CPSC 250 Fall 2011

# Lab 6: Measuring Performance of Sorting Algorithms

***By Dr. Yingwu Zhu, Seattle University***

## 1. <u>Goals</u>

In this lab, you will gain a deeper understanding of two different sorting algorithms, namely insertion sort and quicksort. As addressed in class, insertion sort is quite efficient on small input sizes, while quicksort is quite efficient on large input sizes. For quicksort, the choice of the pivot for partitioning impacts its performance significantly. A well-chosen pivot would provide O(nlogn) performance while an ill-chosen pivot would result in $O(n^2)$ performance. In this lab, you will implement these two sorting algorithm and measure their performance for various input sizes.

## 2. <u>Tasks</u>

**Step1: Create a file mysort.h that includes prototypes of the two aforementioned sorting algorithms. Namely, they are:**

```
void  insertion_sort(int* A, int n);
void  quick_sort(int* A, int n);
```

**Step2: Create a file mysort.cpp that implements the two algorithms. Please try not to refer to the example code we discussed in class.**
Note that for quicksort, to select a better pivot for partitioning, we need to randomly select a pivot or use the median-of-three rule to select a pivot. You are also encouraged use any extra optimization techniques we discussed in class to improve its performance.

**Step3: Create a client/driver program test.cpp that measures the performance of the two algorithms.**

Below, I provide the example code to facilitate the performance measurement (It is OK if you do not understand some code below):

```
/*******************************************************************
 *Author: Yingwu Zhu
 *Purpose: For Seattle U CPSC 250 lab
 *(C)opyright by Yingwu Zhu
 *Date: 11/8/2011, 22:50PM
 */

#include <iostream>
#include <cassert>
#include "mysort.h"
#include <time.h>
#include <sys/time.h>
using namespace std;

void genRandomInputs(int* A, int n); //generate n random integers
/* time elapsed for b-f in microseconds */
int timeElapsed(struct timeval& f, struct timeval& b);
bool isSorted(int* A, int n);  //if the list is sorted or not


int main(int argc, char* argv[]) {
        if (argc < 2) {
                cout << "Usage: " << argv[0] << "  input-size" << endl;
                return  -1;
        }
        int  n = atoi(argv[1]);    //input size
        assert(n > 0);
        int*  A = new int[n];
        assert(A);
        genRandomInputs(A, n);
        struct timeval start, end;

        gettimeofday(&start, NULL);
        insertion_sort(A, n);  //your insertion sort function
        gettimeofday(&end, NULL);
        assert(isSorted(A, n));
        cout << "[insertion sort]: time elpased = " << timeElapsed(end, start)
                  << " microseconds\n";

        genRandomInputs(A, n); //replay the same input by call this again
        gettimeofday(&start, NULL);
        quick_sort(A, n); //your quicksort function
        gettimeofday(&end, NULL);
        assert(isSorted(A, n));
        cout << "[quicksort]: time elpased = " << timeElapsed(end, start)
                  << " microseconds\n";

            delete [] A;
            return 0;

}

void genRandomInputs(int* A, int n) {
        srand(n);
        for (int i = 0; i < n; i++)
                A[i] = rand();
}

int timeElapsed(struct timeval& f, struct timeval& b) {
        return (f.tv_sec * 1000000 + f.tv_usec) - (b.tv_sec * 1000000 + b.tv_usec);

}

bool isSorted(int* A, int n) {
            for (int i = 0; i < n-1; i++)
                        if (A[i] > A[i+1])
                                    return false;
            return true;
}
```

**Step 4: Reuse and adapt previous Makefile and produce the executable file test. Now you need to measure the two algorithms for various input sizes. You may perform an experiment for an input size multiple times in order to minimize the interference of multi-users in CS1.**
For example:

```
./test  1000
```

This will sort a set of 1000 integers using the two algoritgms

**Step 5: Turn in (due by 9:20AM 11/30/2011)**
You need to submit a graph that depicts the performance of the two algorithms for various input sizes (e.g., 100, 1000, $10^4$, $10^5$, $10^6$, …). Please turn in the hardcopy of your results.

## 3. Extra Tasks (Optional)

If you have completed the aforementioned two sorting algorithm, you may practice other sorting algorithms like selection sort, bubble sort, and heapsort.