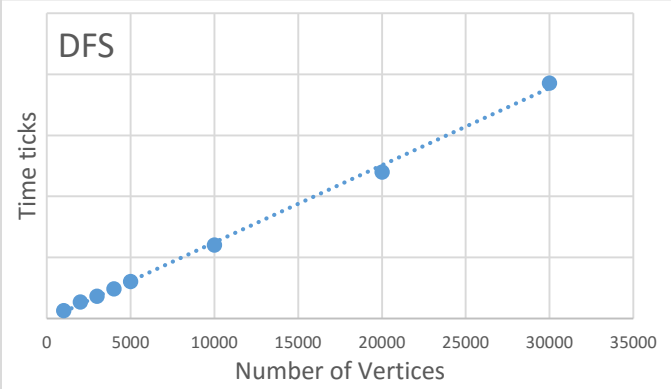
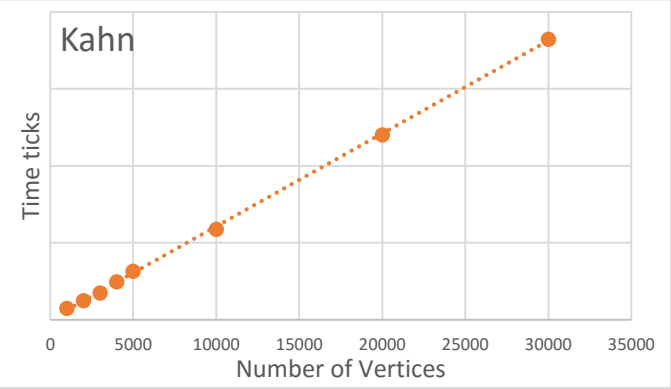
DFS and Kahn Sort Efficiency Analysis

Max Lee 719577
hoso1312@gmail.com

This report will prove and justify the relationship between number of vertices/edges and time taken by two algorithms, DFS and Kahn sort, showing that their efficiency is O(V+E) with appropriate assumptions. It will also discuss why Kahn sort shows quadratic increase with density of the edges and also determine the pathological inputs for both algorithms. All data in this report were generated randomly using daggen.

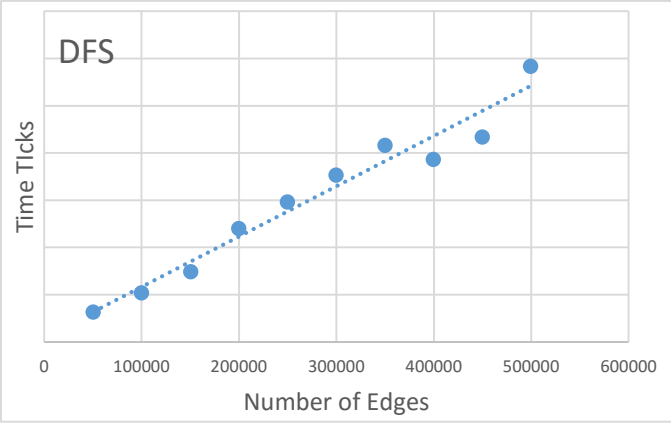
Both Kahn and DFS sorts show linear growth with number of vertices. The following graphs are plotted time (Y) against number of vertices (X), with fixed 0 edges for all data:

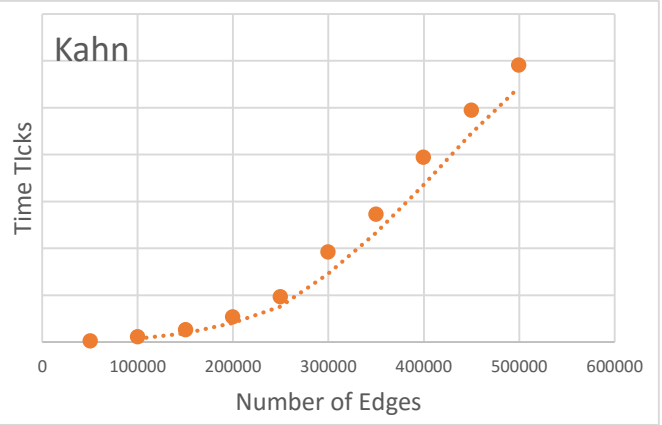




This reflects they both have efficiency O(V) with fixed number of edges. This logical since each vertex is processed only once in the algorithms.

