# On the efficiency of Dijkstra's Algorithm: An Empirical Study

by

**Ken Zyma**

Department of Computer Science
Kutztown University, Kutztown PA
email: kzym650@live.kutztown.edu

**Abstract:** In this paper, an implementation of Dijkstra's Algorithm is proposed which holds the textbook complexity of O(|V|+|E|)log(|V|). This complexity is proven by empirical results of execution time on 2,000 distinct graphs which vary by number of vertices, edge connectivity, and edge cost. Additionally, this paper demonstrates the use of data structures from the c++ standard template library, with a study of time complexity.

**Keywords:** Dijkstra's Algorithm, Minimum Cost Path, Data Structures

## 1.Introduction

Dijkstra's Algorithm (DA) continues to be one of the most celebrated algorithms in the field of computer science and operations research. Used to solve the single source minimum cost algorithm for graphs with non-negative cost edges, this greedy method allows the cost of a path from source to vertices to be relaxed, e.g. overestimated, and iteratively brought to the exact solution. A further explanation of DA is given below. One reason for popularity amongst the computer science community is due to it's association with special data structures (Sniedovich, 2006). Dijkstra's Algorithm can be implemented using an adjacency list or adjacency matrix, binary heap, priority queue, Fibonacci heap, ect. The scope of this work is to explore one implementation, and empirically measure it's efficiency, O(|V|+|E|)log(|V|), using a diverse set of graphs. Graph characteristics such as number of vertices, connectivity, and range of edge costs are isolated to show the effect on this implementation. Section 2 takes a closer look at Dijkstra's Algorithm and the data structures chosen to implement it. Section 3 explores the objectives for this research and how the experiment was constructed. In Section 4, empirical results are analyzed and results given. Lastly, Section 5 concludes this work.

## 2. Dijkstra's Algorithm

## 2.1 Problem Description – Single Source Shortest Path (SSSP)

For each vertex, $V_{1,2..N}$ in a directed graph, G = (V,E), find the minimum cost path from a source Vertex, $V_{source}$.

## 2.2. Dijkstra's Algorithm Overview

First assign every vertex's cost other than the source vertex, $V_{source}$ which is assigned zero, a relaxed cost of infinity. This value will be iteratively improved until the vertex is visited, in which case it will represent the minimum cost path. For implementation, use an ambiguously large number such as maximum integer on the system. Also, set all vertices to an “unvisited” state. Dijkstra's Algorithm consists of N iterations from which each node is visited in order of minimum cost (relaxed) first. While a vertex is visited, the cost to each adjacent vertice is updated if the cost (cost of current path + cost of edge connecting current and adjacent vertex) is lower than the relaxed cost. Note this also means previously visited vertices do not need to be evaluated since they much already represent the minimum cost path. After N iterations, all vertices will be visited and the algorithm terminates. Also, any unconnected vertices of graph (V,E) will still have the initial cost set (infinity or some high value) (Magzhan et al. 2013).

## 2.3 Data Structures for our DA

The choice of data structures for Dijkstra's Algorithm is key to achieving a big O value of (|V|+|E|)log(|V|). The graph was represented using an adjacency lists, where the input file is in the form:

<number of vertices>
<$source_1$><$dest_1$><$cost_1$>
<$source_2$><$dest_2$><$cost_2$>
… … ...
<$source_n$><$dest_n$><$cost_n$>

This implementation of DA is in c++, so from here on out I will speak of the data structures used in terms of the c++ STL. To represent graph (V,E), an STL multimap was used mapping the key, vertex index, to an “edge”, e.g. cost and destination vertex struct. This allowed adding edges, i.e. multimap insertion, in logarithmic time. Additionally, all edges from a vertice can be returned in O(|E|*log(|V|)) amortized time. A complexity of O(log|V|) for key lookup in the multimap and |E| for the number of edges associated with that vertice. An STL vector of edges is returned for simple manipulation (created in |E| time which drops from our big-Oh analysis).

Dijkstra's Algorithm uses a collection of ordered vertices, where each should store it's state [visited/unvisited] and the current minimum cost path from source. A user defined class, named Path, is used for this where the path is stored in an STL vector for efficient appending. This allowed for constant amortized insertion of a new vertex to the path. Every other operation of Class Path is constant time such as setCost, getCost, and isMarked. Additionally, non-member overloaded operator for stream insertion is O(n).

During DA's execution, two other data structures are utilized. A multimap is used to store a collection of class Paths, which maps a vertex number to Path. This gives us all of the advantages of lookup speed mentioned in the previous section. An STL priority queue holds a pointer to the multimap Path's and is used to return a pointer to the minimum path cost vertex at every iteration of the algorithm. The disadvantage of this approach, using pointers, is that anytime front() is called make_heap() must also be called since the cost of a Path may have been altered. Make_heap()' s worst case complexity is linear, however, this will not typically be the case so it is still more efficient than just using a vector or array, O(n) for minimum element retrieval.

**3. Objectives and Methodology**

The purpose of this work is to empirically evaluate and prove the performance of our implementation of DA, versus the classic textbook big O evaluation of (|V|+|E|)log(|V|). To this end, a diverse set of benchmark graphs was generated with the following three characteristics:

(i): The number of vertices tested is 10,20,40,80,160,320,640,1280, and 2560. This gave a good range of small to medium sized graphs.

(ii): The connectivity ranged from out degree of 1 to being fully connected, with three evenly spaced instances between these, for 5 distinct possibilities.

(iii):The cost of an edge took values of 1 (shortest path problem), 1-10, 1-100, and 1-1000. Based on our big O analysis of Dijkstra's Algorithm this should have no effect, however, is included for completeness.

All combinations of these graphs are tested, which allows isolation and analysis of each characteristic and it's impact on our DA. Additionally, 40 distinct graphs of each combination is generated, where the average time is recorded. All time is recorded in microseconds.

## 4. Empirical Results

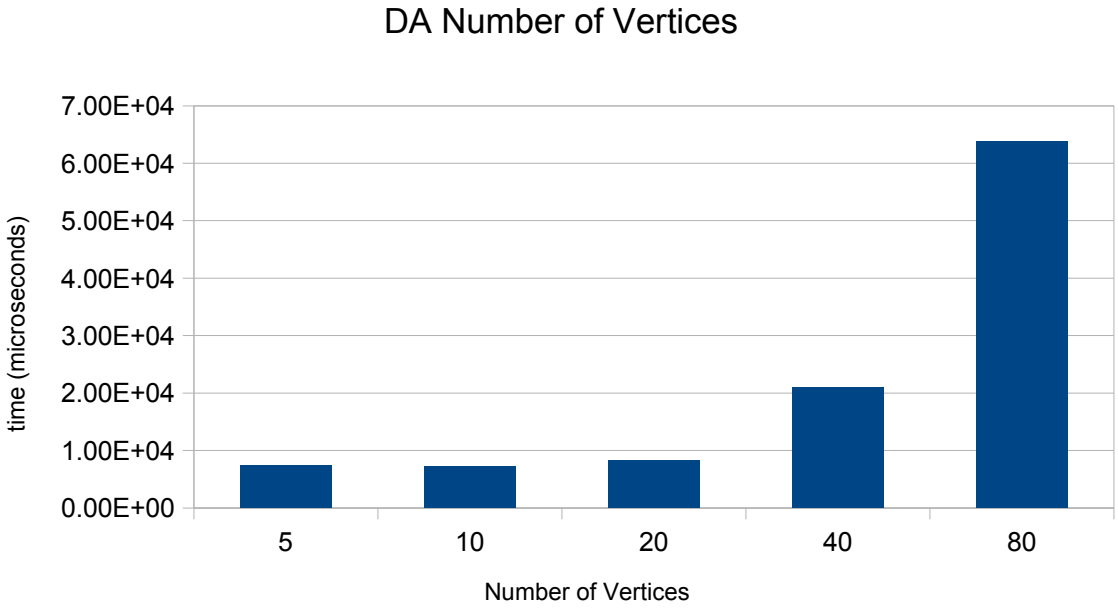
For a full summary of all results gathered from this experiment please refer to Appendix A. Shown below is a summary page of our results gathered:

