Lab 2 Report

Name: Hau Tao

Lab number: 24193 - CSE 330- Data Structures

Winter 2016

1. Status

I completed 100% of the lab implementing insertion sort, bubble sort, and selection sort

2. Complexity analysis

1. Selection sort

n	n^2	Exec Time (sec)	c = Time/n^2
1,000	1,000,000	0.056	5.6*10^-8
2,000	4,000,000	0.124	3.1*10^-8
3,000	9,000,000	0.136	1.51*10^-8
4,000	16,000,000	0.240	1.5*10^-8
5,000	25,000,000	0.440	1.76*10^-8
6,000	36,000,000	0.552	1.533*10^-8
7,000	49,000,000	0.752	1.534*10^-8
8,000	64,000,000	0.864	1.35*10^-8
9,000	81,000,000	1.036	1.279*10-8
10,000	100,000,000	1.352	1.352*10^-8

2. Bubble sort

n	n^2	Exec Time (sec)	c = Time/n^2
1,000	1,000,000	0.020	2*10^-8
2,000	4,000,000	0.068	1.7*10^-8
3,000	9,000,000	0.156	1.733*10^-8
4,000	16,000,000	0.284	1.775*10^-8
5,000	25,000,000	0.440	1.76*10^-8
6,000	36,000,000	0.632	1.756*10^-8
7,000	49,000,000	0.856	1.747*10^-8
8,000	64,000,000	1.112	1.738*10^-8
9,000	81,000,000	1.420	1.753*10-8
10,000	100,000,000	1.784	1.784*10^-8

3. Insertion sort

n	n^2	Exec Time (sec)	c = Time/n^2
1,000	1,000,000	0.020	2*10^-8
2,000	4,000,000	0.096	2.4*10^-8
3,000	9,000,000	0.160	1.78*10^-8
4,000	16,000,000	0.216	1.35*10^-8
5,000	25,000,000	0.308	1.23*10^-8
6,000	36,000,000	0.364	1.01*10^-8
7,000	49,000,000	0.464	9.47*10^-9
8,000	64,000,000	0.540	8.44*10^-9
9,000	81,000,000	0.596	7.36*10-9
10,000	100,000,000	0.852	8.52*10^-9

Sort type	Time complexity			Storage complexity
	Best	Average	Worst	
Selection	O(n^2)	O(n^2)	O(n^2)	O(1)
Buble	O(n)	O(n^2)	O(n^2)	O(1)
Insertion	O(n)	O(n^2)	O(n^2)	O(1)

3. Source Code

1. For the first part of the lab, I implemented the algorithm to implement the selection sort

```
/*************************************************************************************
*******
 * Hau Tao
 * select.cpp
 * 01/26/2016
 * This program implement the selection sort
 *The algorithm divides the input list into two parts: the sublist of items already sorted,
 *which is built up from left to right at the front (left) of the list, and the sublist of
 *items remaining to be sorted that occupy the rest of the list. Initially, the sorted
 *sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds
 * by finding the smallest (or largest, depending on sorting order) element in the unsorted
sublist,
 * exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order)
 *and moving the sublist boundaries one element to the right.

*************************************************************************************
******/

#include <iostream>
#include <stack>
#include <vector>
#include <cstdlib>
using namespace std;
void print_out(vector<int>&v);
void selection_sort(vector<int>&v,int n );
void random_vector( vector<int>& v, int n);

main()
{
    int n;
    cout << "Enter your size (n): \n";
    cin >> n;
    vector<int> v(n);
    srand(time(0));
    random_vector(v,n);
    selection_sort(v, n);
    print_out(v);


}
// selection sort algorithm
void selection_sort(vector<int>&v, int n )
{
    for ( int i = 0; i< n-1; i++){
        for( int j =i+1; j< n; ++j){
               if(v.at(i)>v.at(j))
                      swap(v.at(i), v.at(j));

        }
    }
```

```
}

// print out the sorted vector
void print_out(vector<int>&v)
{
    for( int i =0; i< v.size(); i++)
        cout << v.at(i)<<endl;
}
// create the random number vector
void random_vector( vector<int> &v, int n)
{
    for( int i =0; i< n; i++)
        v.at(i) = rand()/1000000;
}
```

