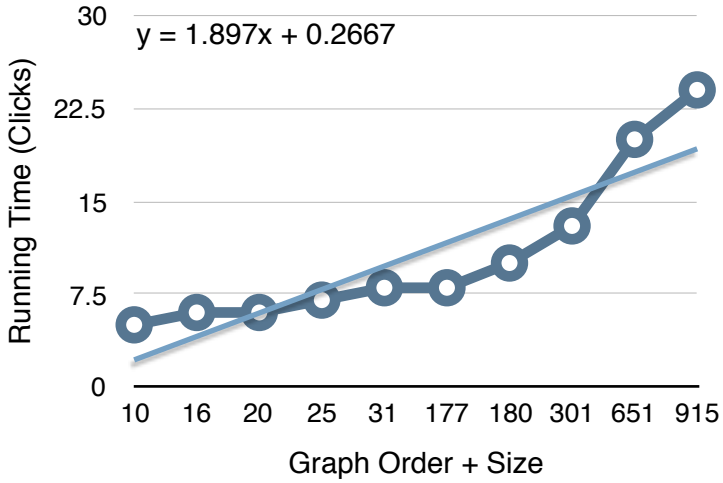
The following report details an analysis of the efficiency in terms of run time of two examples of topological sorting algorithms. The two algorithms examined are Depth-First Search (DFS) and Kahn's Algorithm.

EFFICIENCY:
The efficiency of the two algorithms is to be analysed both theoretically and practically. The following will detail an analysis of the 'Big - O' running times of the algorithms, a numeric analysis of running time compared to input size and projective analysis based on running time trends displayed graphically.

DFS ALGORITHM
The DFS algorithm utilises the idea that a vertex of a graph need only be visited once to determine its place in a topological sorting, more accurately it posits that the set of operations needed to position a vertex in this order only need to be executed once per item and will be the same for all items. The source code for my implementation of the DFS algorithm can be found in the file toposort.c and will be the basis for my following analysis on the algorithm. The DFS uses the assumption that its method ensures every vertex is only visited once and if it is visited more than once implies the graph can't be topologically sorted, that is its a cyclic graph. In order to achieve this once only visitation DFS will only ever terminate a recursive call once all outgoing edges from its current vertex have been examined and uses a logic structure to show that an edge is either still under investigation (denoted visiting in my implementation) or has had all its dependencies and itself investigated (denoted visited in my implementation). This logic structure ensures no vertex will ever have a complete recursive call made once it has been fully investigated as the program will simply skip it and move to the next vertex to be called. This implies the running time of the algorithm is proportional to the number of vertices in the graph and so is O(V) in nature, however this isn't completely true, a look at the recursive call of DFS shows it also activates a call per edge from the vertex and so the running time is actually proportional to the sum of vertices and edges or O(V + E) . The below is a graph of the running time on inputs of varying size and shows time as dependant on the sum of order and size.



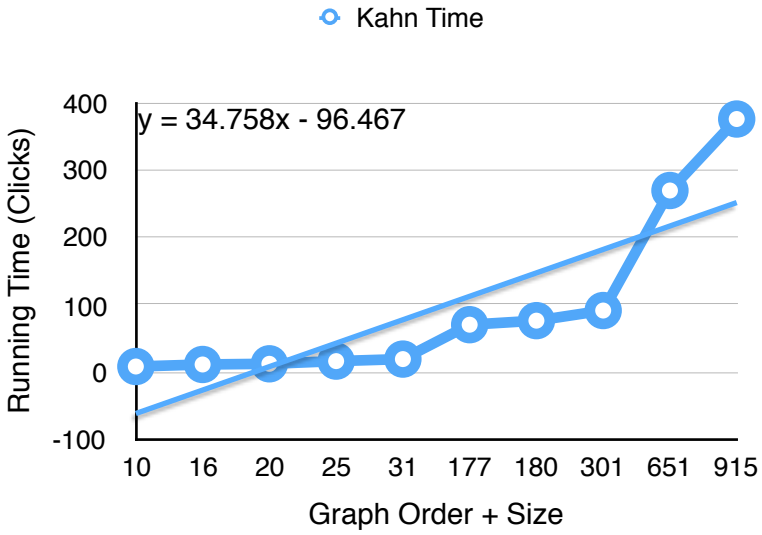
Running Time DFS

Order + Size	DFS Time
10	5
16	6
20	6
25	7
31	8
177	8
180	10
301	13
651	20
915	24

It's interesting to note here that the linear trend line does not actually form anywhere near to a direct match, in fact if one was to insert a polynomial trend line (Appendix A) it would become apparent this is a much closer match and may imply more about the run times of the data structure manipulations being performed within the DFS call, showing an area which could be analysed in order to increase efficiency and bring the algorithm closer to its theoretical running capabilities.