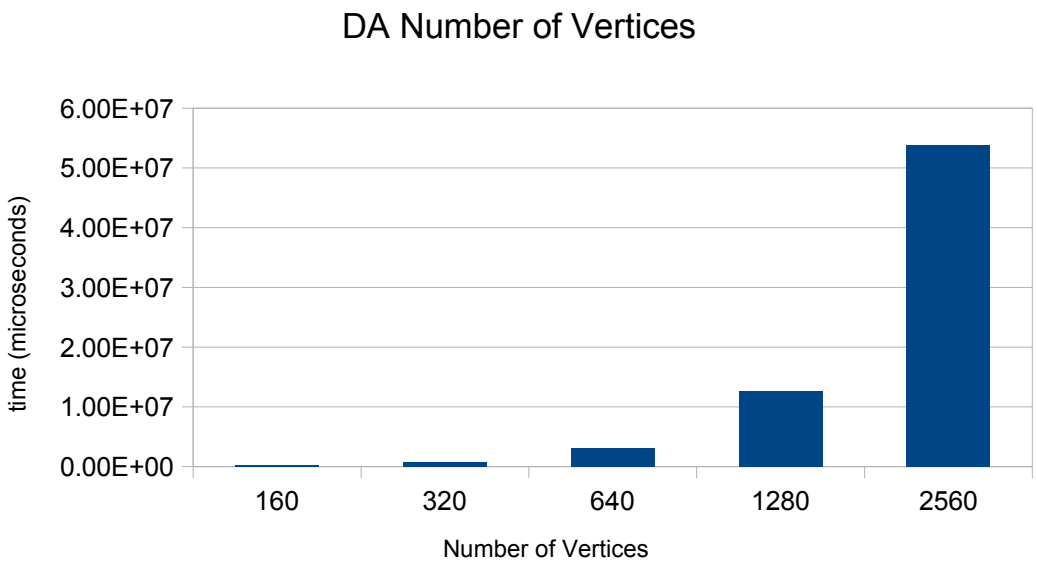
fig 1. Results

| | Out Degree | | | | | |
|---|---|---|---|---|---|---|
| Vertices | 1 | (n-1)/4 | (n-1)/2 | 3(n-1)/4 | n-1 | Total: |
| 5 | 3.22E+03 | 1.32E+02 | 1.99E+03 | 1.08E+03 | 9.46E+02 | 7.36E+03 |
| 10 | 1.77E+03 | 1.02E+03 | 1.38E+03 | 1.68E+03 | 1.35E+03 | 7.20E+03 |
| 20 | 1.74E+03 | 1.18E+03 | 1.74E+03 | 1.55E+03 | 2.02E+03 | 8.23E+03 |
| 40 | 2.13E+03 | 2.37E+03 | 3.49E+03 | 6.34E+03 | 6.64E+03 | 2.10E+04 |
| 80 | 3.80E+03 | 6.38E+03 | 1.27E+04 | 1.99E+04 | 2.09E+04 | 6.37E+04 |
| 160 | 4.32E+03 | 2.92E+04 | 4.56E+04 | 6.81E+04 | 8.02E+04 | 2.27E+05 |
| 320 | 2.92E+03 | 8.92E+04 | 1.59E+05 | 2.32E+05 | 2.87E+05 | 7.70E+05 |
| 640 | 8.24E+03 | 3.31E+05 | 6.24E+05 | 9.13E+05 | 1.16E+06 | 3.03E+06 |
| 1280 | 6.80E+03 | 1.40E+06 | 2.63E+06 | 3.83E+06 | 4.80E+06 | 1.27E+07 |
| 2560 | 1.12E+04 | 6.00E+06 | 1.12E+07 | 1.63E+07 | 2.02E+07 | 5.37E+07 |
| Total: | 4.62E+04 | 7.87E+06 | 1.47E+07 | 2.14E+07 | 2.66E+07 | 7.05E+07 |

The runtime is shown in microsecond's as the average of three tables given in Appendix A where cost was set to the ranges 1, 1-10, 1-100, and 1-1000. This also gives us the total runtime for all instances given a fixed number of vertices (and out degree varies) and a fixed out degree (where number of vertices varies).

