MULTIMEDIA UNIVERSITY®

TCP2101 - ALGORITHM DESIGN AND ANALYSIS

Class Section: TC02/ TT04

Members:

1. Choo Jia Sheng (1142700814)
2. Ng Kang Jie (1142700809)

Question 3:

Perform a comparative analysis of Depth-First Search (DFS) between two implementations of Graph ADT: adjacency matrix and adjacency list.

Introduction

At first, we decided to follow the depth first search implementation as shown in Lab06 tutorial question, where the node is initialized from randomizing undirected relationship nodes. We chose undirected relationship nodes over directed relationship nodes because the approach gives more consistent results in multiple runs and it produces lesser amount of independent nodes. Moreover, the approach would also show more significant difference in results when we are comparing the sparse and dense case.

Unfortunately, 2D matrix with the size of 100,000 consumed a huge amount of memory, which is approximately 10GB. For a smoother program flow, we decided to follow our lecturer's advice and reduce the size to 10,000. For density, we have 6 cases based on the algorithm below:

Type of density	Relational	Non-Relational
Best	-	All
Sparse	10%	90%
Average	50%	50%
Dense	90%	10%
Worst	All	-
Random	Random	Random

Command to Compile the Files

```
g++ -std=c++11 main.cpp
```

Flows of Program for Test Node

1. Create and randomize matrix.
2. Print matrix.
3. Create list with matrix.
4. Print list.
5. Create node with list.
6. Run DFS with list nodes.
7. Print connected nodes.
8. Form DFS List.
9. Print DFS List.

Flows of Program for Runtime Calculation

1. Create and randomize matrix.

2. Create list with matrix.

3. Start clock for matrix running time.

4. Create node with matrix.

5. Run DFS with matrix nodes.

6. End and calculate running time for matrix.

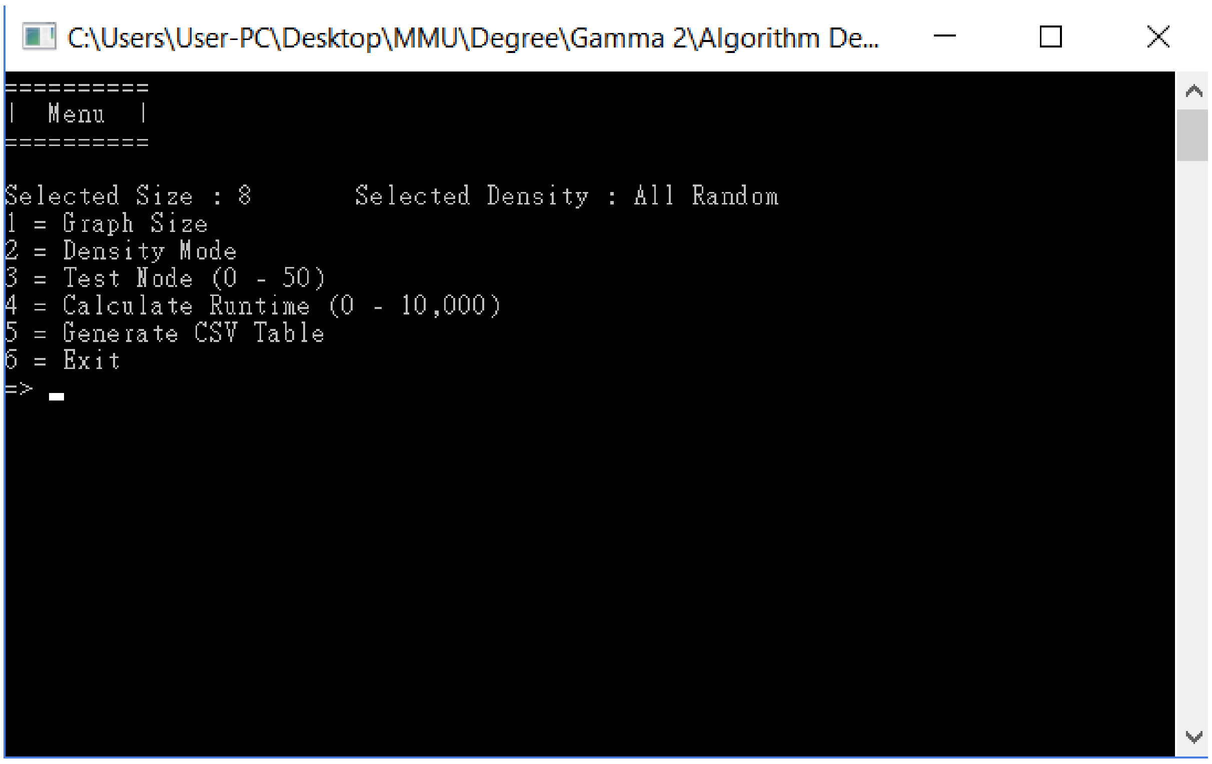
7. Start clock for list running time.

8. Create node with list.

9. Run DFS with list nodes.

10. End and calculate running time for list.

How to Use the Program



```
C:\Users\User-PC\Desktop\MMU\Degree\Gamma 2\Algorithm De...
==========
|  Menu  |
==========

Selected Size : 8       Selected Density : All Random
1 = Graph Size
2 = Density Mode
3 = Test Node (0 - 50)
4 = Calculate Runtime (0 - 10,000)
5 = Generate CSV Table
6 = Exit
=> _
```

1. Main menu contains 6 options where the first and second option (which are size and density) are to modify the produced data set and the selected options will be shown on top of main menu.

2. Test node has a limit of 0 to 50 due to the constraint in size of CMD console. It will display all the results including matrix, list, connected nodes and the list of possible Depth First Search.

3. Calculate Runtime will only show the time required for the machine to convert matrix or list to nodes and perform Depth First Search.

4. Generate CSV Table will prompt user to select a range of data to calculate the running time. It will be saved under .csv file format so that the data could be visualized into graph if needed.

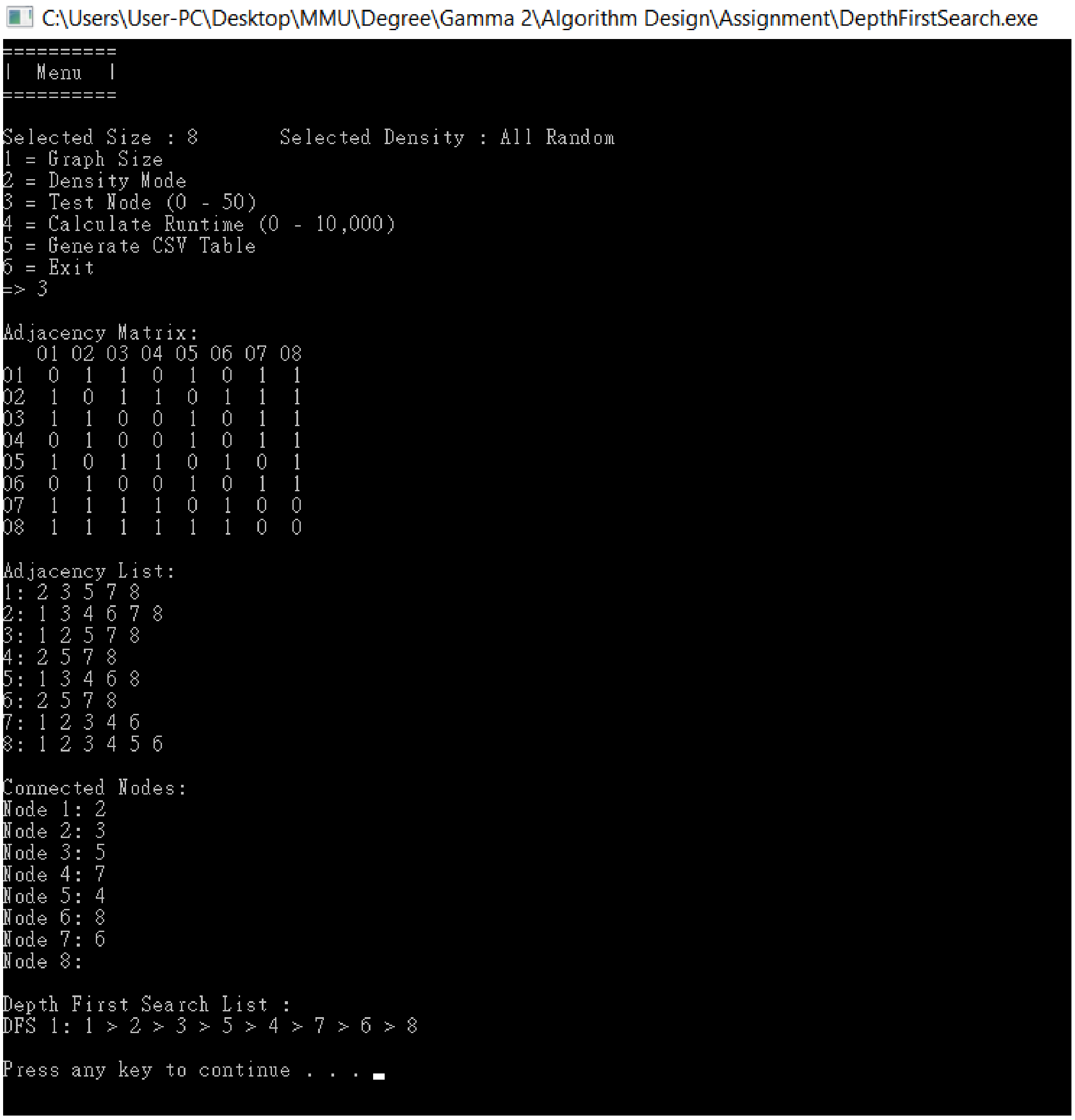
Specification of Computer/ Laptop to Run the Test

CPU : Intel i5-5200U 2.20GHz

OS : Window 10 64-bit

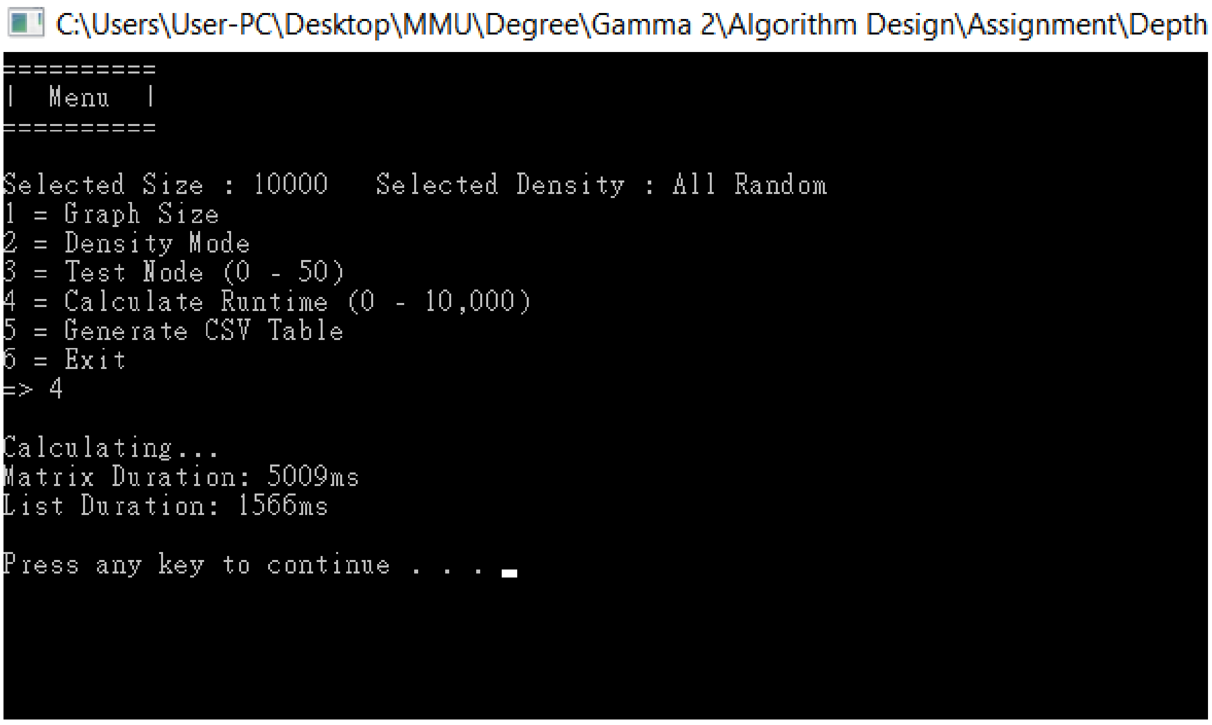
RAM : 8.00GB

Example of 8 Nodes Testing



```
C:\Users\User-PC\Desktop\MMU\Degree\Gamma 2\Algorithm Design\Assignment\DepthFirstSearch.exe
==========
|  Menu  |
==========

Selected Size : 8       Selected Density : All Random
1 = Graph Size
2 = Density Mode
3 = Test Node (0 - 50)
4 = Calculate Runtime (0 - 10,000)
5 = Generate CSV Table
6 = Exit
=> 3

Adjacency Matrix:
   01 02 03 04 05 06 07 08
01  0  1  1  0  1  0  1  1
02  1  0  1  1  0  1  1  1
03  1  1  0  0  1  0  1  1
04  0  1  0  0  1  0  1  1
05  1  0  1  1  0  1  0  1
06  0  1  0  0  1  0  1  1
07  1  1  1  1  0  1  0  0
08  1  1  1  1  1  1  0  0

Adjacency List:
1: 2 3 5 7 8
2: 1 3 4 6 7 8
3: 1 2 5 7 8
4: 2 5 7 8
5: 1 3 4 6 8
6: 2 5 7 8
7: 1 2 3 4 6
8: 1 2 3 4 5 6

Connected Nodes:
Node 1: 2
Node 2: 3
Node 3: 5
Node 4: 7
Node 5: 4
Node 6: 8
Node 7: 6
Node 8:

Depth First Search List :
DFS 1: 1 > 2 > 3 > 5 > 4 > 7 > 6 > 8

Press any key to continue . . .
```

Example of 10,000 Nodes Running Time Test



C:\Users\User-PC\Desktop\MMU\Degree\Gamma 2\Algorithm Design\Assignment\Depth

```
==========
|  Menu  |
==========

Selected Size : 10000   Selected Density : All Random
1 = Graph Size
2 = Density Mode
3 = Test Node (0 - 50)
4 = Calculate Runtime (0 - 10,000)
5 = Generate CSV Table
6 = Exit
=> 4

Calculating...
Matrix Duration: 5009ms
List Duration: 1566ms

Press any key to continue . . .
```