2. For the second part of the lab, I implemented the algorithm to implement the bubble sort

```
/*************************************************************************************************
*******
 * Hau Tao
 * buble.cpp
 * 01/26/2016
 * This program implement the bubble sort
 * The bubble sort makes multiple passes through a list. It compares adjacent items and
exchanges
 * those that are out of order. Each pass through the list places the next largest value in
its proper
 * place. In essence, each item "bubbles" up to the location where it belongs.

*************************************************************************************************
******/
#include <iostream>
#include <stack>
#include <vector>
#include <cstdlib>
using namespace std;
void print_out(vector<int>&v);

void buble_sort (vector<int>&v, int n);

void random_vector( vector<int>& v, int n);

main()
{
    int n;
    cout << "Enter your size (n): \n";
    cin >> n;
    vector<int> v(n);
    srand(time(0));
    random_vector(v,n);
    buble_sort(v,n);
    print_out(v);
```

```
}

// buble sort algorithm
void buble_sort (vector<int>&v, int n)
{
    for (int i = n-1; i > 0; i--){
        for (int j = 0; j < i; j++){
               if (v.at(j)> v.at(j+1))
               swap(v.at(j), v.at(j+1));

        }
    }
}


// print out the sorted vector
void print_out(vector<int>&v)
{
    for( int i =0; i< v.size(); i++)
        cout << v.at(i)<<endl;
}
// create the random number vector
void random_vector( vector<int> &v, int n)
{
    for( int i =0; i< n; i++)
        v.at(i) = rand()/1000000;
}
```

3. For the third part of the lab, I implemented the algorithm to implement the insertion sort

```
/*************************************************************************************
*******
 * Hau Tao
 * buble.cpp
 * 01/26/2016
 * This program implement the insertionsort
 *Insertion sort iterates, consuming one input element each repetition, and growing a sorted
output list.
 * Each iteration, insertion sort removes one element from the input data, finds the
location it belongs
 *within the sorted list, and inserts it there. It repeats until no input elements remain.

*************************************************************************************
******/
#include <iostream>
#include <stack>
#include <vector>
#include <cstdlib>
using namespace std;
void print_out(vector<int>&v);
```

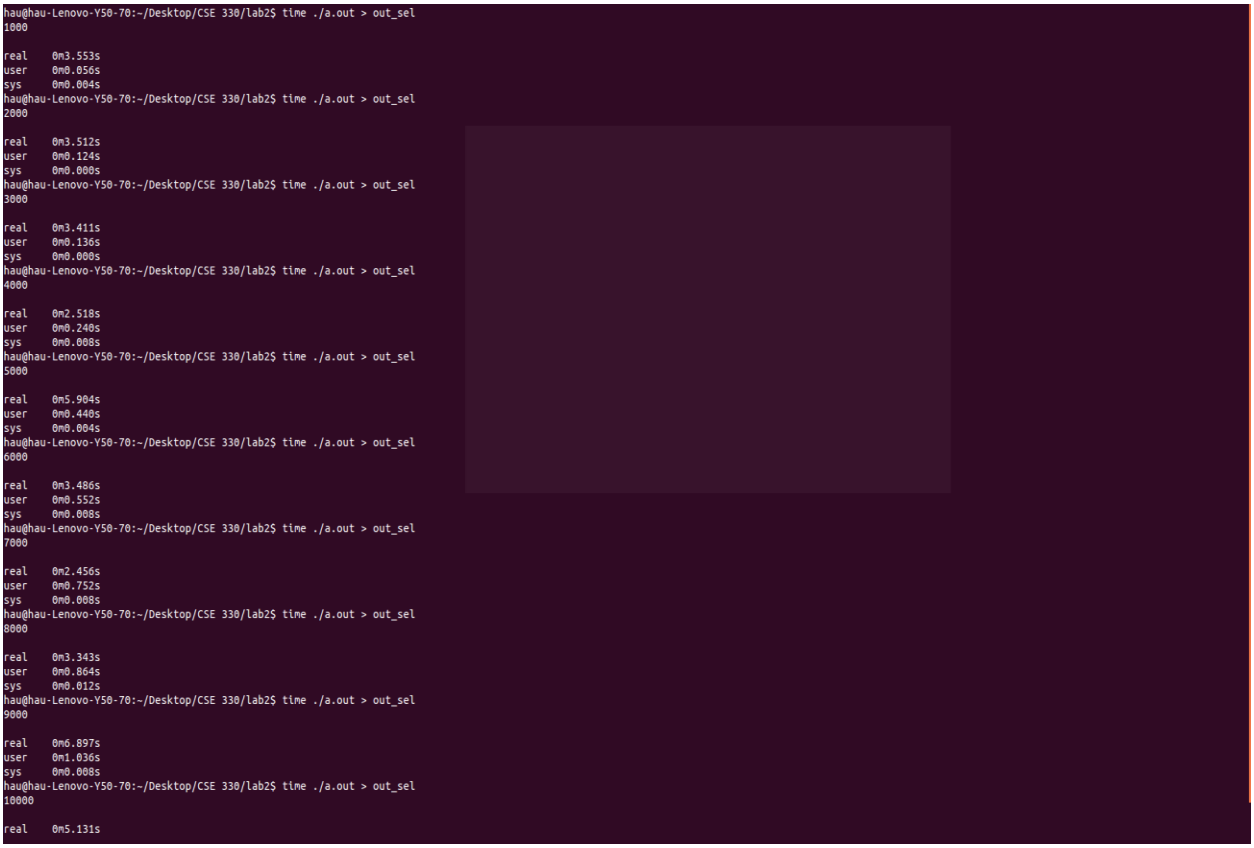
```
void insertion_sort( vector<int>&v, int n);
void random_vector( vector<int>& v, int n);

main()
{
    int n;
    cout << "Enter your size (n): \n";
    cin >> n;
    vector<int> v(n);
    srand(time(0));
    random_vector(v,n);
    insertion_sort(v,n);
    print_out(v);
}
// insertion sort algorithm
void insertion_sort( vector<int>&v, int n)
{
    int elem, j;
    for (int i = 1; i < n; i++){
        for (  elem = v.at(i),  j = i-1; j >= 0 && elem < v.at(j); j--)
               v.at(j+1) = v.at(j);
       v.at(j+1) = elem;
    }
}

// print out the sorted vector
void print_out(vector<int>&v)
{
    for( int i =0; i< v.size(); i++)
        cout << v.at(i)<<endl;
}
// create the random number vector
void random_vector( vector<int> &v, int n)
{
    for( int i =0; i< n; i++)
        v.at(i) = rand()/1000000;
}
```

