FoST Assignment 3: Graphs and Algorithms

Part 3: Time complexity Estimates

- *Depth-First Search:*

Pseudo Code:

```
dfs(n) { // Where 'n' is a node.
    mark n as visited.
    while the current node 'n' has successors { // O(S) where 'S' is the number of successors.
        we call 's' the current successor
        if n is not visited { // O(1)
            recursiveBFS('s') // O(N) where N is the number of nodes.
        }
    }
}
```

So, we have a first loop about successors of current node (O(S)) and a second one for each node (O(N)) => It's represented by the recursive method. So we should add the both big-oh :

=> Time complexity *= O(S) + O(N)*
= O(S + N).

- *Breadth-First Search:*

Pseudo Code:

```
bfs(visited, bfsList, set) {
    for each node 'n' in the set { // O(N)
        if n is not visited { // O(1)
            mark n as visited
            add n in bfsList
        }
        while n has successors { // O(S)
            we call 's' the current successor
            if n is not visited { // O(1)
                add s in set
            }
        }
    }
    if there are some elements in the set { // O(1)
        bfs(visited, bfsList, set) // O(N)
    }
    return bfsList
}
```

Here, we have 3 loops. 2 which traverse each node (O(2 * N)) and a second one for the successors (O(S)).

=> Time complexity = O(2N + S)

Transitive Closure:

Pseudo Code:

```
computeClosure(dg) { // Where « dg » is a graph.
	List of node 'L'
	for each node 'n' in dg { // O(N)
		dfs(n, L) // O(S + N)
		put n and L in map
	}
	return map
}
```

In this case, we have a first casual loop for each node in the graph (O(N)) and we have the recursive DFS method which the time complexity is O(S+N).

***=> Time complexity** = O(N * (S + N))*
*= O(N*N + N*S)*
= O(N^2 + NS)

Connected Components:

Pseudo Code:

```
computeComponents(dg) { // Where « dg » is a graph
	for each node 'n' in dg { // O(N)
		computeComponents(n) // Recursive method => O(S+N)*
		for each collection « coll » in collectionSet { // O(N)
			if coll and connected set have elements in common { // O(1)
				add all connected sets in coll // O(N)
				initialize connected set
			}
		}
		if connected set is not empty { // O(1)
			add connected set in collectionSet
			initialize connected set
		}
	}
	return collectionSet
}
```

```
*computeComponents(n) { // Where 'n' is a node
	mark n as connected
	while n has successors { // O(S)
		call 's' the current successor
		if s is not connected { // O(1)
			computeComponents(s) // O(N)
		}
	}
} => Time complexity of this method = O(S+N)
```

Here, we have some loops to analyze. So, in first, we have a classic loop which traverse all the nodes in the graph (O(N)) and an second nested one(O(N)). But, before we have a recursive method which the time complexity is O(S+N) like the dfs method. After that, we can find two other loops. One which traverse all the collections in the collectionSet (O(N)) and the other one which add all the connected sets in collectionSet (O(N)).

=> Time complexity *= O(N(S+N + N(N^2))*
= O(NS+N^2 + N^3)