# CSCE 221-200: Honors Data Structures and Algorithms
# Assignment Cover Page
# Spring 2021

| | |
|---|---|
| **Name:** | Alexander Akomer |
| **Email:** | alexakomer@tamu.edu |
| **Assignment:** | PA 5 |
| **Grade (filled in by grader):** | |

Please list below all sources (people, books, webpages, etc) consulted regarding this assignment (use the back if necessary):

| CSCE 221 Students | Other People | Printed Material | Web Material (give URL) | Other Sources |
|---|---|---|---|---|
| 1. | 1. | 1. | 1. https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/#:~:text=The%20Time%20complexity%20of%20BFS,and%20E%20stands%20for%20edges. | 1. |
| 2. | 2. | 2. | 2. | 2. |
| 3. | 3. | 3. | 3. | 3. |

Recall that TAMU Student Rules define academic misconduct to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. *Disciplinary actions range from grade penalty to expulsion.*

"On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work. In particular, I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received or given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment."

| | |
|---|---|
| **Signature:** | Alexander Akomer |
| **Date:** | 04/14/2021 |

Alexander Akomer

# PA 5 Report

## 1. *Introduction:*

This assignment is an exercise in calculating efficiency of different graph implementations. We were tasked with creating two different implementations of a graph– one following the structure of an adjacency matrix, and the other following the structure of an adjacency list. To compare the time efficiency of the two, breadth first search (BFS) and depth first search (DFS) algorithms were created. These were tested with different sets of inputs, including a continuous cycle of vertices, a fully connected clique, a graph with a random edge probability of 0.25, and a graph with random edge probability of 0.75. This was repeated with 10 values, 100 values, 1000 values, 10,000 values … until failure memory or time failure (segmentation fault). This was done to measure the time and memory complexity of each structure so that the advantages and disadvantages of each could be easily displayed. The results are as follows.

## 2. *Theoretical analysis:*

Two different graph implementations were required to be implemented alongside two different traversal functions for this assignment. These graph implementations were structurally different; an adjacency matrix was created as well as an adjacency list. BFS and DFS time complexities vary mostly based upon their implementation. For instance, the BFS under an adjacency matrix is known to resolve to $O(v^2)$, where v is the number of vertices in the graph. In fact, the DFS of an adjacency matrix resolves to $O(v^2)$ as well. These complexities occur in the best, worst, and average case of an adjacency matrix implementation for traversals. This is where the adjacency list shines, holding an $O(v + e)$ time complexity for both BFS and DFS in its best case, where v is

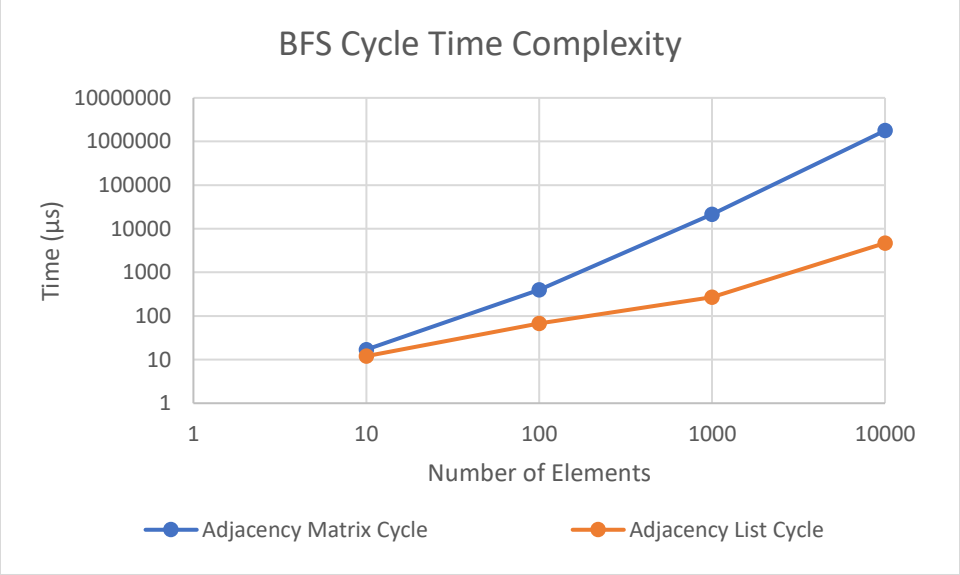
vertices and e is edges. In an average or worst case, adjacency list can also grow to an $O(v^2)$ complexity.

