At the beginning I want to say that files in the first exercise are generated in a way:

<first element><space character><second element><space char>…<nth element><space><slash at the end of the row>

4 files with different graph representations: -adjacency matrix;

-incidence matrix;

-edge list;

-incidence list;

In the second exercise I am using adjacency matrix representation.

Final conclusions:

Edge list – the slowest one – searching through edge list is a linear search of complexity O(E) but it gets E*[el1,el2] sublists in memory.
Advantages: easy to implement; take small amount of RAM to storage
Disadvantages: takes a great deal of time to find an edge in it

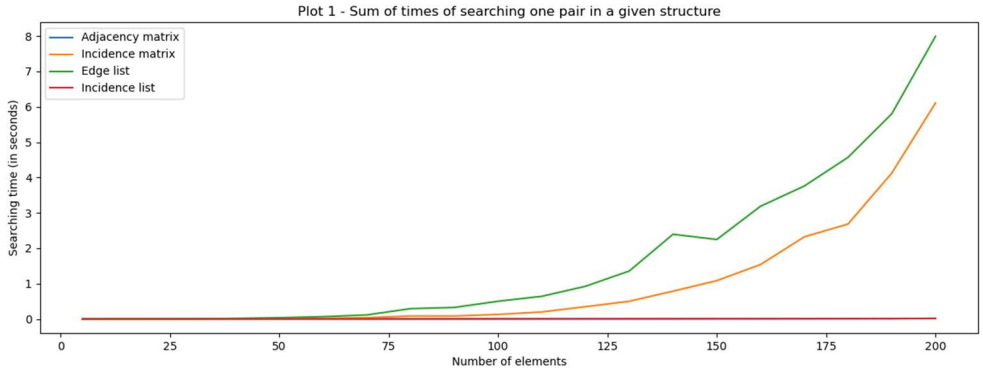
Adjacency matrix – a symmetric matrix which takes $O(V^2)$ RAM memory and give result immediately O(1) complexity
Advantages: fast searching; easy to implement; it takes a lot of RAM but it is not that bad
Disadvantages: I think that memory allocation can be a minus when dealing with big matrices with little number of edges

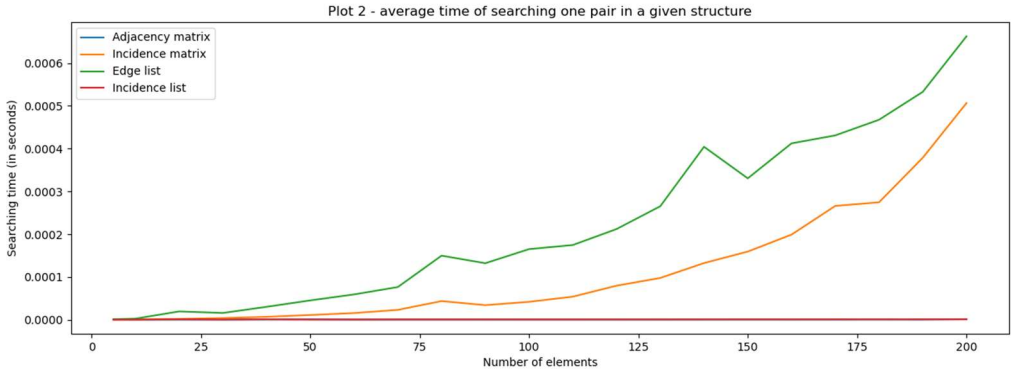
Adjacency/Incidence List – list in which every sublist contains vertices connected to the vertice which number is a sublist's index in an Incidence list
Advantages: fast searching elements in constant O(1) time(in the worst case O(d) where d is the degree of vertex `i` in sublist because that's the of i'th sublist. .
Disadvantages: needed a good imagination to implement it properly. Memory storage in the worst case is O(2E)

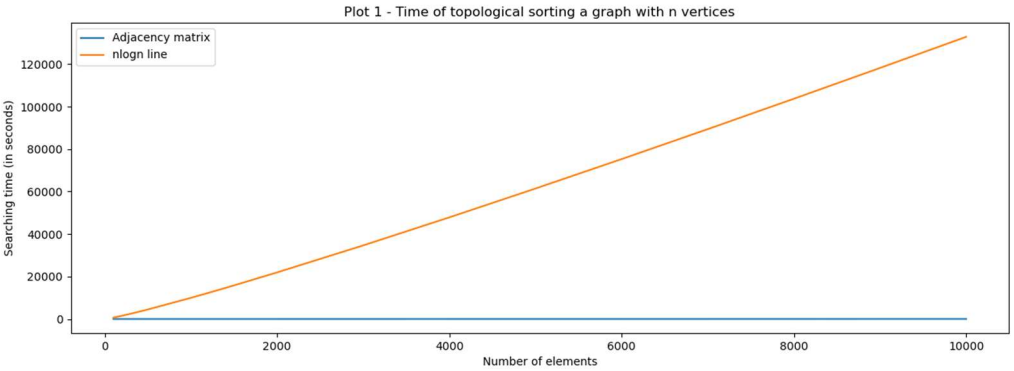
Incidence Matrix – matrix in which every row consists of two 1's and a bunch of zeros which indicates which vertices are connected with each other.
Advantages: easy to implement; slightly better than edge list O(E) search complexity
Disadvantages: very big memory allocated O(E*V) it takes long to generate a file with it.

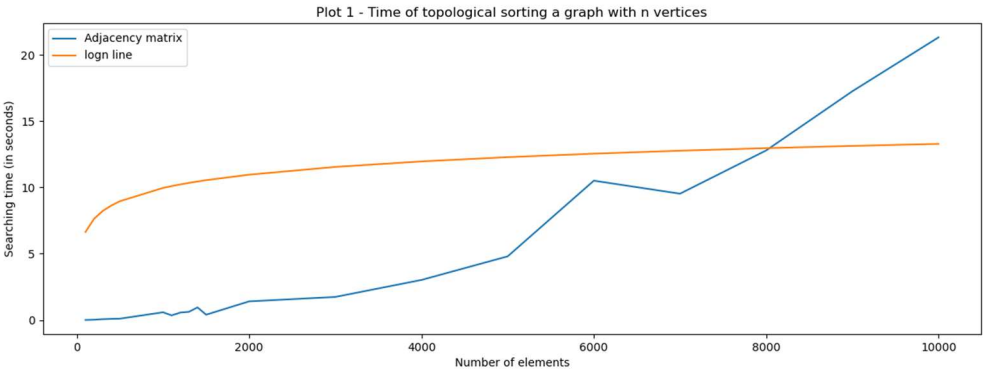
For the second exercise I have chosen the adjacency matrix for graph representation because of it's O(1) complexity of searching edge in it and because it is easy to implement such a matrix.
Topological sorting in my program uses Breadth First Search traverse through a graph with time complexity O(E+V) I have created a Directed Acyclic Graph using property of adjacency matrices such that if you randomly fill a matrix below the main diagonal(with saturation of 0.3) You will always get an adjacency matrix representing Directed Acyclic Graph. Then I have measured the time of sorting this graph in the topological order and put it on a graph with log(n) and nlog(n) line graph for comparison purposes.

Here are the graphs for the first and the second task:


Plot 1 - Sum of times of searching one pair in a given structure
Adjacency matrix
Incidence matrix
Edge list
Incidence list
Searching time (in seconds)
Number of elements


Plot 2 - average time of searching one pair in a given structure
Adjacency matrix
Incidence matrix
Edge list
Incidence list
Searching time (in seconds)
Number of elements


Plot 1 - Time of topological sorting a graph with n vertices
Adjacency matrix
nlogn line
Searching time (in seconds)
Number of elements


Plot 1 - Time of topological sorting a graph with n vertices
Adjacency matrix
logn line
Searching time (in seconds)
Number of elements