

**Preparatory data Structure (CSCI 591)**



**Comparing three Sorting Algorthms**

**Project - VIII**

**April 9, 2020**

**St. Cloud state university**

**Department of Computer Science**

Submitted By: Taddese Erba

# **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**

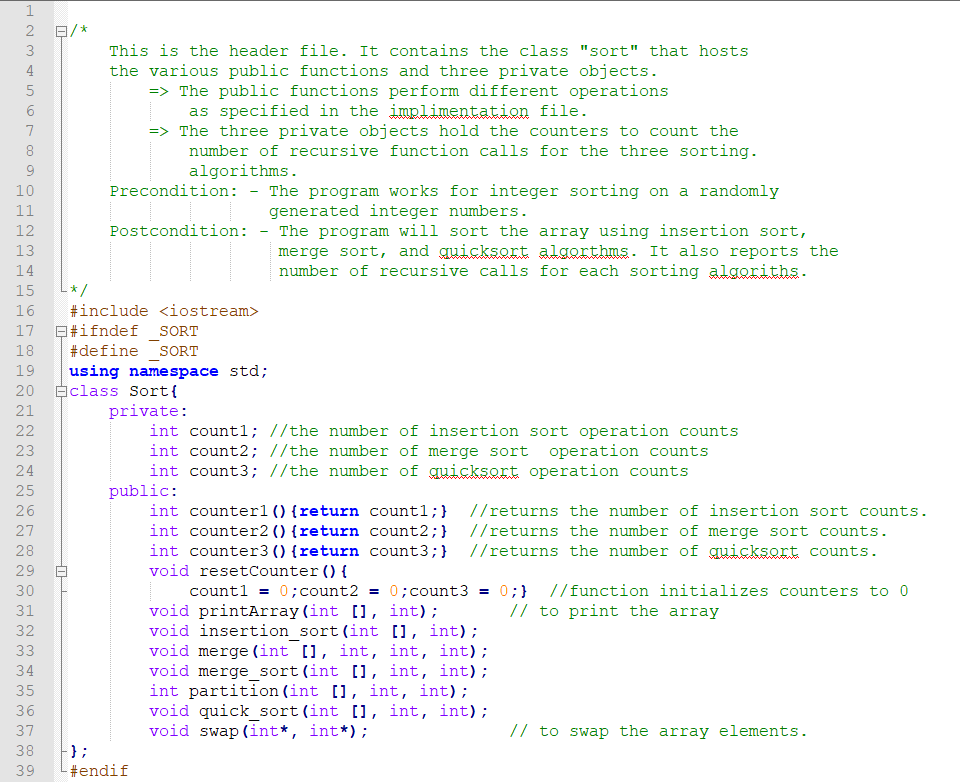
As described in the Data Structure section of this document, there are eleven functions in this project. The functions int counter1() is used to return the number of program execution of the insertion sort algorithm. The functions int counter2() is used to return the number of program execution of the 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. 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**

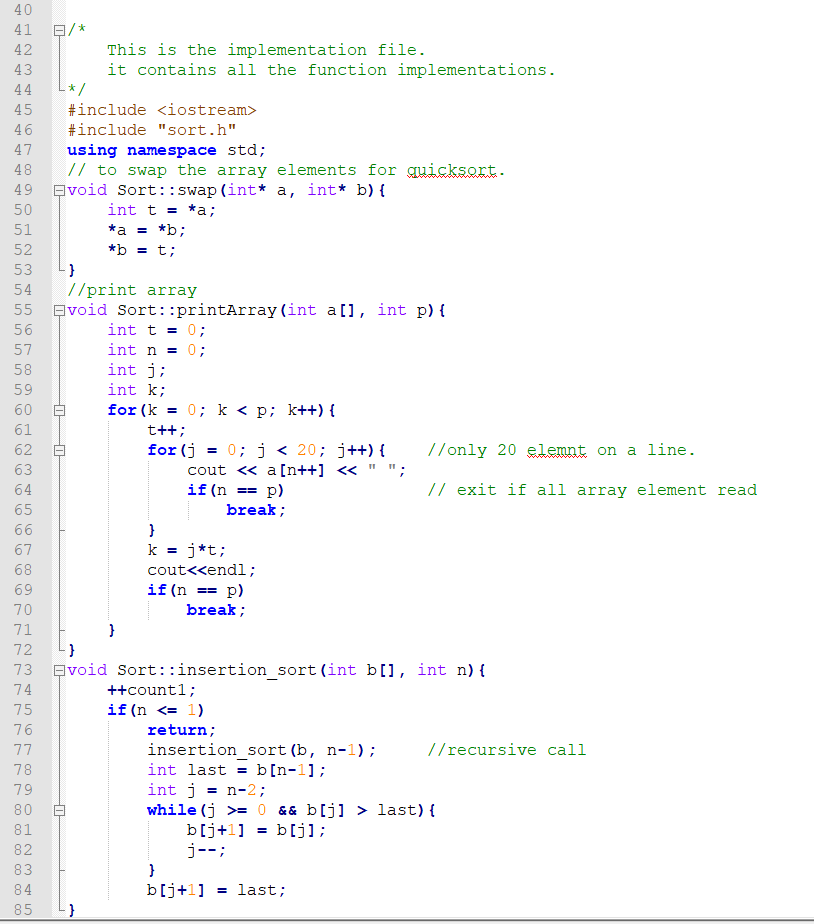
The main() function is used to promote the user to enter the size of the array and the initial smallest value of the array. It hosts the variables, the testing functions and displays the output on the terminal. A series of the statement is printed on the terminal requesting inputs, and functions are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

