DATA STRUCTURES AND ALGORITHMS

By Callum Leonard

THE PROBLEM I AM SOLVING.

- The problem I am solving is to implement two different standard algorithms that traverse a tree structure efficiently and quickly.

- After the algorithms are implemented and working as intended algorithm 1's timing measurements will be directly compared against algorithm 2's to determine which algorithm is faster at traversing through a graph related tree structure.

THE 2 ALGORITHM'S IMPLEMENTED

- BFS (Breadth First Search)
  - For BFS std::queue was used to represent the FIFO data structure required for this algorithm.
  - I represented an adjacency list using a vector as a container for the node edges, as it directly resizes its elements based on the total nodes within the graph.
    - The time complexity for all containers will be O(N) proportional to the number of node edges needed to resize to)
  - Using std::vector allowed for sufficient element space for node edges to be inputted using the data structure std::list.
  - Std::list was used to represent the edge-linking data structure
    - It can push_back into the graph in O(1) constant time, providing the best time complexity possible.
  - Another consideration was a 2D vector, to represent an adjacency list with vector push_back also having a time complexity of O(1).
  - Std::set was also considered for the edge-linking data structure, however std::set has an insert time complexity of O(log N) making the operation to pushback or insert into the graph more expensive than std::list (this would slow down the algorithm computation speed)
- DFS (Depth First Search)
  - For DFS std::stack was used to represent the LIFO data structure required for this algorithm.
  - I implemented the same adjacency list as the BFS algorithm and made the same considerations regarding memory complexity.
  - Although DFS is different, the same considerations applied in BFS also affect DFS in a positive way (computation speed)

PERFORMANCE PROFILING

- A performance profiler was used, and CPU sampling approach was also applied to make sure the program was running as intended, and that no unforeseen bottlenecks existed that could have impacted performance in a negative way.

- For performance profiling the program was executed in release mode which allowed for the computer to compile and execute the code as fast possible.

THE ALGORITHM COMPLEXITY

- I expect the algorithm complexity of BFS to be O(V+E) where V is the vertices/nodes and E is the node edges.
  - In other words the time complexity will be proportional to the number of nodes in the graph.

- I expect the algorithm complexity of DFS to be O(V+E) where V is the number of vertices/nodes and E is the node edges.
  - In other words the time complexity will be proportional to the number of nodes in the graph.

ACCURATELY MEASURING PERFORMANCE OF EACH ALGORITM

- To accurately measure each algorithm's performance and test our theoretical hypothesis both algorithms were tested by timing 11 repetitions after each successful execution.
- Sorting the repetitions into order and computing the median.

- This was done for 3 different varied sizes of node inputs, each varied size of input being the number of nodes/vertices in the graph.
- Repetition timings were in debug mode and not release mode.
- With the repetition size of 11 it produces a more accurate result relating to the overall complexity of the algorithm and the median timings.

ACCURATELY MEASURING PERFORMANCE OF EACH ALGORITM CONTINUED

- Once all median timings for the algorithm were collected, they were directly reviewed to understand whether the O(N) we theorised was correct.
- The medians gathered were then placed onto a line graph for visualisation.
  - With the line graph a better understanding can be gained on execution time versus node input, this can also provide a better view of the time complexity of the algorithm.
- A box plot was also implemented to get more in-depth information regarding the algorithm.
  - The box plot shows useful information clearly such as median, maximum, minimum and outliers.
  - This can be useful to build a broader picture of how the algorithm computes.
- The same steps regarding profiling, timing, graph visualisation and box plots will be repeated with the other algorithm (DFS).
- Both algorithm median timing results will be directly compared against each other in hopes to discover which algorithm is faster in traversing all nodes within a tree structure, based on node input and data structures specified.

CPU SAMPLING (PERFORMANCE PROFILER)

- CPU sampling was applied and the application executed in release mode.
- The suspected bottleneck is relating to the std::cout after each node is popped from the queue or stack.
- As the node input size is increased the std::cout operation will become considerably more expensive and will impact performance in a negative way.
- Hot code: Analysing pieces of code the profiler has identified as the most performance critical.
- *Hotcode in main source.cpp (BFS) – Expected as this is the function call to the algorithm itself*

```
2 (9.09%)    94        graph.BFSAlgorithm(nodeI, visited, graph);
```

- *Hotcode in bfs.cpp – Expected as this is the cout statement relating to the node pop*

```
2 (9.09%)    41            std::cout << "| " << nodeI << " ";
```

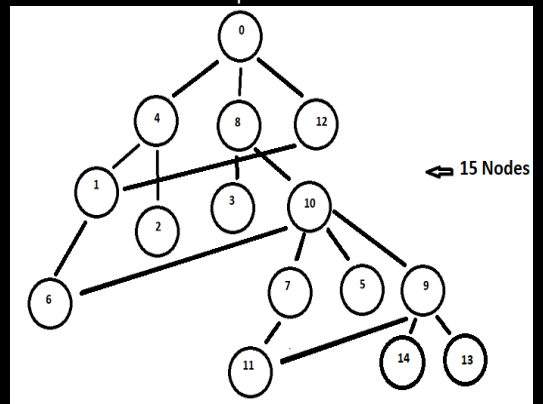
- Both hot code pointers identified by the performance profiler were what was expected.
- There was no other unforeseen bottlenecks/impacts of performance discovered.