4. Sample Run

1. Selection sort



with n = 10 , sample output is:

```
Enter your size (n):
74
358
607
844
1053
1162
1185
1580
1954
2077
```

2. Buble sort

```
real    0m2.618s
user    0m0.040s
sys     0m0.000s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
2000

real    0m2.534s
user    0m0.124s
sys     0m0.000s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
3000

real    0m2.787s
user    0m0.280s
sys     0m0.000s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
4000

real    0m3.538s
user    0m0.272s
sys     0m0.004s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
5000

real    0m3.206s
user    0m0.464s
sys     0m0.008s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
6000

real    0m4.368s
user    0m0.660s
sys     0m0.008s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
7000

real    0m3.415s
user    0m0.872s
sys     0m0.008s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
8000

real    0m3.907s
user    0m1.100s
sys     0m0.012s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
9000

real    0m5.689s
user    0m1.420s
sys     0m0.012s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./a.out > output_buble
10000

real    0m4.957s
user    0m1.736s
sys     0m0.020s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$
```

with n = 10, sample output is:

```
Enter your size (n):
12
54
177
531
821
901
995
1245
1346
1356
```

3. Insertion sort

```
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
1000

real    0m2.126s
user    0m0.020s
sys    0m0.000s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
2000

real    0m5.858s
user    0m0.096s
sys    0m0.004s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
3000

real    0m2.476s
user    0m0.160s
sys    0m0.004s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
4000

real    0m2.959s
user    0m0.216s
sys    0m0.008s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
5000

real    0m3.013s
user    0m0.308s
sys    0m0.004s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
6000

real    0m3.058s
user    0m0.364s
sys    0m0.000s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
7000

real    0m2.476s
user    0m0.464s
sys    0m0.004s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
8000
```

```
real    0m2.376s
user    0m0.540s
sys    0m0.004s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
9000

real    0m4.552s
user    0m0.596s
sys    0m0.008s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$ time ./insert > out_inserttion
10000

real    0m33.764s
user    0m0.852s
sys    0m0.008s
hau@hau-Lenovo-Y50-70:~/Desktop/CSE 330/lab2$
```

with n = 10, sample output is:

```
Enter your size (n):
480
539
609
858
862
1365
1392
1397
1869
2140
```