GRAPH THEORY

TRAVELLING SALESMAN

GROUP MEMBERS

MUHAMMAD SUDAIS 22K-4364
AZKA SAHAR 22K-4494
SUMAIYA WAHEED 22K-4201

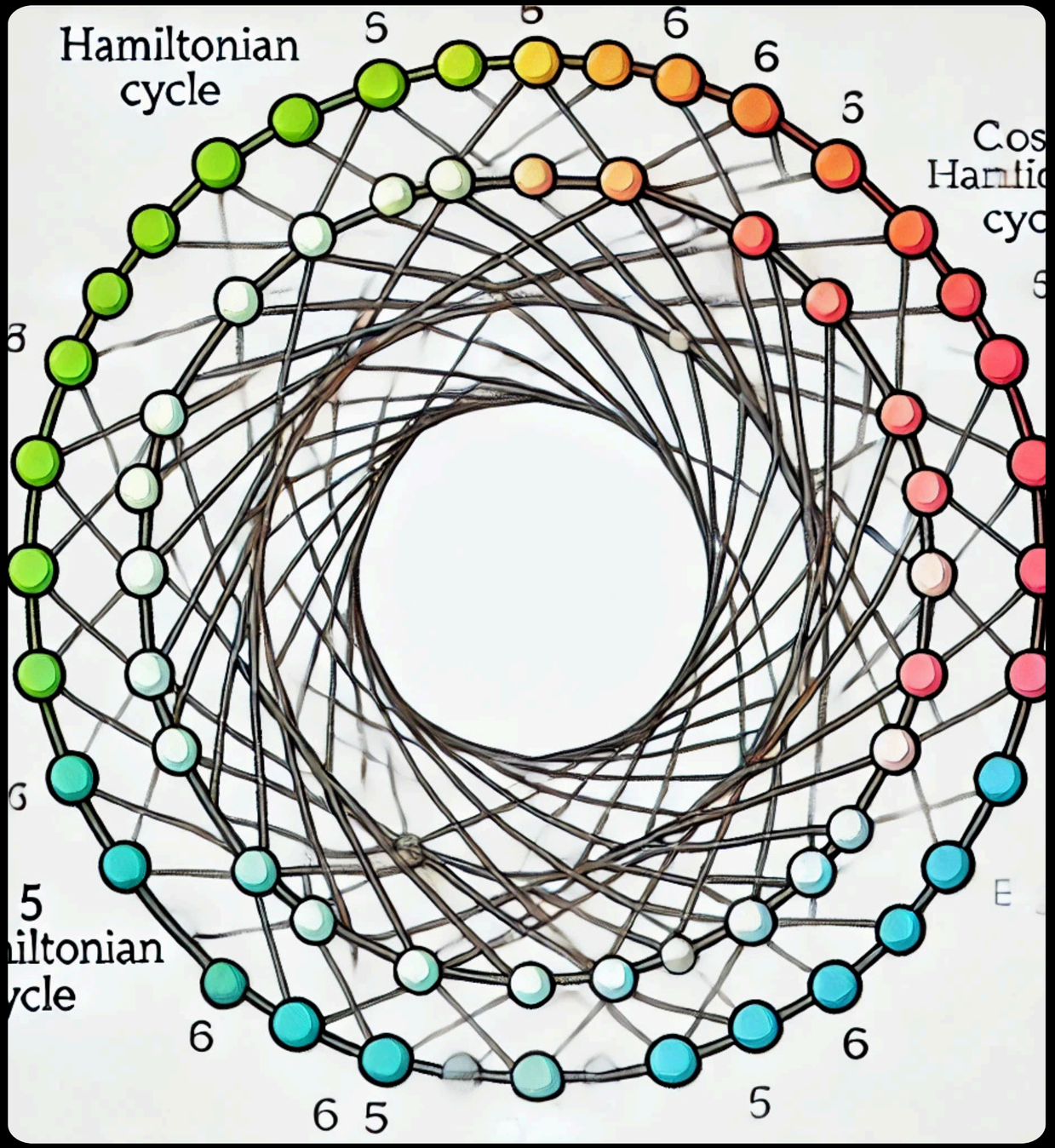
PROBLEM

The Traveling Salesman Problem (TSP) is a classic optimization problem in graph theory. Its primary objective is to find the shortest possible route that allows a salesman to visit a given set of cities (vertices) exactly once and return to the starting city. The distances between each pair of cities are represented as edges in a graph.

0012 7482901 2744103 0592346 8774510 7255

FUNCTION:

0002849916 . 742941 . 1265330 . #18940 DO NOT DELAY



BRUTE FORCE

It involves generating all possible permutations of the vertices and calculating the total distance for each permutation. The solution with the smallest total distance is selected as the optimal solution

List all permutations:

- Generate all possible permutations of the vertices, treating the starting point as fixed.

Calculate the total distance for each permutation:

- For each permutation, calculate the total distance by summing up the distances between consecutive vertices in the cycle, including the return to the starting vertex.

Track the minimum distance:

- Compare the total distance of each permutation and keep track of the minimum distance found.

Select the optimal path:

- Once all permutations have been evaluated, select the path with the smallest total distance as the solution.
- Return the optimal solution:
- Return the permutation that corresponds to the optimal tour with the smallest total distance.

NEAREST INSERTION

It begins with two vertices and gradually inserts the nearest vertex to the current path, minimizing the total distance incrementally.

Start with two vertices:

Choose the pair of vertices with the smallest distance between them and add them to the path.

Mark vertices as visited:

- Mark the selected pair of vertices as visited and add them to the cycle.

Insert nearest unvisited vertex:

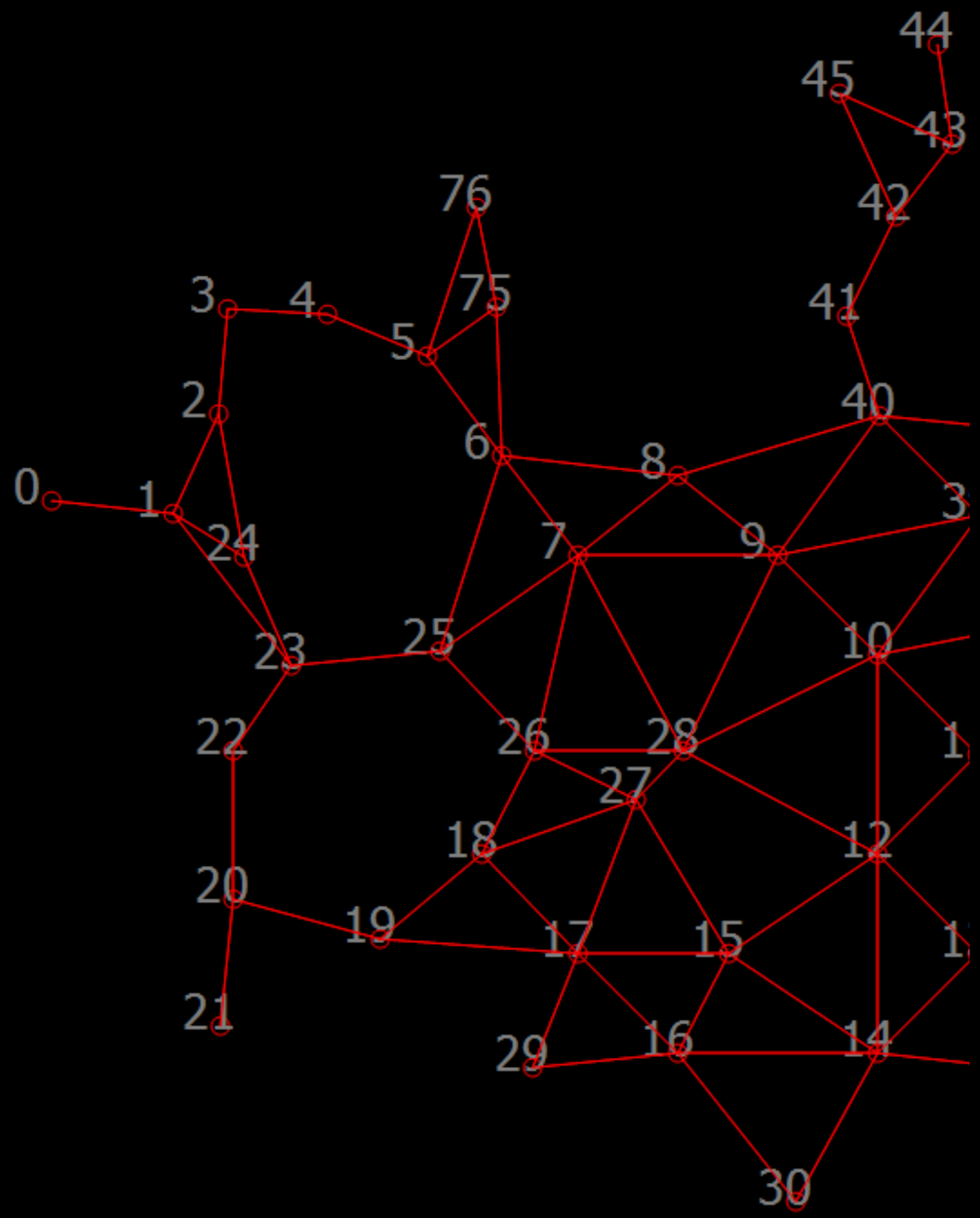
- For each remaining unvisited vertex, calculate the distance to every edge in the current cycle. Insert the vertex where the insertion causes the smallest increase in the total distance.