MORE DETAIL ON NODE INPUT SIZES

- Node input sizes were 15,30 and 60.
- For the original starting node input size (15), the size selected was arbitrary.
- Based on the theorised complexity of the algorithm O(V+E) | O(N) linear it was reasonable for the node input to be doubled.
- This provided a simple way to validate whether the algorithm was O(N) based on the median timings by looking at whether the median timings for 15,30 and 60 nodes were also doubling in execution time (ms).

BFS REPETITIONS WITH 15 AS NODE INPUT SIZE

- Custom Graph Visualisation:



Expected BFS Output:

0,4,8,12,1,2,3,10,6,7,5,9,11,14,13

Actual algorithm output and first repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 9 ms to compute the algorithm.
```

2nd repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 11 ms to compute the algorithm.
```

3rd repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 5 ms to compute the algorithm.
```

4th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 11 ms to compute the algorithm.
```

5th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 9 ms to compute the algorithm.
```

6th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 10 ms to compute the algorithm.
```

7th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 8 ms to compute the algorithm.
```

8th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 10 ms to compute the algorithm.
```

9th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 7 ms to compute the algorithm.
```

10th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 8 ms to compute the algorithm.
```

11th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 |
It took 10 ms to compute the algorithm.
```

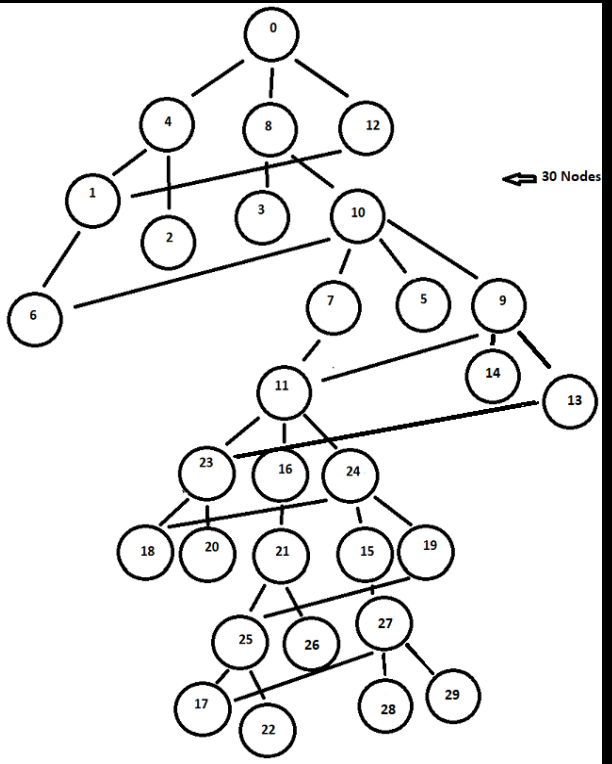
Sorting repetitions into order:

5ms | 7ms| 8ms | 8ms | 9ms | 9ms | 10ms | 10ms | 10ms | 11ms | 11ms

The median execution time for the node input size of 15 is (BFS): 9ms

BFS REPETITIONS WITH 30 AS NODE INPUT SIZE

- Custom Graph Visualisation:



Expected BFS Output:
0,4,8,12,1,2,3,10,6,7,5,9,11,14,13,23,16,24,18,20,21,
15,19,25,26,27,17,22,28,29

Actual algorithm output and first repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 22 ms to compute the algorithm.
```

2nd repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 20 ms to compute the algorithm.
```

3rd repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 16 ms to compute the algorithm.
```

4th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 23 ms to compute the algorithm.
```

5th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 22 ms to compute the algorithm.
```

6th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 19 ms to compute the algorithm.
```

7th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 15 ms to compute the algorithm.
```

8th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 22 ms to compute the algorithm.
```

9th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 19 ms to compute the algorithm.
```

10th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 22 ms to compute the algorithm.
```

11th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 28 | 29 |
It took 21 ms to compute the algorithm.
```

Sorting repetitions into order:

15ms | 16ms| 19ms | 19ms | 20ms | 21ms | 22ms | 22ms | 22ms | 22ms | 23ms

The median execution time for the node input size of 30 is (BFS): 21ms

BFS REPETITIONS WITH 60 AS NODE INPUT SIZE

- Custom Graph Visualisation:

Expected BFS Output:
0,4,8,12,1,2,3,10,6,7,5,9,11,14,13,23,16,24,18,20,21,15,19,25,26,27,17,22,28,29,26,32,36,48,39,29,42,43,50,30,44,28,51,40,46,35,27,38,31,41,33,34,49,37,47,45,54,56,52,58,57,53,55,59

Actual algorithm output and first repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19 | 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 |
| 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43 | 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 35 ms to compute the algorithm.
```

2nd repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 41 ms to compute the algorithm.
```

3rd repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 47 ms to compute the algorithm.
```

4th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 46 ms to compute the algorithm.
```

5th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 37 ms to compute the algorithm.
```

6th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 45 ms to compute the algorithm.
```

7th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 49 ms to compute the algorithm.
```

8th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 47 ms to compute the algorithm.
```

9th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 46 ms to compute the algorithm.
```

10th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 39 ms to compute the algorithm.
```

11th repetition:

```
Visited Nodes: | 0 | 4 | 8 | 12 | 1 | 2 | 3 | 10 | 6 | 7 | 5 | 9 | 11 | 14 | 13 | 23 | 16 | 24 | 18 | 20 | 21 | 15 | 19
| 25 | 26 | 27 | 17 | 22 | 29 | 48 | 39 | 28 | 44 | 31 | 41 | 32 | 36 | 51 | 30 | 46 | 35 | 38 | 37 | 47 | 45 | 42 | 43
| 50 | 40 | 33 | 34 | 49 | 54 | 56 | 52 | 58 | 57 | 53 | 55 | 59 |
It took 41 ms to compute the algorithm.
```