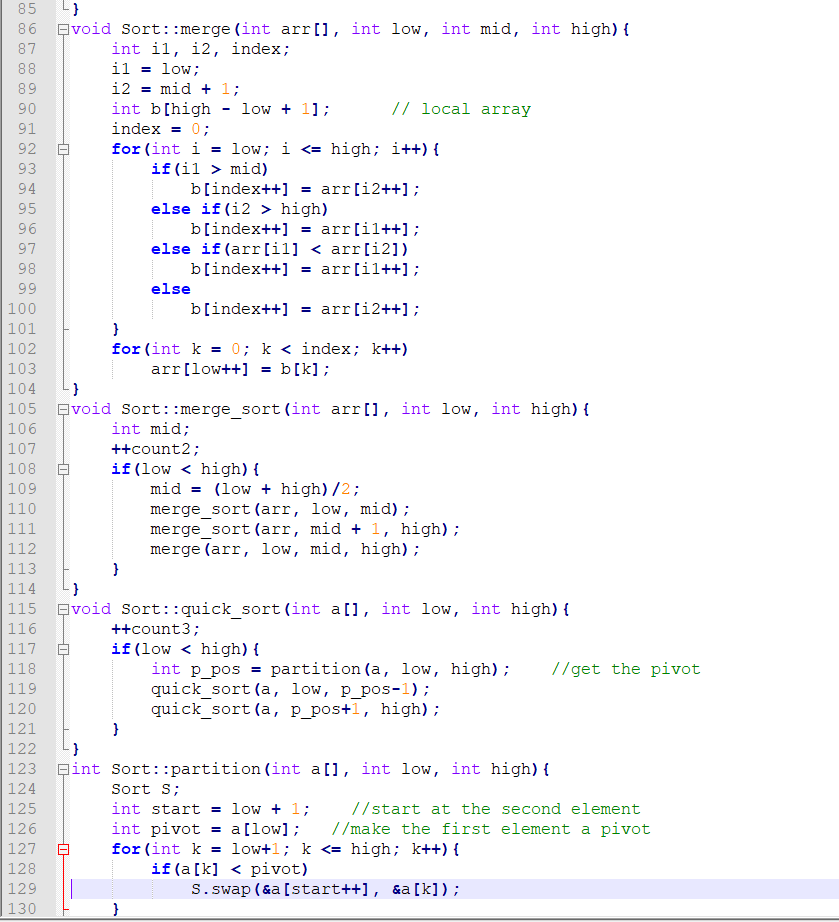
# **Code listing**

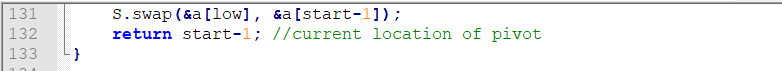
## **The header file (List.h)**



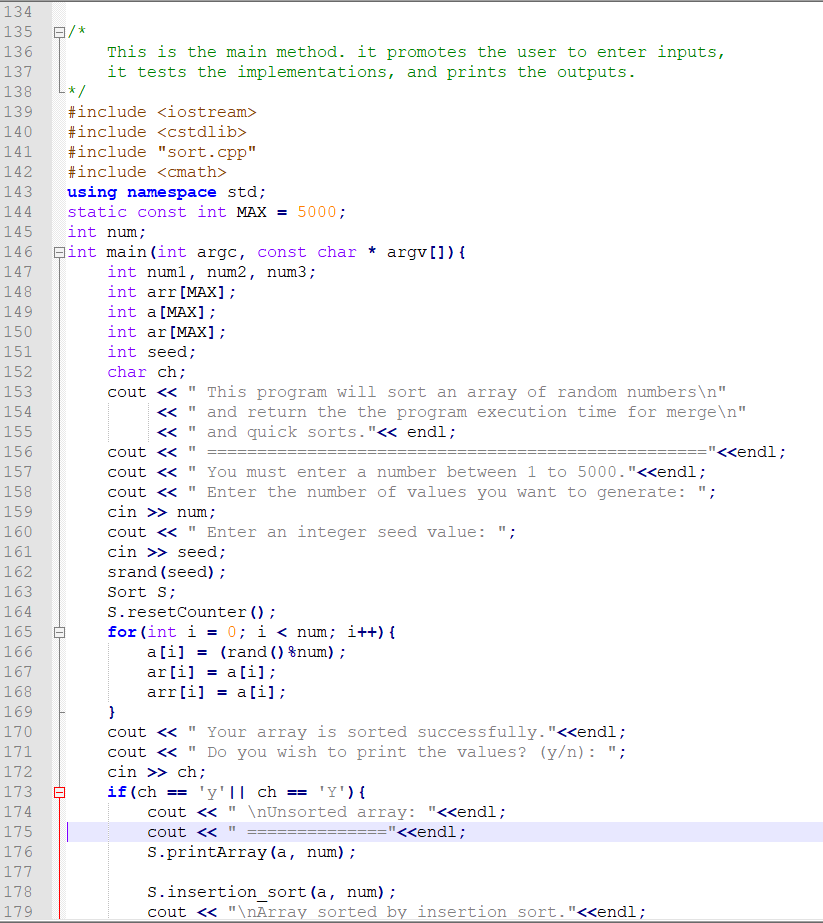
## **The implementation file (List.cpp)**

