Pawan Acharya
CS4412
3/1/2022
Project Report

**Goals:**

- Understand Dijkstra's algorithm in the context of a real world problem.
- Implement a priority queue with worst-case logarithmic operations.
- Compare two different priority queue data structures for implementing Dijkstra's and empirically verify their differences.
- Understand the importance of proper data structures/implementations to gain the full efficiency potential of algorithms.

1. **Implementation of Dijkstra's algorithm:** The source code for the project is submitted in the zip file.
We can find the shortest path between nodes in a graph using Dijkstra's Algorithm. In Particular, we can find the shortest path from a source node to all other nodes in the graph, producing a shortest-path tree. Once a node is visited, it is added as a visited node.
Since, the algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
The algorithm continues until all the nodes have been added to the path.
This algorithm cannot be used when there are one or more negative edges.
Complexity:
Time: $O((|V| + |E|) \log V)$
Space: $O(|V| + |E|)$

**2. Implementation of two version of priority Queue:**

The source code for both the implementations is provided in zip file.

I. Using Python list
II. Using Heap

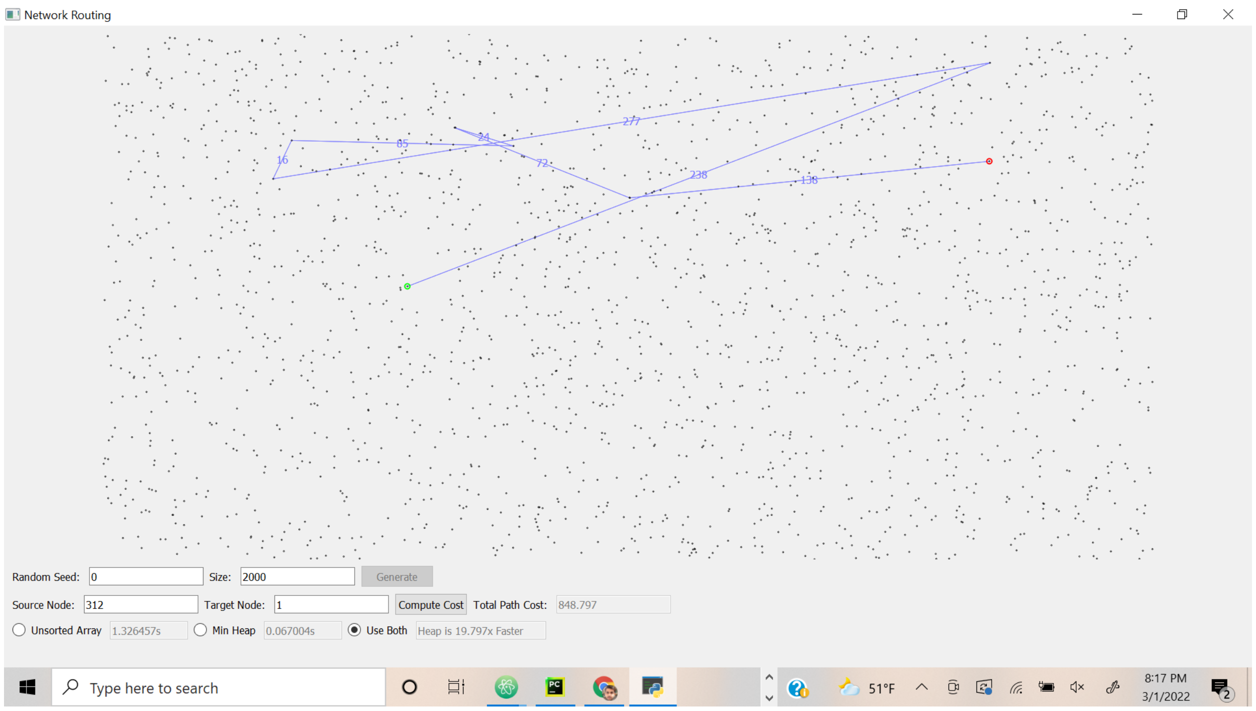
**3. The time complexity of both implementations of the algorithms:**

**I. Using Python list:**
Time complexity: The time complexity to insert to a list is O(1). Similarly, time complexity of O(|V|) as it has to iterate over all of the indexes of the list in the del_min function. Similarly, the decrease_Key takes O(1) time complexity.

**II. Using Heap:** The time complexity to insert to a list is O(log|V|). Similarly, the decrease_Key and del_min both take O(log|V|) time complexity.