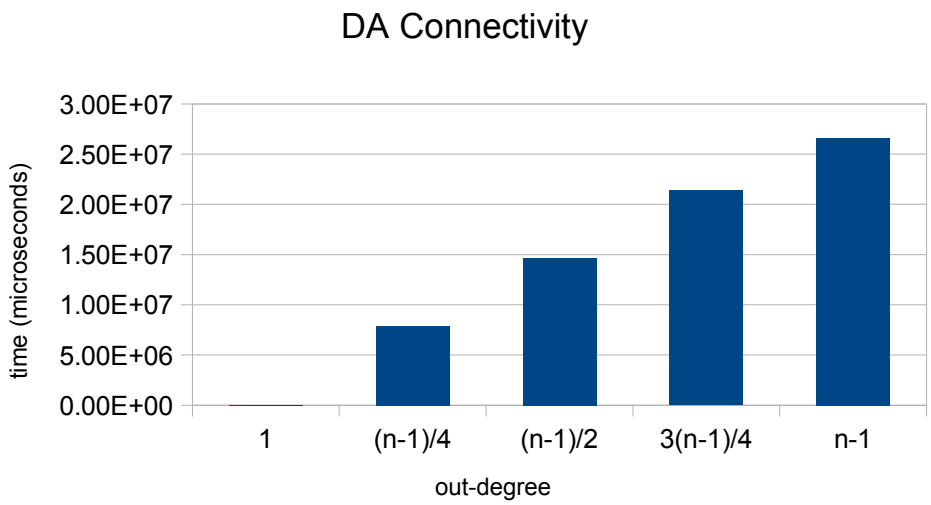
First, we look at how the number of vertices in a graph relates to execution time. In order to better see our results, these have been split into two graphs, the bottom 5 cases, 5-80 vertices, and the top 10, 160-2560.