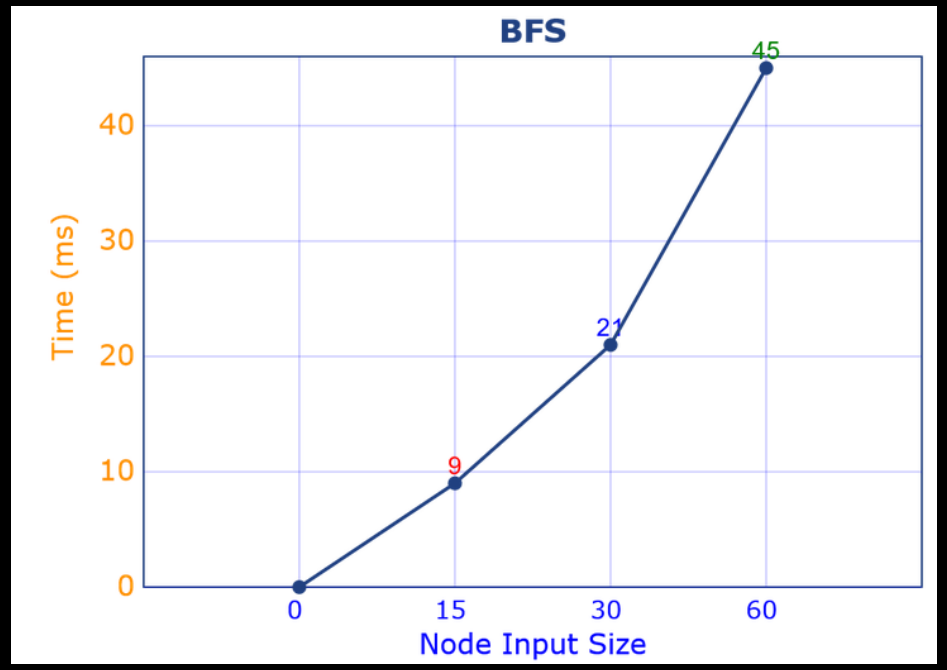
Sorting repetitions into order:

35ms | 37ms| 39ms | 41ms | 41ms | 45ms | 46ms | 46ms | 47ms | 47ms | 49ms

The median execution time for the node input size of 60 is (BFS): 45ms

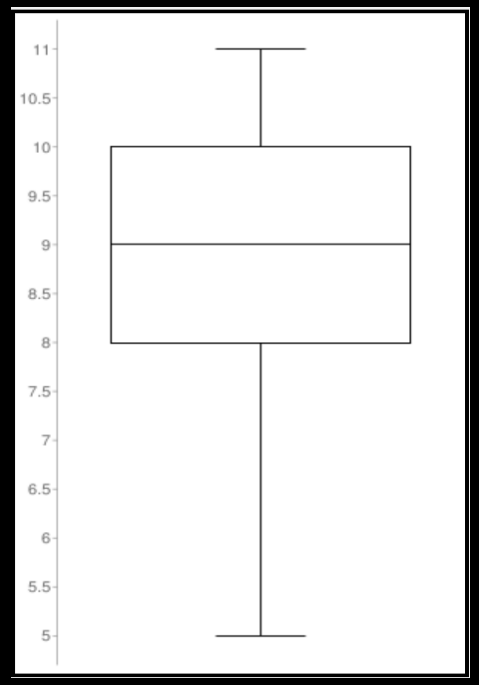
BFS LINE CHART VISUALISATION

- The line chart visualisation shows accurate O(N) visuality for node input sizes 15 and 30, however appears to increase past O(N) and almost to O(N log N) for a node size of 60.
- However, considering the previous 2 node sizes and the medians 9, 21. A node input size of 60 with a 3ms addition to the expected time 21x2ms (42ms) is insignificant, especially when node input sizes are increased and also when other system processes are considered at the time of execution.
- The medians of 9, 21 and 45ms indicate our hypothesis was correct and that BFS O(V+E), O(N) linear time complexity was correct.

