CMP201 - Presentation

By Joseph Roper - 1901881

What problem am I solving?

I am trying to find the most efficient and useful path finding algorithm.

I want to be able too use this algorithm in a tower defence game for the CPU logic.

I will be testing the Lee Algorithm and the Dijkstra Algorithm.

I chose the Lee Algorithm as it is simple and therefore hopefully faster too compute.

I chose the Dijkstra Algorithm as it is more complex as it allows for sideways movement on a grid resulting in smarter CPU's in my game.

Data structures used in Lee Algorithm

My implementation of the Lee algorithm makes use of queues, arrays and structs to function. I chose these three data structures as they have the same time complexity as each other, O(1).

This theoretically should mean that my program computes at the same speed each time it is run regardless of the input size.

In my program I use a queue to store the different vertexes of the map, the queue is then used to calculate the possible movements available to the current vertex.

In my algorithm I use a struct which holds the x and y coordinates of the cell as well as the shortest distance from the source vertex. I used in conjunction with the queue allowing the program too function.

Data structures used in Dijkstra Algorithm

In my Dijkstra Program I have mainly used vector arrays which have a time complexity of O(1), meaning they will always compute at the same speed.

This is good as I am wanting to increase my input data during the tests, technically this should not effect the timings of my program.

I use vector arrays too hold the map information such as what cells have been visited and the coordinates and distance away from the starting point of those cells.

Lee Algorithm Performance

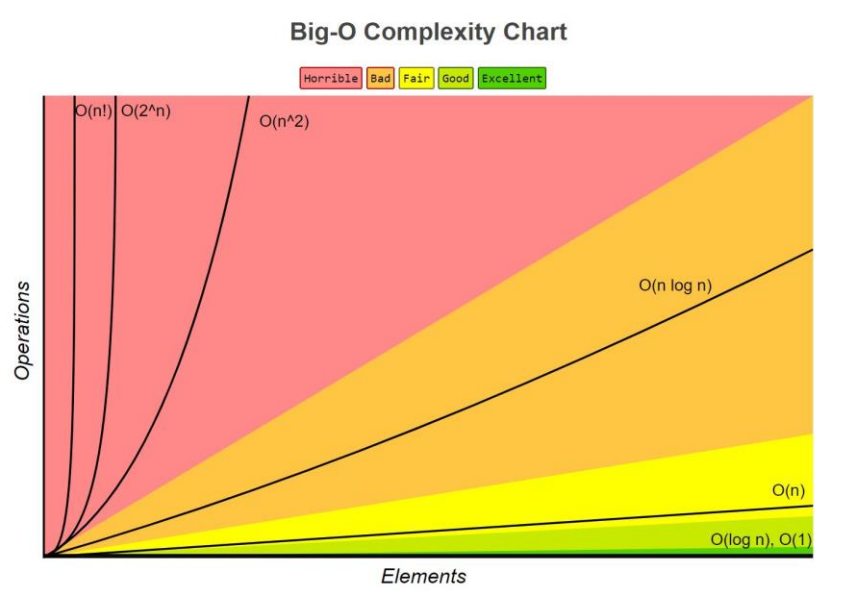
The theoretical time complexity of the Lee Algorithm is O(N). This is a linear time complexity meaning it will take longer the larger the input data.

For Example:

1 item: 1 second

10 items: 10 seconds

100 items: 100 seconds



This matches the times I received from my code. As the size of the input data increases so does times.

Lee Algorithm

9x9 =

Median: 3 (ms)

Mean: 2.6 (ms)

Sum: 13 (ms)

10x10 =

Median: 3 (ms)

Mean: 5.6 (ms)

Sum: 28 (ms)