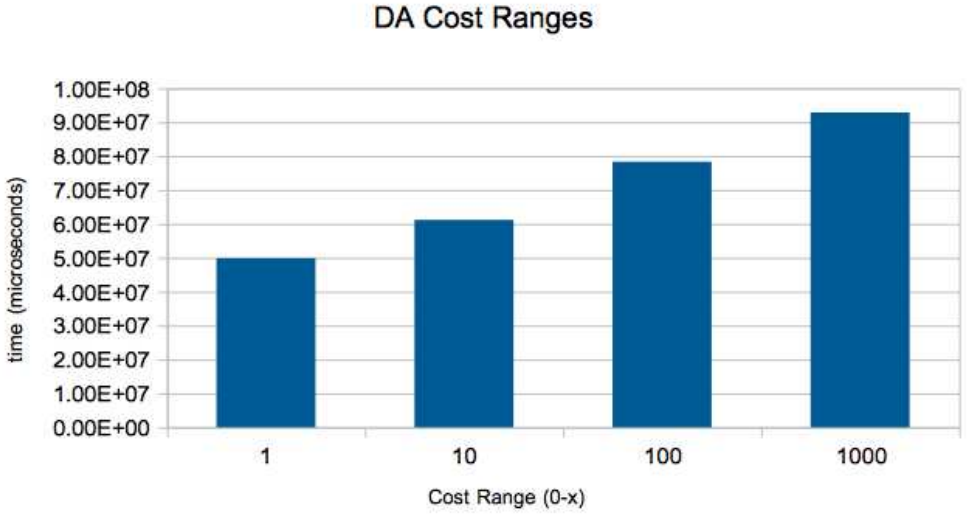

The out degree and cost for all of these cases are the same, while number of vertices is varied. Keep in mind that the number of vertices is doubled for each case, so while the growth appears exponential, this is not the case. Results shown above is what you would expect for O(|V|+|E|)log(|V|), as each time the number of vertices is doubled, runtime is roughly doubled. Keep in mind there is some deviation in cases 5, 10 and 20 due most likely to the fact that once we take a constant startup time(creating objects in c++, function call overhead, ect.) into account, that the actual algorithm O(|V|+|E|)log(|V|) is not noticeable enough to prove anything by runtime. Remember, these times are, after all, on the scale of microsecond's. Additionally, an interesting result shown is as number of vertices rise, they tend to grow further upward from our big-O analysis. This may be attributed to the constant amortized time of creating a vector's, where time of insertion may rise from constant to O(n) to resize the container.

Next, to prove O(|V|+|E|)log(|V|), we look at connectivity, e.g. |E|, the out-degree of our graph. We expect to see that as the graph goes from $|E| = |V|$ to $|V|^2$ , or fully connected, that runtime increases by the same factor.




Above, the number of vertices and cost is the same for all test cases, while out-degree is allowed to vary between five distinct cases. The results shown support my hypothesis, with little deviation from expected, where as connectivity increases execution time increases proportionally.

The last graph characteristic to explore is cost range. As mentioned previously, my hypothesis stated this should exhibit no change in execution time. Below are the results:




Interestingly, although small there does seem to be a correlation between cost increase and an increase in runtime. Keep in mind again the scale being used is a difference of milliseconds. Further testing should be done to verify the correctness of this, however, is most likely due to the increase in time for mathematical functions such as adding path

cost's at execution time with the larger values.

## 5. Conclusion

In this paper, an implementation of Dijstra's Algorithm (DA) was given that was believed to hold the textbook complexity of O(|V|+|E|)log(|V|). The effect of number of vertices, connectivity, and edge cost's for 2,000 distinct graphs was tested on DA to prove this complexity. Based on empirical results it is shown this implementation has the complexity of O(|V|+|E|)log(|V|).

## Appendix A

fig. 1 Cost is 1 (min-path)

| | Out Degree | | | | | |
|---|---|---|---|---|---|---|
| Vertices | 1 | (n-1)/4 | (n-1)/2 | 3(n-1)/4 | n-1 | Total: |
| 5 | 1.49E+02 | 1.34E+02 | 1.34E+03 | 1.17E+03 | 4.53E+02 | 3.25E+03 |
| 10 | 8.81E+02 | 4.31E+02 | 9.42E+02 | 5.17E+02 | 1.38E+03 | 4.15E+03 |
| 20 | 9.62E+02 | 5.98E+02 | 7.96E+02 | 9.96E+02 | 1.16E+03 | 4.52E+03 |
| 40 | 1.18E+03 | 1.29E+03 | 2.34E+03 | 1.25E+04 | 5.29E+03 | 2.26E+04 |
| 80 | 2.10E+03 | 4.74E+03 | 7.53E+03 | 1.09E+04 | 1.58E+04 | 4.11E+04 |
| 160 | 2.22E+03 | 2.26E+04 | 3.06E+04 | 4.80E+04 | 6.78E+04 | 1.71E+05 |
| 320 | 2.51E+03 | 7.51E+04 | 1.27E+05 | 1.80E+05 | 2.27E+05 | 6.11E+05 |
| 640 | 1.33E+04 | 2.41E+05 | 4.60E+05 | 6.83E+05 | 8.92E+05 | 2.29E+06 |
| 1280 | 4.93E+03 | 9.48E+05 | 1.84E+06 | 2.75E+06 | 3.58E+06 | 9.12E+06 |
| 2560 | 1.06E+04 | 3.83E+06 | 7.60E+06 | 1.14E+07 | 1.48E+07 | 3.76E+07 |
| Total: | 3.88E+04 | 5.13E+06 | 1.01E+07 | 1.51E+07 | 1.96E+07 | 4.99E+07 |

