**Sandesh Ghimire**

**CSCI 301 section 3**

**Computer Science 2**

**Project 7**

**26th October**

**Design Document**

**Introduction:**

           Sorting is the process of arranging number, characters, or words systematically. We learned about different algorithms of sorting. In this program we did the algorithm for Merge sort, Quick sort and Insertion Sort. The Big O notation for Merge and Quick sort are (n\**log*n) and Insertion sort is *n*. This Proves that merge sort and Quick sort are faster than Insertion sort. We did it all theoretically on the lector but we are doing all of it practically to see which sort performs faster.

Insertion sort basically works by removing a value from the list, finds the location it belongs within the sorted list and replace it. Quick sort works by selecting a pivot. It then separates the list itself into two parts by checking which part is greater or smaller than pivot. Lastly, the separated sub list sorts recursively. Merge sorts works by dividing the list into different sub lists and sorts each and every sub list recursively.

**Data Structure**

           The data structure we used to do this program is array. We used the array that can hold random integer values. Then the program sorts the Array in ascending order using three different algorithms.

**Functions**

*int Insertion()* : This function basically sorts the array that contains random integers. It starts with a for loop storing the value to the temp variable. It again starts a pre-condition loop to determine weather the is to be sorted or not. At last when the while loop ends the temp value is replaced back to the sorted position.

*int Quick()* : This function takes three parameter that contains an array, and two integer values. It basically sets up the recursion for the Quick sort algorithm.

*int MergeS()* : This function takes three parameter that contains an array, and two integer values. It basically sets up the recursion for the Merge sort algorithm.

*void Partition()* : It takes six parameters which includes an array and five integer variables. Tow of them are called by reference. This function is the heart of Quick sort. It performs the whole sorting algorithm. A while loop is used at first checking if the minimum value is less then the maximum value of not. If the condition is not satisfied the loop terminates. So, it can be called a pre-condition. Then the program starts the conation checking if the values in array are smaller then pivot or not. If the condition is satisfied the program swaps the values of the array.

*void merge()* : This function takes four parameters which includes an array and three integer variables. This function is the heart of the Merge sort algorithm. It starts with a for loop. Inside the for loop again a pre-conditional loop checking the high and mid values. If the condition is satisfied the value are inserted inside the array by sorting.

*void printArray*() : This function prints the arrays which are sorted by the using all the three sorting algorithm. It basically uses the for loop three times, printing every value in the array.

*Swap() and rand()* : Swap and rand are built in function gave by the IDE itself. Swap exchanges the two values. Rand generates the random integer within given boundary.

**The Main Program:**

The main function firstly asks some inputs for the users. After getting the input the array of the required length is generated. The values are inserted in each array by using rand function. Now the main function calls the all three functions of sorting algorithm, lastly printing the values if needed.

**Code**

/\*Sorting is the process of arranging number, characters, or words systematically.

We learned about different algorithms of sorting. In this program we did the algorithm for Merge sort,

Quick sort and Insertion Sort. The Big O notation for Merge and Quick sort are (n\*logn) and Insertion sort is n.

This Proves that merge sort and Quick sort are faster than Insertion sort.

We did it all theoretically on the lector but we are doing all of it practically to see which sort performs faster.

Insertion sort basically works by removing a value from the list, finds the location it belongs within the sorted list and replace it.

Quick sort works by selecting a pivot.

It then separates the list itself into two parts by checking which part is greater or smaller than pivot.

Lastly, the separated sub list sorts recursively.

Merge sorts works by dividing the list into different sub lists and sorts each and every sub list recursively.

\*/

#include <iostream>

#include<iomanip>

#include<cstdlib>

#include <cmath>

using namespace std;

int Insertion(int[], int);

int Quick(int[], int, int);

int MergeS(int[], int, int);

void printArray(int[], int[], int[], int);

void Partition(int[], int, int, int, int & , int & );

void Merge(int[], int, int, int);

int CountQuick = 0;//global variables

int CountInser = 0;//global variables

int CountMerge = 0;//global variables

int Num;//global variables

int main() { //The main function

int Seed;

int Count = 0;

char Yn;

//Asking the input to the users

cout << "Enter the number of values to generate and sort, between 1 and 5000: ";

cin >> Num;

cout << " Enter an integer seed value: ";

cin >> Seed;

cout << "Print the values? (y/n)";

cin >> Yn;

srand(Seed);//ramdomizing the seed

int ToSort[Num];//declaring the array

int TempArray[Num];//declaring the array

int TempArray1[Num];//declaring the array

for (int i = 1; i <= Num; i++) {//starting for loop to insert values to the array

ToSort[i] = (rand() % 5000) + 1;

TempArray[i] = ToSort[i];

TempArray1[i] = ToSort[i];

}

Insertion(ToSort, Num);//calling the function

Quick(TempArray, 0, Num);// calling the function

MergeS(TempArray1, 0, Num);//calling the function

if (Yn == 'y') {

printArray(ToSort, TempArray, TempArray1, Num);

