**Sort Performance Analysis Toolkit Documentation**

By Jordan Rudge

1.0 - Specification

1.1 - Project Proposal

My system will assess the performance of various sorting algorithms by tasking these algorithms to sort various datasets of varying size and ordering with times compared.

I will load these sets from file and output the sorted sets into a new file. The algorithms will be timed during the sorting process and these times will be displayed in a database. The database will store the average times for each specific sort and the corresponding dataset used. Will use a good range of sorting algorithms and datasets.

1.2 - Requirements

* Error handling to be able to take input from the user without returning an error. Can use while loops to ensure a user inputs valid values for the program to use.
* Modular functions for each different sorting algorithm, which can be reused and implemented with different inputs and array sizes depending on the user’s choice.
* Data structures to store and implement data within the program.
* A file handling functionality, from reading files into an array/struct, to the opposite.
* Needs a menu to allow the user to do what they want within the program.

2.0 - Design and Implementation

2.1 – Design (Pseudocode)

2.1.1 – Initialisation

Int z = 0;

Constant int a = 5;

Struct Algorithms {string Algorithms[a] = {all sorting algorithms}; double Times[a];}

2.1.2 – Main Function

If z = 0 {z = z + 1; output message about program;};

Option1 = Menu()

Switch (option1) {

Case 1: {

Option2 = algChoice();

Option2 = option 2 – 1

If option2 = 5 {option2 = -1; main();}

Int array\_choice = arraySizeChoice();

If array\_choice = 6 {array\_choice = 0; main();}

Timer(array\_choice, option2);

Cout << endl;

System(“pause”);

Main(); }

Case 2: {

outputDB;

system(“pause”);

main(); }

case 3: {

exit(); }

2.1.3 – Menu function

Int option = 0;

String choice\_array[4] = “Menu”, “Algorithms”, “Database”, “Exit”;

outputChoices(choice\_array, 4);

cin >> option;

while (!cin) or (option != 1 and option != 2 and option != 3 and option != 4) {

cout << “Pick valid option”;

cin.clear();

cin.ignore(numeric\_limits<int>::max());

cin >> option; }

return option;

2.1.4 – algChoice function

Int option = 0;

String choice\_array[7] = “Algorithms”, “Bubble”, “Quick”, “Insertion”, ”Selection”, “Shell”, “Back”;

outputChoices(choice\_array, 7);

cin >> option;

while (!cin) or (option != 1 and option != 2 and option != 3 and … option != 6) {

cout << “Pick valid option”;

cin.clear();

cin.ignore(numeric\_limits<int>::max());

cin >> option; }

return option;

2.1.5 – arraySizeChoice function

Int option = 0;

String choice\_array[7] = “Array Size”, “Extra Small”, “Small”, “Medium”, ”Large”, “Extra Large”, “Menu”;

outputChoices(choice\_array, 7);

cin >> option;

while (!cin) or (option != 1 and option != 2 and option != 3 and … option != 6) {

cout << “Pick valid option”;

cin.clear();

cin.ignore(numeric\_limits<int>::max());

cin >> option; }

return option;

2.1.6 – outputChoices function

Cout << array[0] << endl;

For (Int I = 1; I < n; i++) { cout << I << array[i] << endl;}

Cout << “Choose an option” << endl;

For (int j = 1; j < n; j++) {if (j<=1) {cout << “ “ << j;} else { cout << “ | “ << j;}}

Cout << “): “;

2.1.7 - Timer function

Int size = arraySize(array\_choice);

Int\* tempArray = new int[size]

For (int I = 0; I < size; i++) { tempArray[i] = rand(100) + 1;}

start timer;

switch (option2 + 1) {

case 1: bubbleSort(tempArray, size);

case 2: quicksort(tempArray, 0, (size – 1);

case 3: insertionSort(tempArray, size);

case 4: selectionSort(tempArray, size);

case 5: shellSort(tempArray, size);

stop timer;

delete[] tempArray;

time = finish – start;

cout << time << “s” << endl;

inputDB(alg.Algorithms[option2], time, array\_choice);

2.1.8 – outputDB function

Ifstream inFile;

Cout << “\t\t\t\t\t Size of Array”;

Int size[5];

For (int I = 1, I < a + 1; i++) { size[I – 1] = arraySize(i);}

Printf(“\t\t%15i%15i%15i%15i%15i”, size[0], size[1], size[2], size[3], size[4]);

For (int j = 0; j < a j++) {

string file\_name = alg.Algorithms[j] + “.txt”;

inFile.open(file\_name);

cout << “\n\t” << alg.Algorithms[j];

for (int k = 0; I < a; k++) {if inFile >> alg.Times[k]) {} else { alg.Times[k] = 0; }

inFile.close();

Printf(“\t\t%15i%15i%15i%15i%15i”, alg.Times[0], alg.Times[1], alg.Times[2], alg.Times[3], alg.Times[4]); }

2.1.9 – inputDB function

Array\_choice -= 1;

Int n = 5;4ifstream inFile;

File\_name += “.txt”;

inFile.open(file\_name);

for (int I = 0; I < n; i++) {if inFile >> alg.Times[i]) {} else { alg.Times[i] = 0; }

inFile.close();

if (alg.Times[array\_choice] == 0) {alg.Times[array\_choice] = time; }

else { alg.Times[array\_choice] = (alg.Times[array\_choice] + time) / 2; }

ofstream outFile;

outFile.open(file\_name);

for (int j = 0; j < n; j++) { outFile << alg.Times[j] << endl; }

outFile.close();

2.1.10 – arraySize function

Int sizes[] = { 100, 500, 1000, 5000, 10000 };

Int array\_sie = sizes[n – 1];

Return array\_size;

2.2 – Implementation

2.2.1 – Use Case Diagram

A close up of a map

Description automatically generated

2.2.2 – Implementation problems

I changed the program to instead generate a random array of data instead of loading from a file as I specified in my proposal. While I did not necessarily have any problems implementing this, I did feel however that generating a new array of data each time would improve the program by giving the user a more accurate result for sorting the average array of data for a particular size.

2.3 – Testing

2.3.1 – White Box Testing

2.3.1.1 – Definition

Testing based on an analysis of the internal structure of the component or system.

2.3.1.2 – Method

The functions that output the choice menus (menu, algChoice, arraySizeChoice) could be analysed to find what type of input the function will respond to, and first test out which inputs the functions will legally take and see if they produce the intended response. Then other inputs can be attempted, for example, symbols, letters, or numbers outside of the valid range. These inputs should not break the program, and the user should be prompted to re-enter an input. The legal inputs should correspond correctly to the user’s choice, and this can be verified by checking the files in the project folder for an updated time for a particular algorithm and the size of array chosen. In the similar way, the database can be tested to ensure the right times are being output to the user.

2.3.2 – Black Box Testing

2.3.2.1 – Definition

Testing, either functional or non-functional, without reference to the internal structure of the component or system.

2.3.2.2 - Method

Almost the same as white box, however the “legal” inputs from the user will just be determined by the output from the program to the user. Also the user will only be able to use the database to check if the right algorithm and array size was chosen corresponding to what the user entered.