fig. 2 Cost ranges from 1-10

| | Out Degree | | | | | |
|---|---|---|---|---|---|---|
| Vertices | 1 | (n-1)/4 | (n-1)/2 | 3(n-1)/4 | n-1 | Total: |
| 5 | 6.61E+03 | 1.36E+02 | 1.28E+03 | 1.10E+03 | 4.61E+02 | 9.58E+03 |
| 10 | 2.03E+03 | 1.92E+03 | 1.31E+03 | 6.42E+02 | 6.60E+02 | 6.56E+03 |
| 20 | 2.34E+03 | 1.30E+03 | 1.70E+03 | 2.02E+03 | 1.79E+03 | 9.15E+03 |
| 40 | 1.14E+03 | 3.68E+03 | 3.52E+03 | 4.10E+03 | 7.06E+03 | 1.95E+04 |
| 80 | 5.64E+03 | 8.33E+03 | 1.27E+04 | 1.38E+04 | 1.67E+04 | 5.72E+04 |
| 160 | 6.34E+03 | 2.79E+04 | 4.81E+04 | 7.51E+04 | 7.71E+04 | 2.35E+05 |
| 320 | 3.13E+03 | 8.07E+04 | 1.46E+05 | 2.11E+05 | 2.61E+05 | 7.02E+05 |
| 640 | 7.21E+03 | 3.06E+05 | 5.60E+05 | 8.27E+05 | 1.04E+06 | 2.74E+06 |
| 1280 | 6.79E+03 | 1.20E+06 | 2.31E+06 | 3.38E+06 | 4.22E+06 | 1.11E+07 |
| 2560 | 1.28E+04 | 5.04E+06 | 9.65E+06 | 1.41E+07 | 1.75E+07 | 4.63E+07 |
| Total: | 5.41E+04 | 6.67E+06 | 1.27E+07 | 1.86E+07 | 2.31E+07 | 6.12E+07 |

fig. 3 Cost ranges from 1-100

| | Out Degree | | | | | |
|---|---|---|---|---|---|---|
| Vertices | 1 | (n-1)/4 | (n-1)/2 | 3(n-1)/4 | n-1 | Total: |
| 5 | 3.18E+03 | 1.31E+02 | 4.29E+02 | 4.05E+02 | 8.92E+02 | 5.03E+03 |
| 10 | 1.89E+03 | 4.82E+02 | 2.25E+03 | 1.60E+03 | 1.60E+03 | 7.83E+03 |
| 20 | 2.07E+03 | 1.43E+03 | 3.01E+03 | 1.14E+03 | 3.64E+03 | 1.13E+04 |
| 40 | 4.45E+03 | 2.96E+03 | 3.57E+03 | 4.63E+03 | 9.17E+03 | 2.48E+04 |
| 80 | 4.98E+03 | 5.82E+03 | 1.10E+04 | 2.10E+04 | 2.43E+04 | 6.71E+04 |
| 160 | 3.35E+03 | 3.32E+04 | 5.41E+04 | 7.19E+04 | 8.85E+04 | 2.51E+05 |
| 320 | 2.74E+03 | 9.56E+04 | 1.75E+05 | 2.63E+05 | 3.18E+05 | 8.54E+05 |
| 640 | 5.94E+03 | 3.83E+05 | 7.13E+05 | 1.02E+06 | 1.28E+06 | 3.40E+06 |
| 1280 | 8.25E+03 | 1.63E+06 | 2.97E+06 | 4.24E+06 | 5.26E+06 | 1.41E+07 |
| 2560 | 1.07E+04 | 6.95E+06 | 1.25E+07 | 1.81E+07 | 2.20E+07 | 5.96E+07 |
| Total: | 4.75E+04 | 9.11E+06 | 1.65E+07 | 2.37E+07 | 2.90E+07 | 7.83E+07 |

