CSE331: Data Structures and Algorithms

Merge Sort Lab Report

Name: Anthony Amgad Fayek

Program: CESS

ID: 19P9880

The Full Project is in a GitHub Repository Below

Here are the used libraries and definitions:

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cstdlib>
#define LENGTH 10000
using namespace std;
```

Part 2A:

Writing the mergesort and merge functions (this includes the counter (the variable "step") that is required in Part 3C:

```
int merge(int arr[], int s, int m, int l) {
	int step = 5;
	int* firstarr = new int[m - s + 1];
	int* secondarr = new int[l - m];
	for (int i = 0; i < m - s + 1; i++) {
		firstarr[i] = arr[s + i];
		step++;
	}
	for (int i = 0; i < l - m; i++) {
		secondarr[i] = arr[m + i +1];
		step++;
	}
	int i = 0, j = 0, k = s;

	while (i < m - s + 1 && j < l - m) {
		if (firstarr[i] < secondarr[j]) {
			arr[k] = firstarr[i];
			i++;
			step += 2;
		}
		else {
			arr[k] = secondarr[j];
			j++;
			step += 2;
		}
		k++;
		step++;
	}
	while (i < m - s + 1) {
		arr[k] = firstarr[i];
		i++; k++;
		step += 3;
	}
	while (j < l - m) {
		arr[k] = secondarr[j];
		j++; k++;
		step += 3;
	}

	delete[] firstarr;
	delete[] secondarr;
	return step;
}
```

```
int mergeSort(int arr[], int s, int l) {
	int step = 1;
	if (s >= l)
		return step;
	int mid = (s + l) / 2;
	step += mergeSort(arr, s, mid);
	step += mergeSort(arr, mid + 1, l);
	step += merge(arr, s, mid, l);
	return step;
}
```

Part 2B,D:

Creating the main function which reads n items using another function from the file generated and executes the merge algorithm on 10, 100, 1000 and 10000 elements and writes a file that includes pairs of n and clock time it took (the full generated txt is in the GitHub repository linked below):

```
int main() {
	int arr[LENGTH];
	createRandFile();
	readFile(arr, LENGTH);

	ofstream sFile("clockFile.txt");
	int x[LENGTH];

	for (int i = 10; i <= 10000; i *= 10) {
		for (int j = 0; j < i; j++) {
			x[j] = arr[j];
		}
		clock_t time = clock();
		mergeSort(x, 0, i - 1);
		sFile << i << ',' << (float)(clock() - time) / CLOCKS_PER_SEC << endl;
	}

	system("pause");
	return 0;
}
```

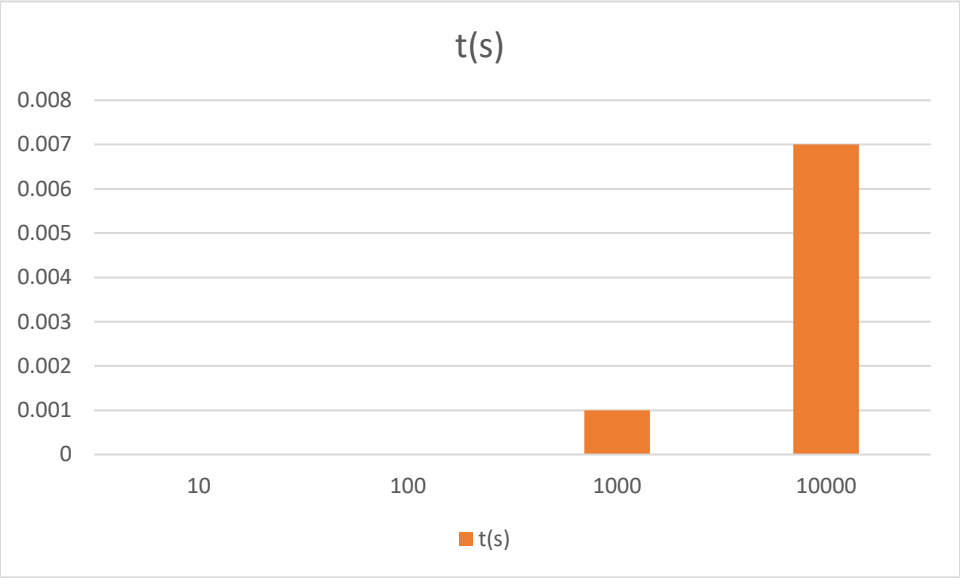
Part 2C:

Writing a C++ function to generate 10,000 random numbers between 1 and 100 and save them in a file (the full generated txt is in the GitHub repository linked below):

