Preparatory data Structure (CSCI 591)

Project - VIII

Comparing three Sorting Algorithms

Submitted By: Taddese Erba

April 9, 2020

St. Cloud state university

Department of Computer Science

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

Design Document

Introduction

A sorting algorithm is an algorithm that puts elements in a certain order. The order of the elements is often governed by certain comparison rules that a programmer is interested in. Sorting is often employed to optimize the operation of other algorithms that operate on the elements.

There are several sorting algorithms in computer science. The most common ones are Insertion Sort, Merge Sort, and Quicksort. Sorting algorithms use numerical values to sort elements of a data structure. This project explores the use of sorting algorithms and their data structure and compares the time complexity of these sorting algorithms.

Data Structure

The program has three distinct files. The `sort.h` file contains all the declaration of the required member functions and variable. It is the framework for `sort` class implementation. It consists of three private variables that are used for counting the number of program executions. Furthermore, the `sort` class contains eleven functions each with their operations as discussed in the following section of this document.

Functions

As described in the Data Structure section of this document, there are eleven functions in this project. The functions `int counter1()` is used to return the number of program execution of the insertion sort algorithm. The functions `int counter2()` is used to return the number of program execution of the merge sort algorithm. The functions `int counter3()` is used to return the number of program execution of the quicksort algorithm. The three counting functions take no argument and return the number of cycles the program executes. The `void resetCounter()` is used to initialize the counters.

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

It takes no argument and returns none. The `void printArray(int [], int)` function is used to print the unsorted and sorted arrays. It takes the array and its size as an argument but does not return anything. The `void insertion_sort(int [], int)` function takes the array and its size as its argument performs sorting operation on the array. The functions, `void merge(int [], int, int, int)` is used to merge the two halves of the array. The functions, `void merge_sort(int [], int, int)` is used to recursively sort the two halves of the array. The functions, `int partition(int [], int, int)` is used to rearrange the elements of the array in the two halves of the array based on their value compared to the element of the array called the pivot. It takes the array, the starting index of the array, and the ending index of the array as its arguments and returns an integer value, the index of the new pivot position. The functions, `void quick_sort(int [], int, int)` is used to recursively sort the two halves of the array. The functions, `void swap(int*, int*)` is used swap the array elements to rearrange them and put them in their correct position.

The Main Program

The `main`() function is used to promote the user to enter the size of the array and the initial smallest value of the array. It hosts the variables, the testing functions and displays the output on the terminal. A series of the statement is printed on the terminal requesting inputs, and functions are called from the class to do the job. It also reports what the user wants to see in the output.

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

Code listing

a. The header file (List.h)

```
1
2   /*
3       This is the header file. It contains the class "sort" that hosts
4       the various public functions and three private objects.
5           => The public functions perform different operations
6              as specified in the implimentation file.
7           => The three private objects hold the counters to count the
8              number of recursive function calls for the three sorting.
9              algorithms.
10      Precondition: - The program works for integer sorting on a randomly
11                      generated integer numbers.
12      Postcondition: - The program will sort the array using insertion sort,
13                       merge sort, and quicksort algorthms. It also reports the
14                       number of recursive calls for each sorting algoriths.
15  */
16  #include <iostream>
17  #ifndef _SORT
18  #define _SORT
19  using namespace std;
20  class Sort{
21      private:
22          int count1; //the number of insertion sort operation counts
23          int count2; //the number of merge sort  operation counts
24          int count3; //the number of quicksort operation counts
25      public:
26          int counter1(){return count1;}  //returns the number of insertion sort counts.
27          int counter2(){return count2;}  //returns the number of merge sort counts.
28          int counter3(){return count3;}  //returns the number of quicksort counts.
29          void resetCounter(){
30              count1 = 0;count2 = 0;count3 = 0;}  //function initializes counters to 0
31          void printArray(int [], int);       // to print the array
32          void insertion_sort(int [], int);
33          void merge(int [], int, int, int);
34          void merge_sort(int [], int, int);
35          int partition(int [], int, int);
36          void quick_sort(int [], int, int);
37          void swap(int*, int*);              // to swap the array elements.
38  };
39  #endif
```

