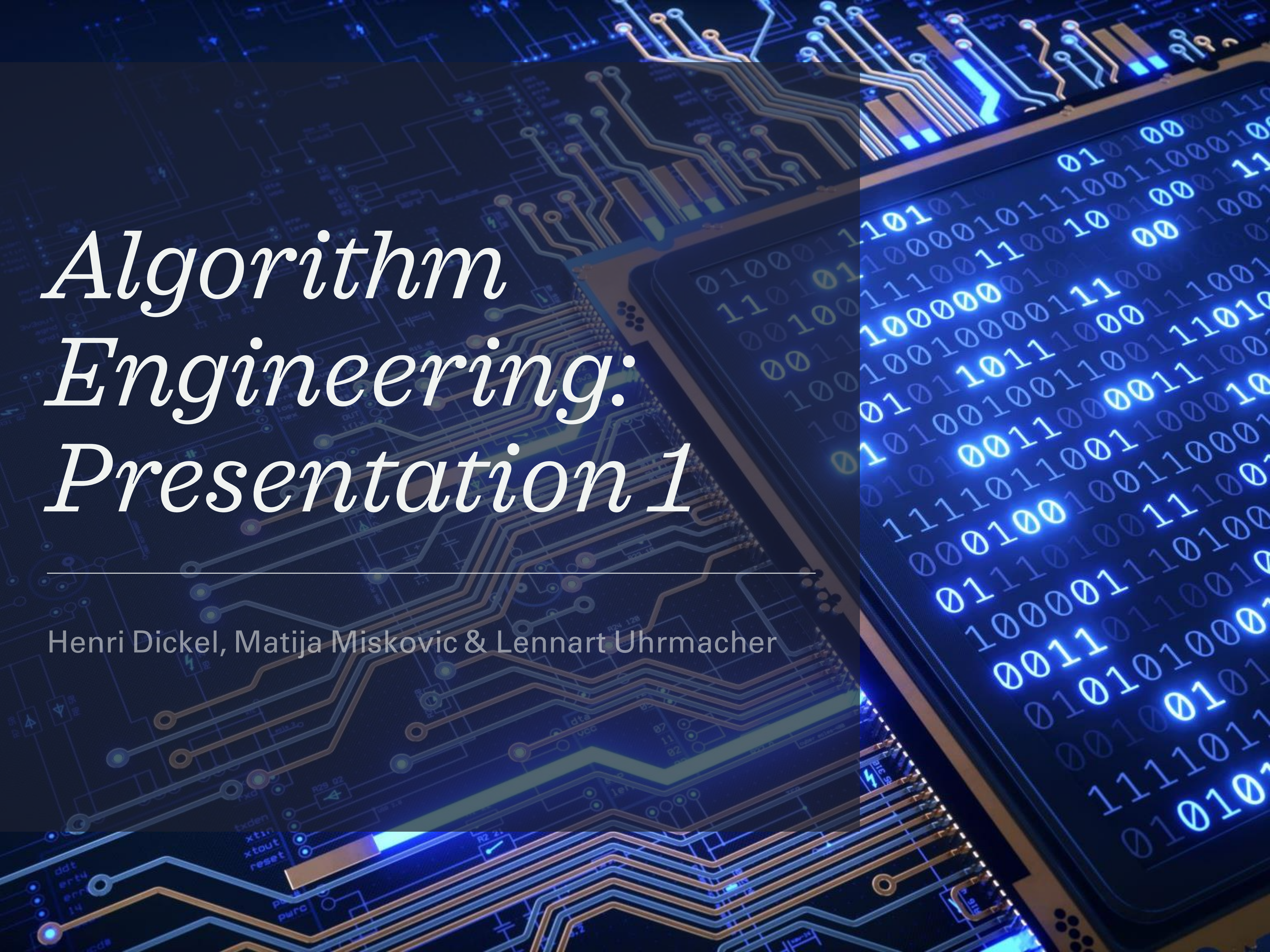


Algorithm Engineering: Presentation 1

Henri Dickel, Matija Miskovic & Lennart Uhrmacher

Our approach

- → Programming language: Java
- → Standard Java, Streams, no external libraries
- → Python for creating plots
- → Object-oriented implementation:
    - → Every node is an Object
    - → Each node holds the information about it's outgoing neighbors

```
class Node
{
        String label;
        List<Node> outNeighbors;

        boolean deleted;
        int visitIndex;
}
```

Our approach

- Solver class that executes the main algorithm
- One class for each algorithm:
  - Preprocessing
  - Is the graph a DAG?
  - Find first cycle
  - Log class for printing the result and debug information
- All of these classes offer static methods

Our approach

→ Nodes are not actually deleted, only labeled:

```
for(Node node: cycle)
{
        node.delete();
        List<Node> S = dfvsBranch(graph, k  - 1);
        node.unDelete();
        if(S != null)
        {
                S.add(node);
                return S;
        }
}
```

Finding the next cycle

→ Algorithm traverses the graph recursively (DFS)

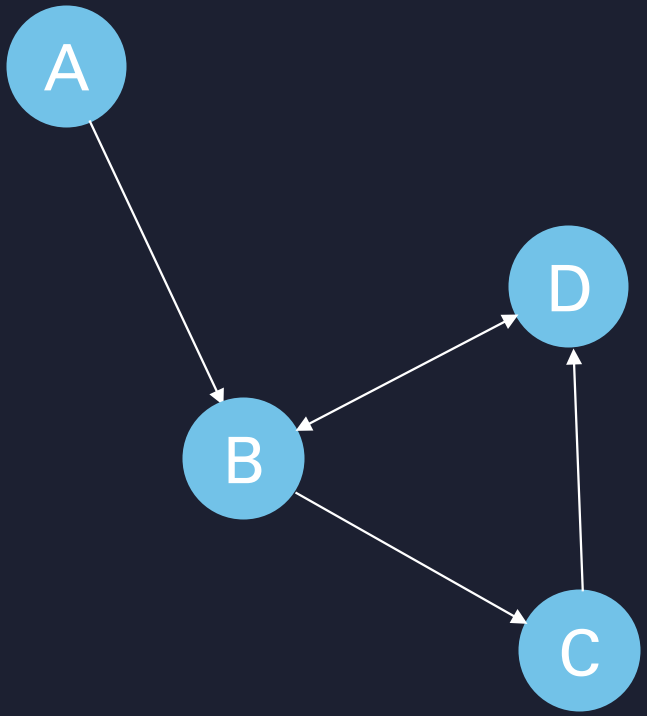
→ Visited nodes get marked with an index