**Total Complexity of the Dijkstra's Algorithm:**
Complexity:
Time: $O((|V| + |E|) \log V)$
Space: $O(|V| + |E|)$

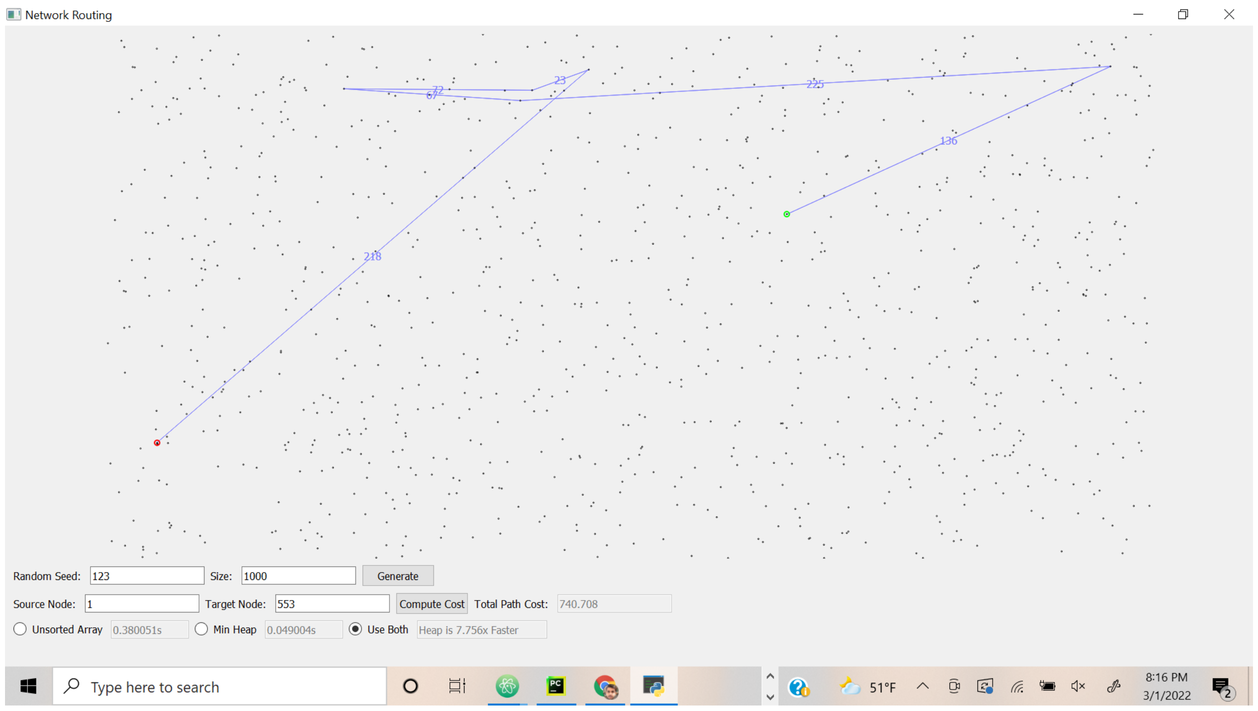
## 4. Results(Screenshots):



Random Seed: 0 Size: 2000 Generate

Source Node: 312 Target Node: 1 Compute Cost Total Path Cost: 848.797

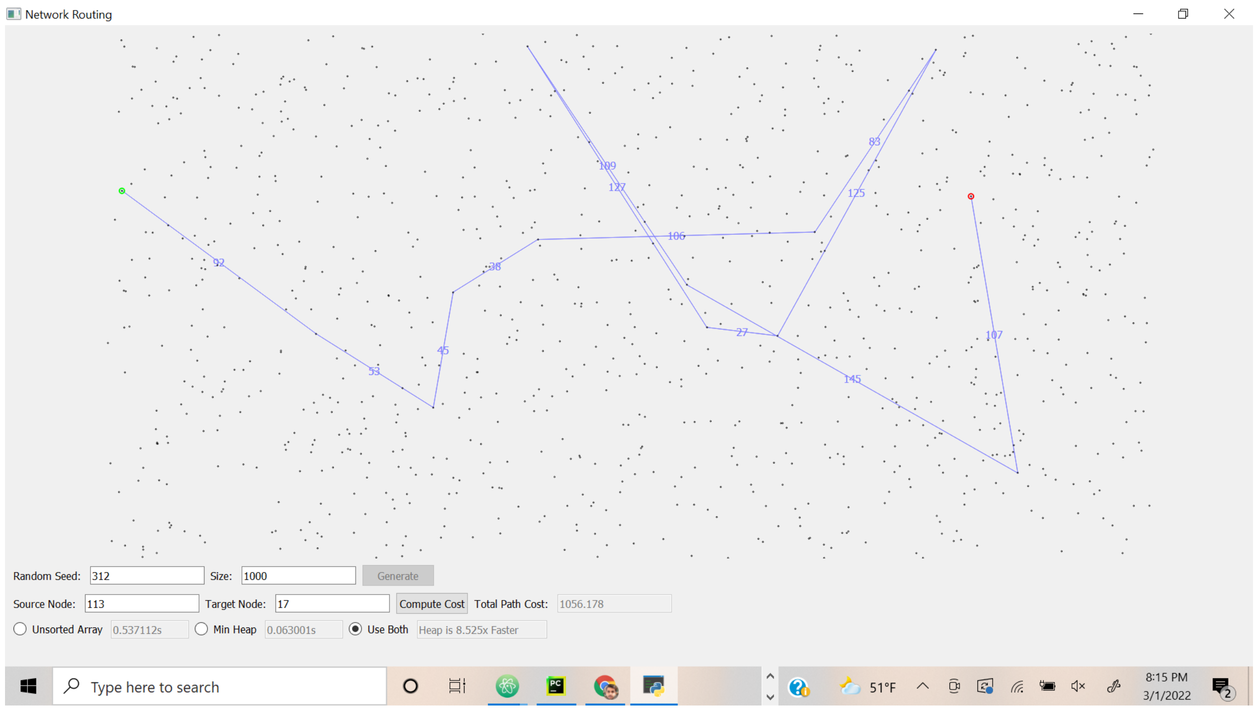
Unsorted Array 1.326457s Min Heap 0.067004s [✓ Use Both] Heap is 19.797x Faster