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

b. The implementation file (`List.cpp`)

```
40
41  /*
42      This is the implementation file.
43      it contains all the function implementations.
44  */
45  #include <iostream>
46  #include "sort.h"
47  using namespace std;
48  // to swap the array elements for quicksort.
49  void Sort::swap(int* a, int* b){
50      int t = *a;
51      *a = *b;
52      *b = t;
53  }
54  //print array
55  void Sort::printArray(int a[], int p){
56      int t = 0;
57      int n = 0;
58      int j;
59      int k;
60      for(k = 0; k < p; k++){
61          t++;
62          for(j = 0; j < 20; j++){     //only 20 elemnt on a line.
63              cout << a[n++] << " ";
64              if(n == p)               // exit if all array element read
65                  break;
66          }
67          k = j*t;
68          cout<<endl;
69          if(n == p)
70              break;
71      }
72  }
73  void Sort::insertion_sort(int b[], int n){
74      ++count1;
75      if(n <= 1)
76          return;
77          insertion_sort(b, n-1);      //recursive call
78          int last = b[n-1];
79          int j = n-2;
80          while(j >= 0 && b[j] > last){
81              b[j+1] = b[j];
82              j--;
83          }
84          b[j+1] = last;
85  }
```

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

```
85  }
86  void Sort::merge(int arr[], int low, int mid, int high){
87      int i1, i2, index;
88      i1 = low;
89      i2 = mid + 1;
90      int b[high - low + 1];      // local array
91      index = 0;
92      for(int i = low; i <= high; i++){
93          if(i1 > mid)
94              b[index++] = arr[i2++];
95          else if(i2 > high)
96              b[index++] = arr[i1++];
97          else if(arr[i1] < arr[i2])
98              b[index++] = arr[i1++];
99          else
100             b[index++] = arr[i2++];
101     }
102     for(int k = 0; k < index; k++)
103         arr[low++] = b[k];
104 }
105 void Sort::merge_sort(int arr[], int low, int high){
106     int mid;
107     ++count2;
108     if(low < high){
109         mid = (low + high)/2;
110         merge_sort(arr, low, mid);
111         merge_sort(arr, mid + 1, high);
112         merge(arr, low, mid, high);
113     }
114 }
115 void Sort::quick_sort(int a[], int low, int high){
116     ++count3;
117     if(low < high){
118         int p_pos = partition(a, low, high);    //get the pivot
119         quick_sort(a, low, p_pos-1);
120         quick_sort(a, p_pos+1, high);
121     }
122 }
123 int Sort::partition(int a[], int low, int high){
124     Sort S;
125     int start = low + 1;    //start at the second element
126     int pivot = a[low];   //make the first element a pivot
127     for(int k = low+1; k <= high; k++){
128         if(a[k] < pivot)
129             S.swap(&a[start++], &a[k]);
130     }
```

```
131     S.swap(&a[low], &a[start-1]);
132     return start-1; //current location of pivot
133 }
```

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

c. The testing file (main.cpp)

```
134
135 /*
136     This is the main method. it promotes the user to enter inputs,
137     it tests the implementations, and prints the outputs.
138 */
139 #include <iostream>
140 #include <cstdlib>
141 #include "sort.cpp"
142 #include <cmath>
143 using namespace std;
144 static const int MAX = 5000;
145 int num;
146 int main(int argc, const char * argv[]){
147     int num1, num2, num3;
148     int arr[MAX];
149     int a[MAX];
150     int ar[MAX];
151     int seed;
152     char ch;
153     cout << " This program will sort an array of random numbers\n"
154          << " and return the the program execution time for merge\n"
155          << " and quick sorts."<< endl;
156     cout << " ================================================="<<endl;
157     cout << " You must enter a number between 1 to 5000."<<endl;
158     cout << " Enter the number of values you want to generate: ";
159     cin >> num;
160     cout << " Enter an integer seed value: ";
161     cin >> seed;
162     srand(seed);
163     Sort S;
164     S.resetCounter();
165     for(int i = 0; i < num; i++){
166         a[i] = (rand()%num);
167         ar[i] = a[i];
168         arr[i] = a[i];
169     }
170     cout << " Your array is sorted successfully."<<endl;
171     cout << " Do you wish to print the values? (y/n): ";
172     cin >> ch;
173     if(ch == 'y'|| ch == 'Y'){
174         cout << " \nUnsorted array: "<<endl;
175         cout << " =============="<<endl;
176         S.printArray(a, num);
177
178         S.insertion_sort(a, num);
179         cout << "\nArray sorted by insertion sort."<<endl;
```