Repeat until all vertices are visited:

- Continue inserting the nearest vertex into the cycle until all vertices are included.

Complete the cycle:

- Once all vertices are in the cycle, return to the starting vertex to complete the tour.



NEAREST NEIGHBOR

The Nearest Neighbor Algorithm provides a greedy approach to approximate the solution to the TSP by making locally optimal choices.

Choose a Starting City:
Begin at any randomly selected city or a predefined starting point.

Find the Nearest Neighbor:
Identify the unvisited city closest to the current city (the one with the smallest distance).

Visit the City:
Move to the nearest unvisited city and mark it as visited.

Repeat:
Continue finding and visiting the nearest unvisited city until all cities have been visited.

Return to the Starting City:
Once all cities are visited, return to the starting city to complete the tour.

Calculate Total Distance:
Sum up the distances between consecutive cities in the tour to determine the total weight of the path.

COMPARISON

NEAREST NEIGHBOR

Time Complexity: O(n^2)

This algorithm is a greedy approximation, which means it doesn't guarantee an optimal solution but gives a fairly good solution quickly.

NEAREST INSERTION

Time Complexity: O(n^2)

This algorithm gives a better result than Nearest Neighbor, but it still doesn’t guarantee the optimal solution. It’s more balanced between computational efficiency and result quality.

BRUTE FORCE

Time Complexity: O(n!)

Exact optimal solution: This algorithm guarantees the best possible solution, but it becomes unfeasible for large inputs.

PROJECT ANALYSIS

74921 59300 4

18863 70782 0

Global

PROGRAMING LANGUAGES

LANGUAGE 1:
Python

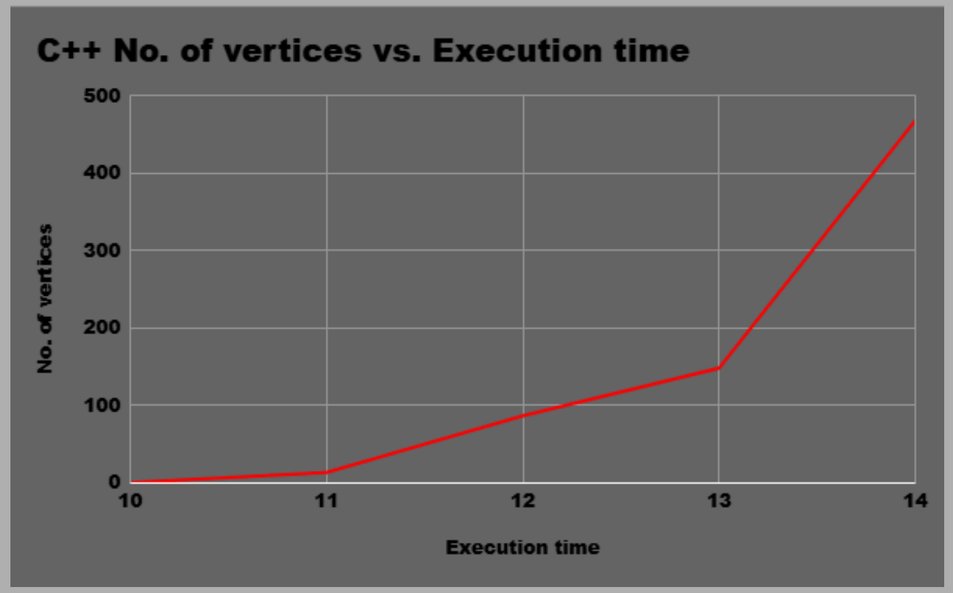
LANUAGE 2:
C++

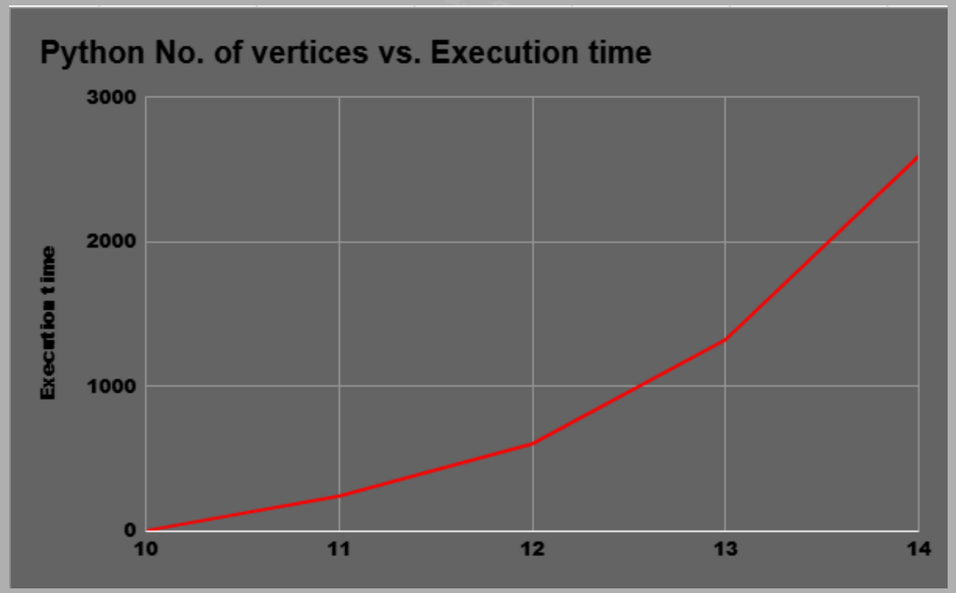
FUNCTION:

0012 7482901 2744103 0592346 8774510 7255

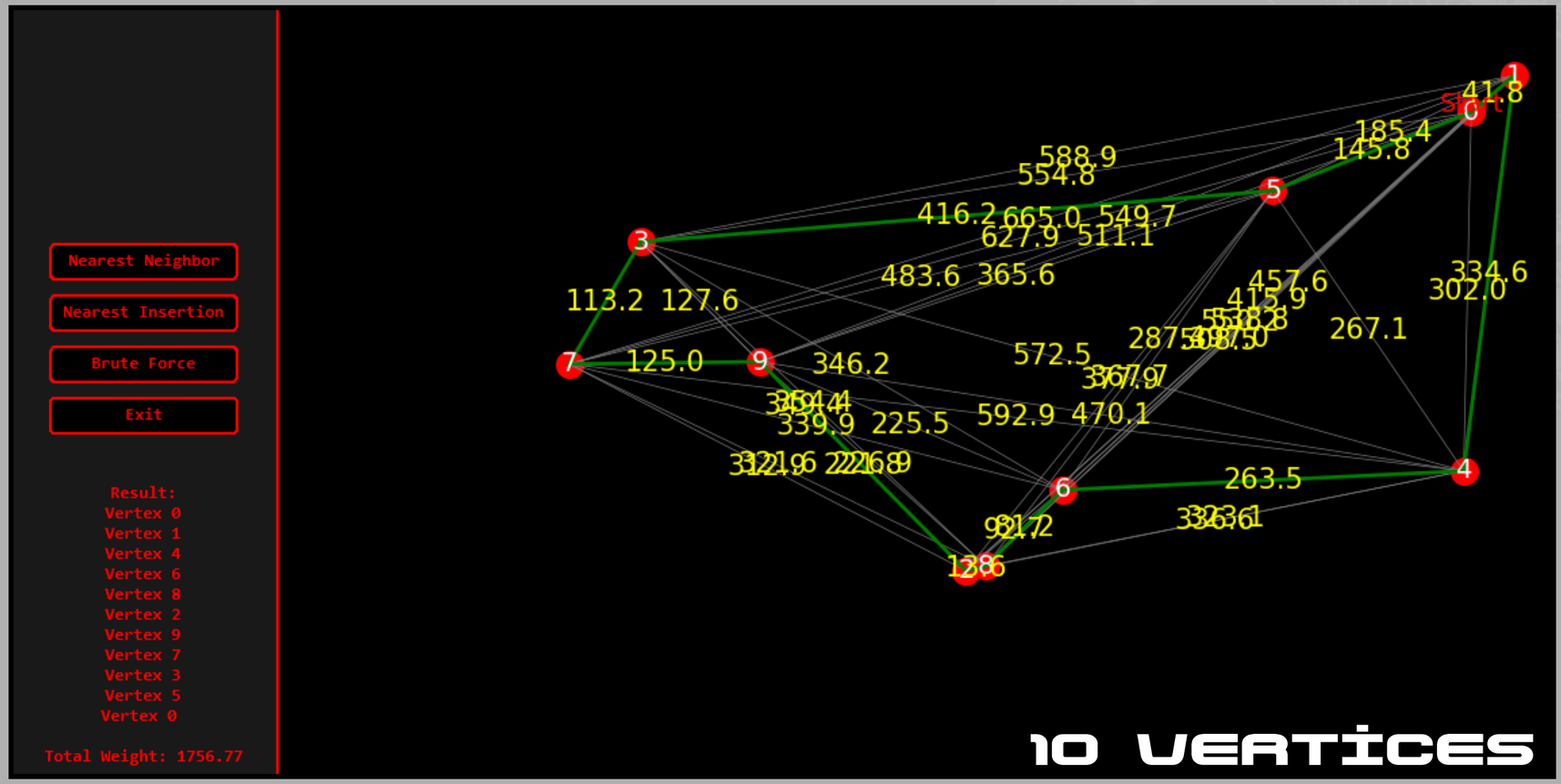
0002849916 . 742941 . 1265330 . #18940 DO NOT DELAY

BRUTE FORCE

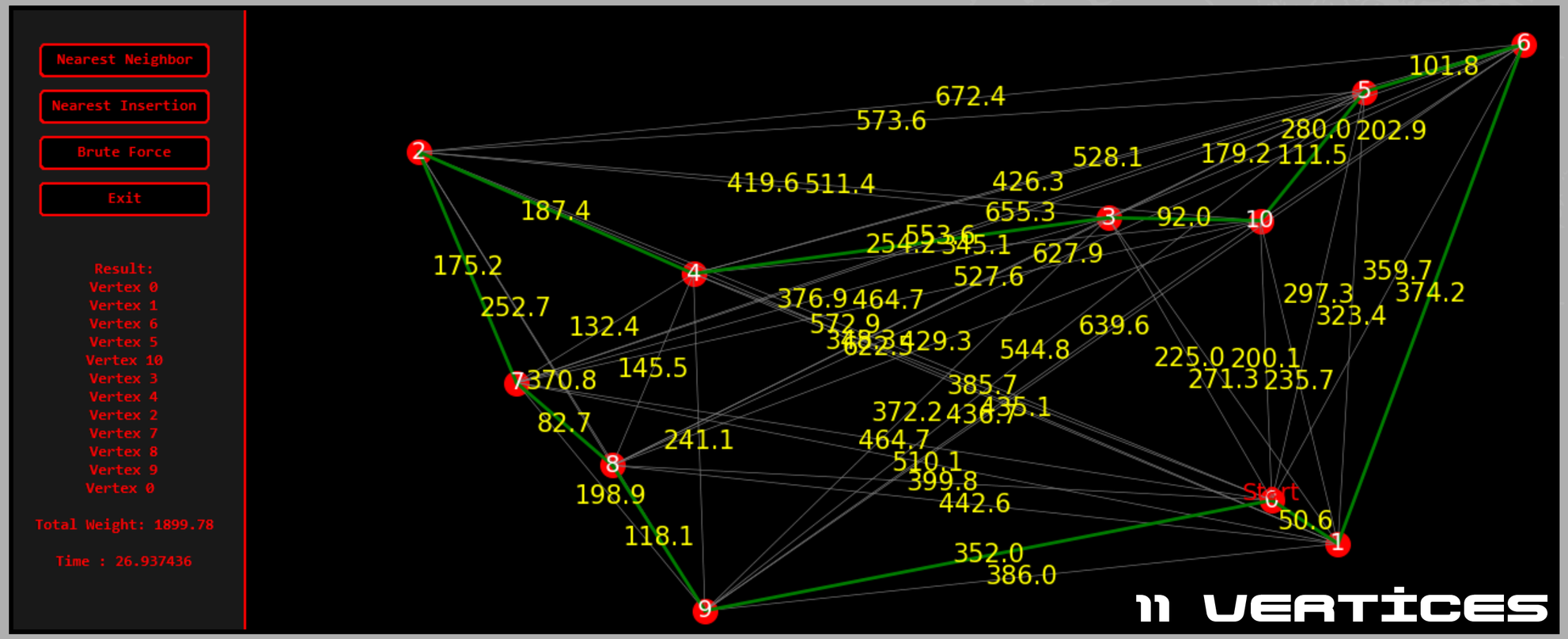

C++ No. of vertices vs. Execution time
500
400
300
200
100
0
No. of vertices
10
11
12
13
14
Execution time


Python No. of vertices vs. Execution time
3000
2000
1000
0
Execution time
10
11
12
13
14

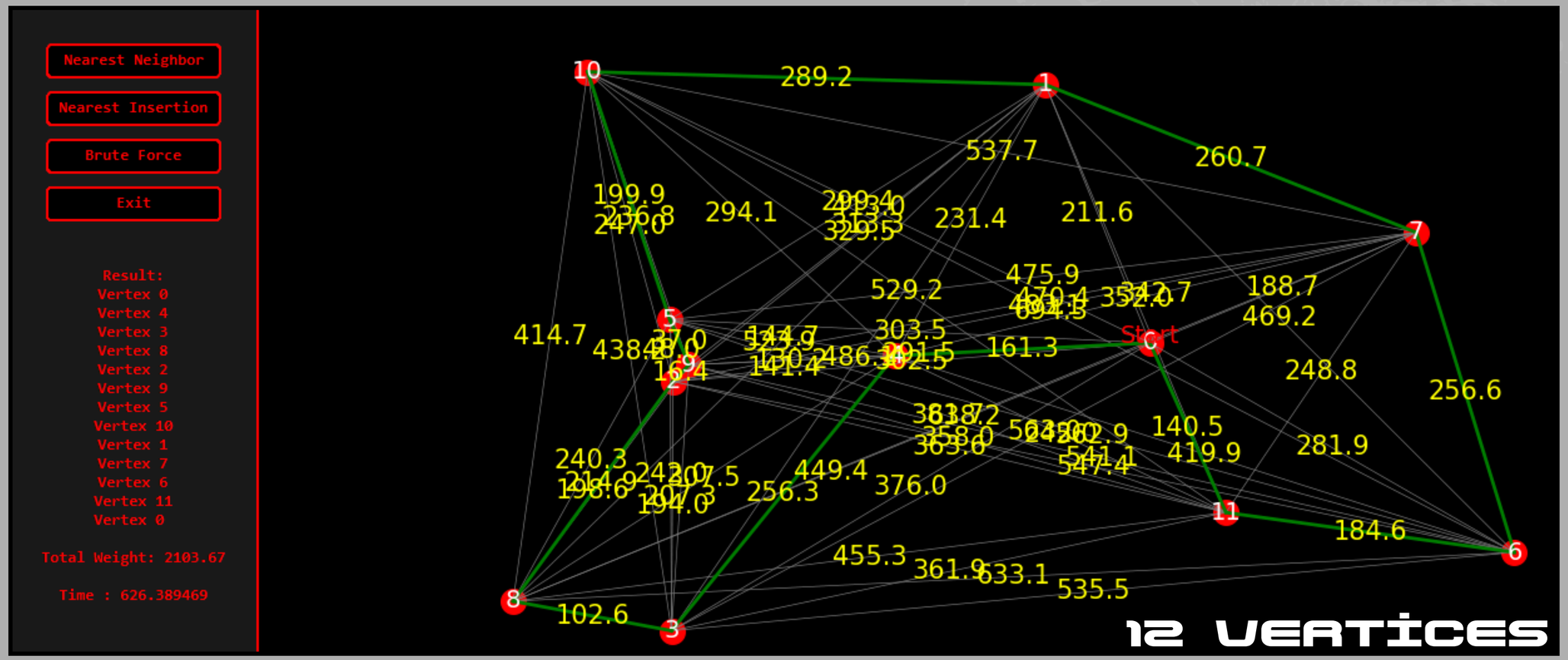
BRUTE FORCE


Nearest Neighbor
Nearest Insertion
Brute Force
Exit
Result:
Vertex 0
Vertex 1
Vertex 4
Vertex 6
Vertex 8
Vertex 2
Vertex 9
Vertex 7
Vertex 3
Vertex 5
Vertex 0
Total Weight: 1756.77
10 VERTİCES