→ If a new visited node is already marked, a cycle is found

→ Running time is O(|V|) in the worst case, when each node is visited once

→ Will be replaced by BFS in the future

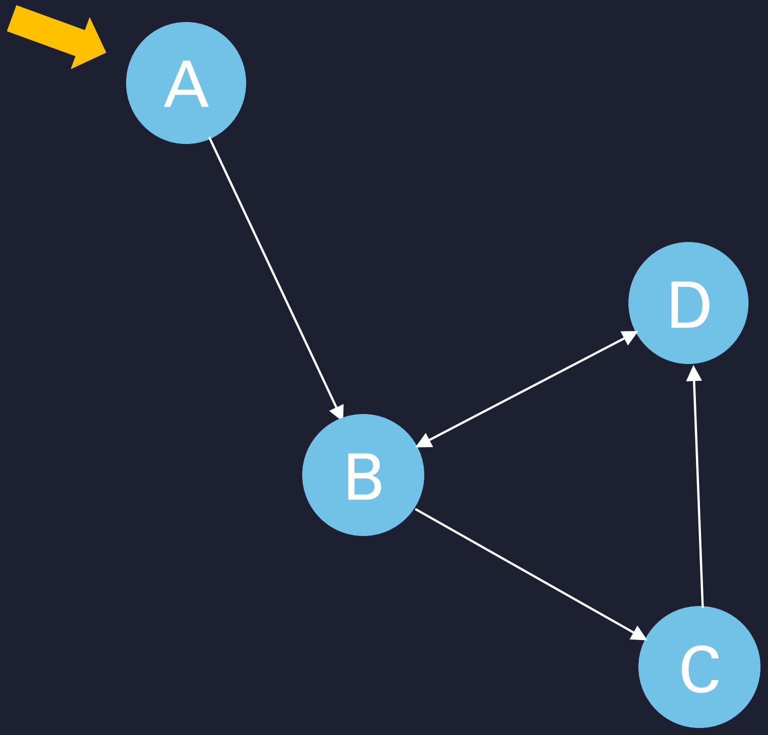
Finding the next cycle



cycleStartIndex: -1

cycle: ∅

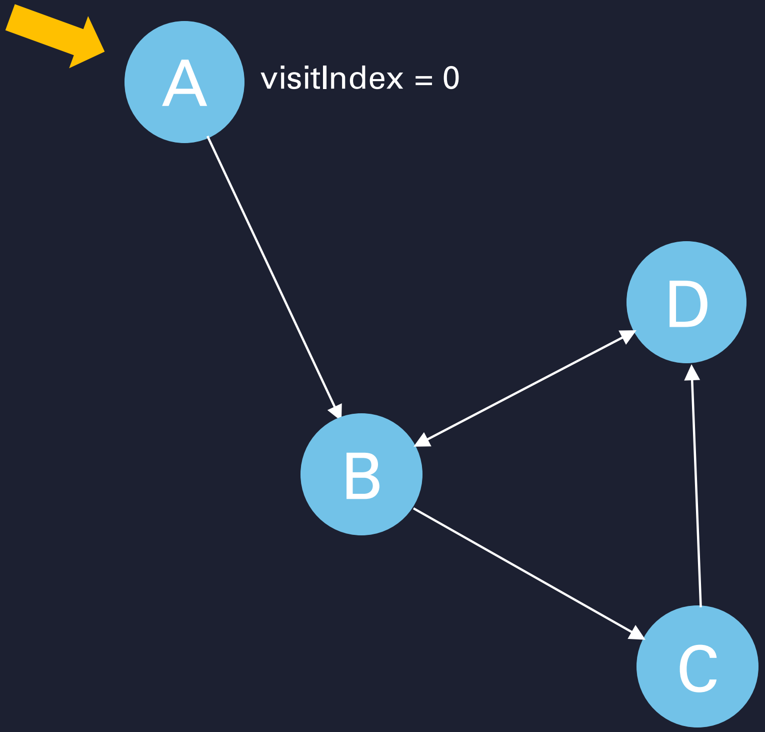
Finding the next cycle



cycleStartIndex: -1

cycle: ∅