}

cout<<endl;//Printing the values for counts

cout << " The Count for Insertion Sort is " << CountInser << endl;

cout << "The Count for Merge Sort is " << CountMerge << endl;

cout << "The Count for Quick Sort is " << CountQuick << endl;

return 0;

}

int Quick(int TempArray[], int Min, int Max) {//quick function is used for recursion

int pivot;

int lastS1, FirstS3;

if (Min < Max) {//starting the pre-condition

pivot = TempArray[Min];//giving value to pivot

Partition(TempArray, Min, Max, pivot, lastS1, FirstS3);//calling the function

Quick(TempArray, Min, lastS1);//calling the function

Quick(TempArray, FirstS3, Max);//calling the function

}

}

int Insertion(int ToSort[], int n) {//function to perform insertion sort

int j, Temp, i;//declaring variable

for (i = 1; i <= n; i++) {//starting for loop

Temp = ToSort[i];//giving the first value to the temp

j = i - 1;

while (j >= 0 && ToSort[j] > Temp) {//starting while loop

++CountInser;

ToSort[j + 1] = ToSort[j];//exchanging the value

j = j - 1;

}

ToSort[j + 1] = Temp;//giving back the temp value

}

}

void printArray(int ToSort[], int TempArray[], int TempArray1[], int n) {//function to print the sorted arrays

int i;

int Count = 0;

cout << "=======================================================================================================================" << endl;

cout << endl;

cout << "-------------------------------------------------------Insertion Sort-----------------------------------------------------" << endl;

cout << endl;

for (i = 1; i < n; i++) {//starting for loop

cout << ToSort[i] << " ";//printing the datas in the array

}

cout << endl;

cout << endl;

cout << endl;

cout << "----------------------------------------------------------Quick sort--------------------------------------------------" << endl;

cout << endl;

for (i = 1; i < n; i++) {

cout << TempArray[i] << " ";

}

cout << endl;

cout << endl;

cout << "----------------------------------------------------------Merge sort--------------------------------------------------" << endl;

cout << endl;

for (i = 0; i < n - 1; i++) {

cout << TempArray1[i] << " ";

}

cout << endl;

cout << "=======================================================================================================================" << endl;

}

void Partition(int TempArray[], int Min, int Max, int pivot, int & i, int & j) {//function to sort using quick sort algorithm

int FirstU = Min;

int LastS1 = Min - 1;

int FirstS3 = Max + 1;

while (FirstU < FirstS3) {//starting the pre-condition loop

++CountQuick;

if (TempArray[FirstU] < pivot) {//checking if first value us less then pivot

++LastS1;//increament

swap(TempArray[FirstU], TempArray[LastS1]);//exchanging the value

++FirstU;//increase value by 1

} else if (TempArray[FirstU] == pivot) {//another condition if first is not satisfied

++FirstU;

} else {//if all not

--FirstS3;//decreasing the value

swap(TempArray[FirstU], TempArray[FirstS3]);//exchanging the vale

}

}

i = LastS1;

j = FirstS3;

}

int MergeS(int TempArray1[], int low, int high) {//function for merge sort

int mid;

if (low < high) {//starting the pre condition

mid = (low + high) / 2;//giving mid value

MergeS(TempArray1, low, mid);//calling the function recursively

MergeS(TempArray1, mid + 1, high);//calling the function recursively

Merge(TempArray1, low, mid, high);//calling the function

}

}

void Merge(int TempArray1[], int low, int mid, int high) {//merge sort algorithm

int a[Num];//local copy of array

int i1, i2, index;

for (index = low; index <= high; ++index) {//starting the for loop

a[index] = TempArray1[index];//giving the value to the local array

}

i1 = low;

i2 = mid + 1;

index = low;

