

# Assignment 3

Francisco Hernandez

fjhernan@uark.edu

ID: 010882940

Late days used: 0

Late Days Left: 4/5

I seem repetitive saying this each report (I will continue to do so until the end of semester), but to state the obvious: applying the algorithms into actual code really does help give a visual and solid understanding. I feel like I just went through my own enlightenment era.

This project really did a toll on me and I wanted it to end (the assignment or me) halfway through. The Heap and Quick sort algorithms were a bit more straight forward, but the merge sent me through some loops (get it).

## Code

As always, the MakeFile made me love life. Using the respective additions to make to run each necessary sort algorithm was the best, and made me look into make files a bit. Really interesting, will make some basic ones for my own little projects.

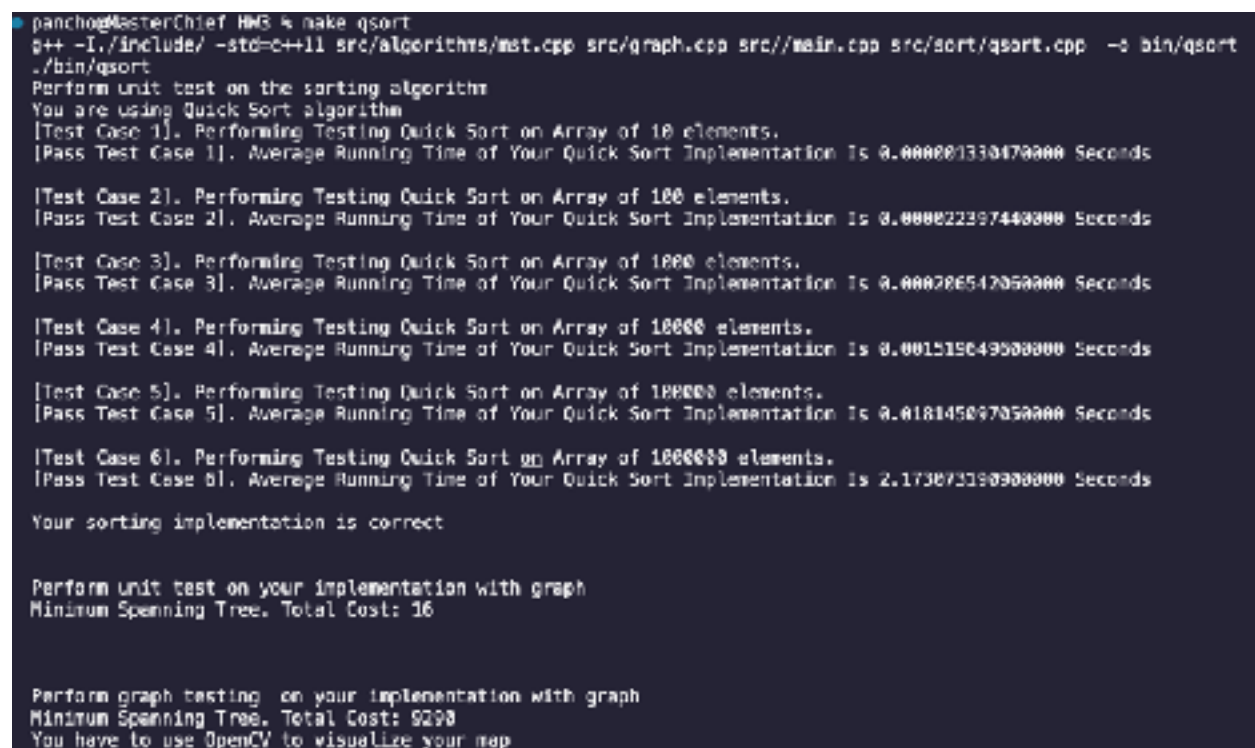
The first problems I get usually in all of my homework has been segmentation fault 11s. But now I am getting more used to them, and I find the problem quicker. It happened trying to implement my quick sort, but I quickly saw that I accidentally used 'j <= r' instead of simply '<', and it was trying to access a memory it did not have access to. Easy fix.

I had a problem and I did not realize it until the end and I stared at my screen for hours. Until it was that I was using exit(0); instead of return;. That little mistake made me cry and drink a Monster Zero (they are so good). Until I decided

to try that based on the fact that I had nothing to lose... and I thought my implementation was correct. It was that, it worked, I cried.

The error that really tickled my patience was in merge sort. I spent an embarrassing amount of time trying to make the merge() function from the Algorithm cpp library that was brought in from the graph.hpp header file. I thought that was what we were supposed to. Until I gave up of seeing so many lines of error that I could build a bridge across the Grand Canyon, and I decided to try to figure out how to make my own merge function. I re read the slides from lectures, and I honestly pulled old reliable [geeksforgeeks.com](http://www.geeksforgeeks.com) and [stackoverflow.com](http://stackoverflow.com) to glue in the missing pieces of my understanding. I managed to, and it worked! Thank the heavens! Our usual, I will attack some screenshots of each sort running successfully, with their respective captions below their images.