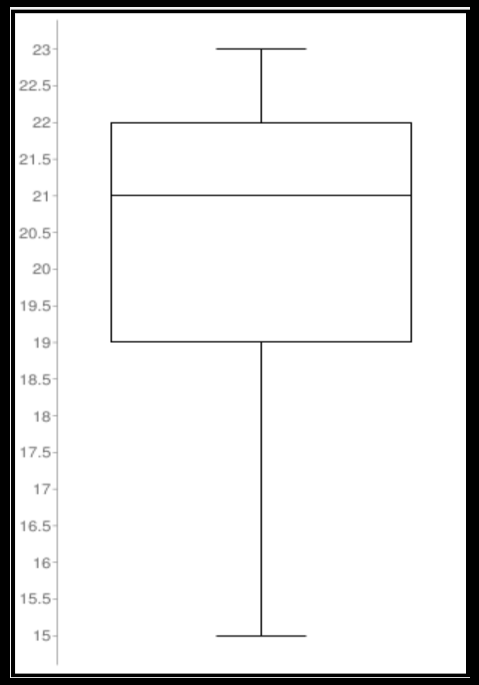


BFS BOX PLOT VISUALISATION 15,30 AND 60 NODES

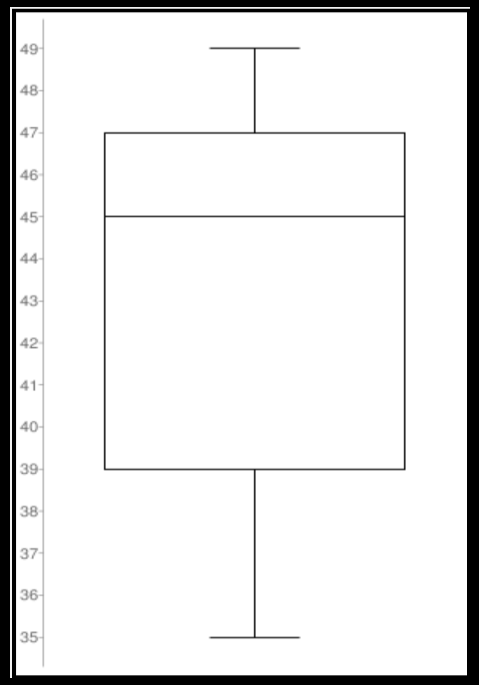
- 15 Nodes



30 Nodes:

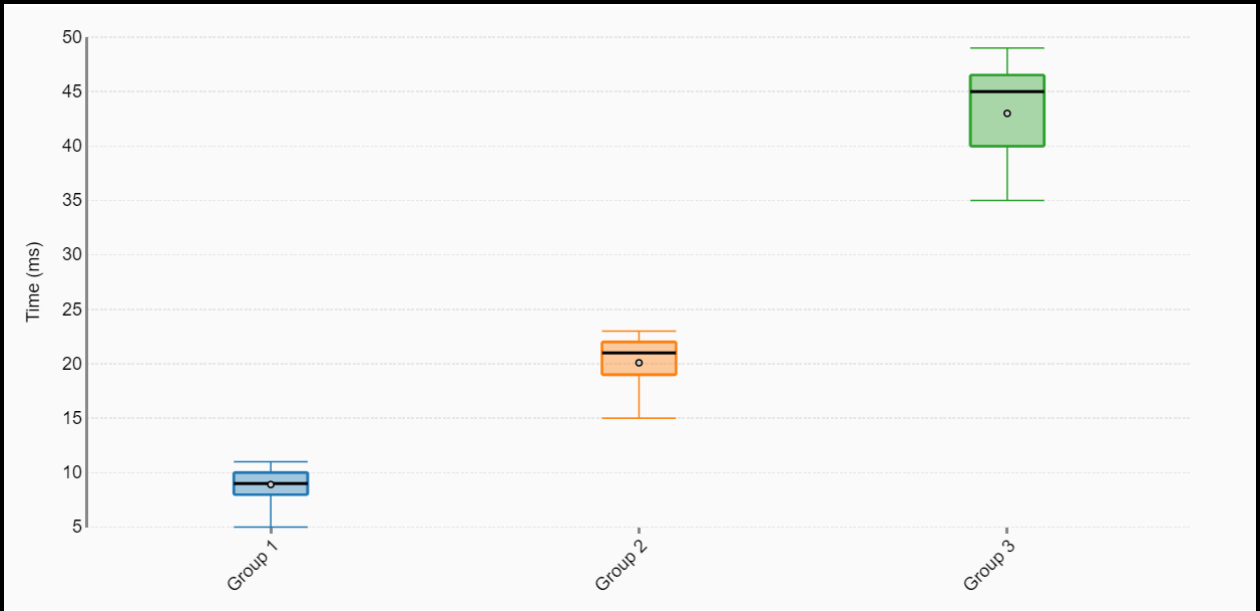


60 Nodes:



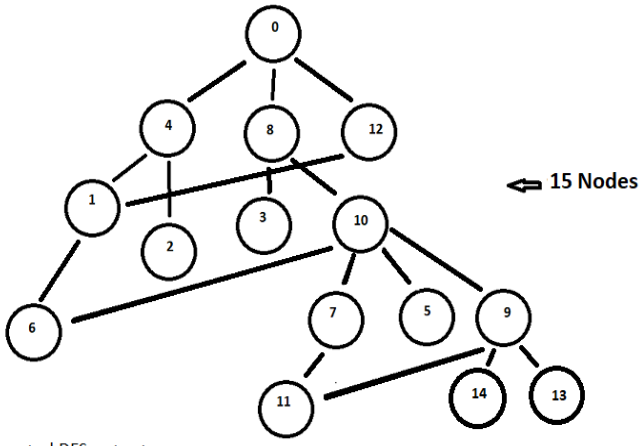
BFS MULTI BOX PLOT VISUALISATION

- By taking the individual (singular) box plots we can then construct a multi box plot graph (multiple box plots contained within a single graph). This can give us a more concrete understanding of our algorithm, and again point towards its overall time complexity.
- Key: Y axis time(ms) | X axis Group1: 15 nodes | Group2: 30 nodes | Group3: 60 nodes
- By looking at the positioning of the box plots alone relative to the graph it is clear that the graph is O(N). We can also easily look at each box plots minimum, maximum and median to give us a broader understanding of how the algorithm operates based on different node input sizes.



