Project 9

**COMPARING THREE SORTING**

**ALGORITHMS**

Full Name: Prashul Shrestha

Section #: 2

Project #: 9

Due Date: April 17, 2017

**Design** **Document**

**Introduction**

**Big O** notation is used in Computer Science to describe the performance or complexity of an algorithm. **Big O** specifically describes the worst-case scenario, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

Big-O is used in calculating the time taken by different sorting algorithms to graph their different time taken to sort arrays. In this project implements and exercises these algorithms and an earlier algorithm whose time is *O(n2)*. This investigation should confirm our theoretical observations about their behaviors.

**Data** **Structures**

This program uses three **arrays** [] and one **dynamic array[]** of size inputted by the user. First the user is prompted to enter the size of the array, then a random seed number to initialize elements into the dynamic array[]. After the elements have been initialized in the array, we call the first function Insertion\_Sort and then after that we put the sorted array into another array named Insertion Array[]. And, we delete the items in the **dynamic array[]** and reinitialize elements into it using the **rand()** and the step of Insertion\_sort continues with Merge Sort and Quick Sort. In this way, we get the returned count value of each algorithm. And later print the list if user wants to or not and print the count of the three algorithms.

**Functions**

The program uses **six** functions to implement Big O. The functions are called from main() and some are member function to return the result within the function which called it. The list of the functions are given below:

* Int Insertion\_Sort(int Array[], int size)– This will sort the array using Insertion Algorithm and return the merge count.
* Int Merge\_sort (int Array[], int low, int high): This will sort the array using merge sorting Algorithm with the help of another array and return the merge count.
* Void Merge(int arr[], int low, int mid, int high): Help the function Merge\_sort to sort the array into another array.
* Int quick\_sort(int arr[], int low, int high)– Returns the quick­\_count and sort the arrays using quick sort algorithm.
* Void Partition (int arr[], int low, int mid, int high): Help Quick­\_sort function to partition the array into corresponding length.

**Menu**() is a the main function from where we call all those functions by value and save the returned count of each algorithm.

**The Main Program**

**Menu**() is a the main function from where we call all those functions by value and save the returned count of each algorithm. Here, first the user is prompted to enter the size of the array, then a random seed number to initialize elements into the dynamic array[]. After the elements have been initialized in the array, we call the first function Insertion\_Sort and then after that we put the sorted array into another array named Insertion Array[]. And, we delete the items in the **dynamic array[]** and reinitialize elements into it using the **rand()** and the step of Insertion\_sort continues with Merge Sort and Quick Sort. In this way, we get the returned count value of each algorithm. And later print the list if user wants to or not and print the count of the three algorithms.

User Document

**Big O** notation is used in Computer Science to describe the performance or complexity of an algorithm. **Big O** specifically describes the worst-case scenario, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

Big-O is used in calculating the time taken by different sorting algorithms to graph their different time taken to sort arrays. In this project implements and exercises these algorithms and an earlier algorithm whose time is *O(n2)*. This investigation should confirm our theoretical observations about their behaviors.

The main program named **BigO**.**cpp** can be compiled and run, using the code:

**g++ BigO**.**cpp**

**a.out**

**g++** function will compile the function and make it ready to be run using **a.out**. The function will prompt the following output:

**Ouptut:**

**Hello, Today we are going to find the BIG-O for three sorting algorithms.**

**-----------------------**

**Press 'c' to continue and 'q' to Quit-> c**

**.**

**.**

**.**

**Enter the number of values to generate and sort, between 1 and 5000: 25**

**Enter a random number seed: 12**

**Print the List('y' for Yes and 'n' for No): y**

**Insertion Sort:**

**Unsorted Array:**

**{ 5077, 5629, 6233, 9057, 6558, 6155, 7949, 9931, 6983, 7375, 9078, 8634, 9665, 7229, 9703, 7375, 8081, 8015, 9969, 7064, 6895, 6058, 5226, 6779, 9858, }**

**Sorted Array:**

**{ 0, 5077, 5226, 5629, 6058, 6155, 6233, 6558, 6779, 6895, 6983, 7064, 7229, 7375, 7375, 7949, 8015, 8081, 8634, 9057, 9078, 9665, 9703, 9858, 9931, }**

**Merge Sort:**

**Unsorted Array:**

**{ 7956, 7443, 8341, 6343, 9757, 5690, 9859, 9173, 7310, 5292, 5345, 5515, 7954, 6868, 5890, 9584, 8465, 7658, 8409, 8358, 6156, 9600, 7457, 6300, 5768, }**

**Sorted Array:**

**{ 7956, 7443, 8341, 6343, 9757, 5690, 9859, 9173, 7310, 5292, 5345, 5515, 7954, 6868, 5890, 9584, 8465, 7658, 8409, 8358, 6156, 9600, 7457, 6300, 5768, }**

**Quick Sort:**

**Unsorted Array:**

**{ 6045, 8286, 7507, 5151, 9945, 6175, 6833, 5197, 9795, 6809, 7831, 6995, 7839, 9980, 7433, 6685, 9616, 5271, 5753, 7057, 5445, 9241, 8096, 8089, 6290, }**

**Sorted Array:**

**{ 5151, 5197, 5271, 5445, 5753, 6045, 6175, 6290, 6685, 6809, 6833, 6995, 7057, 7433, 7507, 7831, 7839, 8089, 8096, 8286, 9241, 9616, 9795, 9945, 9969, }**

**Insertion Sort: 155**

**Merge Sort: 317**

**Quick Sort: 164**

**-----------------------**

**Press 'c' to continue and 'q' to Quit-> q**

**.**

**.**

**.**

**Summary**

Since I used dynamic array to play around the numbers initialized in the array, I learnt the use of dynamic arrays in the real context.

Other than that, I used different values and check the time count for each algorithm and I got the following result in the graphs I have attached. From the graphs, we can confirm that the Big O of algorithms that was discussed in class. Yes, the operations vary with the initial arrangement of the value because already sorted array need not be sorted again.

**TESTING**

**CODING**