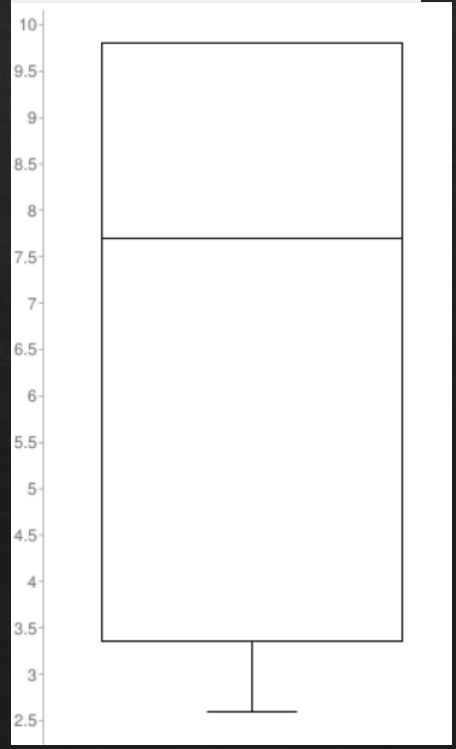
11x11 =

Median: 3 (ms)

Mean: 9.8 (ms)

Sum: 49 (ms)

Median: 7.7
Minimum: 2.6
Maximum: 9.8
First quartile: 3.35
Third quartile: 9.8
Interquartile Range: 6.45



Dijkstra Algorithm Performance

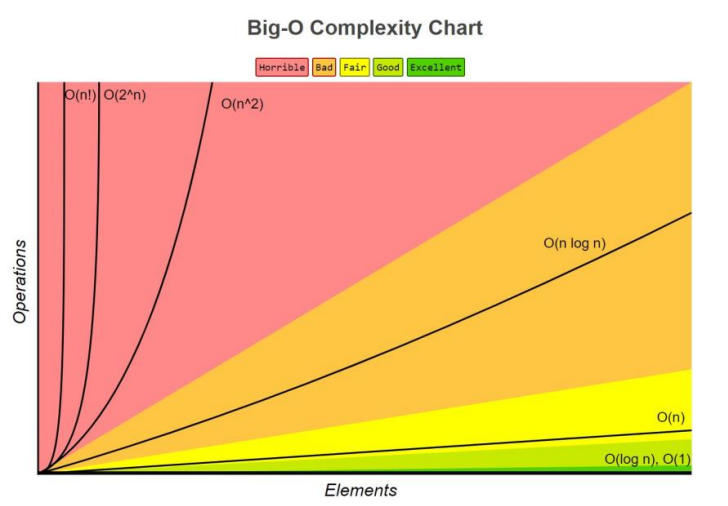
The theoretical time complexity of the Dijkstra Algorithm is $O(Nn^2)$. This is a quadratic time complexity meaning that it is proportional to the input size squared. Meaning it should be a lot slower than the lee algorithm.

For Example:

1 item: 1 second

10 items: 100 seconds

100 items: 10000 seconds



This matches the times I received from my code. As the size of the input data increases so does times.

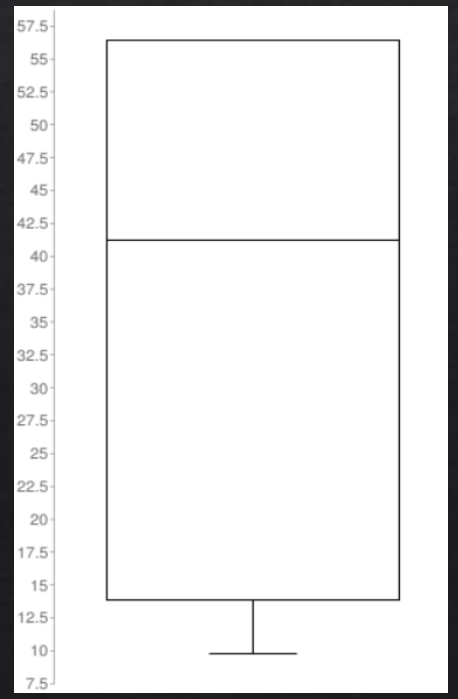
Dijkstra Algorithm

9x9 =
Median: 6 (ms)
Mean: 9.8(ms)
Sum: 49 (ms)

10x10 =
Median: 6 (ms)
Mean: 26 (ms)
Sum: 130 (ms)

11x11 =
Median: 6 (ms)
Mean: 56.4 (ms)
Sum: 282 (ms)

Median: 41.2
Minimum: 9.8
Maximum: 56.4
First quartile: 13.85
Third quartile: 56.4
Interquartile Range: 42.55



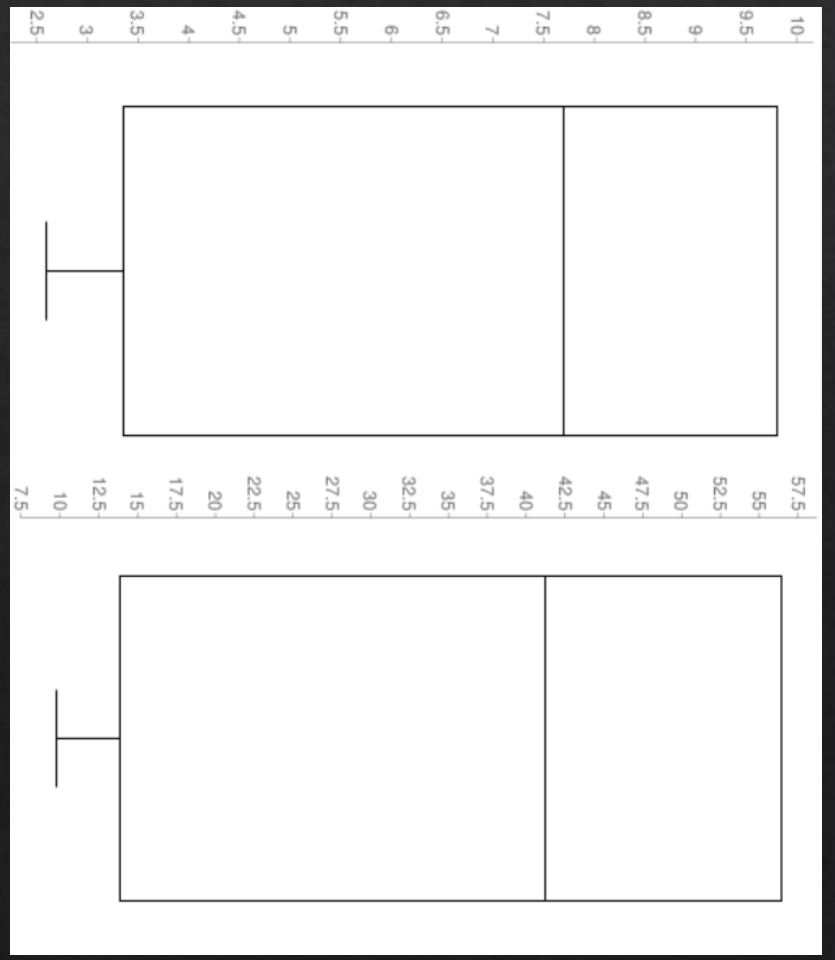
Algorithm Performance Comparison

- It is clear that the lee algorithm is faster than the Dijkstra algorithm as you can see from the results.

Lee Algorithm	Dijkstra Algorithm
9x9 =	9x9 =
Median: 3 (ms)	Median: 6 (ms)
Mean: 2.6 (ms)	Mean: 9.8(ms)
Sum: 13 (ms)	Sum: 49 (ms)
10x10 =	10x10 =
Median: 3 (ms)	Median: 6 (ms)
Mean: 5.6 (ms)	Mean: 26 (ms)
Sum: 28 (ms)	Sum: 130 (ms)
11x11 =	11x11 =
Median: 3 (ms)	Median: 6 (ms)
Mean: 9.8 (ms)	Mean: 56.4 (ms)
Sum: 49 (ms)	Sum: 282 (ms)

Median: 7.7
Minimum: 2.6
Maximum: 9.8
First quartile: 3.35
Third quartile: 9.8
Interquartile Range: 6.45

Median: 41.2
Minimum: 9.8
Maximum: 56.4
First quartile: 13.85
Third quartile: 56.4
Interquartile Range: 42.55



Overview

Even thought the lee algorithm is clearly faster I am still going to choose the Dijkstra algorithm as my final choice as it computes the data in seconds, meaning there is nothing to worry about when being implemented into my game. Also the movement capabilities on the map is much greater with the Dijkstra algorithm as it allows diagonal movement which will give me more options for map design with CPU movement in mind.