KAHN SORT ALGORITHM

The Kahn sort algorithm works on a very different premise to DFS and relies on the properties of a unique set of vertices known as sources. A source vertex is one that has no incoming edges, that is it is not dependant. With this property any edges from this vertex can confidently be put after it in a topological sort as they are all dependants and the vertex itself can be placed at the head of the sort list as it is not dependant. Kahn's algorithm seeks to impose this property on all vertices so that this form of analysis can be used to construct a sorted listed. The kahn sort algorithm first finds all vertices without edges an operation of O(V) time as it must perform a check on all vertices. Once this has been constructed one at a time these vertices are cycled through and removed from this list to be added to the end of the topologically sorted list. This selected vertex then has all its outgoing edges removed and if this causes any of its dependants to then become source-like they are in turn added to the list of sources and the next source is selected to be analysed. The afore-described process will eventually examine all vertices as all vertices will eventually become sources given the graph is acyclic so this is another O(V) operation, with a bit of imagination we can see that all the edges will also be examined as every edge must be removed in order to successfully perform a sort and so we have an operation of O(E) as well. Therefore the running time of this algorithm should theoretically be proportional to O(V) + O(V) + O(E) which can be simplified to the expression O(V + E). The below diagram displays the running time on increasing inputs.

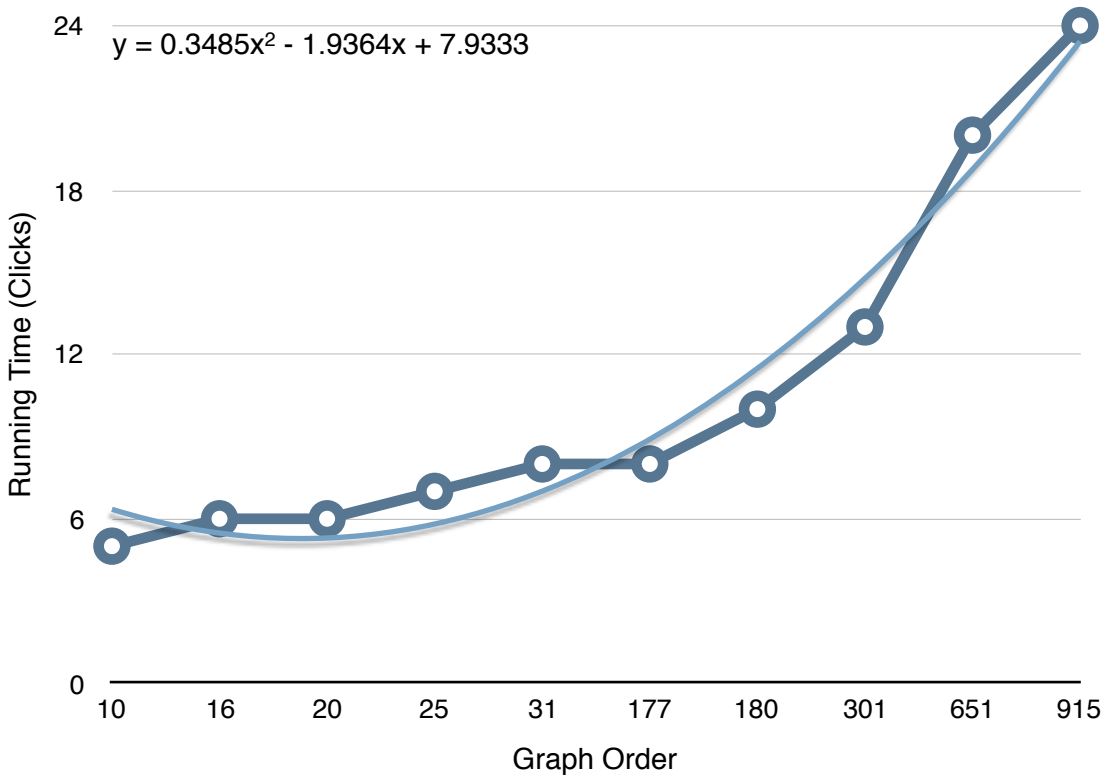


Running Time Kahn

Order + Size	Kahn Time
10	8
16	11
20	12
25	16
31	19
177	70
180	76
301	91
651	269
915	375

What is interesting to note from this graph is the obvious leap in running time between inputs using Kahn sort and DFS by a factor in some cases greater than 10. This implies that on inputs lower than 915 in total size (still a relatively small input size) that DFS would be a more efficient choice of algorithm to use. Probable causes of this gap in run time is the high-level data structure manipulations required to successfully run a Kahn sort. Unlike DFS which relied solely on boolean markers and recursive calls, Kahn sort, as I have written it relies on nested while loops (analogous to recursive structures) and also the suite of functions required to remove an edge from a graph. This action takes quite a lot of computing power and time requiring at least two calls for the outgoing and incoming sides of an edge and also while loop comparisons to find the right locations to delete in each list of edges respectively. It appears the simplicity of the DFS approach has done well to improve overall efficiency. Another note interest would be the apparent disjoint between the linear trend line and the graph of run times, an exponential trend line appears to fit much better (Appendix B) and may imply inefficiencies in the implementation of Kahn sort used in this program.

APPENDIX A:



APPENDIX B: