COMP20007 Design of Algorithms
Semester 1 2016
Assignment 1
Ivan Ken Weng Chee 736901
Topological Sorting Algorithms Analysis

Depth-First Search Algorithm (dfs_sort)[1]

Description :

```
                                                        Time    Space
dfs <- empty list to contain sorted vertices                    O(V)
boolean <- array of structs to keep track of vertices           O(V)
for each unmarked vertex V in graph do                  O(V)
        visit(V)

function visit_node(vertex V)
        if V has temporary mark or permanent mark
                return
        mark V temporarily                              O(1)
        for each vertex W with an edge E from V to W do O(E)
                visit(W)
        mark V permanently                              O(1)
        unmark V temporarily                            O(1)
        prepend V to dfs                                O(1)
return dfs
```

Complexity :

Time : $O(V) + O(E) + 4 * O(1) \in O(|V| + |E|)$
The algorithm visits every vertex only once. Visited edges are not processed in the visit function.

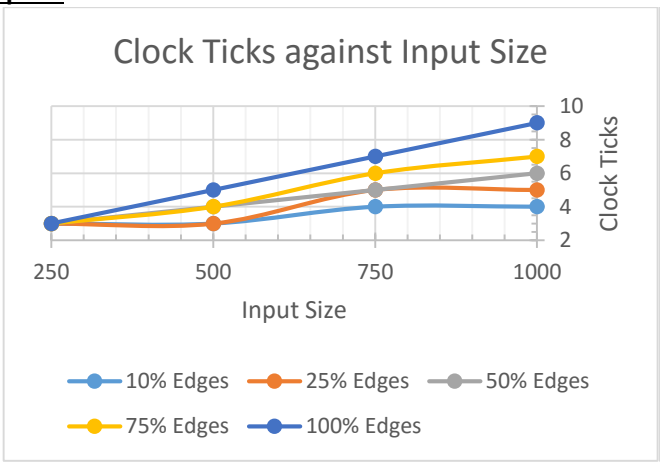
Space : $2 * O(V) \in O(V)$
Topological sort of the graph requires storing an equal number of vertices in the graph

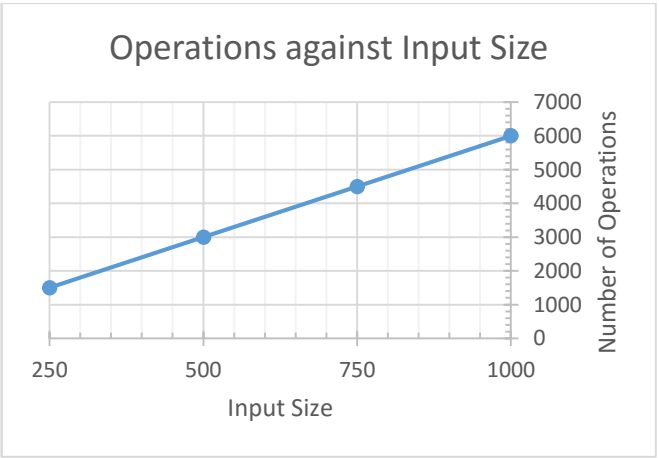
Pathological Inputs :
No pathological input exists for this particular algorithm since its best case equals its worst case.

Timing Results :

Input Size	10% Edges		25% Edges		50% Edges		75% Edges		100% Edges	
	Clock Ticks	Operations	Clock Ticks	Operations	Clock Ticks	Operations	Clock Ticks	Operations	Clock Ticks	Operations
250	3	1500	3	1500	3	1500	3	1500	3	1500
500	3	3000	3	3000	4	3000	4	3000	5	3000
750	4	4500	5	4500	5	4500	6	4500	7	4500
1000	4	6000	5	6000	6	6000	7	6000	9	6000

Graphs :





Description :

	Time	Space
kahn <- empty list to contain sorted vertices		O(V)
source <- empty list to contain vertices without incoming edges		O(V)
for all vertices in graph do	O(V)	
if vertex has no incoming edges		
prepend to source	O(1)	
while source contains vertices do	O(V)	
pop vertex V from source	O(1)	
prepend V to kahn	O(1)	
for each vertex W with an edge E from V to W:	O(E)	
pop W from V's outgoing edges	O(1)	
delete V from W's incoming edges	O(E)	
if W has no more incoming edges		
prepend W to source	O(1)	
return reversed(kahn)	O(V)	

Complexity :

Time : [O(V) + O(1)] + [O(V) * [O(E)] ^ 2 + 4O(1)] + [O(V)] ∈ O(|V| * |E|^2),
Algorithm is asymptotically much slower than the O(|V| + |E|) in Wikipedia.

Space : 2 * O(V) ∈ O(V)
Topological sorting of the graph requires V vertices to be stored, and at most V source vertices.

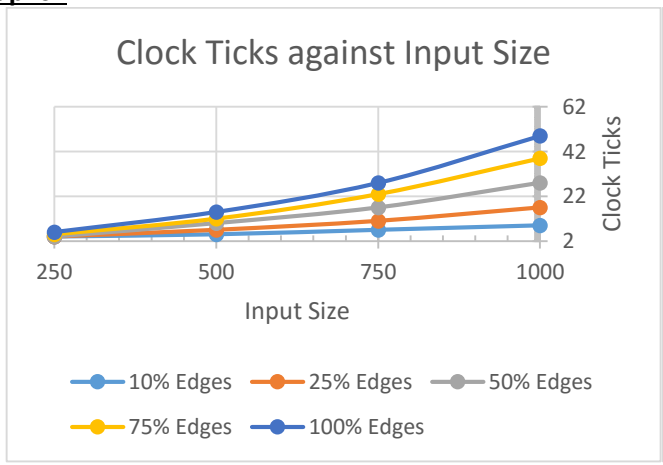
Pathological Inputs :

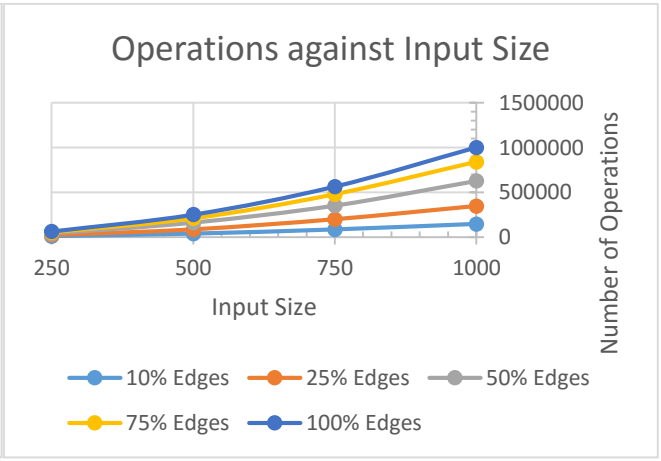
Worst case behaviour when graph has the maximum number of edges - all vertices in a graph are connected to each other. Maximum of E(E-1)(E-2)(E-3)… = E! edges

Timing Results :

Input Size	10% Edges		25% Edges		50% Edges		75% Edges		100% Edges	
	Clock Ticks	Operations	Clock Ticks	Operations	Clock Ticks	Operations	Clock Ticks	Operations	Clock Ticks	Operations
250	4	9371	4	22297	4	38974	5	53643	6	63000
500	5	36868	7	86059	10	160544	12	208448	15	251000
750	7	83885	11	198696	17	351229	23	478510	28	564000
1000	9	145657	17	346141	28	627517	39	840609	49	1002000

Graphs :





Reference :

[1]https://en.wikipedia.org/wiki/Topological_sorting#Kahn.27s_algorithm