N = node, E = Edge

Depth-First Search as implemented in X.MyDFS.dfs(DirectedGraph<E> graph)

dfs_non_recrusive_method:
First I loop through every node in the graph.

In the non-recursive method I check if the collection with soon-to-visit-nodes is not empty. I then remove the node that will be checked, looping through all its edges. With each edge I add it to the soon-to-visit-nodes-collection and it's starts over.

In worst case i have to loop through every node's edges once, that makes the time complexity: **O(N+E)/O(N+C+E)? c=node in collection**

dfs_recrusive:
First I loop through every node in the graph.
In the recursive method I loop through every edge to one specific node.

In worst case we have to loop through every node's edges once, that makes the time complexity: **O(N+E)**

Breadth-First Search as implemented in X.MyBFS.bfs(DirectedGraph<E> graph)

bfs_non_recursive:
BFS is almost the same as DFS non-recursive. Except the BFS visit the nodes in another order. Therefore the worst case and the time complexity is the same as in DFS: **O(N+E)**

Transitive Closure as implemented in X.MyTransitiveClosure.computeClosure(DirectedGraph<E> graph)

First I loop through every node in graph.
On every node I do a DFS to get it's "Reachables".

In worst case we need to go through all the nodes and do a DFS on it, therefore the time complexity is: **O(N*N+E)**

Connected Components as implemented in X.MyConnectedComponents.computeComponents(DirectedGraph<E> graph)

First I loop through the graph, for each node that isn't visited I do a DFS to get it's connections.
Then I loop through the returnCollections and for each collection I check if it has a element in common with the connections to the specific node.

This way i only go through the nodes once. Because of the nested loop the time complexity is: **O(n^2*N+E)**