```
void createRandFile() {
	ofstream mfile("unsortedFile.txt");
	srand(time(0));

	for (int i = 0; i < LENGTH; i++) {
		mfile << ((rand() % 100) + 1) << endl;
	}
}
```

Part 3A:

The "clockFile.txt" created in the main function is then imported into excel and a graph is generated from the data. (the full generated excel is in the GitHub repository linked below):



Part 3B:

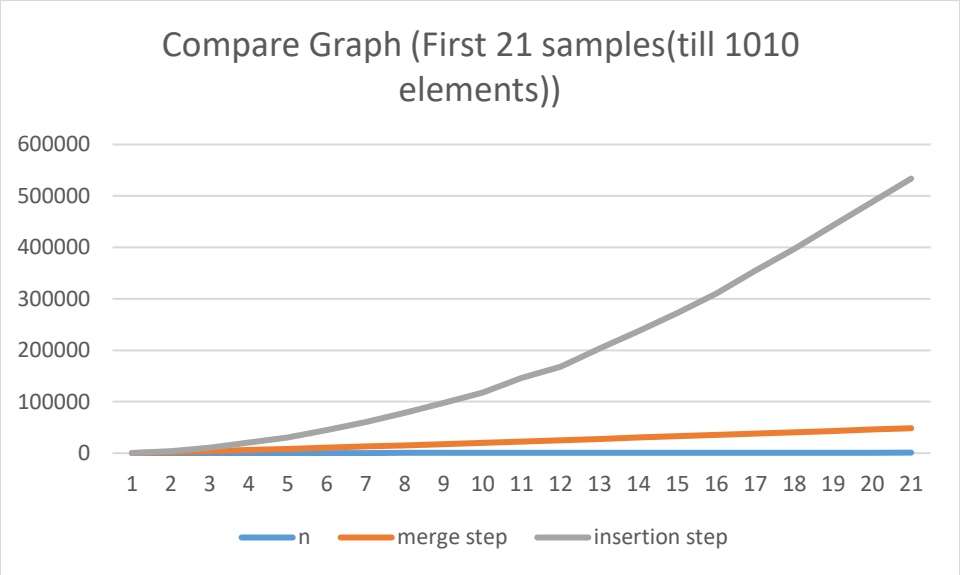
The difference in time isn't clear between small numbers like 10 and 100 but when 1000 and 10000 are added, they are shown a little better. However, it's still not that long of a time as when 10000 elements are getting sorted takes 0.007 seconds.

Part 3C:

To compare the results of the merge sort to the insertion sort the createRandomFile has to be edited to create numbers between 0 and 10000. In addition to that we have to calculate the steps with the same step 50 as the insertion sort and generate a txt file which is then added to the excel, as well as add the remaining createSortedFile function. (The full generated files are in the GitHub repository linked below):

```
void createRandFile() {
	ofstream mfile("unsortedFile.txt");
	srand(time(0));

	for (int i = 0; i < LENGTH; i++) {
		mfile << ((rand() % LENGTH) + 1) << endl;
	}
}
```

```
void readFile(int arr[], int l) {
	ifstream mfile("unsortedFile.txt");
	for (int i = 0; i < l; i++) {
		mfile >> arr[i];
	}
}

void createSortedFile(int arr[]) {
	ofstream mfile("sortedFile.txt");

	for (int i = 0; i < LENGTH; i++) {
		mfile << arr[i] << endl;
	}
}

int main() {
	int arr[LENGTH];
	createRandFile();
	readFile(arr, LENGTH);

	ofstream sFile("clockFile.txt");
	int x[LENGTH];

	for (int i = 10; i <= 10000; i *= 10) {
		for (int j = 0; j < i; j++) {
			x[j] = arr[j];
		}
		clock_t time = clock();
		mergeSort(x, 0, i - 1);
		sFile << i << ',' << (float)(clock() - time) / CLOCKS_PER_SEC << endl;
	}

	ofstream bFile("stepFile.txt");
	int y[LENGTH];

	for (int i = 10; i < 10000; i += 50) {
		for (int j = 0; j < i; j++) {
			y[j] = arr[j];
		}
		bFile << i << ',' << mergeSort(y, 0, i - 1) << endl;
	}

	bFile << 10000 << ',' << mergeSort(arr, 0, LENGTH - 1) << endl;

	createSortedFile(arr);
	system("pause");
	return 0;
}
```



It is noticeable that the merge sort takes way less steps to sort the same number of elements.

GitHub Repository:
https://github.com/Anthony-Amgad/CSE331MergeSort19P9880