Visualization

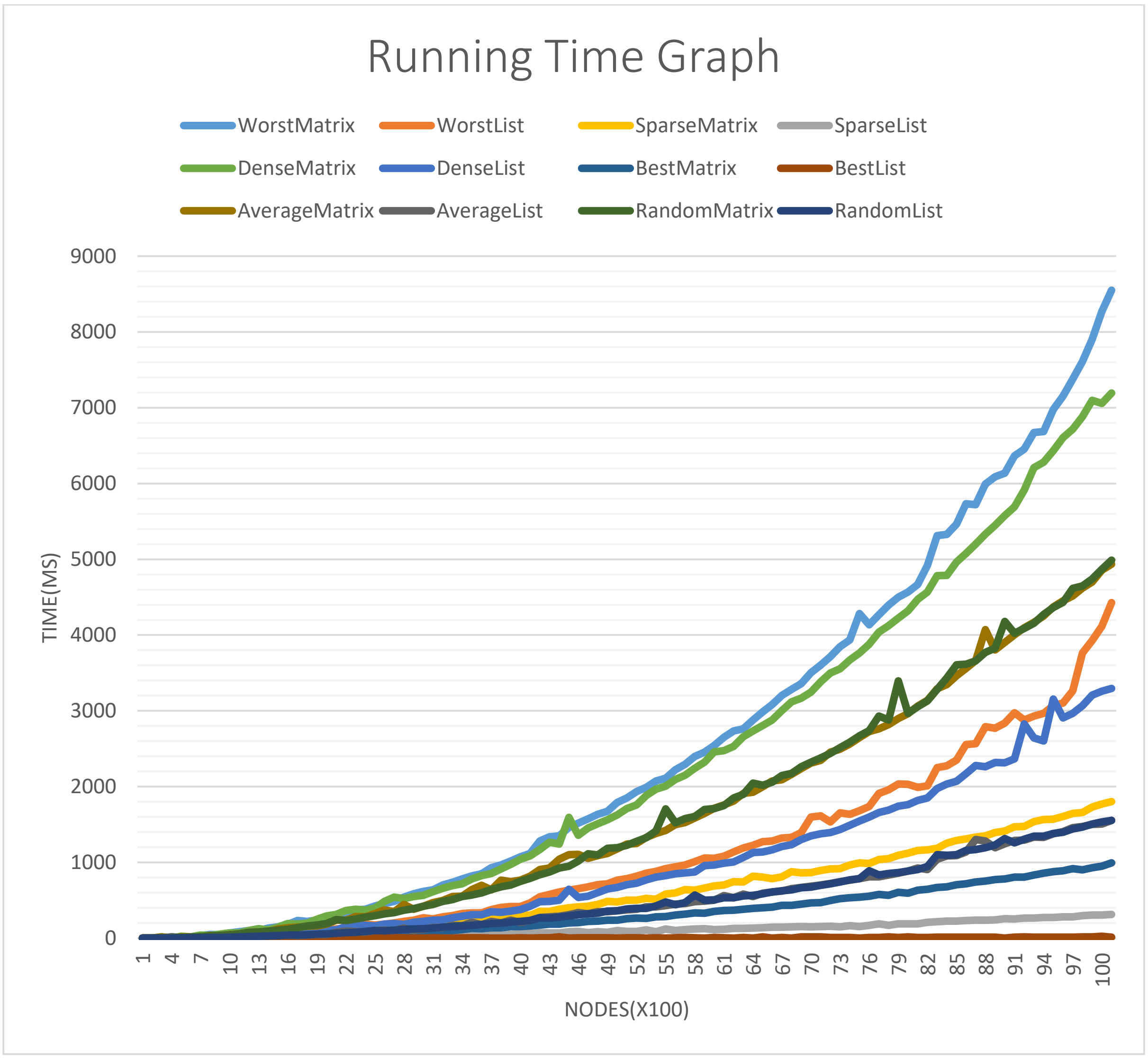


Diagram 1: Overall Graph Comparison

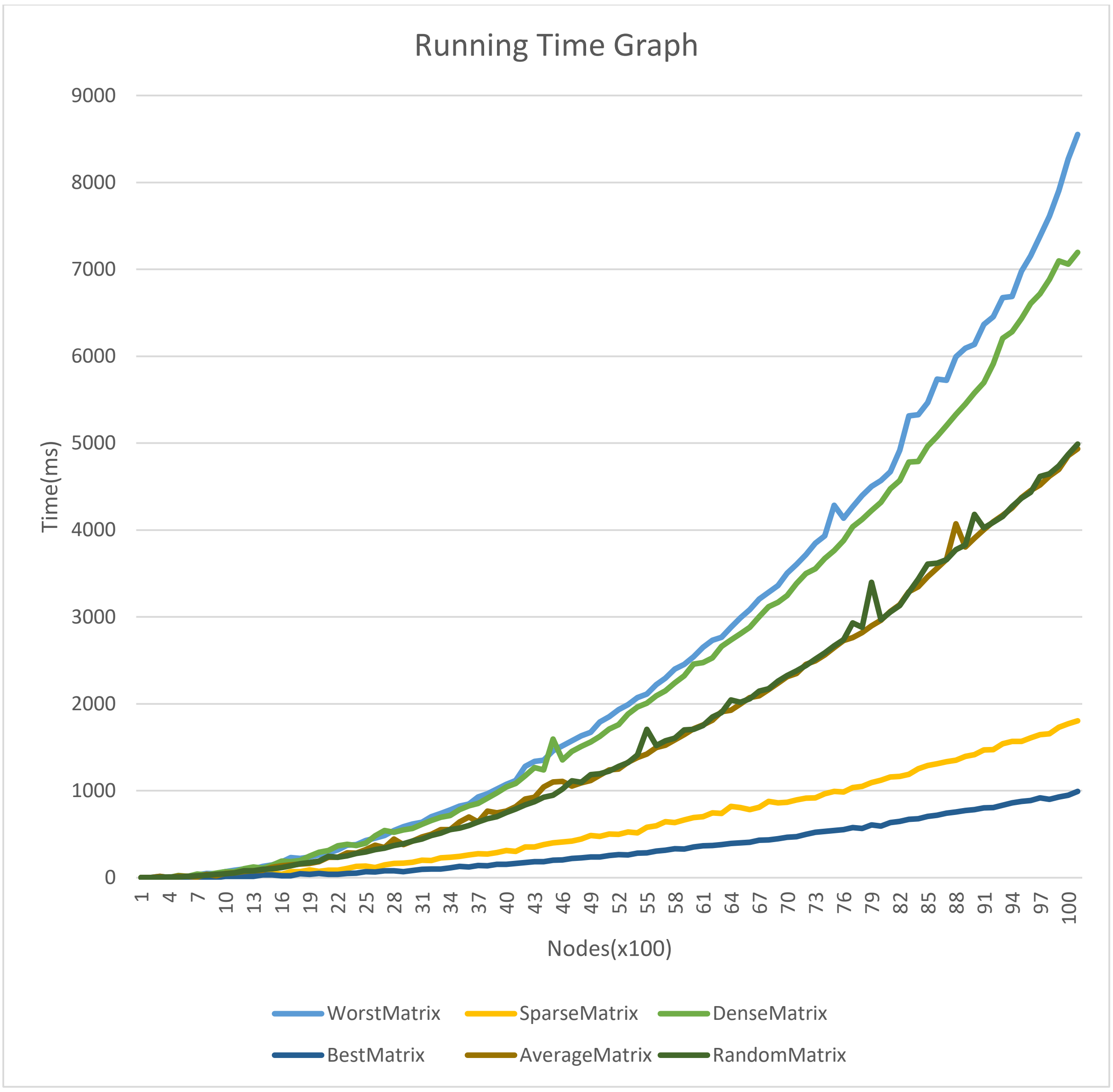


Diagram 2: Matrix Graph Comparison

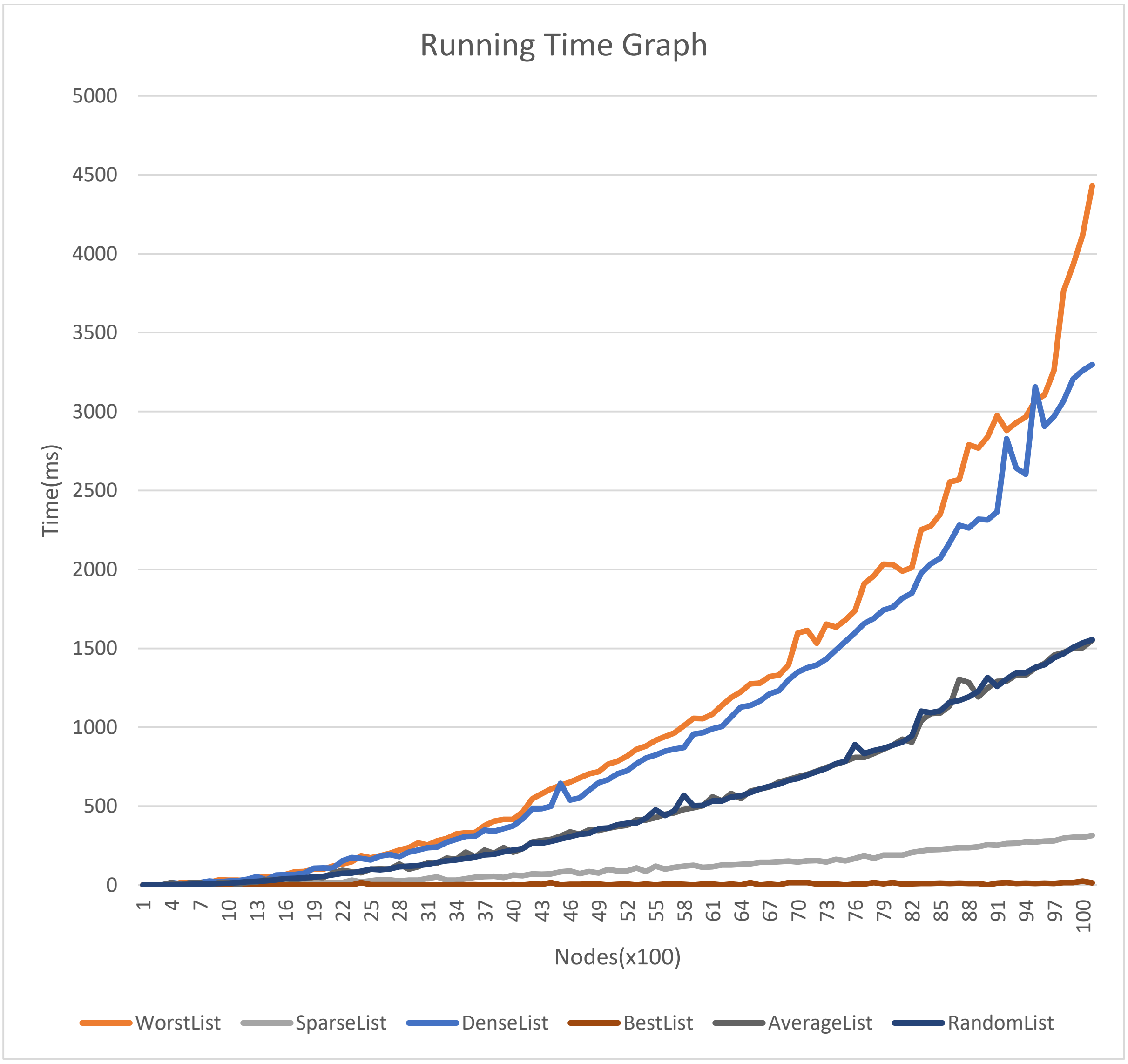


Diagram 3: List Graph Comparison