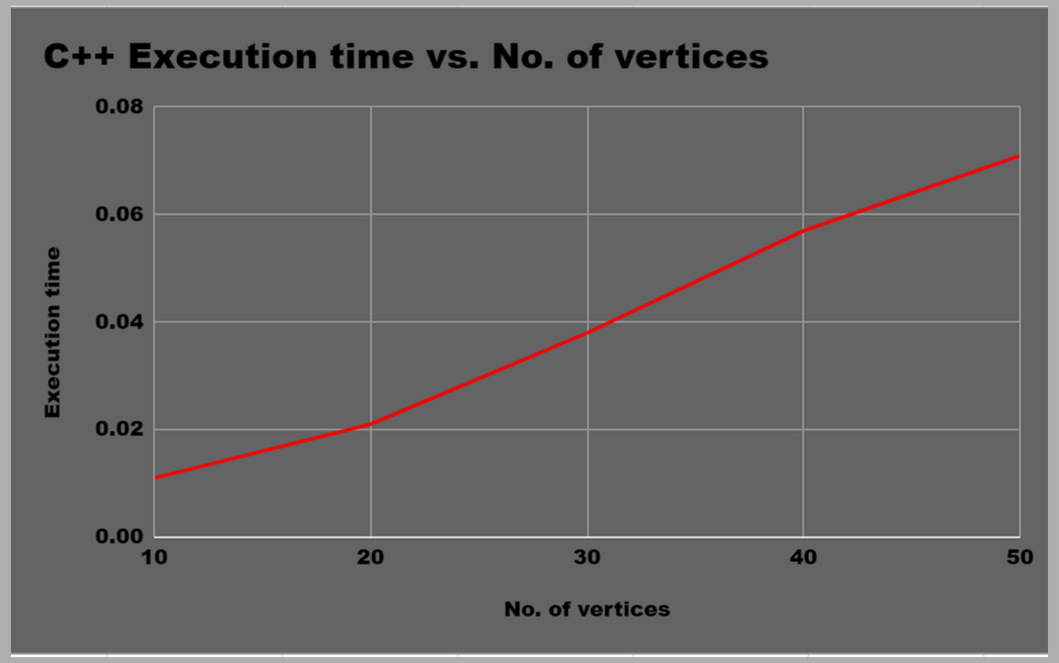
BRUTE FORCE

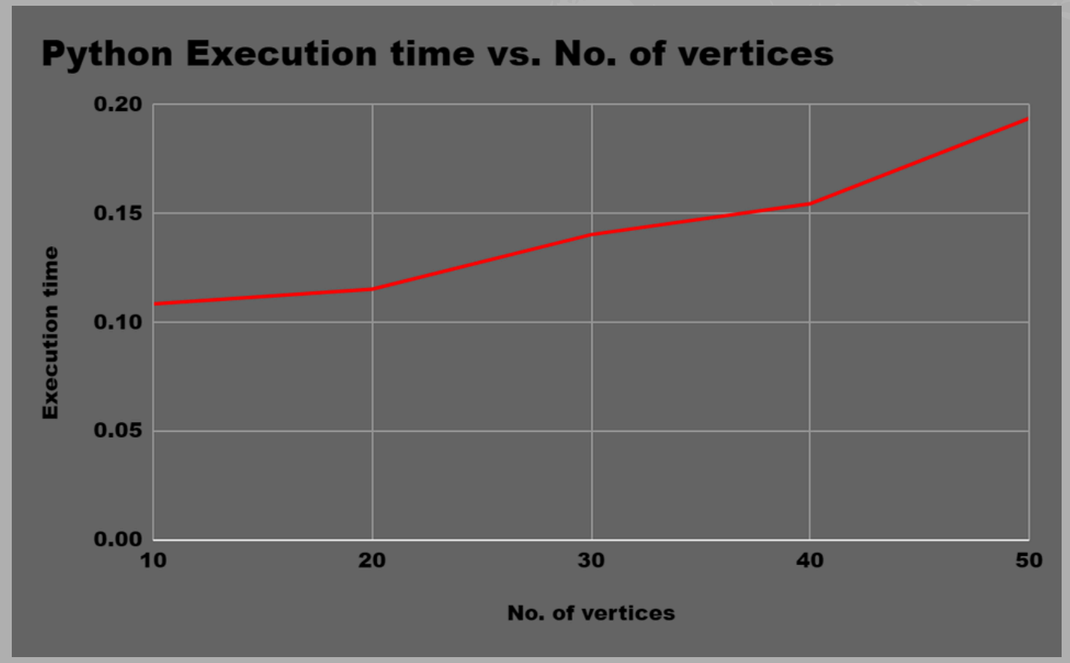

Nearest Neighbor
Nearest Insertion
Brute Force
Exit
Result:
Vertex 0
Vertex 1
Vertex 6
Vertex 5
Vertex 10
Vertex 3
Vertex 4
Vertex 2
Vertex 7
Vertex 8
Vertex 9
Vertex 0
Total Weight: 1899.78
Time : 26.937436
672.4
573.6
101.8
280.0
202.9
528.1
179.2
111.5
419.6
511.4
426.3
187.4
655.3
92.0
553.6
254.2
345.1
627.9
175.2
527.6
359.7
297.3
374.2
323.4
252.7
376.9
464.7
572.9
639.6
132.4
622.5
429.3
544.8
225.0
200.1
145.5
370.8
385.7
271.3
235.7
372.2
436.7
435.1
82.7
241.1
464.7
510.1
399.8
198.9
442.6
Start
50.6
118.1
352.0
386.0
11 VERTİCES