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

```
180         cout << "================================"<<endl;
181         S.printArray(a, num);
182 
183         S.merge_sort(ar, 0, num-1);
184         cout << "\nArray sorted by merge sort."<<endl;
185         cout << "============================="<<endl;
186         S.printArray(a, num);
187 
188         S.quick_sort(arr, 0, num-1);
189         cout << "\nArray sorted by quick sort."<<endl;
190         cout << "============================="<<endl;
191         S.printArray(a, num);
192     }
193     else if(ch == 'n'|| ch == 'N'){
194         S.insertion_sort(a, num);
195         num1 = S.counter1();
196         int exc = pow((num1/2), 2);
197         cout << " Insertion sort count: "<< exc<<endl;
198 
199         S.merge_sort(ar, 0, num);
200         num2 = S.counter2();
201         int ex = num2*log(num2);
202         cout << " Merge sort count: "<< ex<<endl;
203 
204         S.quick_sort(arr, 0, num);
205         num3 = S.counter3();
206         int e = num3*log(num3);
207         cout << " Quick sort count: "<< e<<endl;
208     }
209     else{
210         cout << " The choice is unavailable."<<endl;
211         cout << " Run the program again and make the right choice."<<endl;
212     }
213     return 0;
214 }
```

Test Results

I will be providing the partial run here. However, I am attaching the whole run at the end of this document for reference.

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

```
This program will sort an array of random numbers
and return the the program execution time for merge
and quick sorts.
==================================================
You must enter a number between 1 to 5000.
Enter the number of values you want to generate: 980
Enter an integer seed value: 23
Your array is sorted successfully.
Do you wish to print the values? (y/n): n
Insertion sort count: 239121
Merge sort count: 14858
Quick sort count: 9566

--------------------------------
Process exited after 25.03 seconds with return value 0
Press any key to continue . . .
```

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

```
 This program will sort an array of random numbers
 and return the the program execution time for merge
 and quick sorts.
 ================================================
 You must enter a number between 1 to 5000.
 Enter the number of values you want to generate: 77
 Enter an integer seed value: 5
 Your array is sorted successfully.
 Do you wish to print the values? (y/n): y

Unsorted array:
 ===============
54 49 12 40 17 14 57 16 60 69 32 62 58 74 41 72 28 10 61 76
43 58 8 29 64 56 40 64 49 51 49 38 31 17 15 62 44 10 42 67
53 57 20 32 26 51 64 18 18 42 37 46 29 63 15 66 43 61 64 34
36 65 4 27 46 53 68 20 0 60 60 19 76 28 18 46 13

Array sorted by insertion sort.
================================
0 4 8 10 10 12 13 14 15 15 16 17 17 18 18 18 19 20 20 26
27 28 28 29 29 31 32 32 34 36 37 38 40 40 41 42 42 43 43 44
46 46 46 49 49 49 51 51 53 53 54 56 57 57 58 58 60 60 60 61
61 62 62 63 64 64 64 64 65 66 67 68 69 72 74 76 76

Array sorted by merge sort.
=============================
0 4 8 10 10 12 13 14 15 15 16 17 17 18 18 18 19 20 20 26
27 28 28 29 29 31 32 32 34 36 37 38 40 40 41 42 42 43 43 44
46 46 46 49 49 49 51 51 53 53 54 56 57 57 58 58 60 60 60 61
61 62 62 63 64 64 64 64 65 66 67 68 69 72 74 76 76

Array sorted by quick sort.
=============================
0 4 8 10 10 12 13 14 15 15 16 17 17 18 18 18 19 20 20 26
27 28 28 29 29 31 32 32 34 36 37 38 40 40 41 42 42 43 43 44
46 46 46 49 49 49 51 51 53 53 54 56 57 57 58 58 60 60 60 61
61 62 62 63 64 64 64 64 65 66 67 68 69 72 74 76 76

--------------------------------
Process exited after 6.248 seconds with return value 0
Press any key to continue . . .
```