Network Routing

Random Seed: 123 Size: 1000 Generate

Source Node: 1 Target Node: 553 Compute Cost Total Path Cost: 740.708

Unsorted Array 0.380051s Min Heap 0.049004s [✓ Use Both] Heap is 7.756x Faster


Network Routing

Random Seed: 312 Size: 1000 Generate

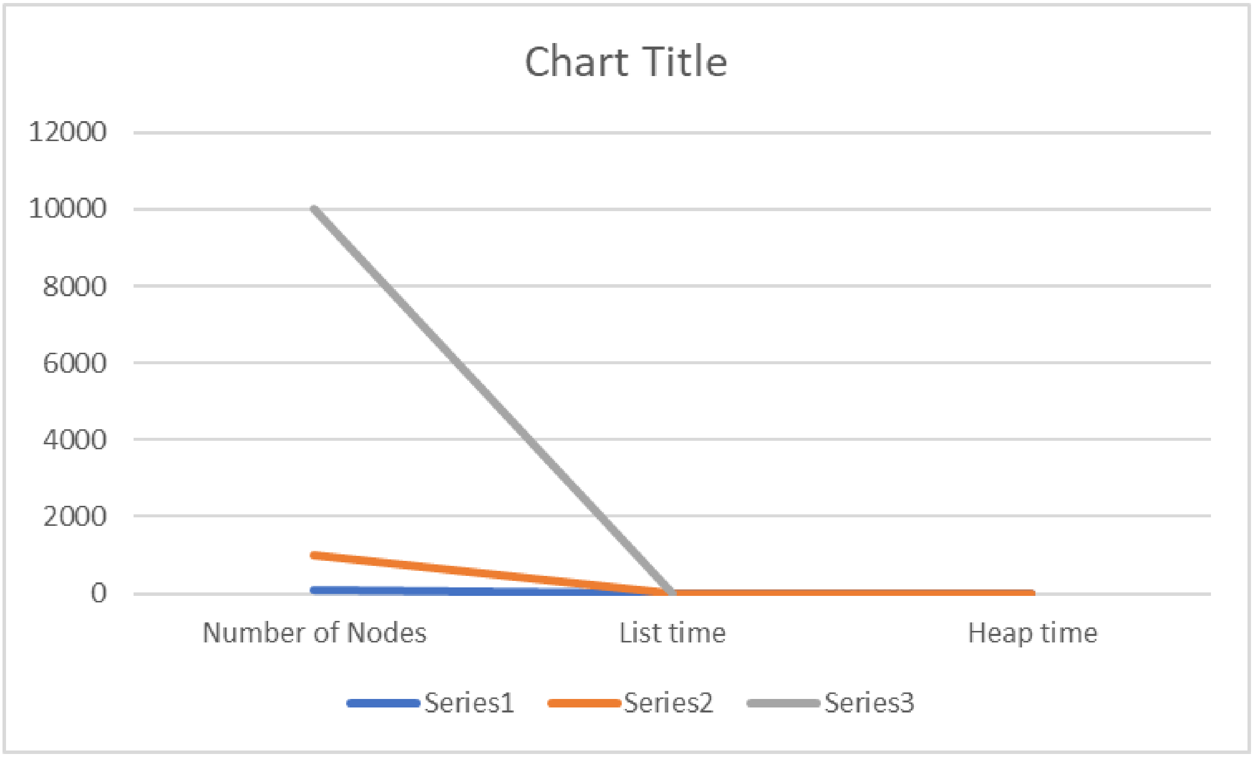
Source Node: 113 Target Node: 17 Compute Cost Total Path Cost: 1056.178

Unsorted Array 0.537112s Min Heap 0.063001s [✓ Use Both] Heap is 8.525x Faster

**5. Comparison of the empirical time complexity for Array vs. Heap:**

For very large values of nodes, my program didn’t work, meaning it didn’t responded.

| **Number of Nodes** | **List time** | **Heap time** | Approximated |
|---|---|---|---|
| 100 | 0.007999 | 0.002999 | 0.003 |
| 1000 | 0.442106 | 0.038999 | 0.05 |
| 10000 | 44.084056 | 0.551007 | 10 |
|  | Program didn't run |  |  |



Analysis of this comparison:
The graphing of average time complexity for array and heap implementations revealed that the array implementation has an O|V^2| runtime, whereas the binary heap implementation has a nlogn runtime as depicted in the diagram above. The run time for sizes 10 to 1000 did not differ significantly, but as soon as the size is increased, the run time increases dramatically.

REFERENCES: Project Assignment Description,
https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/