With varying number of edges, two sorts show different behaviours. The following graph are number of edges against time, with fixed 1000 vertices for all data:





DFS shows linear relationship (therefore, efficiency of O(E).) but Kahn shows initial quadratic growth, followed by linear growth after a certain point. This behaviour can be proved in the following graphs using Kahn sort: Graph A compromises of data taken with increasing vertex count (therefore, increasing edge count), with each series having different edge creation probability. Graph B shows time against density, where data was extracted using 100% probability in daggen (Appendix). Note that it goes linear is due to rapid increase then stabilization in edge density (explained later).

Graph A shows that quadratic behaviour reduces with edge creation probability – which means when the edges are less dense, quadratic behaviour reduces. Graph B shows cubic relation between time and density, which explains the behaviour Graph A, affecting the expected linearity. This relationship with density is logical since more dense edges (ie. each vertex has more edges) will result in longer time going through linked in-list for each vertex in out-list when deleting a vertex.

From this fact (that Kahn sort has to go through linked lists), we know that Kahn sort takes significantly longer time with increasing size of the graph. It also takes about twice the time when there are no edges. This is reflected in the following data tables:

Vertices	*Edges*	*V+E*	*DFS*	*Kahn*
1000	0	1000	**64**	**150**
2000	0	2000	**136**	**248**
3000	0	3000	**183**	**349**
4000	0	4000	**243**	**496**
5000	0	5000	**302**	**632**

Vertices	*Edges*	*V+E*	*DFS*	*Kahn*
1000	50351	51351	**1268**	**28830**
1000	99978	100978	**2078**	**113183**
1000	150327	151327	**2976**	**265968**
1000	199677	200677	**4809**	**540094**
1000	249540	250540	**5934**	**972810**

Vertices	*Edges*	*V+E*	*DFS*	*Kahn*
100	4950	5050	**335**	**3642**
200	19900	20100	**390**	**16093**
300	44850	45150	**1050**	**58132**
400	79800	80200	**1620**	**148484**
500	124750	125250	**2531**	**297066**

This shows that there is room for improvement for Kahn sort. Instead of linked lists, if we had in/out lists as array where searching an element in sorted array takes O(log n) and updating O(1), edge deletion when deleting a vertex will be much faster. Using hashed array with vertex id will ideally make both search and update O(1).

In conclusion of overall efficiency. It is clear for DFS – it has efficiency of O(V+E) independent of density as shown in previous graphs. However, for Kahn, we need to assume that the density of the edges stay constant – which is a reasonable assumption as in real-life, a new vertex does not have chance of getting connected with all other existing vertices unlike daggen algorithm. With this, we get the following data >

Vertices	*Edges*	*V+E*	*Density*	*DFS*	*Kahn*	*%*
1000	199937	200937	199.937	4242	539320	40
2000	399854	401854	199.927	9841	2120270	20
4000	799117	803117	199.7793	19865	5620613	10
8000	1599085	1607085	199.8856	40824	13950959	5
20000	4001905	4021905	200.0953	101076	41031049	2