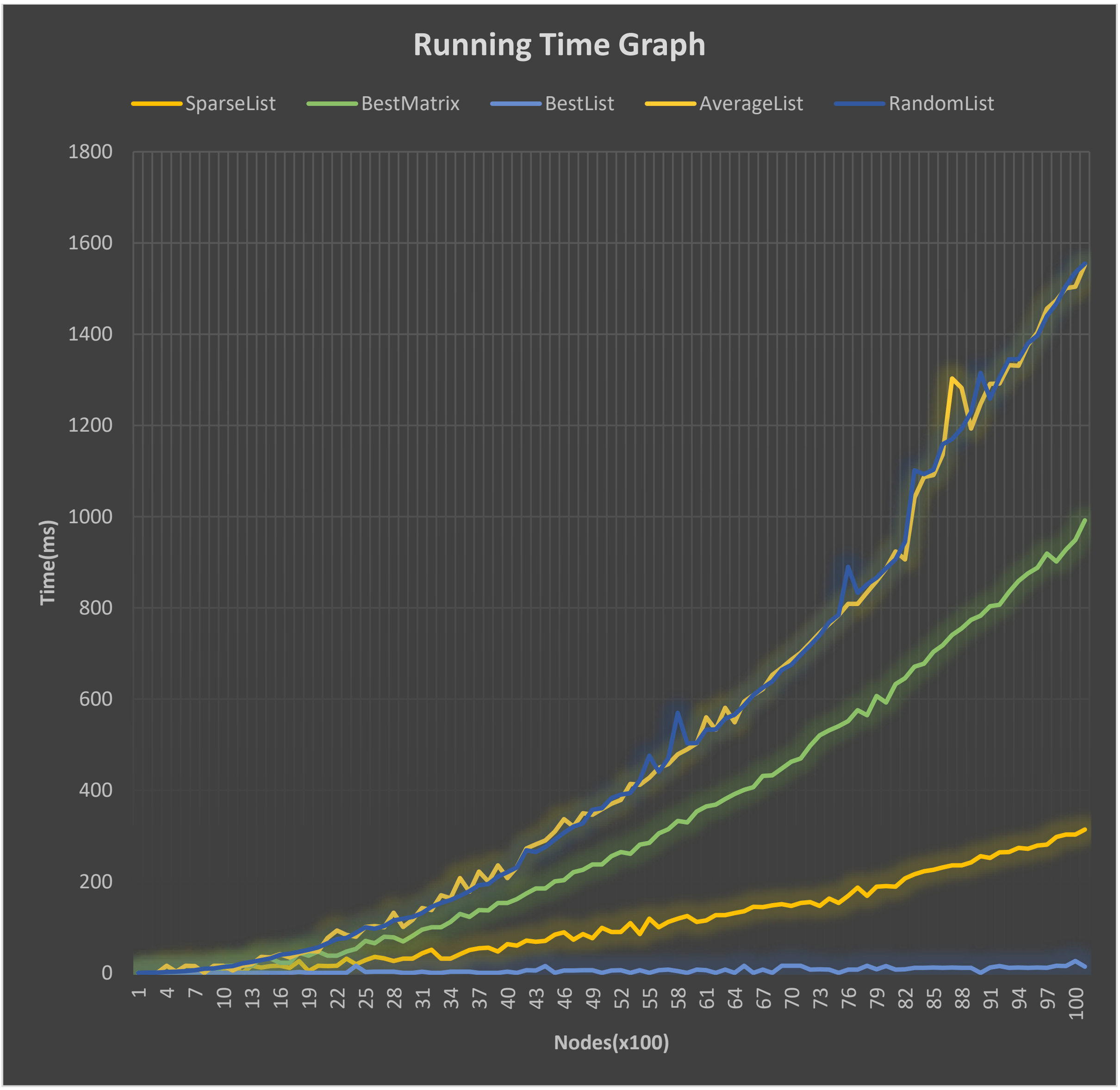


Diagram 4: Top 5 Fastest

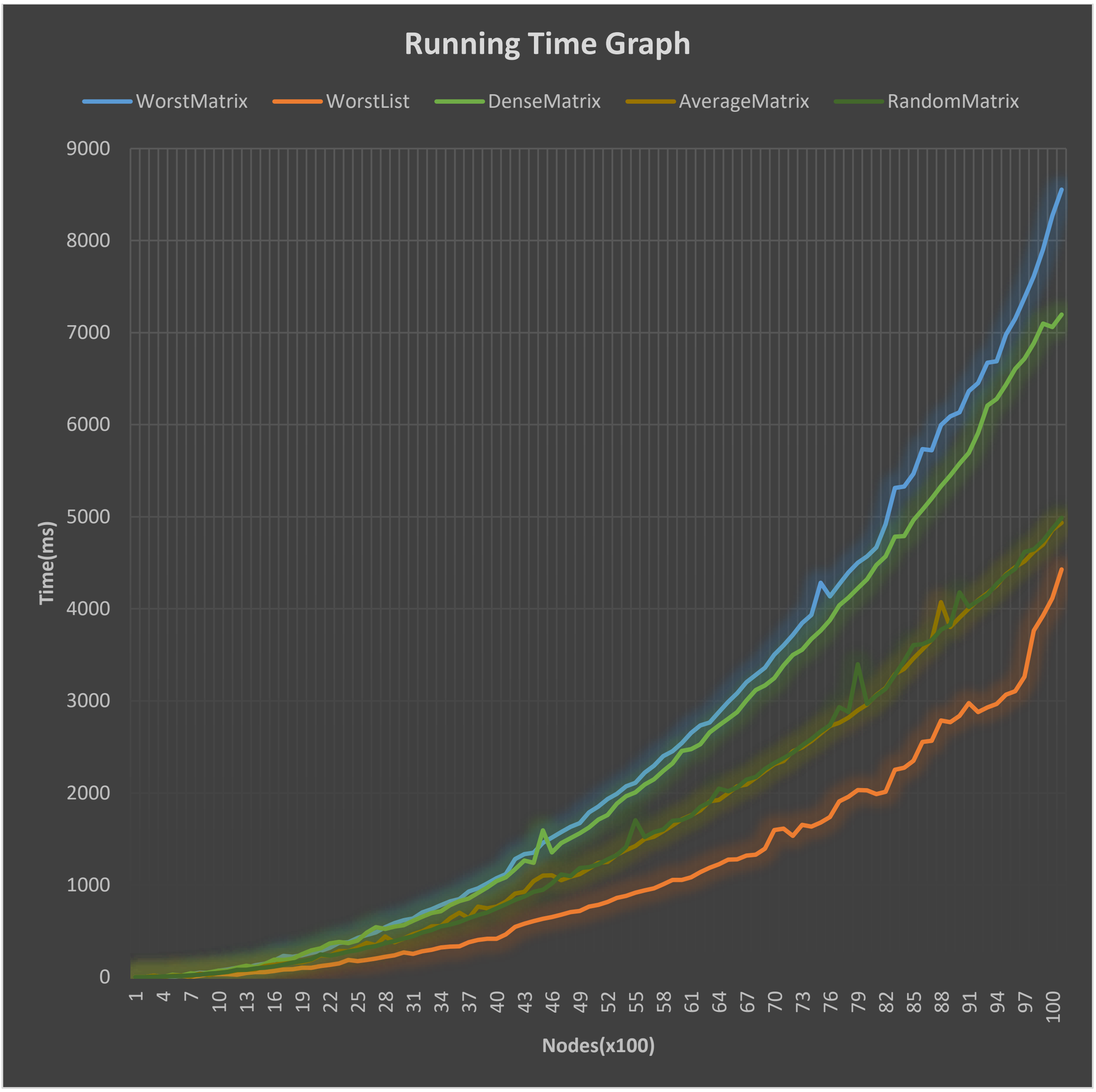


Diagram 5: Top 5 Slowest

Conclusion

In the algorithm we wrote, the running time of adjacency list is faster than adjacency matrix in all cases, no matter sparse or dense. It is because the algorithm did not cover the part of randomizing the matrix and creating of list from matrix, which is initializing the data and irrelevant to include in the running time. The time taken is only counted from the conversion of matrix/ list into node until the completion of DFS to ensure consistency. Last but not least, from the graph, we could also conclude that sparse is faster than dense in both matrix and list.

Appendix

1. 10,000 nodes with the interval of 100, hence only 100 results were generated. Included the results for 6 types of density.

RunningTime100
Jump.xlsx

2. Full results of 10,000 nodes with a random density.

RunningTimeRan
domFull.xlsx