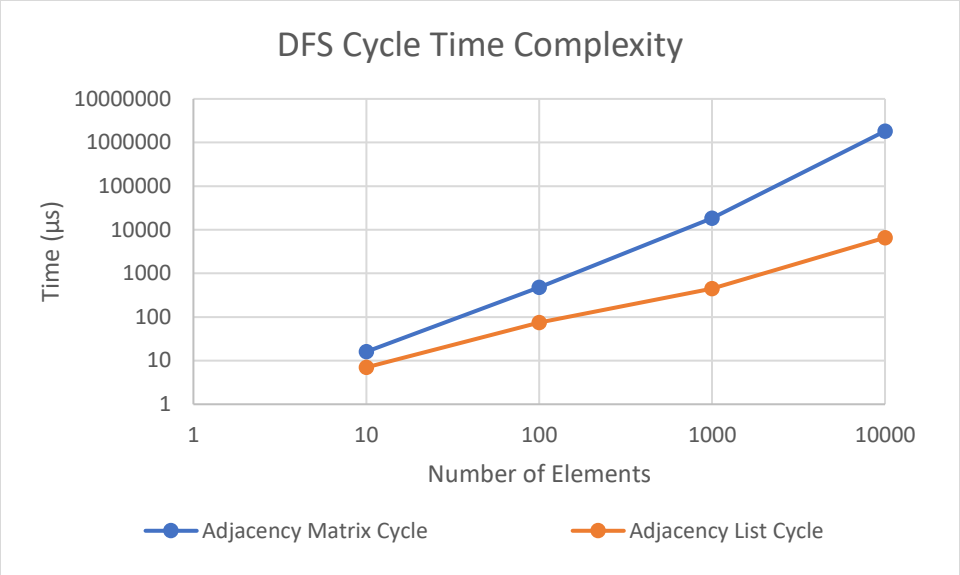
The adjacency matrix is also extremely taxing in terms of memory, requiring a matrix able to hold $v^2$ elements. The combination of this time complexity and memory inefficiency results in early segmentation faults. The adjacency list, only needing to hold v elements in its best case, struggles less with segmentation faults (specifically in best case).