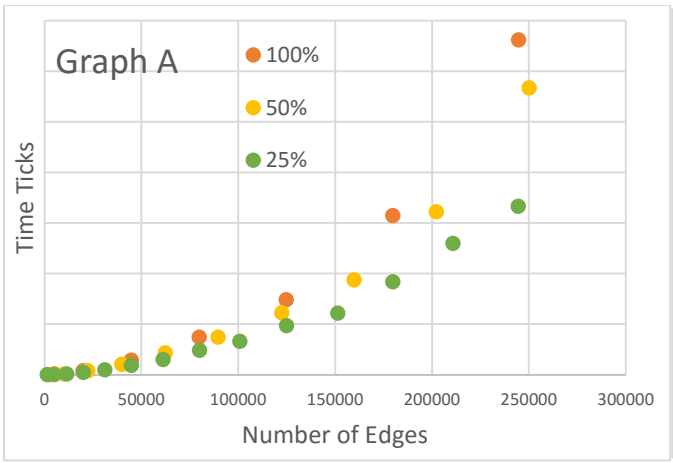
This is plotted into two graphs in the appendix, which are both linear graphs. Therefore, with reasonable assumption, both DFS and Kahn have efficiency of O(V+E). In real life, density will stabilize into constant at one point since gradient of E/V against V+E is E/(V(V+E)). Denominator growth > numerator growth.

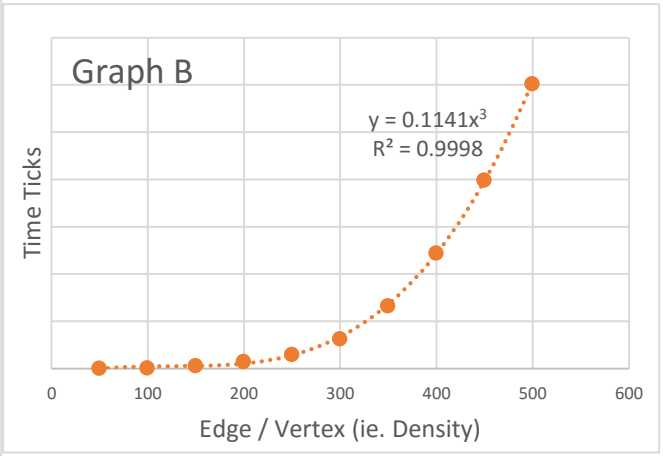
We can now conclude the pathological input for each sort algorithms. DFS does not have pathological input. The algorithm uses visited array which takes O(1) to search (by id) or update. It has no bottleneck in the design – no linked lists and size of input purely determines the run time. You cannot possibly make algorithm run over-time with same size of input.

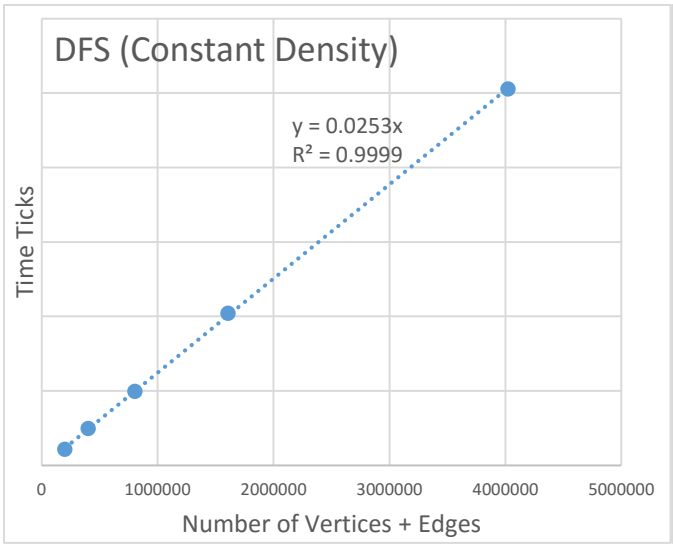
This is not the case for Kahn sort. Due to usage of linked list – the bottle neck – the arrangement of list affects the run time. The worst case for Kahn sort is a graph where all vertices are connected to all others and when deleting a vertex, the in-lists are arranged such that search for a vertex takes O(n) where n is the size of the list. Such generation of graph is extremely complex. Graph with increasing vertex count with 100% edge probability is satisfactory to represent the outcome of pathological input (appendix) as its difference is only by constant compared to real pathological input.