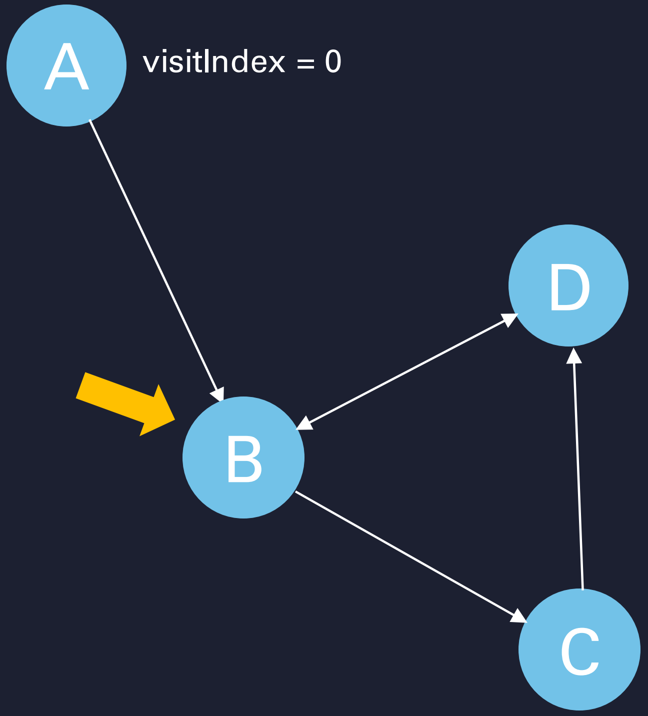
Finding the next cycle



cycleStartIndex: -1

cycle: Ø

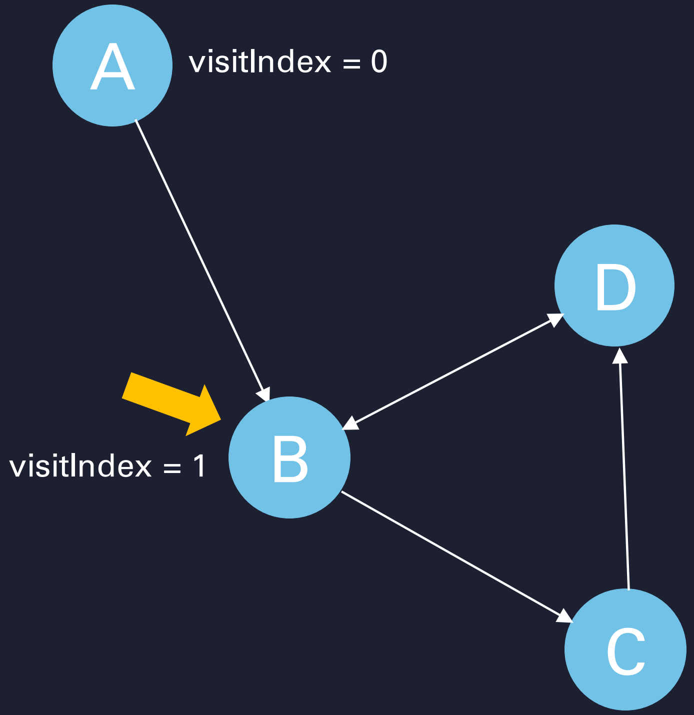
Finding the next cycle



cycleStartIndex: -1

cycle: ∅

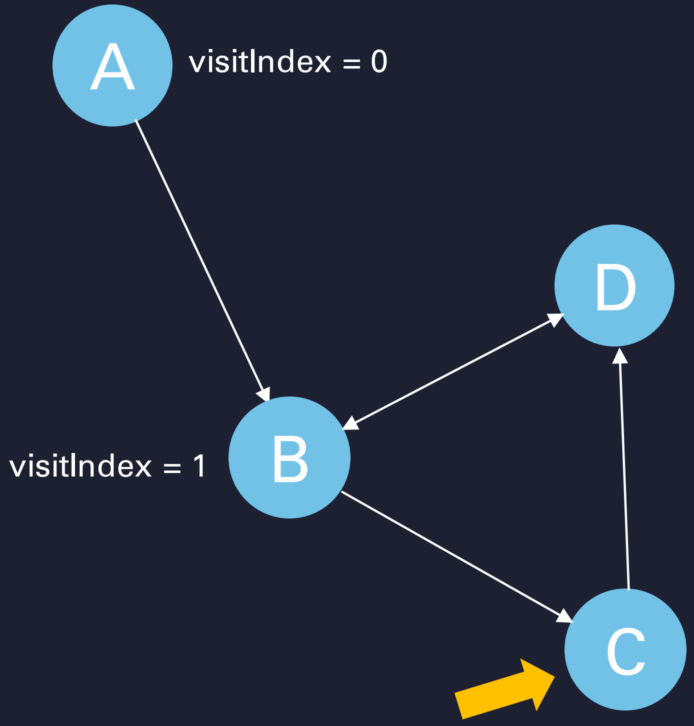
Finding the next cycle



cycleStartIndex: -1

cycle: Ø

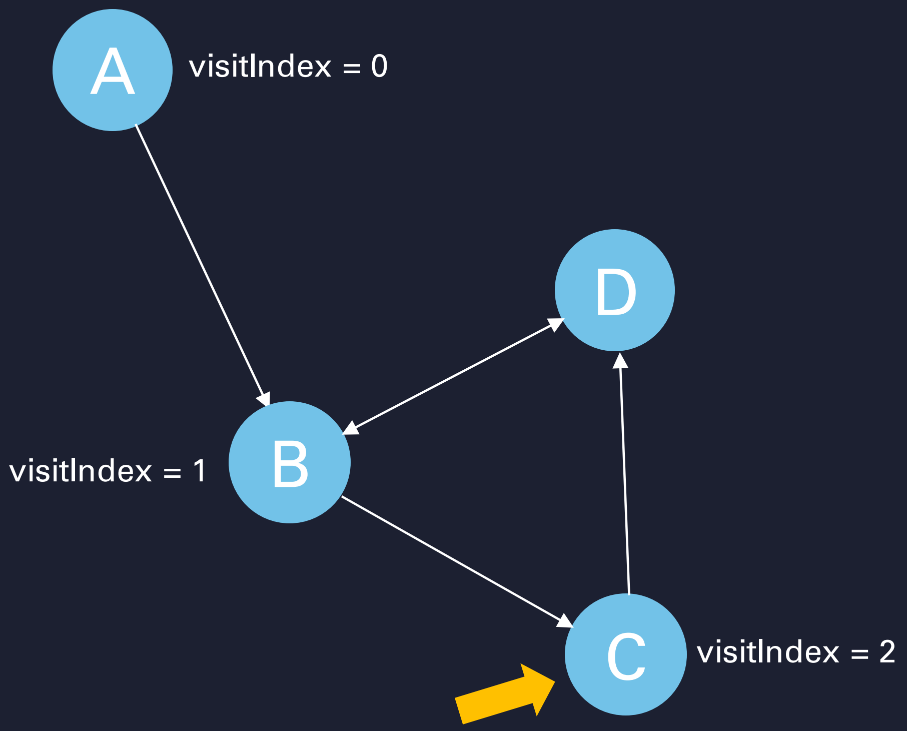
Finding the next cycle



cycleStartIndex: -1

cycle: ∅