DFS REPETITIONS WITH 15 AS NODE INPUT SIZE

- Custom Graph Visualisation:



Expected DFS output:

0,4,1,6,10,8,3,7,11,9,14,13,5,12,2

Actual algorithm output and first repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 10 ms to compute the algorithm.
```

2nd repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 6 ms to compute the algorithm.
```

3rd repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 8 ms to compute the algorithm.
```

4th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 8 ms to compute the algorithm.
```

5th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 9 ms to compute the algorithm.
```

6th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 12 ms to compute the algorithm.
```

7th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 7 ms to compute the algorithm.
```

8th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 9 ms to compute the algorithm.
```

9th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 10 ms to compute the algorithm.
```

10th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 8 ms to compute the algorithm.
```

11th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 9 | 14 | 13 | 5 | 12 | 2 |
It took 10 ms to compute the algorithm.
```

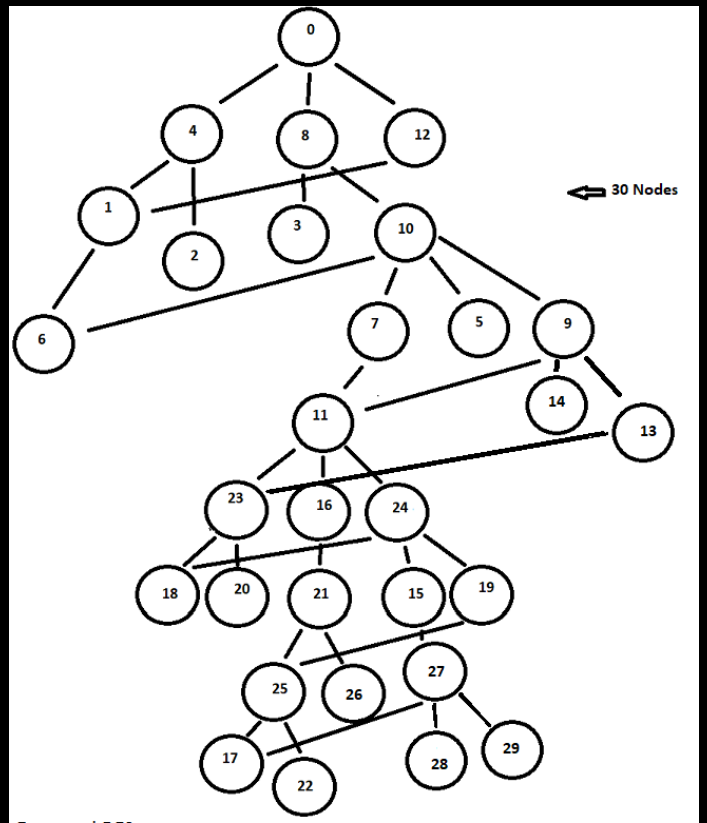
Sorting repetitions into order:

6ms | 7ms| 8ms | 8ms | 8ms | 9ms | 9ms | 10ms | 10ms | 10ms | 12ms

The median execution time for the node input size of 15 is (DFS): 9ms

DFS REPETITIONS WITH 30 AS NODE INPUT SIZE

- Custom Graph Visualisation:



Expected DFS output:
0,4,1,6,10,8,3,7,11,23,18,24,15,27,17,25,21,16,26,22,19,28,29,20,13,9,14,5,12,2

Actual algorithm output and first repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
 29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 23 ms to compute the algorithm.
```

2nd repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 19 ms to compute the algorithm.
```

3rd repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 23 ms to compute the algorithm.
```

4th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 20 ms to compute the algorithm.
```

5th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 20 ms to compute the algorithm.
```

6th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 19 ms to compute the algorithm.
```

7th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 13 ms to compute the algorithm.
```

8th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 20 ms to compute the algorithm.
```

9th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 17 ms to compute the algorithm.
```

10th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 19 ms to compute the algorithm.
```

11th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 19 | 28 |
29 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 20 ms to compute the algorithm.
```

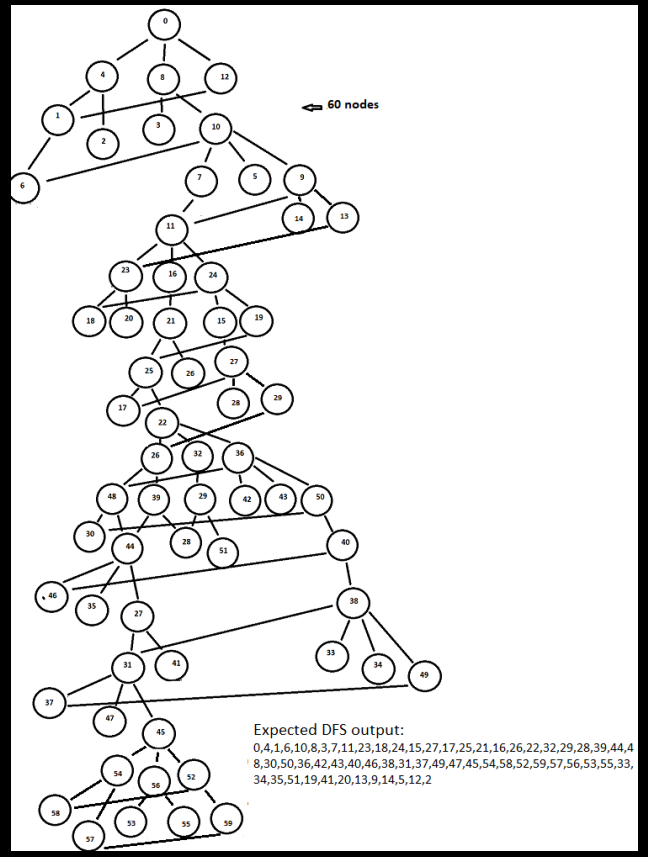
Sorting repetitions into order:

13ms | 17ms| 19ms | 19ms | 19ms | 20ms | 20ms | 20ms | 20ms | 23ms | 23ms

The median execution time for the node input size of 30 is (DFS): 20ms

DFS REPETITIONS WITH 60 AS NODE INPUT SIZE

- Custom Graph Visualisation:



Actual algorithm output and first repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 49 ms to compute the algorithm.
```

2nd repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 44 ms to compute the algorithm.
```

3rd repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 33 ms to compute the algorithm.
```

4th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 44 ms to compute the algorithm.
```

5th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 39 ms to compute the algorithm.
```

6th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 40 ms to compute the algorithm.
```

7th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 36 ms to compute the algorithm.
```

8th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 44 ms to compute the algorithm.
```

9th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 39 ms to compute the algorithm.
```

10th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 43 ms to compute the algorithm.
```

11th repetition:

```
Visited Nodes: | 0 | 4 | 1 | 6 | 10 | 8 | 3 | 7 | 11 | 23 | 18 | 24 | 15 | 27 | 17 | 25 | 21 | 16 | 26 | 22 | 32 | 29 |
28 | 39 | 44 | 48 | 30 | 50 | 36 | 42 | 43 | 40 | 46 | 38 | 31 | 37 | 49 | 47 | 45 | 54 | 58 | 52 | 59 | 57 | 56 | 53 |
55 | 33 | 34 | 35 | 51 | 19 | 41 | 20 | 13 | 9 | 14 | 5 | 12 | 2 |
It took 42 ms to compute the algorithm.
```

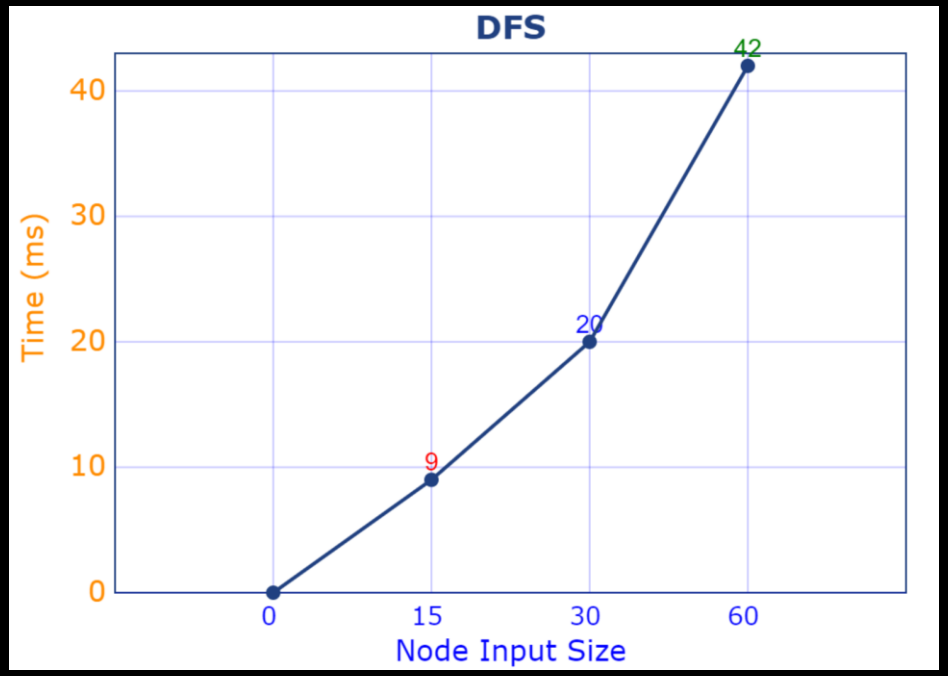
Sorting repetitions into order:

33ms | 36ms| 39ms | 39ms | 40ms | 42ms | 43ms | 44ms | 44ms | 44ms | 49ms

The median execution time for the node input size of 60 is (DFS): 42ms

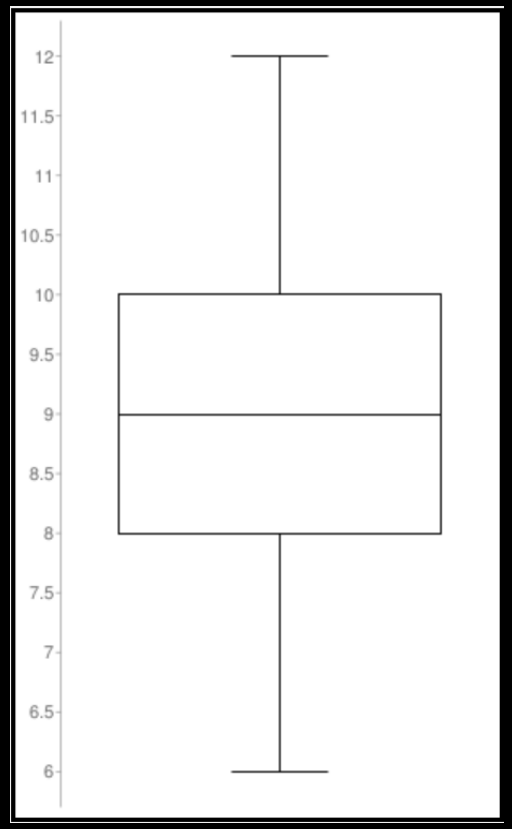
DFS LINE CHART VISUALISATION

- The line chart visualisation shows accurate O(N) visuality for node input sizes 15 and 30, however appears to increase past O(N) and almost to O(N log N) for a node size of 60.
- However, considering the previous 2 node sizes and the medians 9, 20. A node input size of 60 with a 2ms addition to the expected time 20x2ms (42ms) is insignificant, especially when node input sizes are increased and also when other system processes are considered at the time of execution.
- The medians of 9, 20 and 42ms indicate our hypothesis was correct and that BFS O(V+E), O(N) linear time complexity was correct.