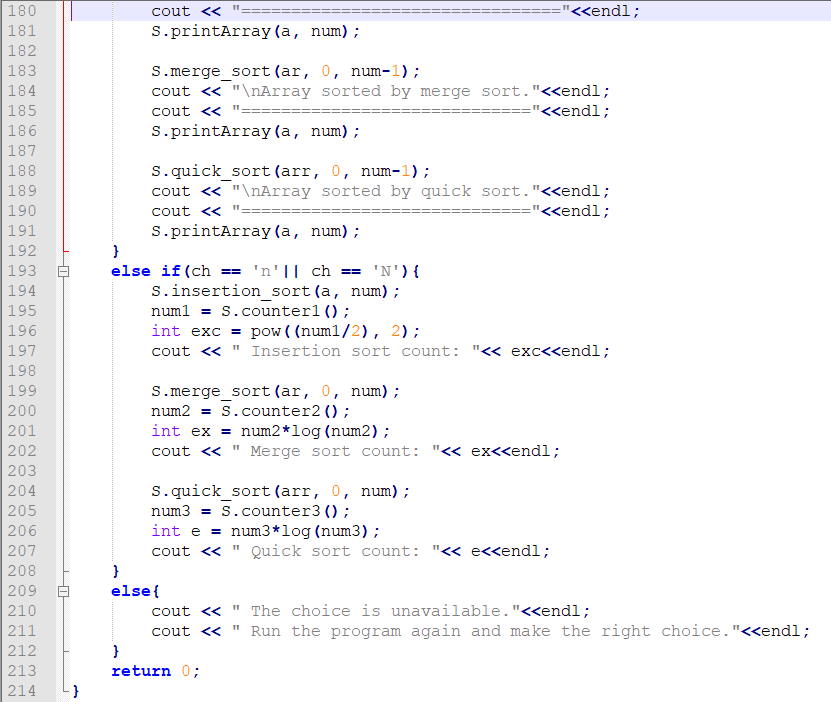






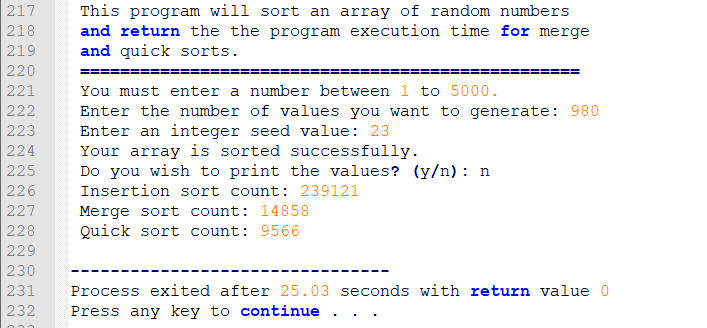
## **The testing file (main.cpp)**

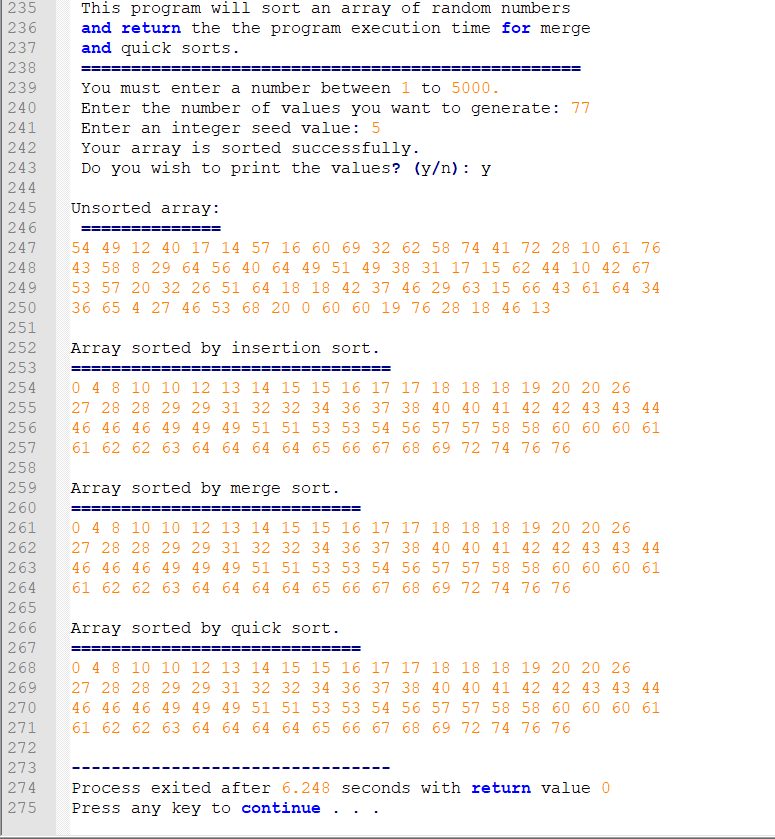




# **Test Results**

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





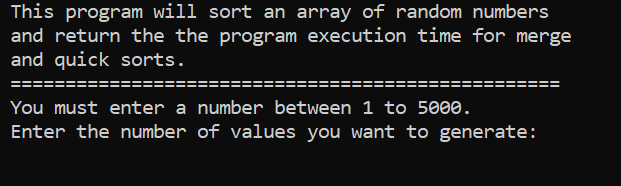
# **User document**

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

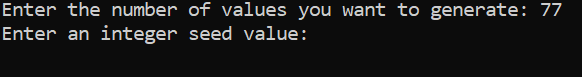
* Run the main.cpp. To compile and run the program, enter the following command to on the terminal window.

g++ -o main main.cpp

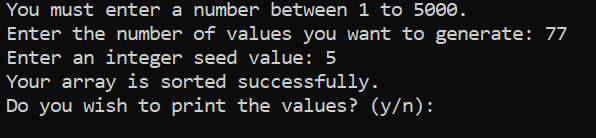
* The program will compile and open the following window:



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

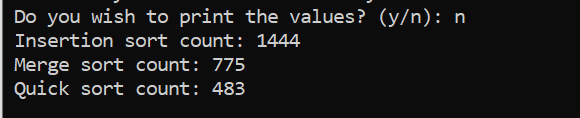


* Enter a pseudo number seed and press enter.

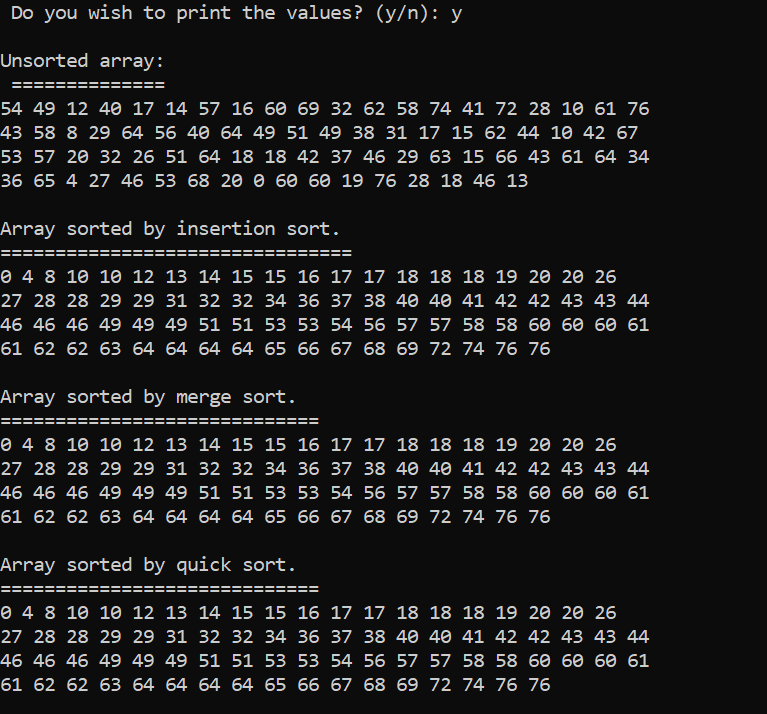


Finally, choose what you want to see.

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



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



# **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 |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 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.