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

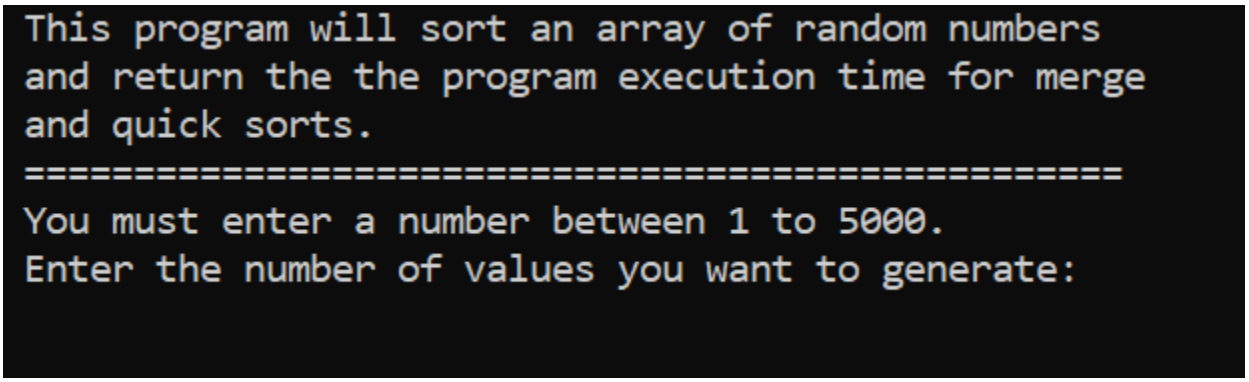
User document

This program can perform queuing simulation operations. To run the program, you must perform the following steps.

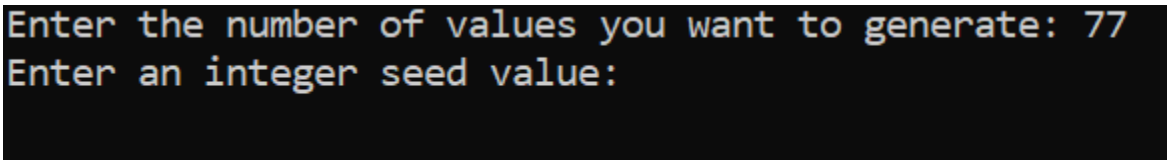
- Run the `main.cpp`. To compile and run the program, enter the following command to on the terminal window.

  ```
  g++ -o main main.cpp
  ```

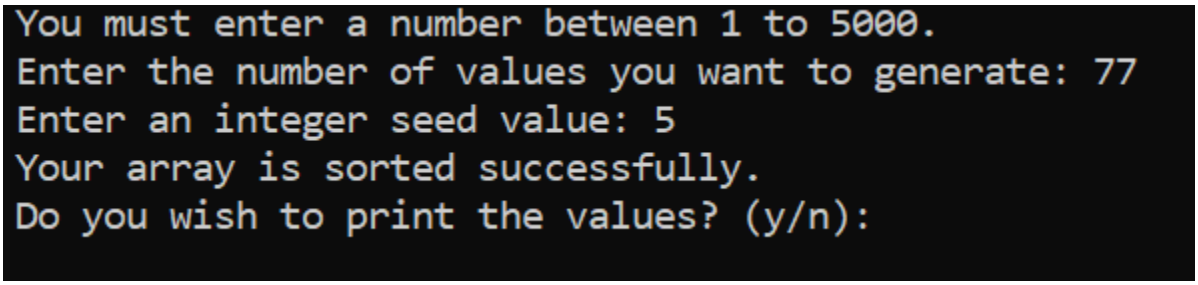
- The program will compile and open the following window:



- Once the window opens, enter an integer number less than 5000 as the size of the array .



- Enter a pseudo number seed and press enter.



Finally, choose what you want to see.

i. If you want to see the statistics only enter n(N):

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

```
Do you wish to print the values? (y/n): n
Insertion sort count: 1444
Merge sort count: 775
Quick sort count: 483
```

ii. If you want to see the sorted array, enter y(Y).