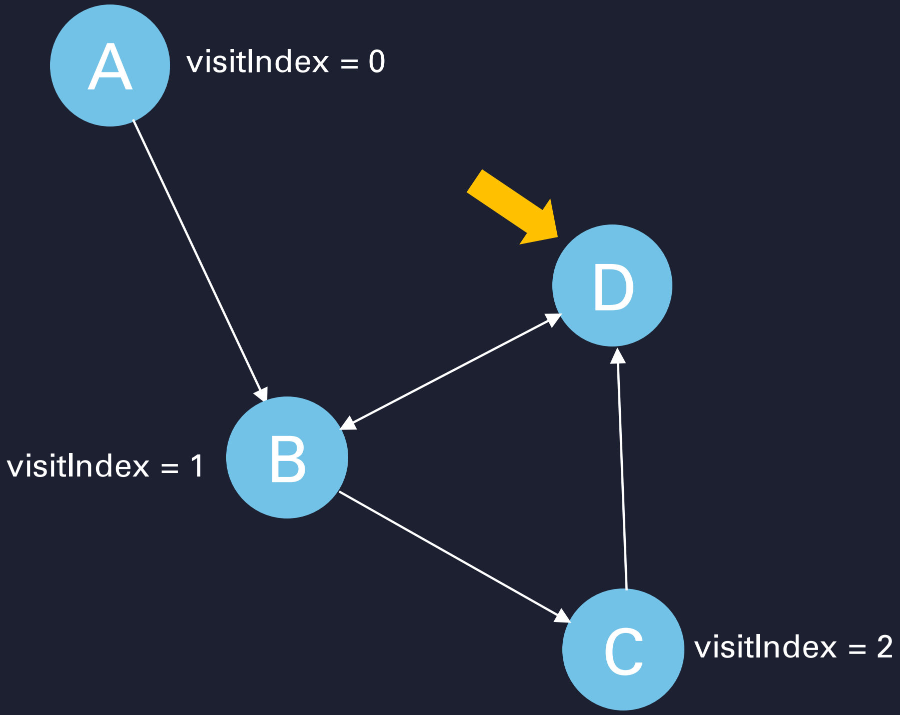
Finding the next cycle



cycleStartIndex: -1

cycle: ∅

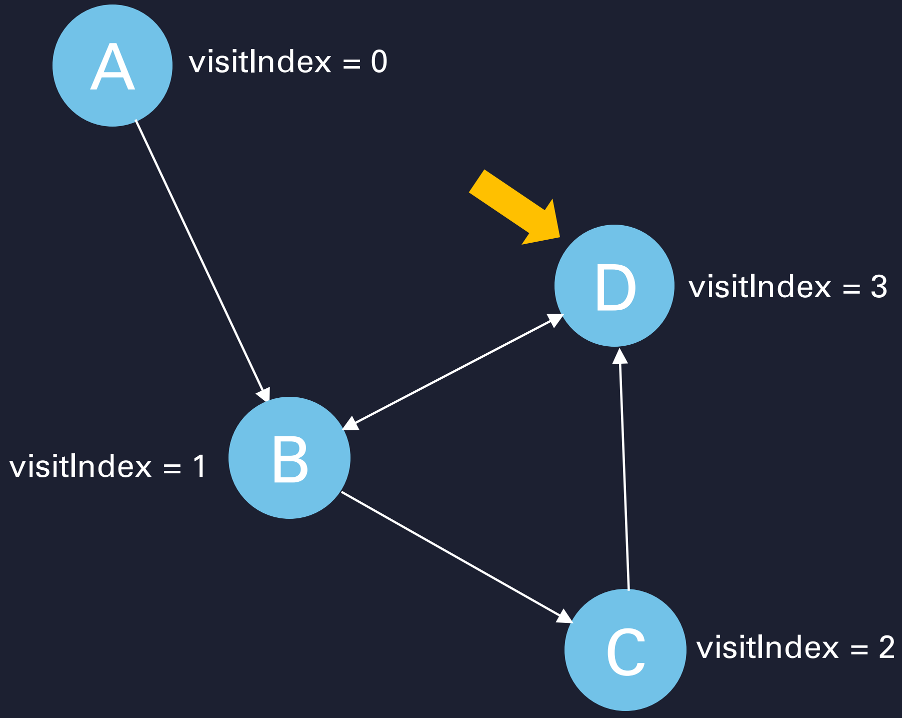
Finding the next cycle



cycleStartIndex: -1

cycle: ∅

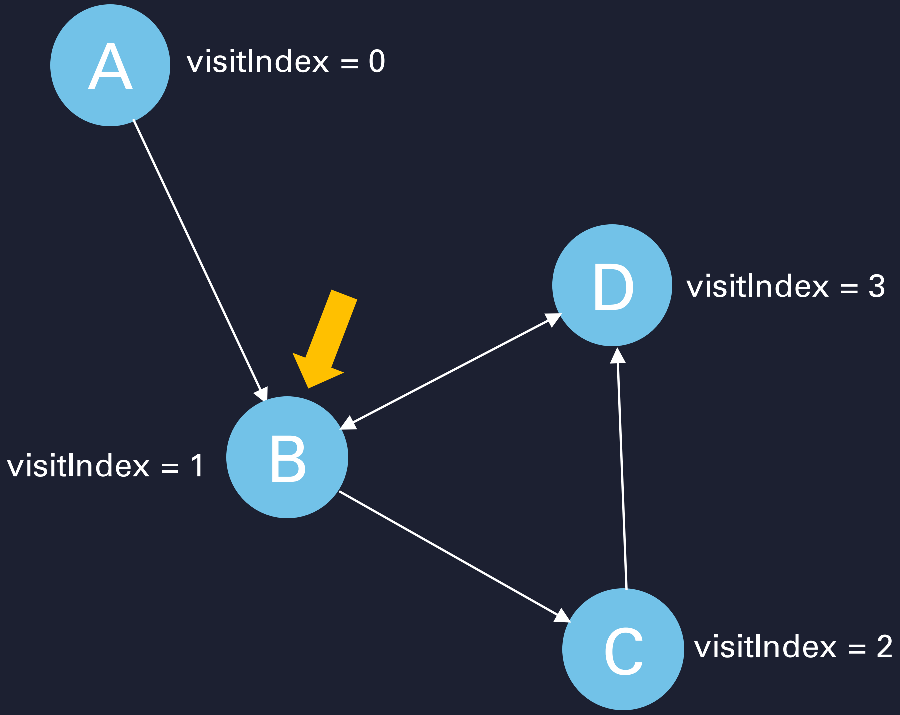
Finding the next cycle



cycleStartIndex: -1

cycle: ∅

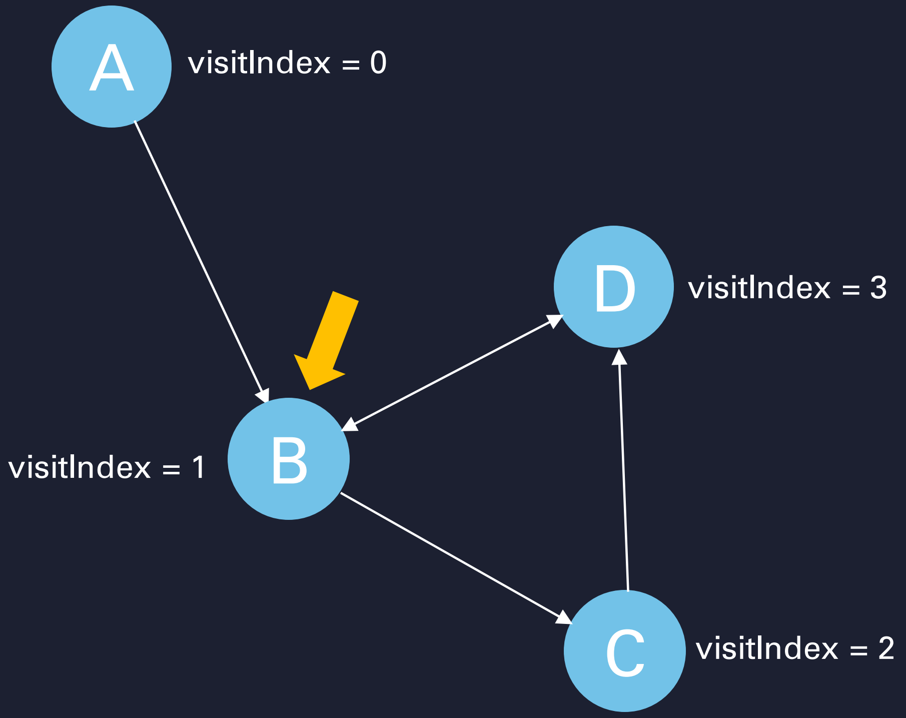
Finding the next cycle



cycleStartIndex: -1

cycle: ∅

