    Linux elapsed time
//	int elapsed_secs = elapsed( startTime, endTime)
	cout << "Time (ms): " << elapsed_secs << endl;
	return 0;
}



// array initialization with random numbers
void InitArray(vector<int> &array, int randMax)
{
	int size = array.size();

	for (int i = 0; i < size;)
	{
		int tmp = (randMax == -1) ? rand() : rand() % randMax;
		bool hit = false;
		for (int j = 0; j < i; j++)
		{
			if (array[j] == tmp)
			{
				hit = true;
				break;
			}
		}
		if (hit)
		{
			continue;
		}
		array[i] = tmp;
		i++;
	}
}
```

```
// Function to Print Array
void PrinArray(const vector<int> &array, string name)
{
        int size = array.size();

        for (int i = 0; i < size; i++)
                cout << name << "[" << i << "] = " << array[i] << endl;
}

// Function to calculate elapsed time is using gettimeofday
// int elapsed( timeval &startTime, timeval &endTime )
// {
//          return ( endTime.tv_sec - startTime.tv_sec ) * 1000000
//          + ( endTime.tv_usec - startTime.tv_usec );
// }
```