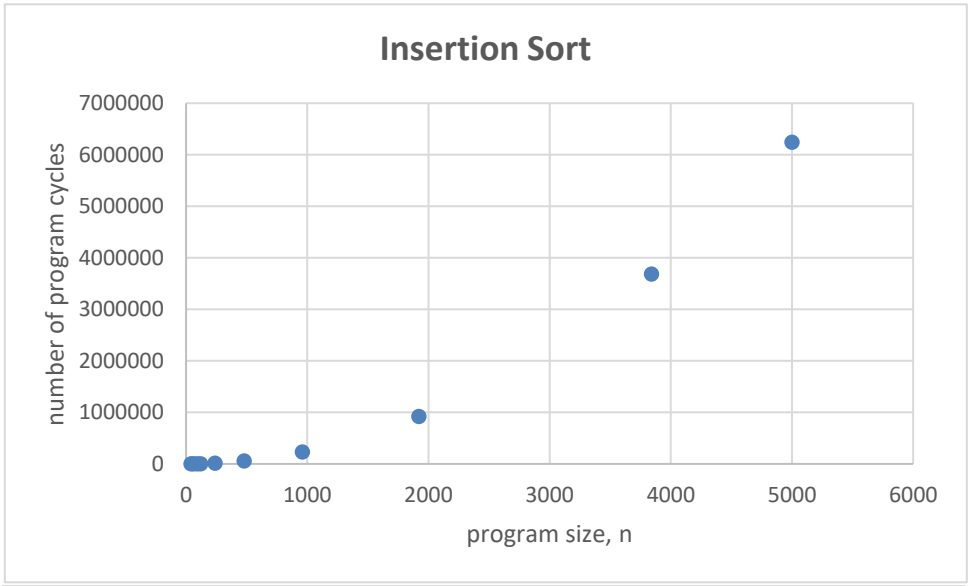
```
 Do you wish to print the values? (y/n): y

Unsorted array:
 ==============
54 49 12 40 17 14 57 16 60 69 32 62 58 74 41 72 28 10 61 76
43 58 8 29 64 56 40 64 49 51 49 38 31 17 15 62 44 10 42 67
53 57 20 32 26 51 64 18 18 42 37 46 29 63 15 66 43 61 64 34
36 65 4 27 46 53 68 20 0 60 60 19 76 28 18 46 13

Array sorted by insertion sort.
================================
0 4 8 10 10 12 13 14 15 15 16 17 17 18 18 18 19 20 20 26
27 28 28 29 29 31 32 32 34 36 37 38 40 40 41 42 42 43 43 44
46 46 46 49 49 49 51 51 53 53 54 56 57 57 58 58 60 60 60 61
61 62 62 63 64 64 64 64 65 66 67 68 69 72 74 76 76

Array sorted by merge sort.
============================
0 4 8 10 10 12 13 14 15 15 16 17 17 18 18 18 19 20 20 26
27 28 28 29 29 31 32 32 34 36 37 38 40 40 41 42 42 43 43 44
46 46 46 49 49 49 51 51 53 53 54 56 57 57 58 58 60 60 60 61
61 62 62 63 64 64 64 64 65 66 67 68 69 72 74 76 76

Array sorted by quick sort.
============================
0 4 8 10 10 12 13 14 15 15 16 17 17 18 18 18 19 20 20 26
27 28 28 29 29 31 32 32 34 36 37 38 40 40 41 42 42 43 43 44
46 46 46 49 49 49 51 51 53 53 54 56 57 57 58 58 60 60 60 61
61 62 62 63 64 64 64 64 65 66 67 68 69 72 74 76 76
```

Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

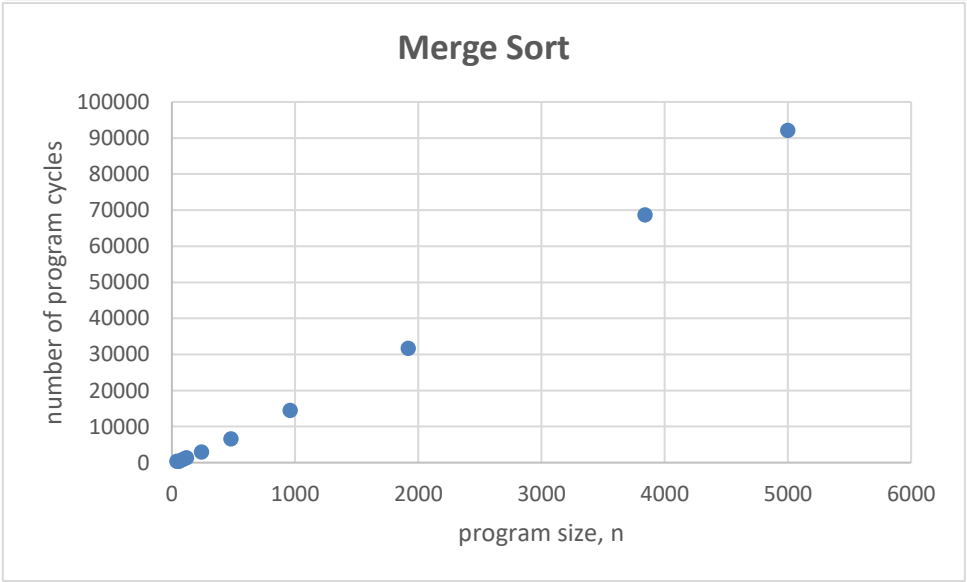
Summery

The sorting algorithms in this project use the array of integers that contain randomly generated numbers by the `rand()` function. The array is passed to each function to be sorted and a counter is incremented each time the functions make a call to themselves. From the obtained counters, the characteristic operations are calculated accordingly to compare if it agrees to the theoretical values. The values for different program sizes are tabulated and an approximate graph is produced for each of the three sorting algorithms.

40	361
60	841
90	1936
120	3481
240	14161
480	57121
960	229441
1920	919681
3840	3682567
5000	6245001

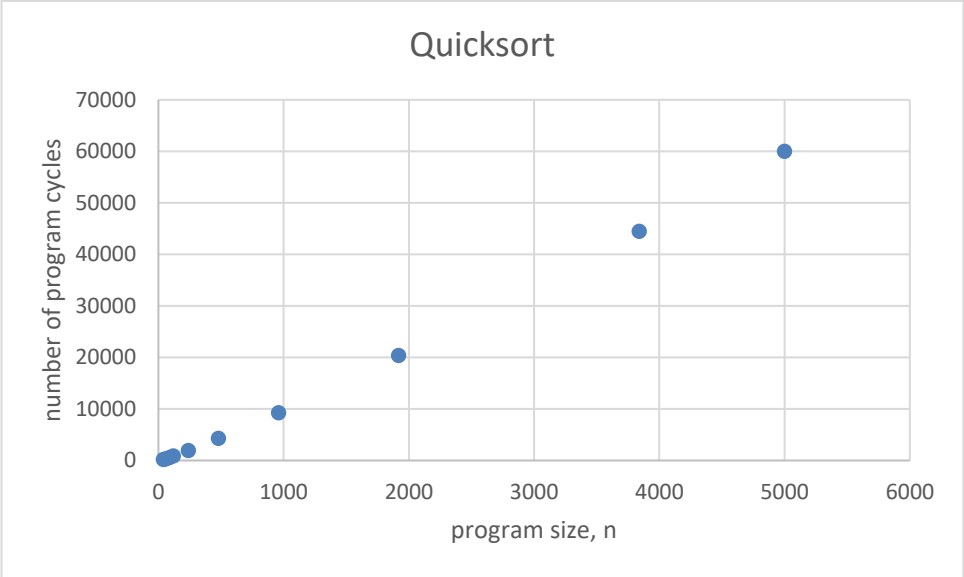


40	350
60	374
90	934
120	1315
240	2963
480	6592
960	14515
1920	31692
3840	68708
5000	92103



Taddese Erba
Section – I
Project – Seven
Due: April 9, 2020

40	215
60	339
90	562
120	873
240	1913
480	4284
960	9288
1920	20373
3840	44455
5000	60027



The project is quite strong in terms of difficulty. The underlying principles of the three sorting algorithms, however, effectively revealed what is going on within the algorithmic codes. As it is shown in the tables and the graph results, it is possible to see that the time complexity of the algorithms is in agreement with what is discussed in class. The results tend to vary with the initial arrangements of the values particularly for the quicksort. Of all the three algorithms, the insertion algorithm has the highest number of execution cycles.

By completing this project, I have gained a significant level of confidence and the necessary knowledge to work with sorting algorithms. I don't see how I can further improve these programs at this time. However, as I apply them frequently, I think there can be a way these programs be improved and used more practically.