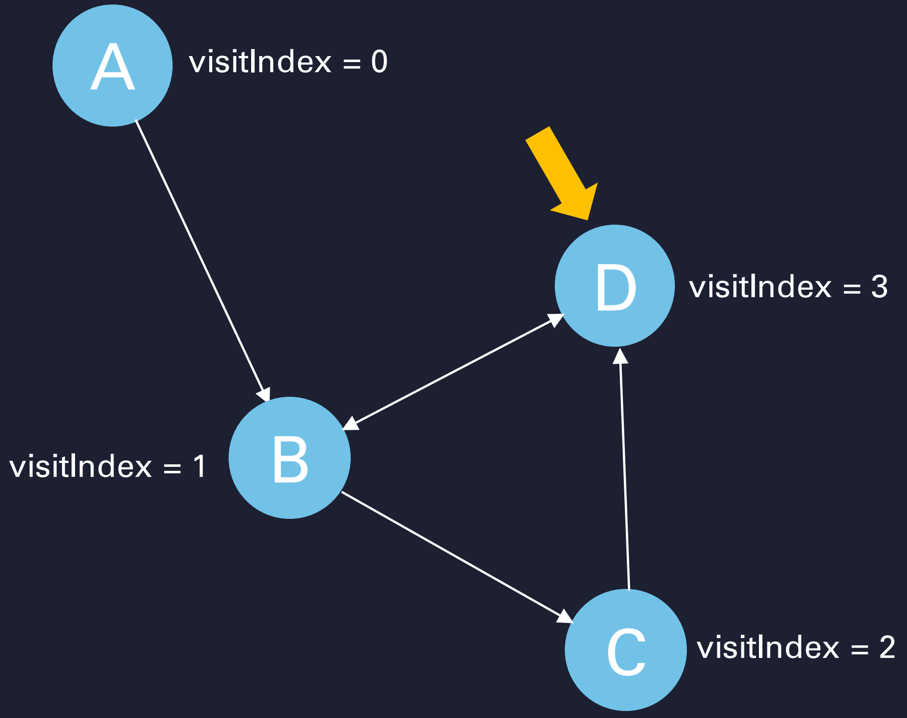
Finding the next cycle



cycleStartIndex: 1

cycle: B

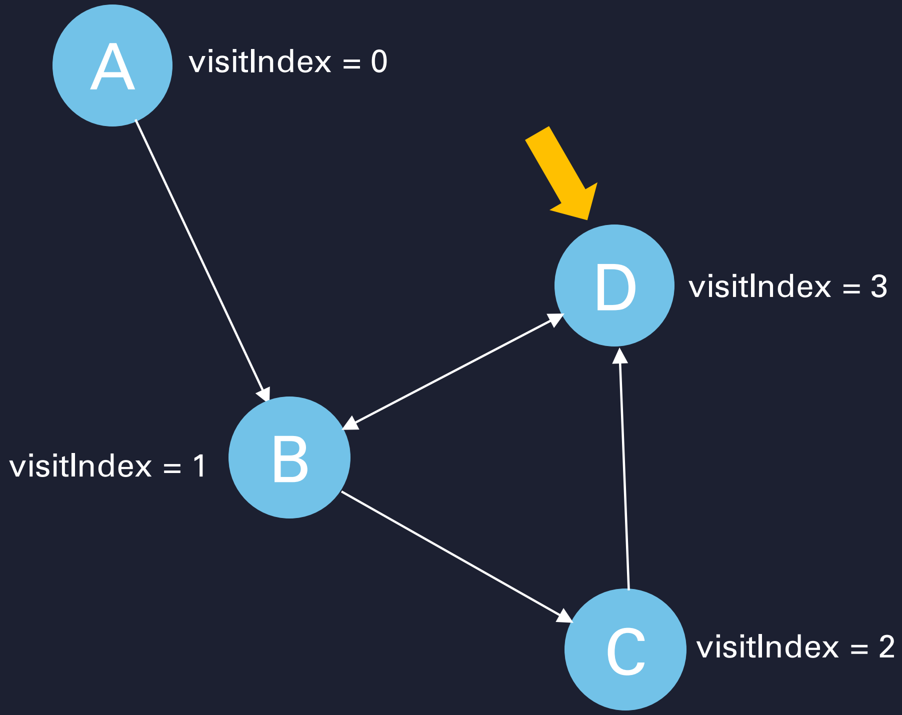
Finding the next cycle



cycleStartIndex: 1

cycle: B

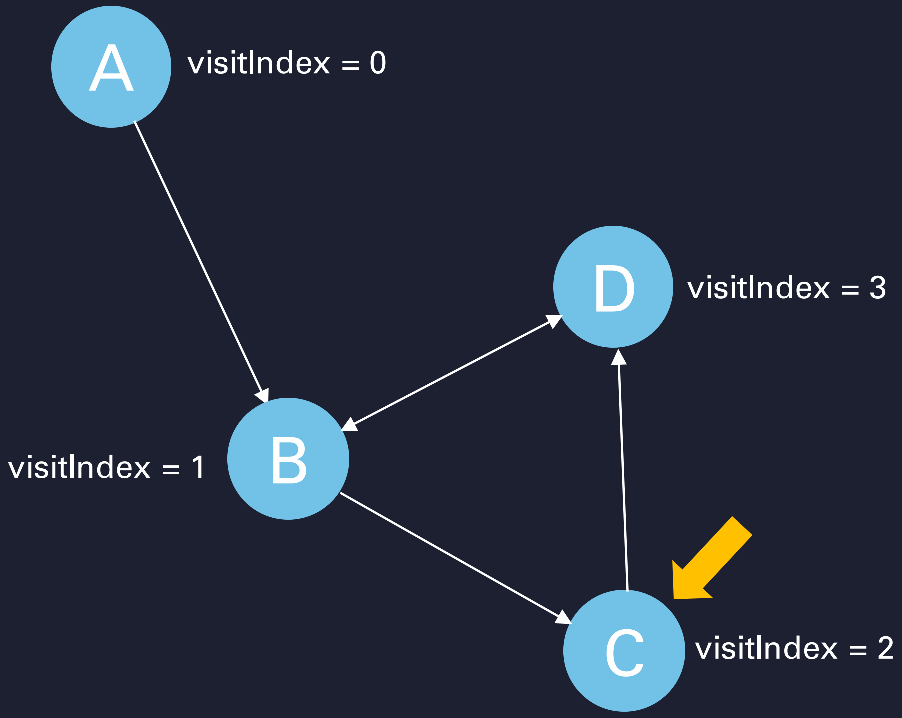
Finding the next cycle



cycleStartIndex: 1

cycle: B, D

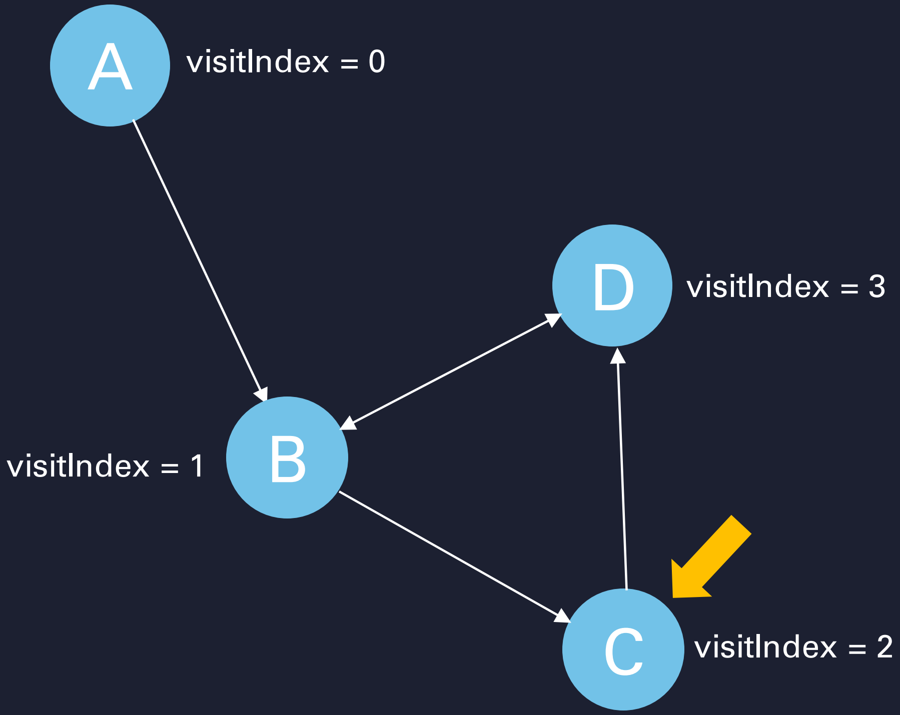
Finding the next cycle



cycleStartIndex: 1