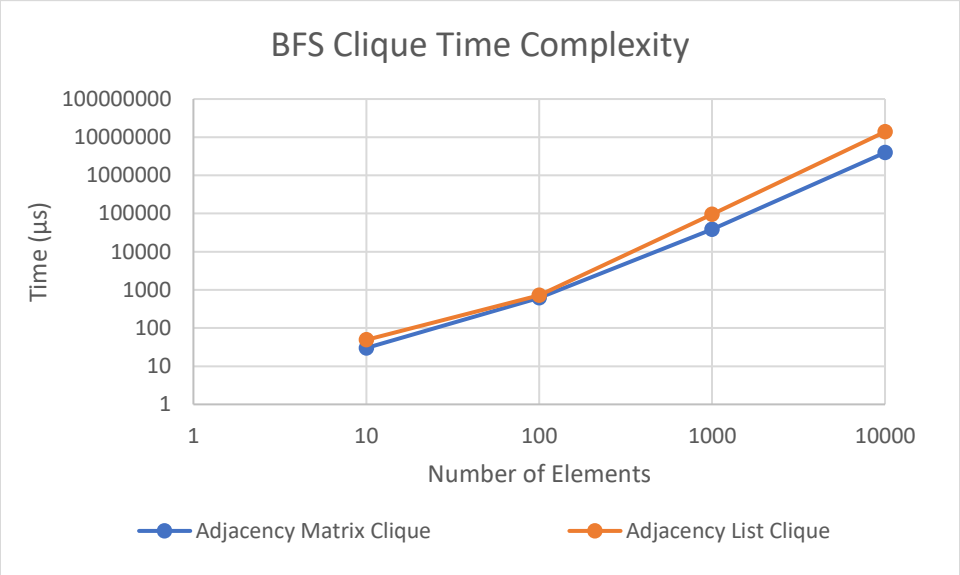
3. *Experimental Setup:*

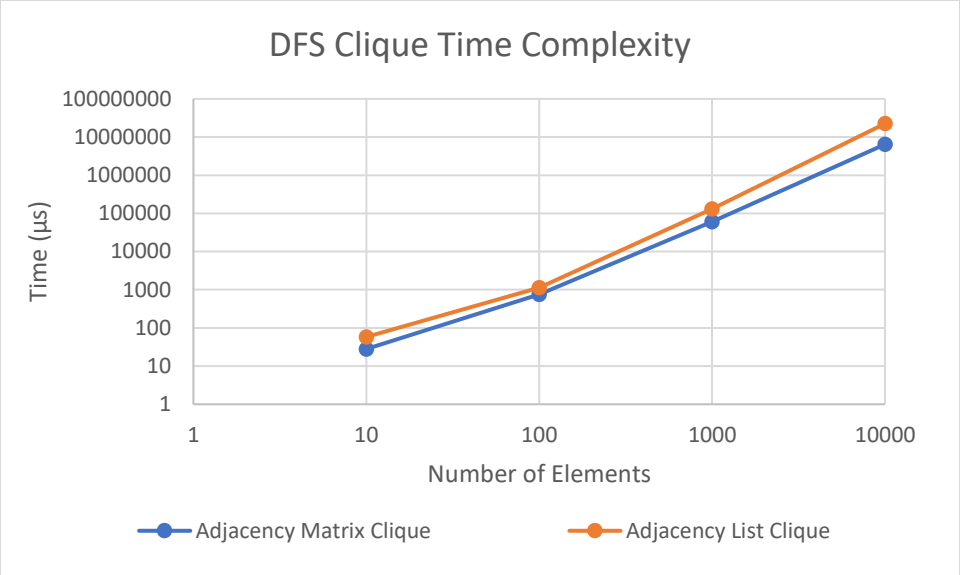
The machine the program was run on for my trials was a Surface Book 2 with 16 GB of RAM, an i7-8650U @ 1.90 GHz, and 63.6 GB / 475 GB free on the C: drive. My timing mechanism was the code attached on Canvas, which included the chrono package. This package allowed me to run a high-resolution clock and output the runtime in microseconds for each trial. Because the instructions only called for one trial of each function call for each sorting algorithm, that is the all of the data that was recorded, which may lead to faults in accuracy.