while (i1 <= mid && i2 <= high) {//starting the while loop

++CountMerge;//increment

if (a[i1] < a[i2]) {//if condition to check the value of array

TempArray1[index] = a[i1];//give the value to the main array from local copy

++i1;//increment

} else {//if not

TempArray1[index] = a[i2];//give the value to the main array from local copy

++i2;//increment

}

++index;//increment

}

while (i1 <= mid) {//while loop for mid value

TempArray1[index] = a[i1];//give the value to the main array from local copy

++i1;//increment

++index;//increment

}

while (i2 <= high) {//while loop for high value

++CountMerge;//increment

TempArray1[index] = a[i2];//give the value to the main array from local copy

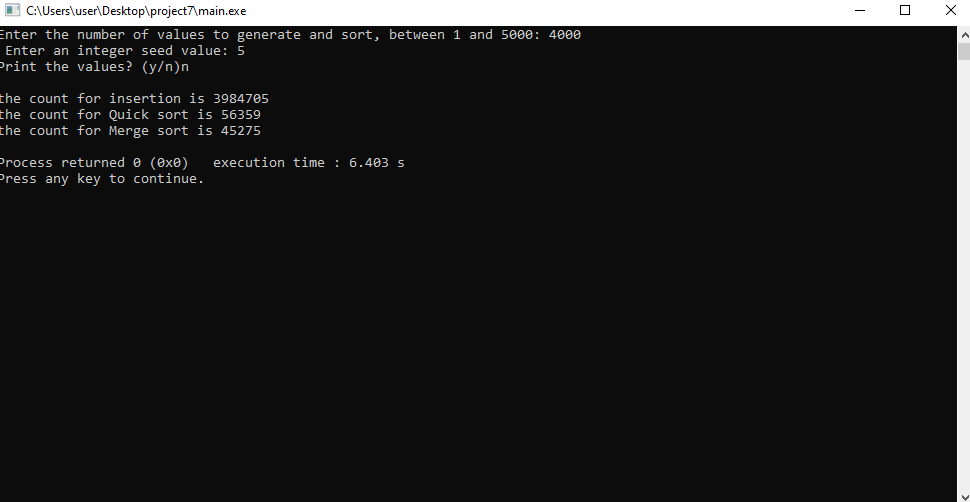
++i2;//increment

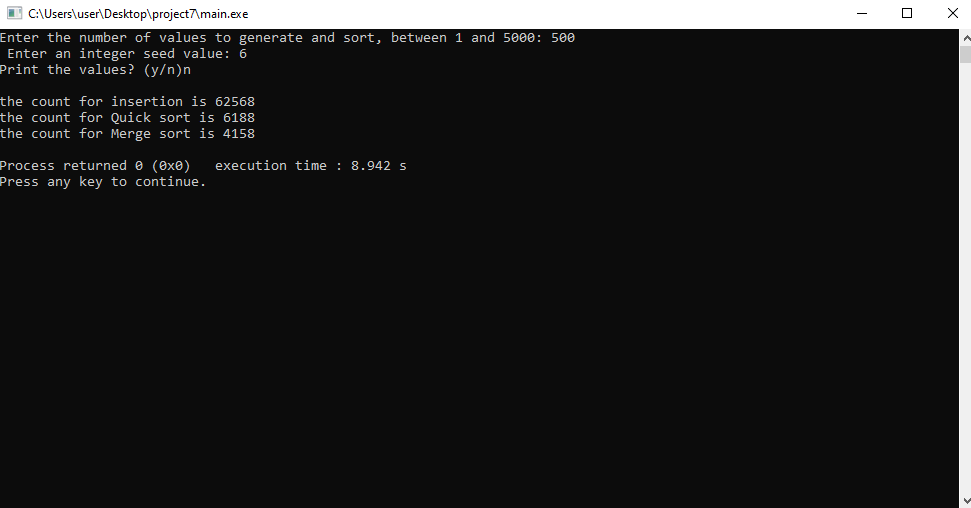
++index;//increment

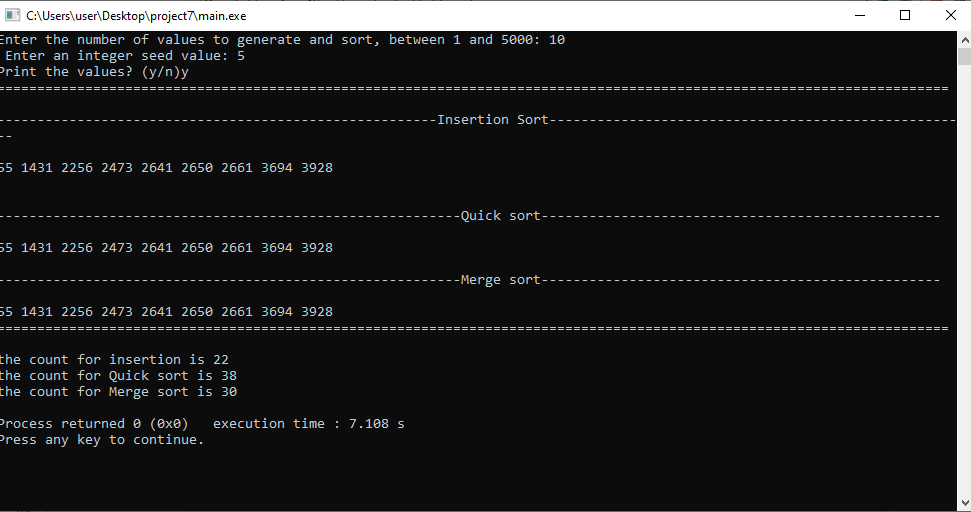
}

}

**Test**

****

****

****

**User Document**

To compile the program simply enter:

*g++ -o main main.cpp*

To run the program, enter. */main,* then you can input an integer. When the program opens it asks user to enter values.

For example, when you run the program it asks input:

*Prompt>./main*

*Enter the number of values to generate and sort, between 1 and 5000: 700*

*Enter an integer seed value: 4*

*Print the values? (y/n) y*

If you enter those three lines the program displays the required output.

**Summary**

In this project, we learned about different sorting algorithm. We did thee different algorithm for sorting i.e. Insertion sort, merge sort, and quick sort. According to the result we found that merge sort and quick sort are most efficient while we have a large amount of data. But Insertion sort was also good while we have small amount of data. This program proves that Merge sort is the most efficient sorting algorithm.

Yes, the results confirm our discussions of the algorithms' big-O times in class.