cycle: B, D

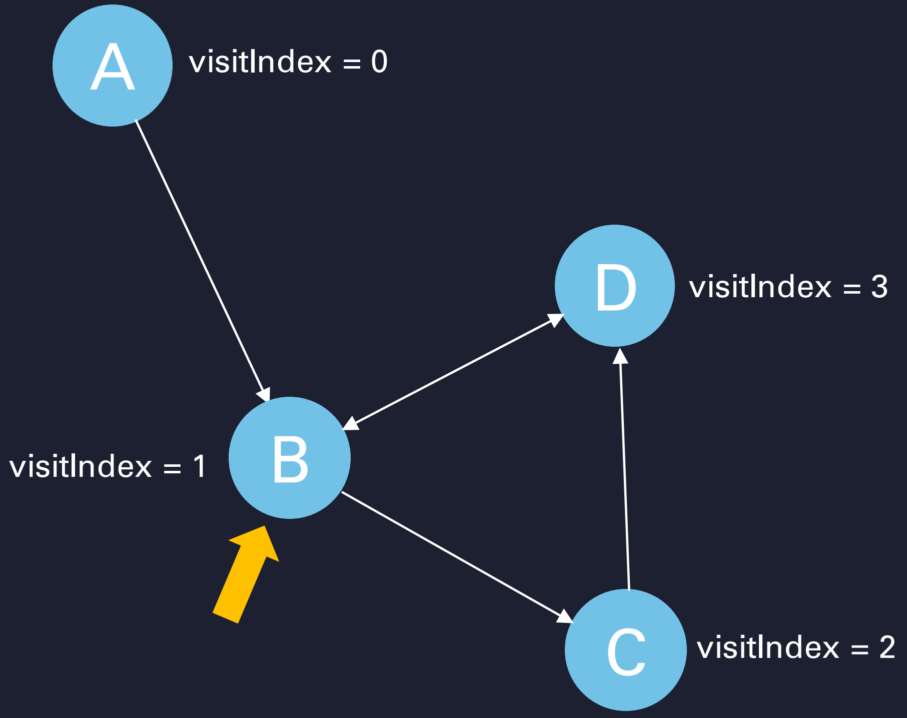
Finding the next cycle



cycleStartIndex: 1

cycle: B, D, C

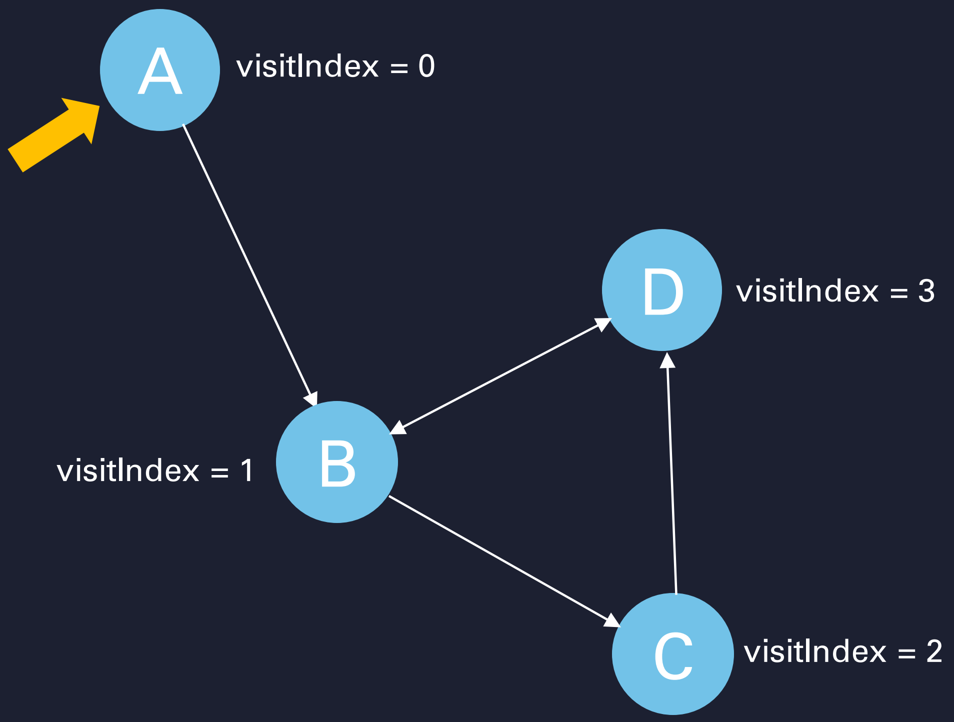
Finding the next cycle



cycleStartIndex: 1

cycle: B, D, C

Finding the next cycle



cycleStartIndex: 1

cycle: B, D, C

Tarjan's Algorithm

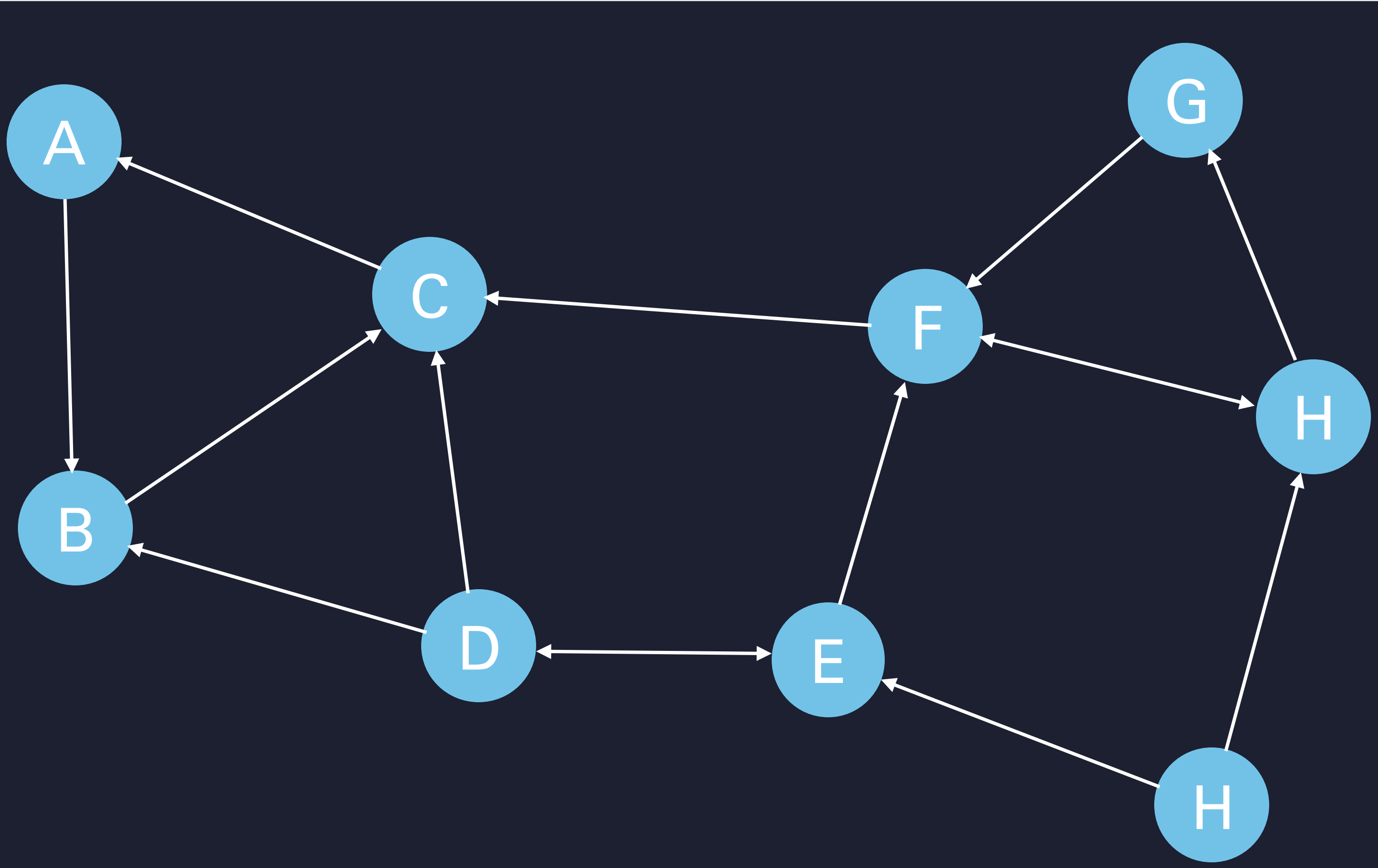
→ Algorithm finds cyclic components in the graph

→ Cyclic component: set of nodes, which have any cyclic connection

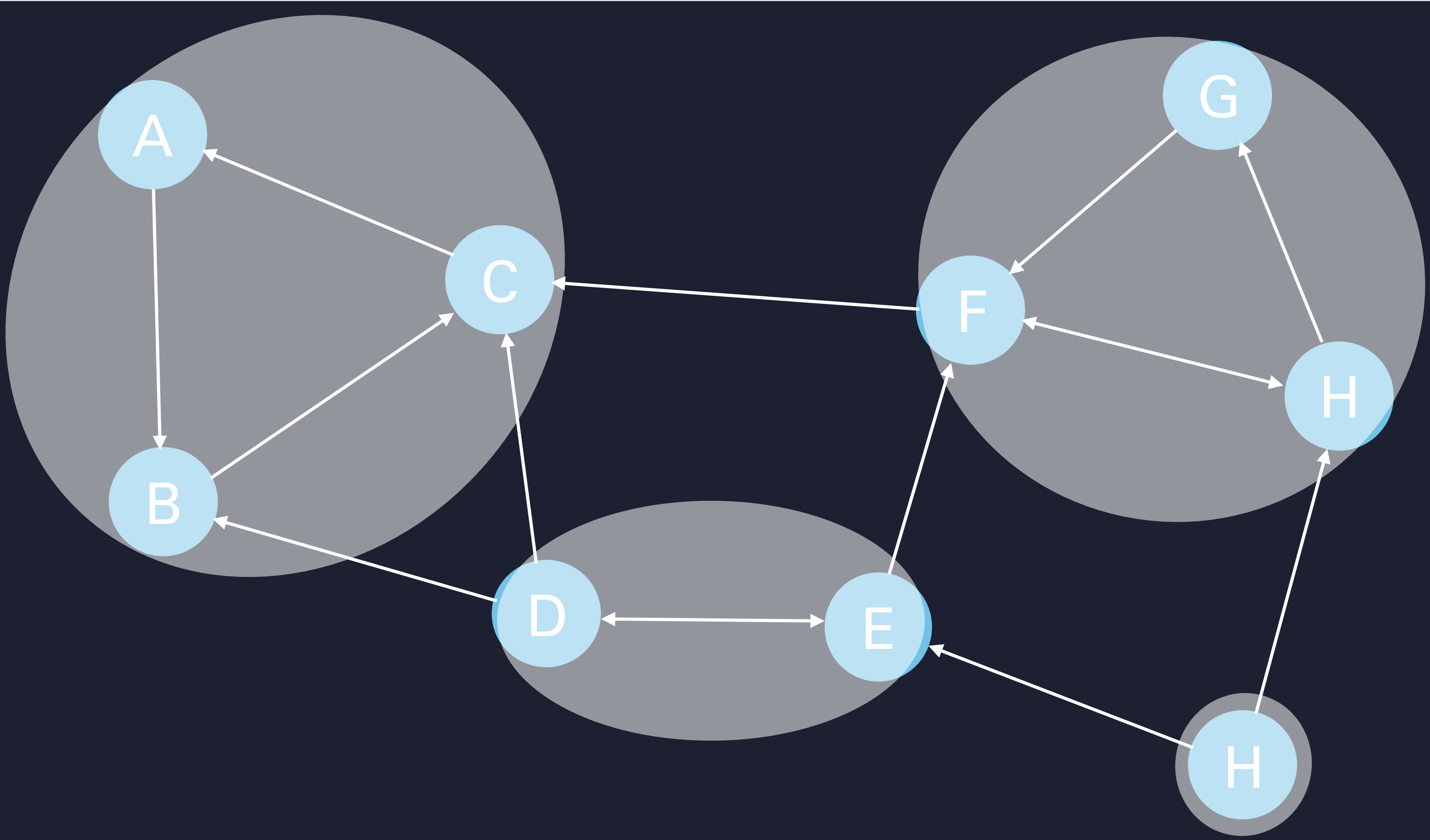
→ Linear running time: O(|V| + |E|)