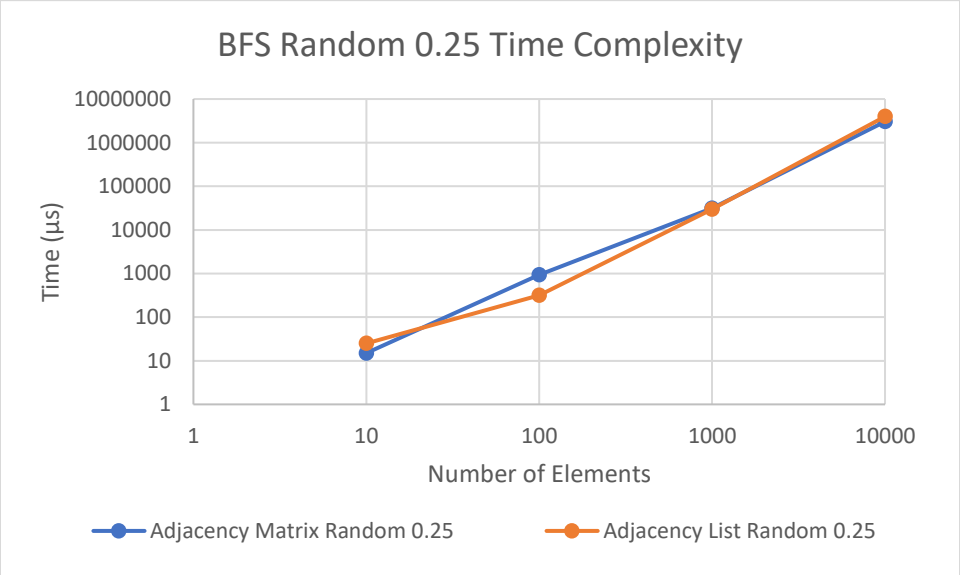
4. *Experimental Results:*

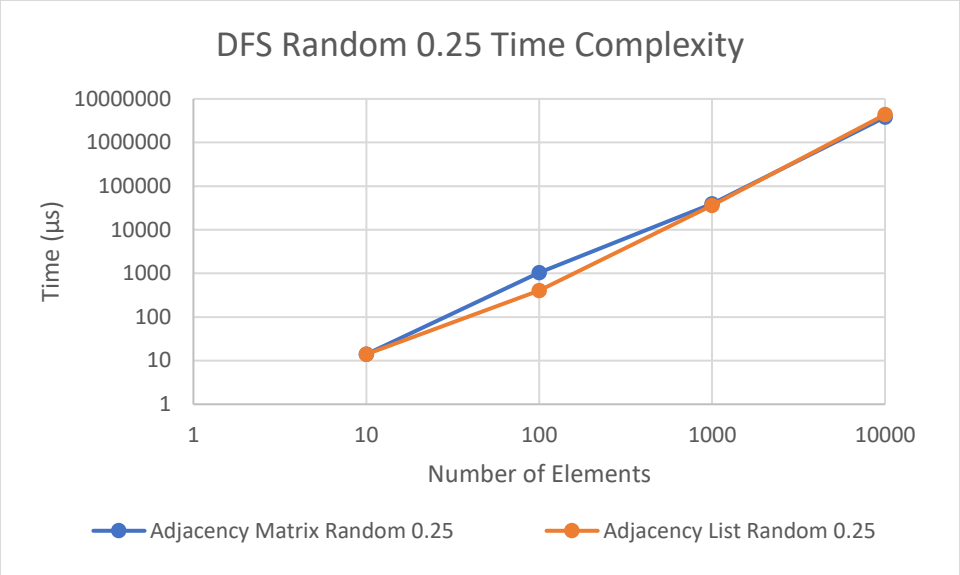



DFS Cycle Time Complexity
10000000
1000000
100000
10000
1000
100
10
1
Time (µs)
1
10
100
1000
10000
Number of Elements
Adjacency Matrix Cycle
Adjacency List Cycle


BFS Clique Time Complexity
100000000
10000000
1000000
100000
10000
1000
100
10
1
Time (µs)
1
10
100
1000
10000
Number of Elements
Adjacency Matrix Clique
Adjacency List Clique


DFS Clique Time Complexity
100000000
10000000
1000000
100000
10000
1000
100
10
1
Time (µs)
1
10
100
1000
10000
Number of Elements
Adjacency Matrix Clique
Adjacency List Clique


BFS Random 0.25 Time Complexity
10000000
1000000
100000
10000
1000
100
10
1
Time (µs)
1
10
100
1000
10000
Number of Elements
Adjacency Matrix Random 0.25
Adjacency List Random 0.25




Finally, the Stanford Facebook data set boasted a time of:

2511561 µs for the Adjacency Matrix

1376116 µs for the Adjacency List

After testing the two different traversal algorithms on the structurally different graph implementations, the following experimental conclusions are made:

(v is the number of vertices, e is the number of edges)

Adjacency Matrix:

BFS: $O(v^2)$ for best, average, and worst case

DFS: $O(v^2)$ for best, average, and worst case

Adjacency List:

BFS: $O(v + e)$ for best case, $O(v^2)$ for average and worst case

DFS: $O(v + e)$ for best case, $O(v^2)$ for average and worst case

The runtime data closely matches the theoretical analysis of each traversal algorithm dependent on their structure. The graphs show that in all cases, the adjacency matrix maintains a steady $O(v^2)$ constraint. This is even more obvious in the final graphs in which the number of vertices is held constant while the edges are assigned with random probability; in both the sparsely and almost fully-connected graphs, the adjacency matrix graph stays steady as edges have no role in its time complexity (asymptotically). The graphs show a different story for the adjacency list, which, in both the DFS and BFS algorithms, maintains an efficiency of $O(v + e)$ in its best case (which is represented by the cyclical testing in the first two graphs). This linear efficiency is due to the fact that each vertex has one neighbor, and the associated structure therefore becomes a vector that is v long with a single node in each respective linked list. This is much more memory and time efficient than the $v^2$ adjacency matrix. However, as average and worst cases were reached (including the fully-connected clique, the worst possible case), $O(v^2)$ efficiency was quickly reached for both BFS and DFS (as seen in the last six graphs). The Stanford data set represented a more average case with relatively dense connections, and therefore had a similar asymptotical outcome between both graph implementations. The number of edges still had a greater effect on the adjacency list than the adjacency matrix, as evidenced by the random graphs in the end in which the adjacency list becomes more efficient than the adjacency matrix in a sparsely populated

graph and less efficient in a denser graph containing more edges. Despite the limited trials and slight discrepancies, every algorithm is bounded exactly as depicted in the theoretical analyses.