BRUTE FORCE

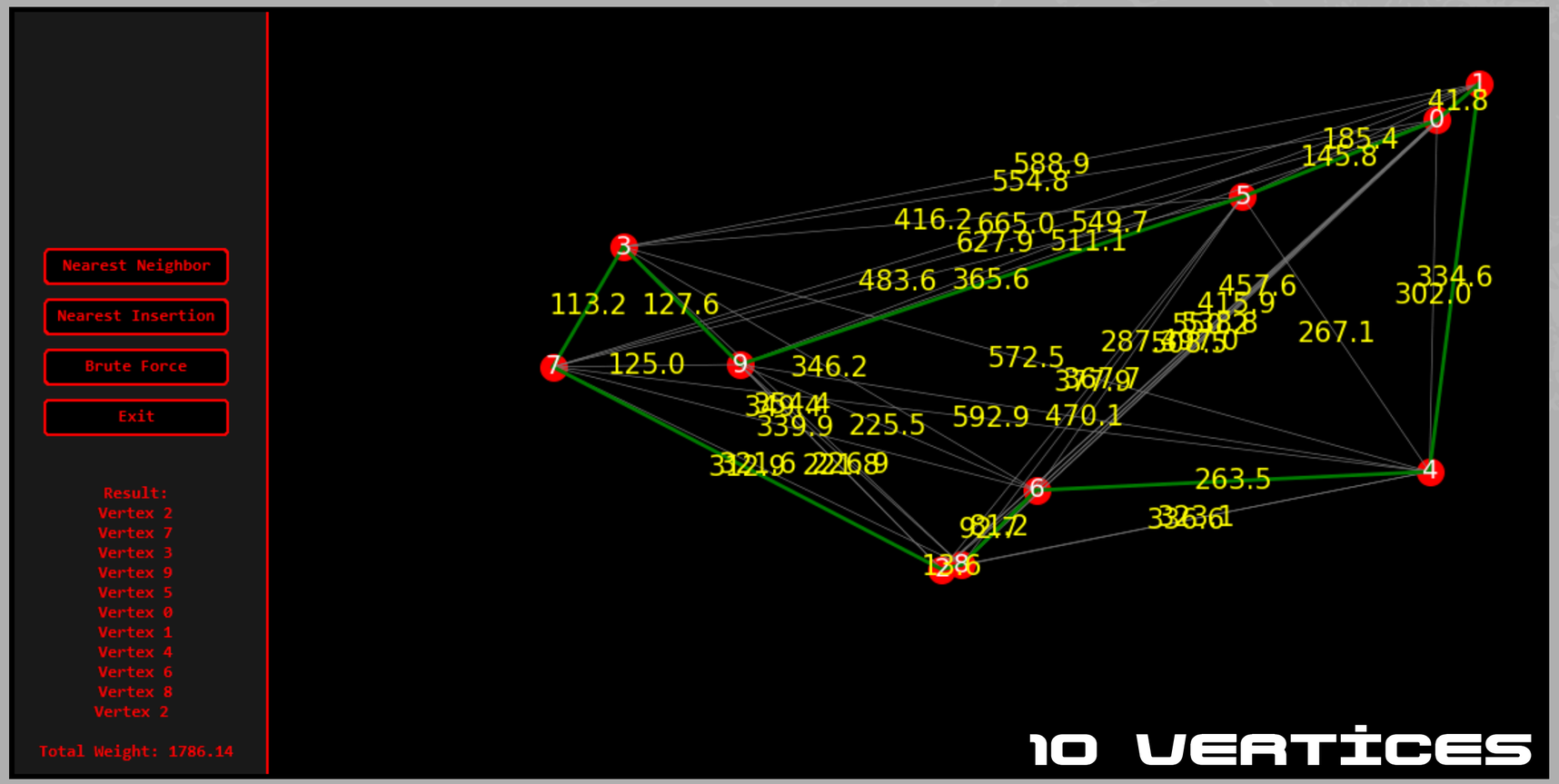

Nearest Neighbor
Nearest Insertion
Brute Force
Exit
Result:
Vertex 0
Vertex 4
Vertex 3
Vertex 8
Vertex 2
Vertex 9
Vertex 5
Vertex 10
Vertex 1
Vertex 7
Vertex 6
Vertex 11
Vertex 0
Total Weight: 2103.67
Time : 626.389469
Start
12 VERTİCES

NEAREST INSERTION

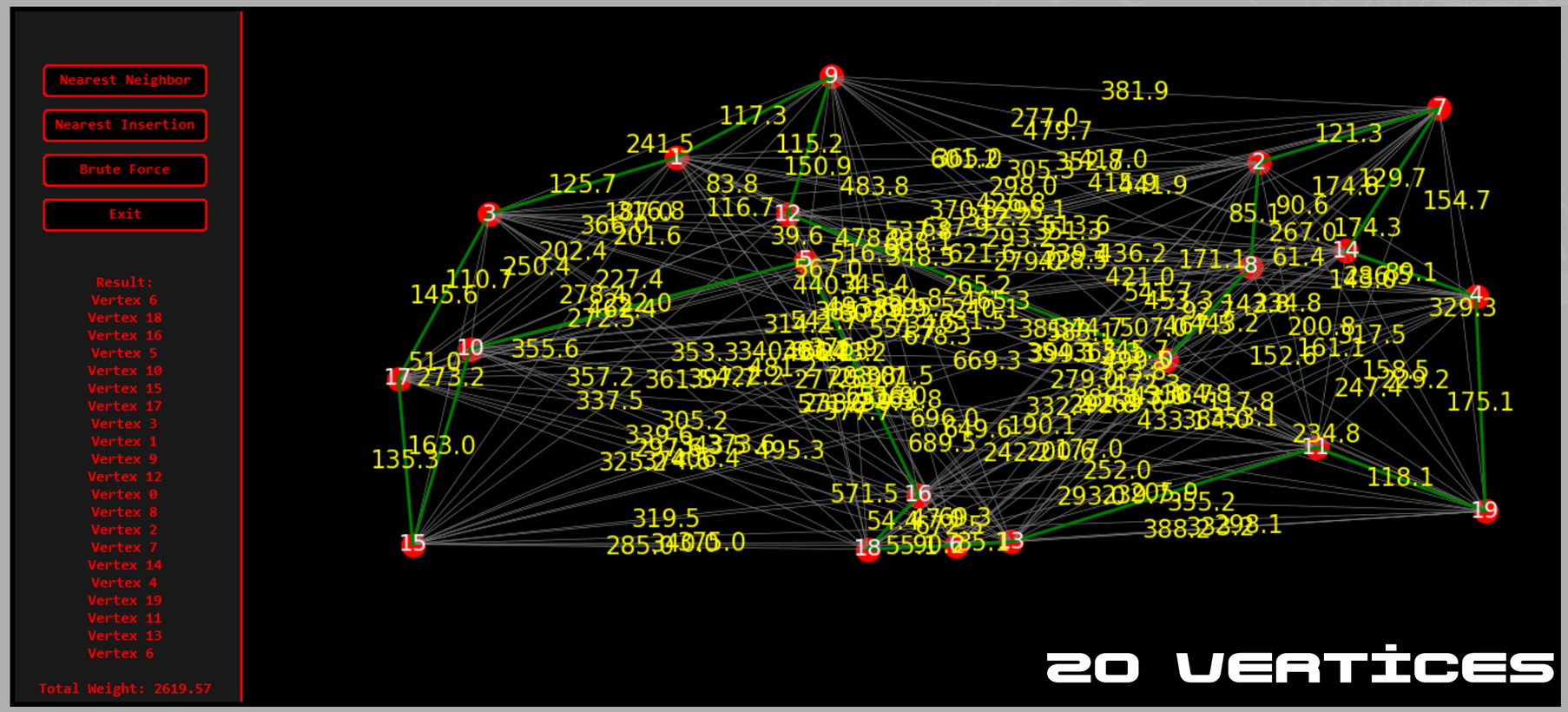




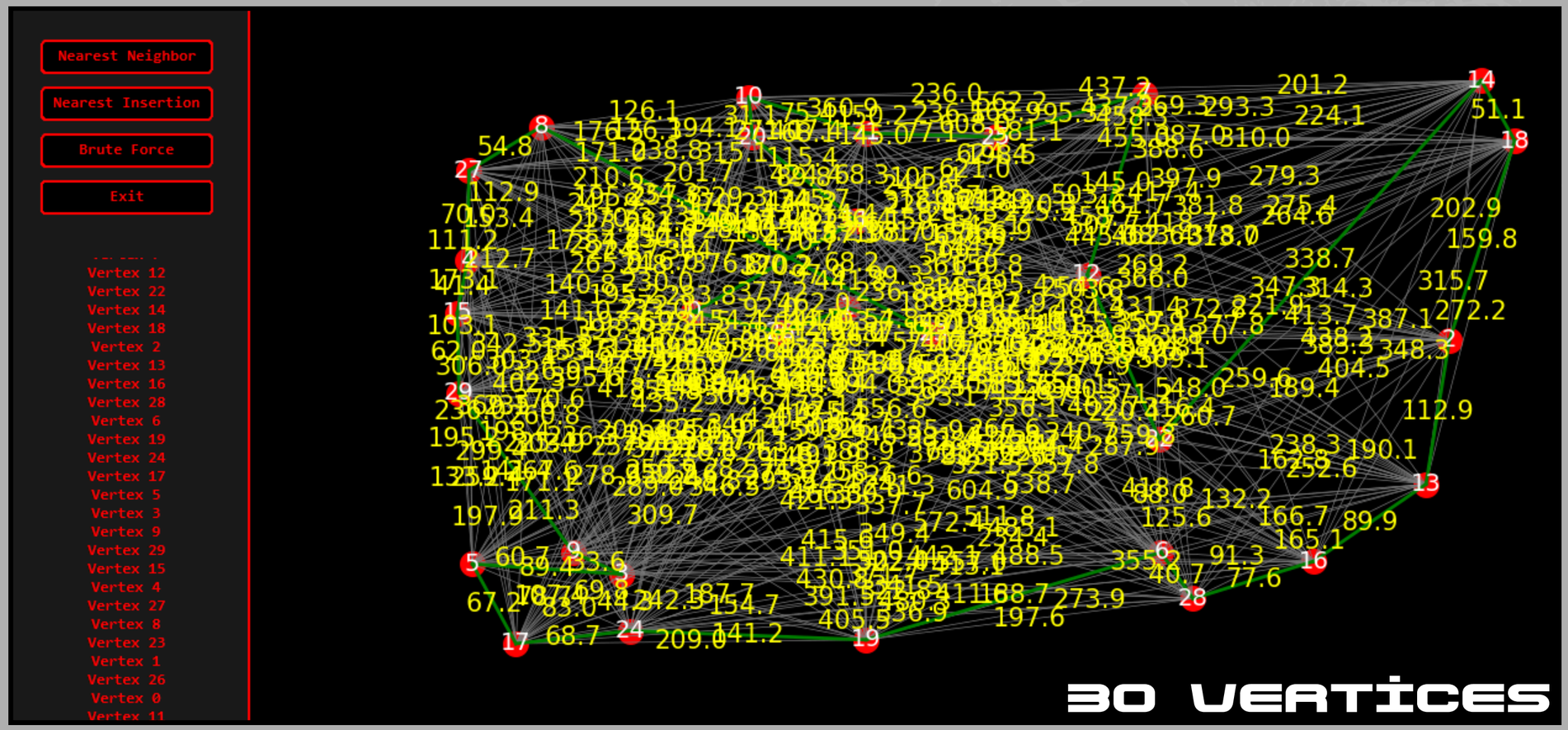
NEAREST INSERTION



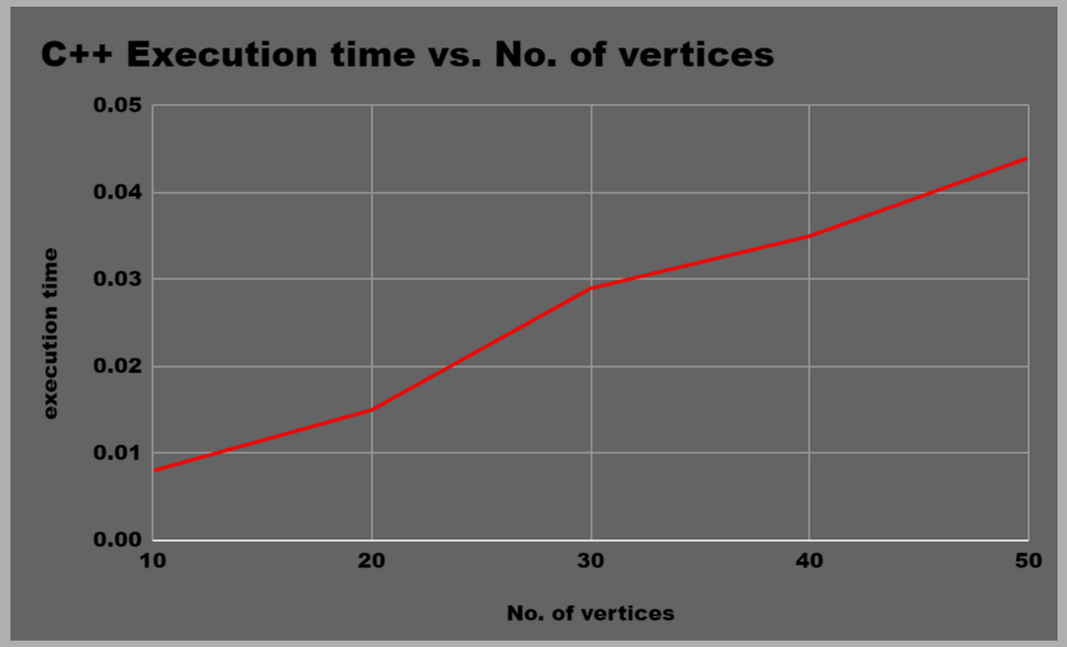
NEAREST INSERTION

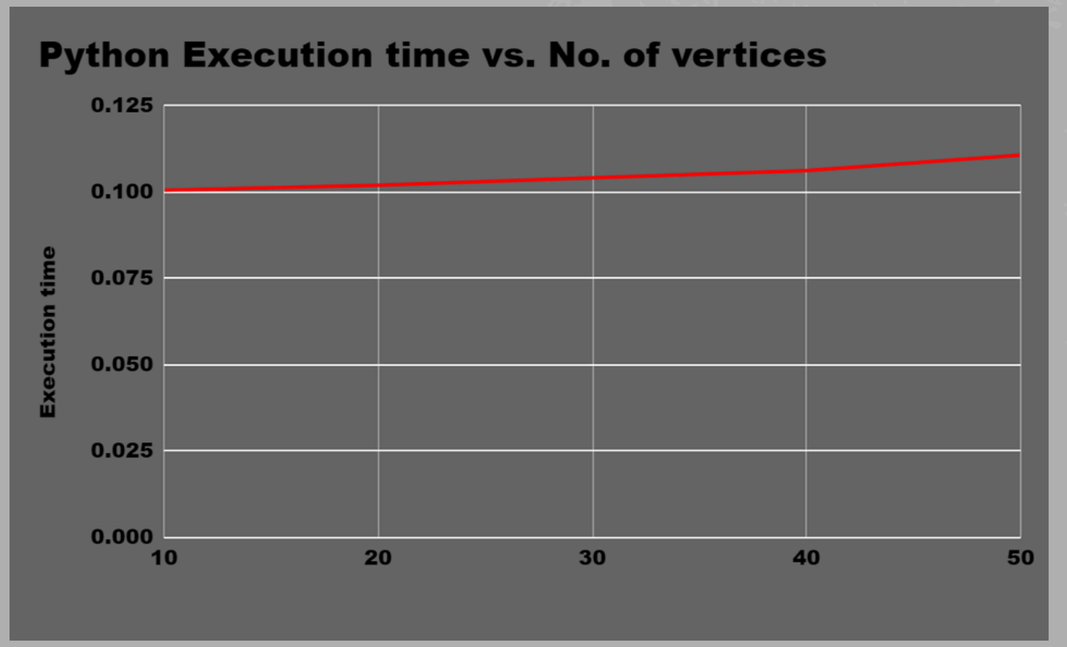


NEAREST INSERTION

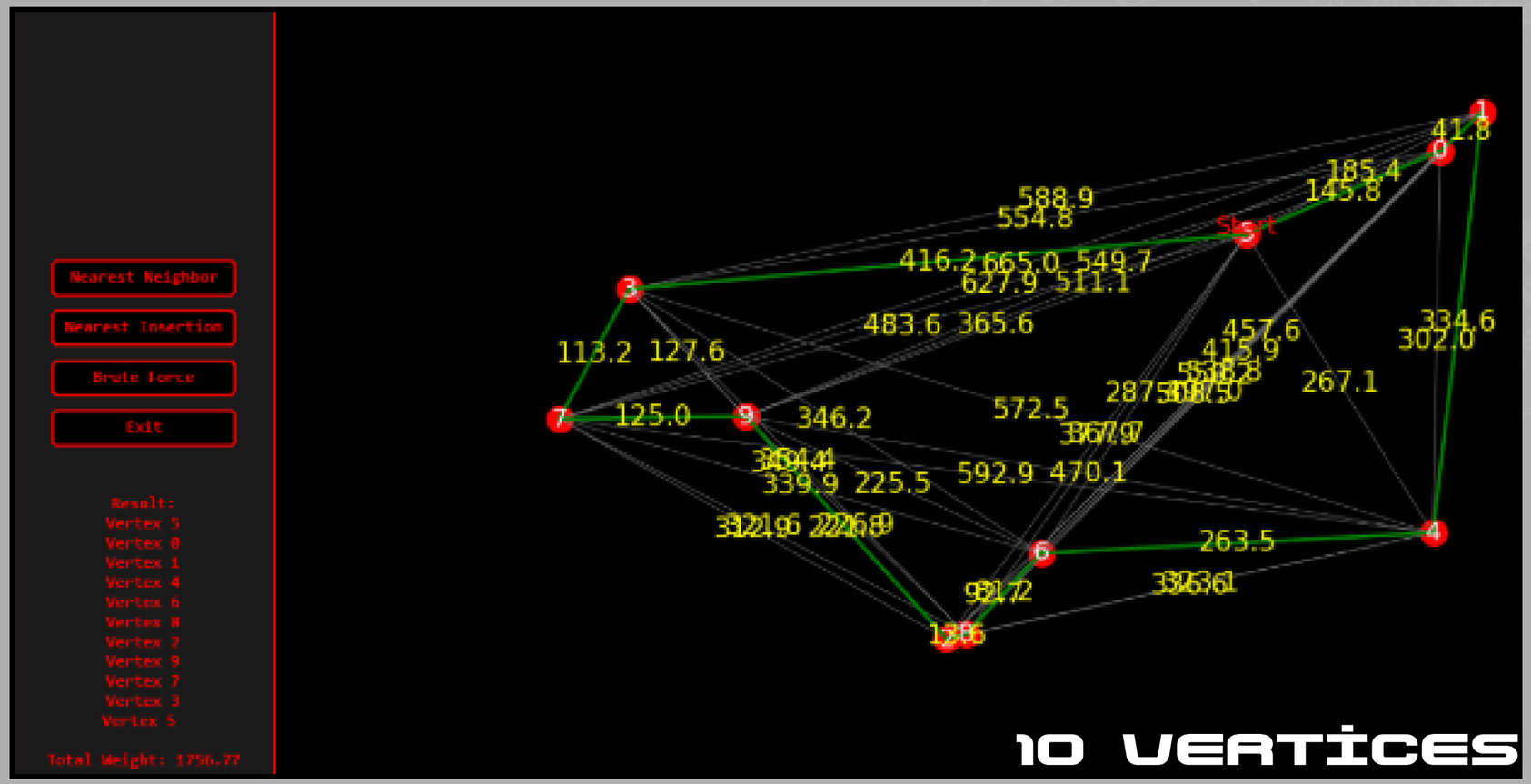


NEAREST NEIGHBOR

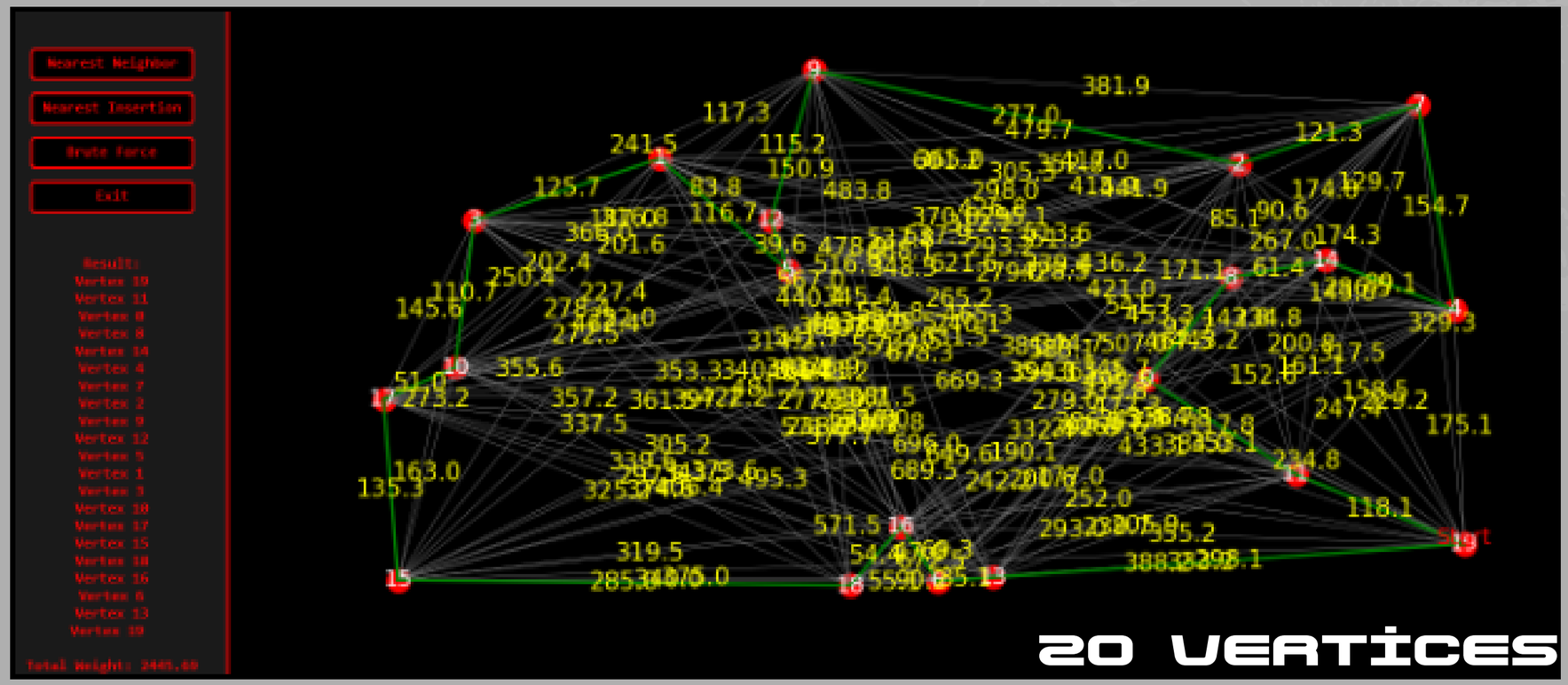




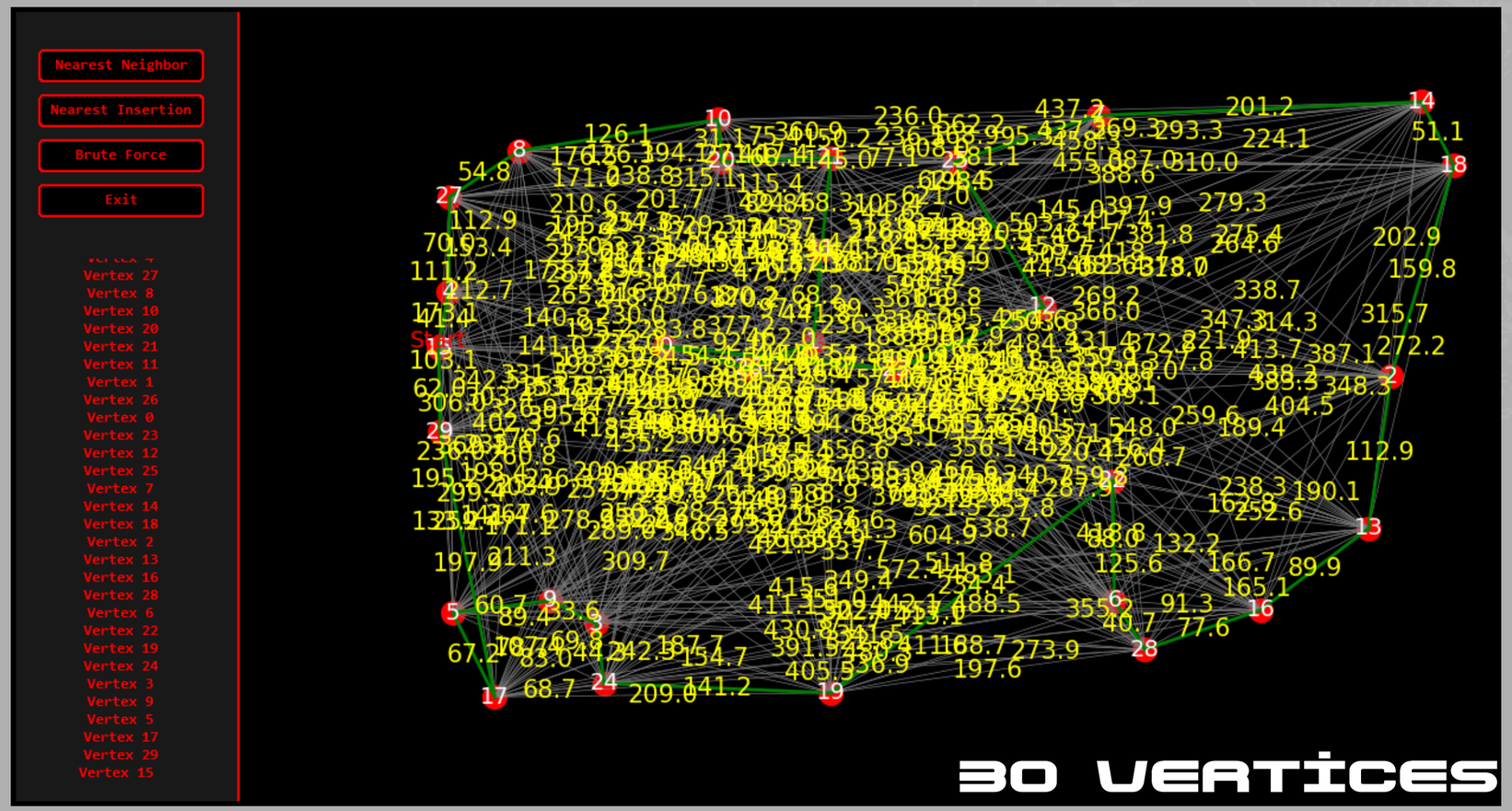
NEAREST NEIGHBOR

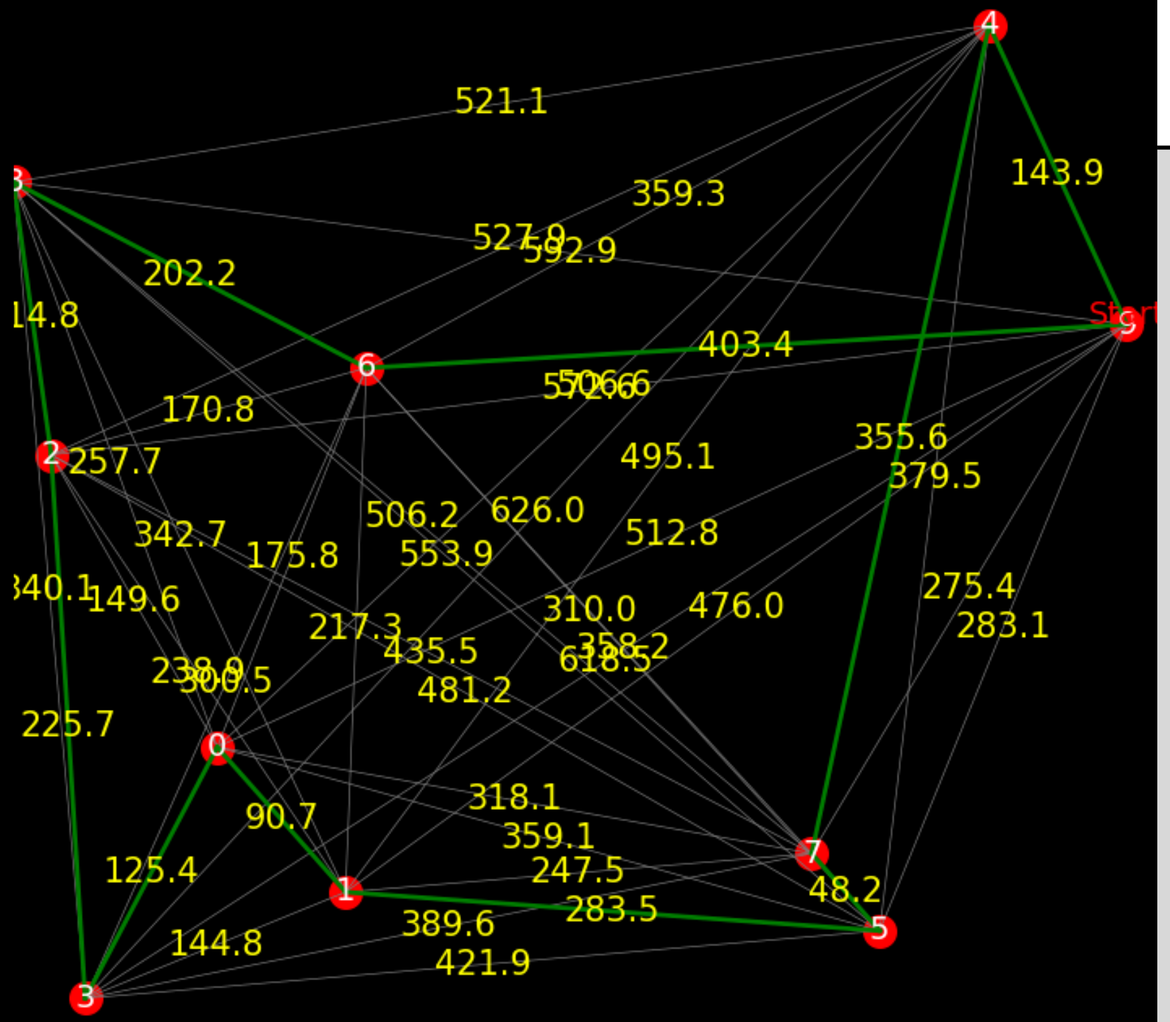


NEAREST NEIGHBOR



NEAREST NEIGHBOR

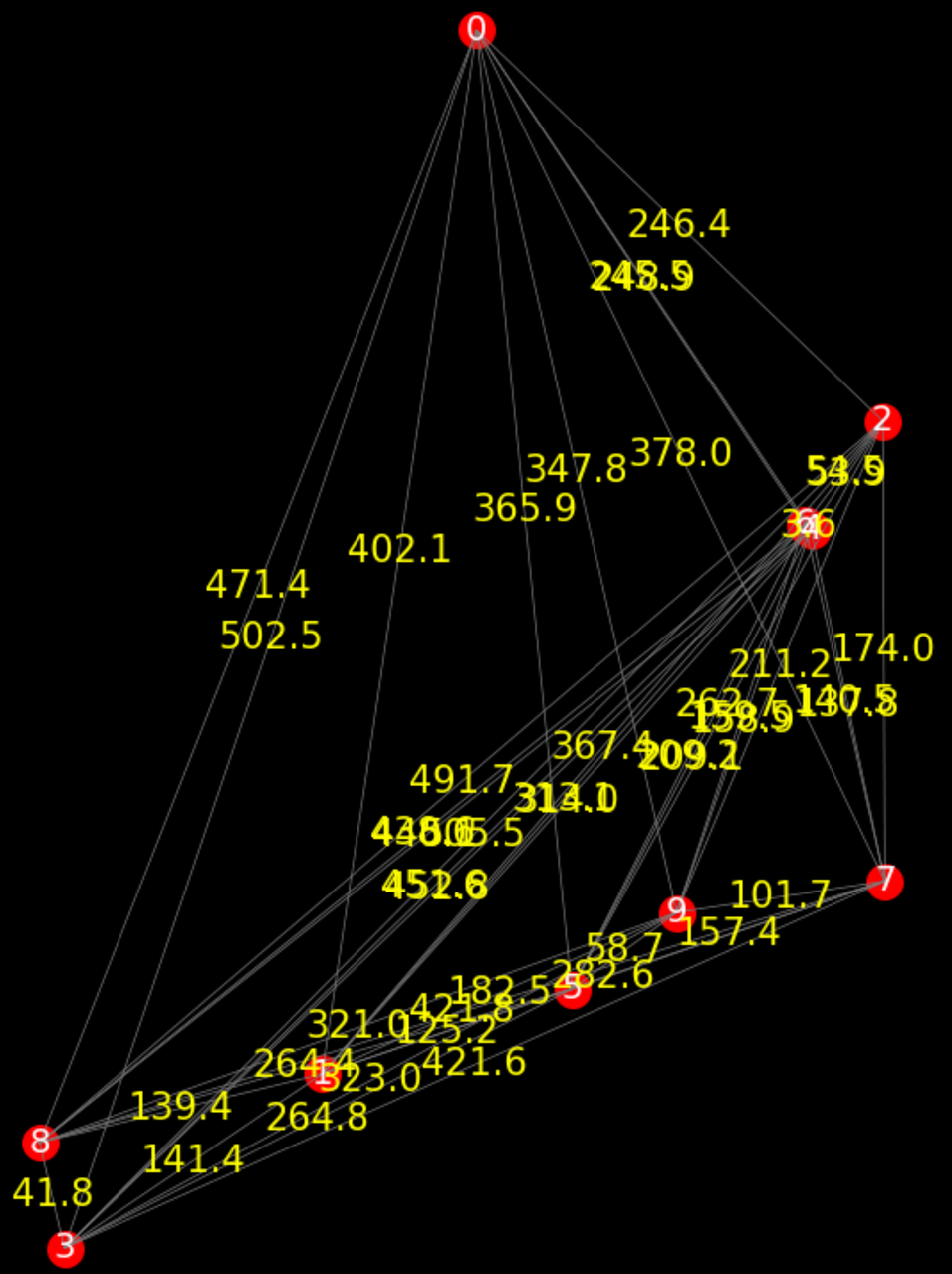




BREAKDOWN

- C++: C++ is faster because it is compiled, meaning it turns your code into machine instructions that the computer understands directly. This makes it very quick when performing heavy calculations, like solving the Traveling Salesman Problem (TSP).

- Python: Python is interpreted, meaning it reads and runs your code line by line. This takes longer because there's extra work involved in managing the code while it runs. Python also uses more memory, which can slow things down, especially with complex algorithms.



BREAKDOWN

- Python is easier to learn with simpler syntax and automatic memory management. It's slower in execution but great for web apps, data analysis, AI, and rapid development due to its vast libraries. It's portable and beginner-friendly.

- C++ is faster and provides more control over memory and system resources, making it ideal for performance-heavy tasks like game development and embedded systems. It has a steeper learning curve and requires manual memory management, but it offers greater control and efficiency.

THANKYOU