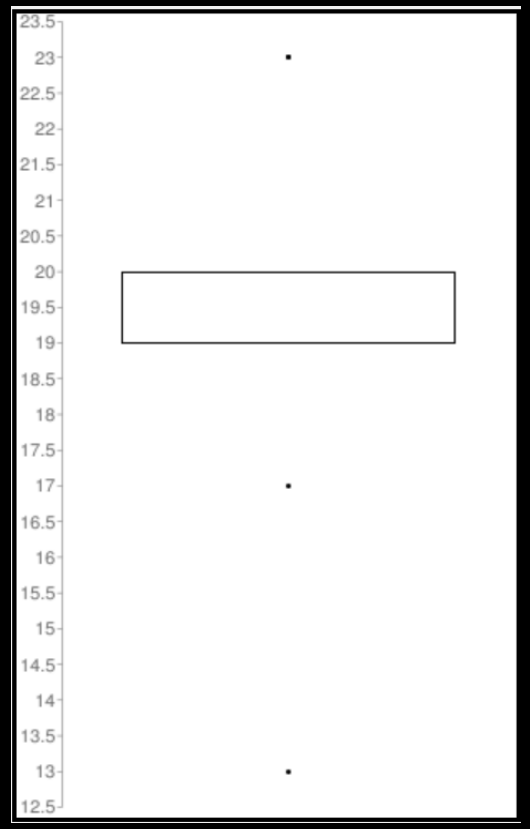


DFS BOX PLOT VISUALISATION 15,30 AND 60 NODES

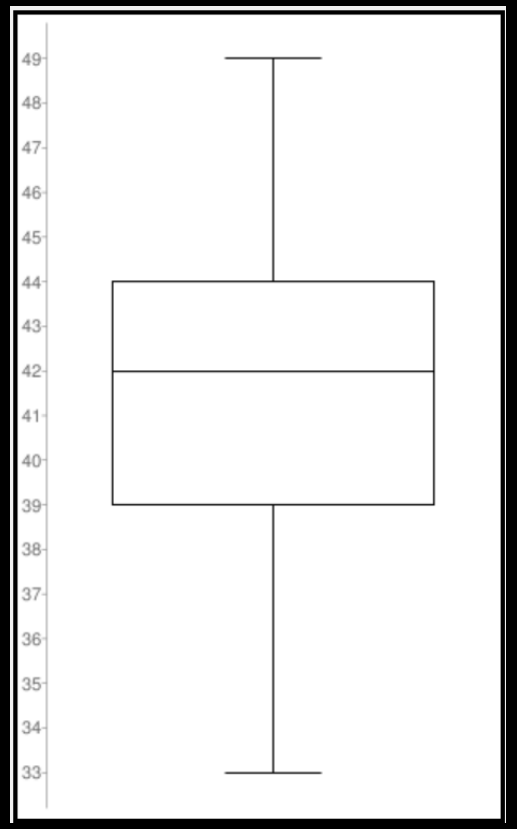
- 15 Nodes



30 Nodes

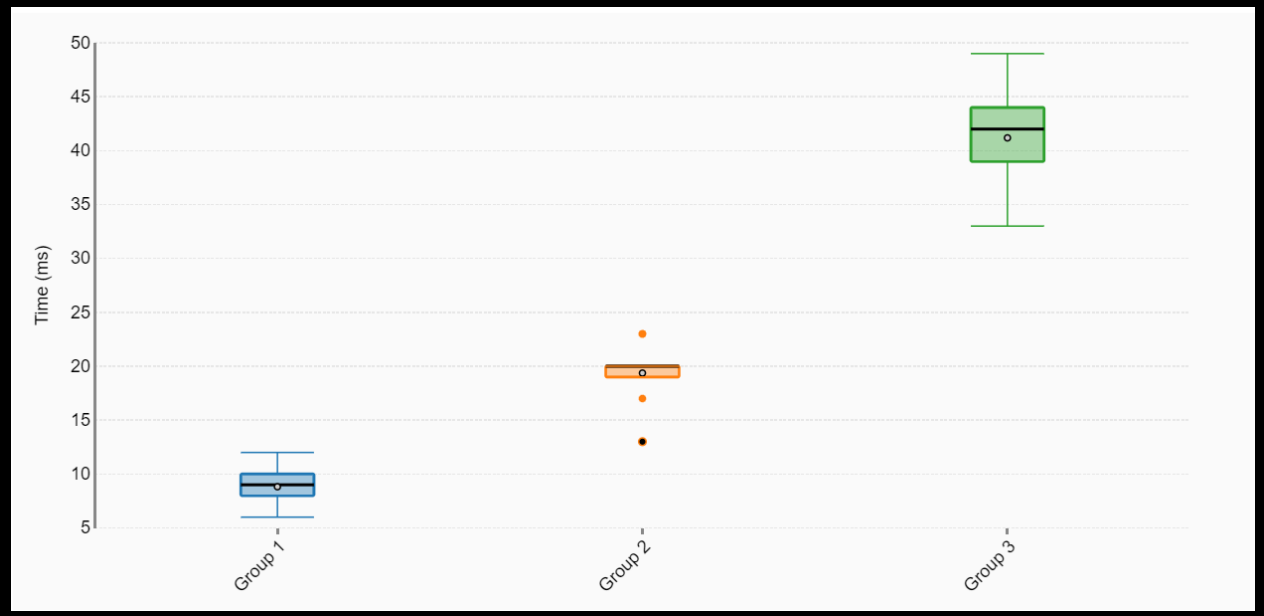


60 Nodes



DFS MULTI BOX PLOT VISUALISATION

- By taking the individual (singular) box plots we can then construct a multi box plot graph (multiple box plots contained within a single graph). This can give us a more concrete understanding of our algorithm, and again point towards its overall time complexity.
- Key: Y axis time(ms) | X axis Group1: 15 nodes | Group2: 30 nodes | Group3: 60 nodes
- By looking at the positioning of the box plots alone relative to the graph it is clear that the graph is O(N). We can also easily look at each box plots minimum, maximum and median to give us a broader understanding of how the algorithm operates based on different node input sizes.