fig. 4 Cost ranges from 1-1000

| | Out Degree | | | | | |
|---|---|---|---|---|---|---|
| Vertices | 1 | (n-1)/4 | (n-1)/2 | 3(n-1)/4 | n-1 | Total: |
| 5 | 2.94E+03 | 1.25E+02 | 4.90E+03 | 1.65E+03 | 1.98E+03 | 1.16E+04 |
| 10 | 2.28E+03 | 1.26E+03 | 1.02E+03 | 3.95E+03 | 1.75E+03 | 1.03E+04 |
| 20 | 1.57E+03 | 1.39E+03 | 1.45E+03 | 2.05E+03 | 1.48E+03 | 7.94E+03 |
| 40 | 1.76E+03 | 1.53E+03 | 4.52E+03 | 4.18E+03 | 5.04E+03 | 1.70E+04 |
| 80 | 2.48E+03 | 6.63E+03 | 1.97E+04 | 3.41E+04 | 2.66E+04 | 8.95E+04 |
| 160 | 5.36E+03 | 3.32E+04 | 4.95E+04 | 7.74E+04 | 8.74E+04 | 2.53E+05 |
| 320 | 3.28E+03 | 1.05E+05 | 1.87E+05 | 2.73E+05 | 3.43E+05 | 9.12E+05 |
| 640 | 6.54E+03 | 3.94E+05 | 7.62E+05 | 1.13E+06 | 1.42E+06 | 3.71E+06 |
| 1280 | 7.23E+03 | 1.83E+06 | 3.40E+06 | 4.95E+06 | 6.14E+06 | 1.63E+07 |
| 2560 | 1.08E+04 | 8.18E+06 | 1.50E+07 | 2.16E+07 | 2.67E+07 | 7.15E+07 |
| Total: | 4.42E+04 | 1.06E+07 | 1.95E+07 | 2.81E+07 | 3.47E+07 | 9.28E+07 |

fig 5. Results summary, average of all four cost cases

| | Out Degree | | | | | |
|---|---|---|---|---|---|---|
| Vertices | 1 | (n-1)/4 | (n-1)/2 | 3(n-1)/4 | n-1 | Total: |
| 5 | 3.22E+03 | 1.32E+02 | 1.99E+03 | 1.08E+03 | 9.46E+02 | 7.36E+03 |
| 10 | 1.77E+03 | 1.02E+03 | 1.38E+03 | 1.68E+03 | 1.35E+03 | 7.20E+03 |
| 20 | 1.74E+03 | 1.18E+03 | 1.74E+03 | 1.55E+03 | 2.02E+03 | 8.23E+03 |
| 40 | 2.13E+03 | 2.37E+03 | 3.49E+03 | 6.34E+03 | 6.64E+03 | 2.10E+04 |
| 80 | 3.80E+03 | 6.38E+03 | 1.27E+04 | 1.99E+04 | 2.09E+04 | 6.37E+04 |
| 160 | 4.32E+03 | 2.92E+04 | 4.56E+04 | 6.81E+04 | 8.02E+04 | 2.27E+05 |
| 320 | 2.92E+03 | 8.92E+04 | 1.59E+05 | 2.32E+05 | 2.87E+05 | 7.70E+05 |
| 640 | 8.24E+03 | 3.31E+05 | 6.24E+05 | 9.13E+05 | 1.16E+06 | 3.03E+06 |
| 1280 | 6.80E+03 | 1.40E+06 | 2.63E+06 | 3.83E+06 | 4.80E+06 | 1.27E+07 |
| 2560 | 1.12E+04 | 6.00E+06 | 1.12E+07 | 1.63E+07 | 2.02E+07 | 5.37E+07 |
| Total: | 4.62E+04 | 7.87E+06 | 1.47E+07 | 2.14E+07 | 2.66E+07 | 7.05E+07 |

**References**

Sniedovich M. (2006). Dijstra's algorithm revisited: the dynamic programming connexion. Control and Cybernetics v. 35.3.

Magzhan K., Jani H. (2013). A review and evaluations of shortest path algorithms. International Journal of Scientific and Technology Research. v. 2.6. 2277-8616