**Figure 3.1**



```
pancho@MasterChief: HW3 % make qsort
g++ -I./include/ -std=c++11 src/algorithms/mst.cpp src/graph.cpp src/main.cpp src/sort/qsart.cpp -o bin/qsart
./bin/qsart
Perform unit test on the sorting algorithm
You are using Quick Sort algorithm
[Test Case 1]. Performing Testing Quick Sort on Array of 10 elements.
[Pass Test Case 1]. Average Running Time of Your Quick Sort Implementation Is 0.000001330470000 Seconds

[Test Case 2]. Performing Testing Quick Sort on Array of 100 elements.
[Pass Test Case 2]. Average Running Time of Your Quick Sort Implementation Is 0.000022397440000 Seconds

[Test Case 3]. Performing Testing Quick Sort on Array of 1000 elements.
[Pass Test Case 3]. Average Running Time of Your Quick Sort Implementation Is 0.000286542050000 Seconds

[Test Case 4]. Performing Testing Quick Sort on Array of 10000 elements.
[Pass Test Case 4]. Average Running Time of Your Quick Sort Implementation Is 0.001515649500000 Seconds

[Test Case 5]. Performing Testing Quick Sort on Array of 100000 elements.
[Pass Test Case 5]. Average Running Time of Your Quick Sort Implementation Is 0.018145097050000 Seconds

[Test Case 6]. Performing Testing Quick Sort on Array of 1000000 elements.
[Pass Test Case 6]. Average Running Time of Your Quick Sort Implementation Is 2.173073390000000 Seconds

Your sorting implementation is correct

Perform unit test on your implementation with graph
Minimum Spanning Tree. Total Cost: 16

Perform graph testing on your implementation with graph
Minimum Spanning Tree. Total Cost: 9290
You have to use OpenCV to visualize your map
```

In figure 1.1 we can see that the quick sort (sort) algorithm is working correctly, as it passed the tests pre-written in the main file by Dr. Khoa Luu and TAs.

Figure 3.2

```
g++ -I./include/ -std=c++11 src/algorithms/test.cpp src/graph.cpp src/main.cpp src/sort/msort.cpp -o bin/msort
./bin/msort
Perform unit test on the sorting algorithm
You are using Merge Sort algorithm
[Test Case 1]. Performing Testing Merge Sort on Array of 10 elements.
[Pass Test Case 1]. Average Running Time of Your Merge Sort Implementation Is 0.000003554170000 Seconds

[Test Case 2]. Performing Testing Merge Sort on Array of 100 elements.
[Pass Test Case 2]. Average Running Time of Your Merge Sort Implementation Is 0.000050799950000 Seconds

[Test Case 3]. Performing Testing Merge Sort on Array of 1000 elements.
[Pass Test Case 3]. Average Running Time of Your Merge Sort Implementation Is 0.000691717430000 Seconds

[Test Case 4]. Performing Testing Merge Sort on Array of 10000 elements.
[Pass Test Case 4]. Average Running Time of Your Merge Sort Implementation Is 0.006002477530000 Seconds

[Test Case 5]. Performing Testing Merge Sort on Array of 100000 elements.
[Pass Test Case 5]. Average Running Time of Your Merge Sort Implementation Is 0.060063279170000 Seconds

[Test Case 6]. Performing Testing Merge Sort on Array of 1000000 elements.
[Pass Test Case 6]. Average Running Time of Your Merge Sort Implementation Is 7.755971004300000 Seconds

Your sorting implementation is correct

Perform unit test on your implementation with graph
Minimum Spanning Tree. Total Cost: 16

Perform graph testing on your implementation with graph
Minimum Spanning Tree. Total Cost: 9298
You have to use OpenCV to visualize your map
```

Same as last caption, but for the merge sort (msort).

Figure 3.3

```
g++ -I./include/ -std=c++11 src/algorithms/test.cpp src/graph.cpp src/main.cpp src/sort/hsort.cpp -o bin/hsort
./bin/hsort
Perform unit test on the sorting algorithm
You are using Heap Sort algorithm
[Test Case 1]. Performing Testing Heap Sort on Array of 10 elements.
[Pass Test Case 1]. Average Running Time of Your Heap Sort Implementation Is 0.000002182320000 Seconds

[Test Case 2]. Performing Testing Heap Sort on Array of 100 elements.
[Pass Test Case 2]. Average Running Time of Your Heap Sort Implementation Is 0.000027502900000 Seconds

[Test Case 3]. Performing Testing Heap Sort on Array of 1000 elements.
[Pass Test Case 3]. Average Running Time of Your Heap Sort Implementation Is 0.000317332970000 Seconds

[Test Case 4]. Performing Testing Heap Sort on Array of 10000 elements.
[Pass Test Case 4]. Average Running Time of Your Heap Sort Implementation Is 0.002341424500000 Seconds

[Test Case 5]. Performing Testing Heap Sort on Array of 100000 elements.
[Pass Test Case 5]. Average Running Time of Your Heap Sort Implementation Is 0.020092829900000 Seconds

[Test Case 6]. Performing Testing Heap Sort on Array of 1000000 elements.
[Pass Test Case 6]. Average Running Time of Your Heap Sort Implementation Is 3.511692245200001 Seconds

Your sorting implementation is correct

Perform unit test on your implementation with graph
Minimum Spanning Tree. Total Cost: 16

Perform graph testing on your implementation with graph
Minimum Spanning Tree. Total Cost: 9298
You have to use OpenCV to visualize your map
```

This is the heap sort (hsort).