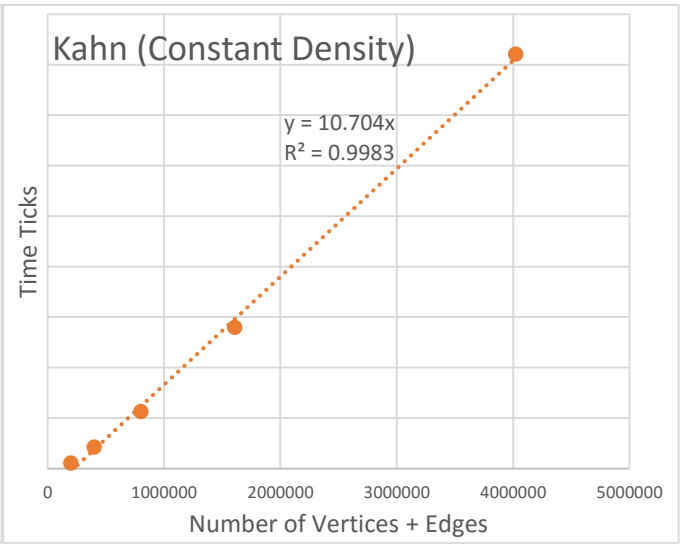
Since 100% probability means maximum edges, E=V*(V-1)/2 and since density has not stabilized, efficiency is O(V+E(E/4V)+(E/V)^3) – each vertex processed once and to eliminate an edge, each vertex has E/V edges on average, and half of them are in-edge on average (therefore E/2V) and processing one edge reduces the number of edges by 1 until 0 therefore E/4V.

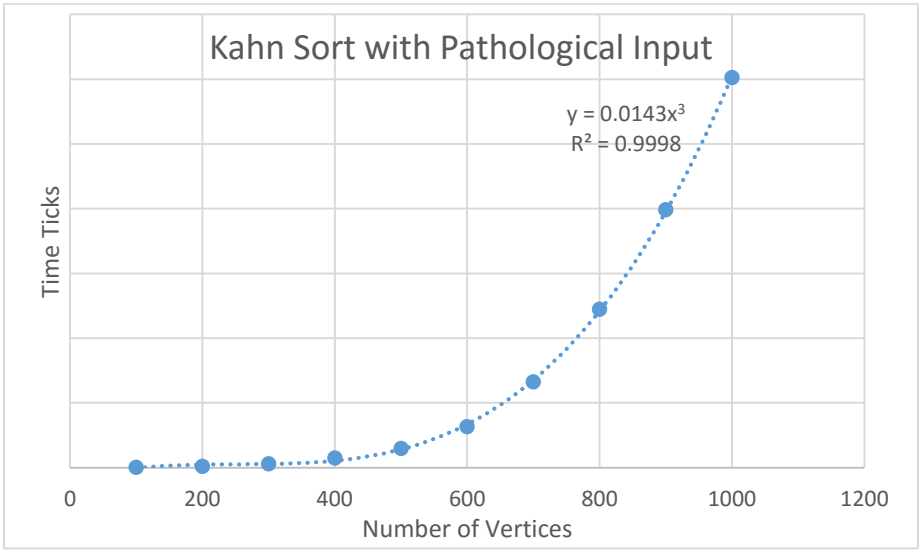
V+E(E/4V)+(E/V)^3 = V+V*(V-1)/2*((V-1)/8)+((V-1)/2)^3 = O(V^3). This is clearly reflected in the graph.











Vertices	*Edges*	*Time Ticks*
100	4950	3642
200	19900	16093
300	44850	58132
400	79800	148484
500	124750	297066
600	179700	629375
700	244650	1323268
800	319600	2444854
900	404550	3984463
1000	499500	6026060