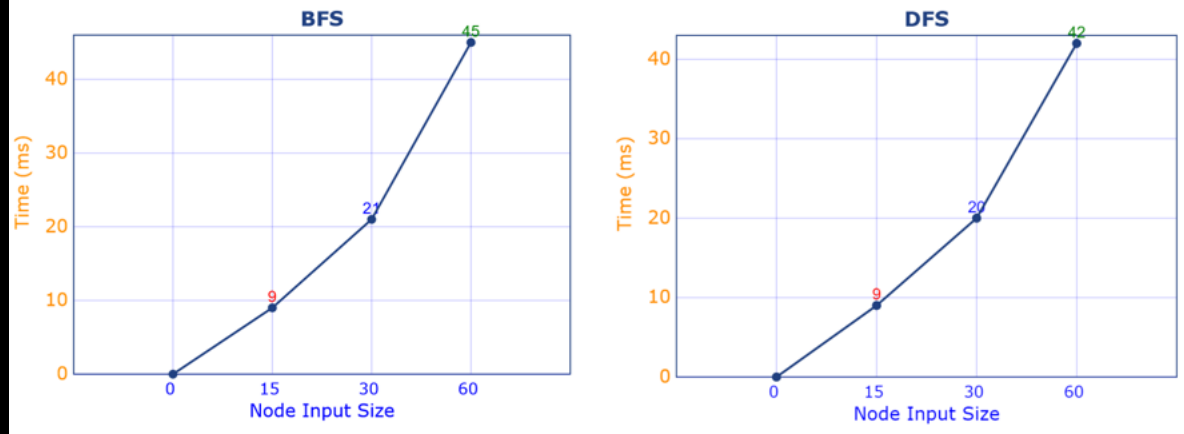


DIRECT COMPARISONS

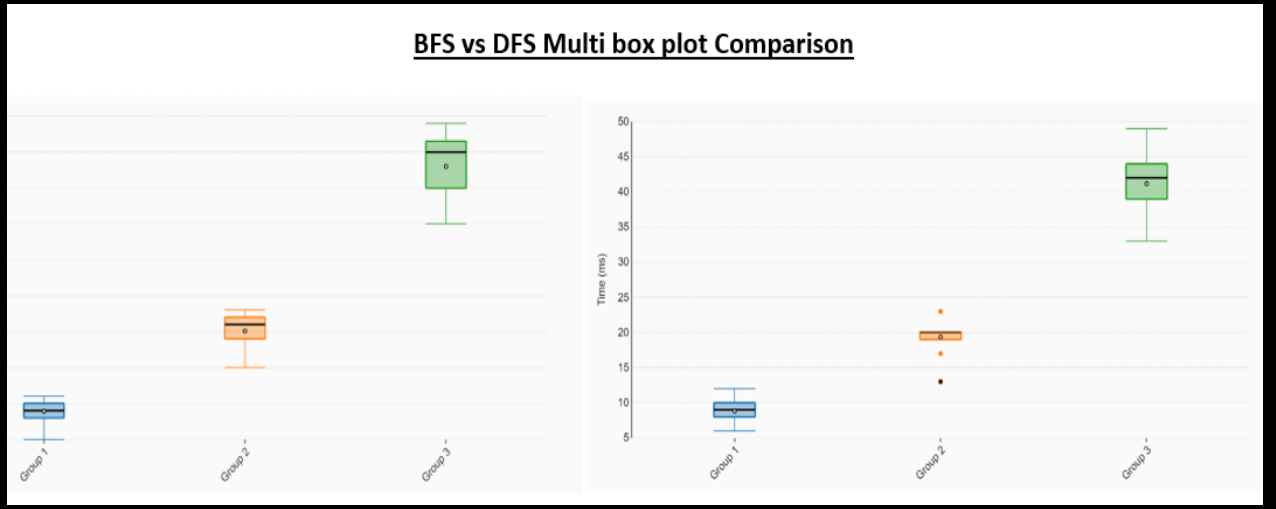
BFS vs DFS Median Timing Direct Comparisons

Node Input Size	15	30	60
BFS	9(ms)	21(ms)	45(ms)
DFS	9(ms)	20(ms)	42(ms)
Difference (ms)	0(ms)	1(ms)	3(ms)

BFS vs DFS Line Graph Comparisons



BFS vs DFS Multi box plot Comparison



CONCLUSION

- Looking at all the successful benchmarking results captured, our original hypothesis regarding O(V+E) O(N) for BFS and DFS is correct.
- By analysing the direct median outputs, the line chart graphs and the box plots there is a clear double in the computation time (ms) for each double in node input size.
- BFS is O(V+E) O(N) || DFS is O(V+E) O(N)