→ We used it once before the main algorithm

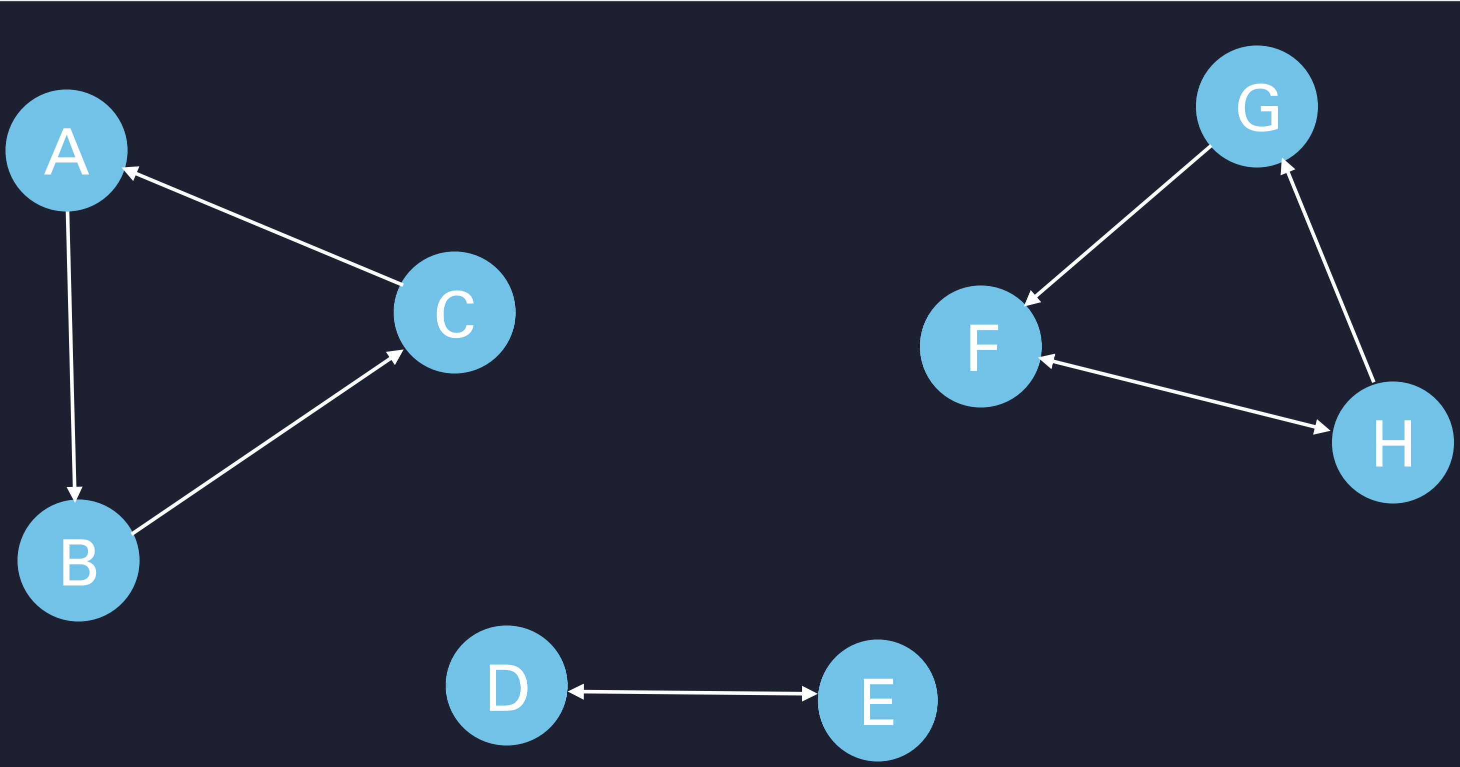
Tarjan's Algorithm



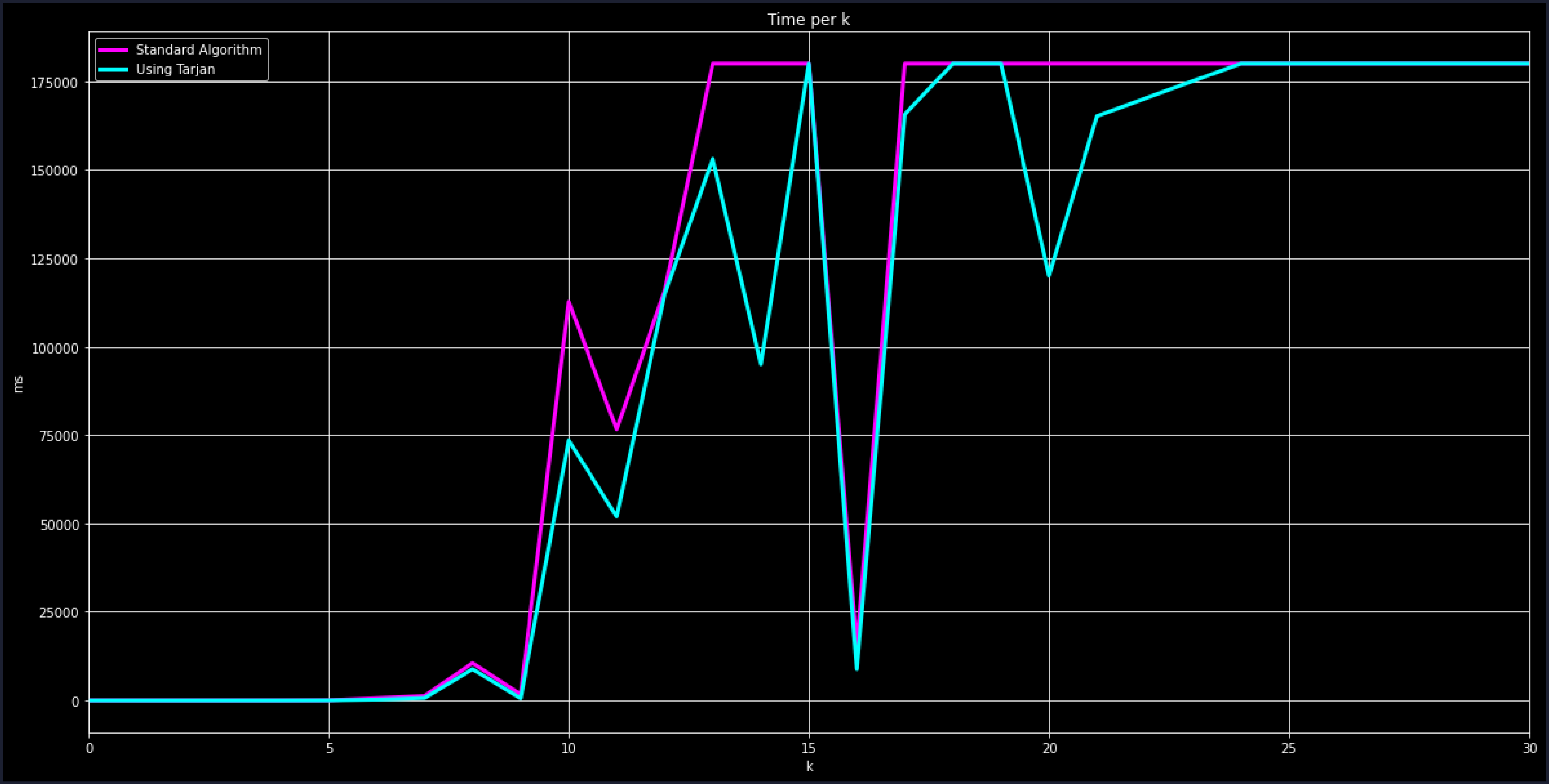
Tarjan's Algorithm



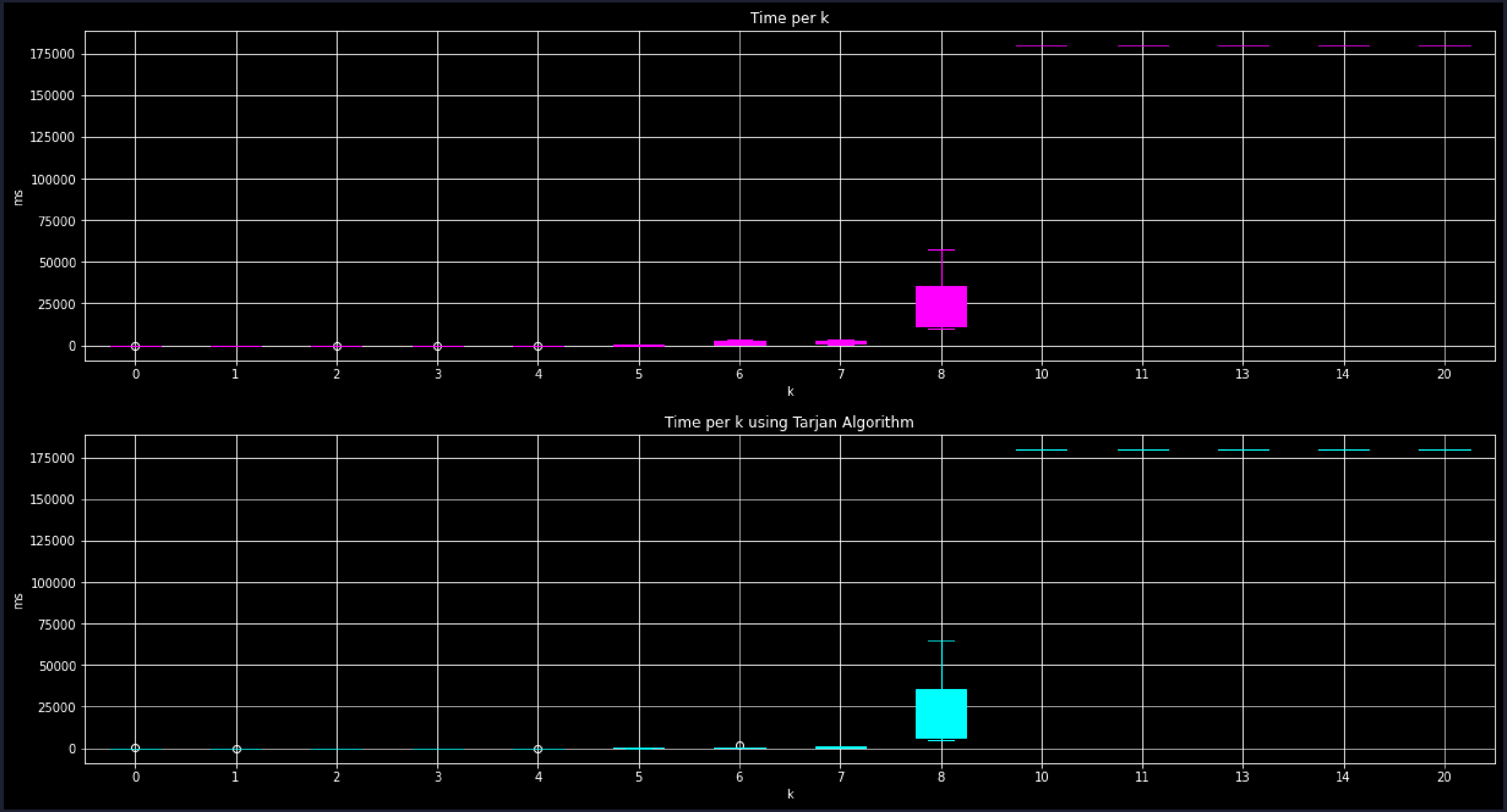
Tarjan's Algorithm



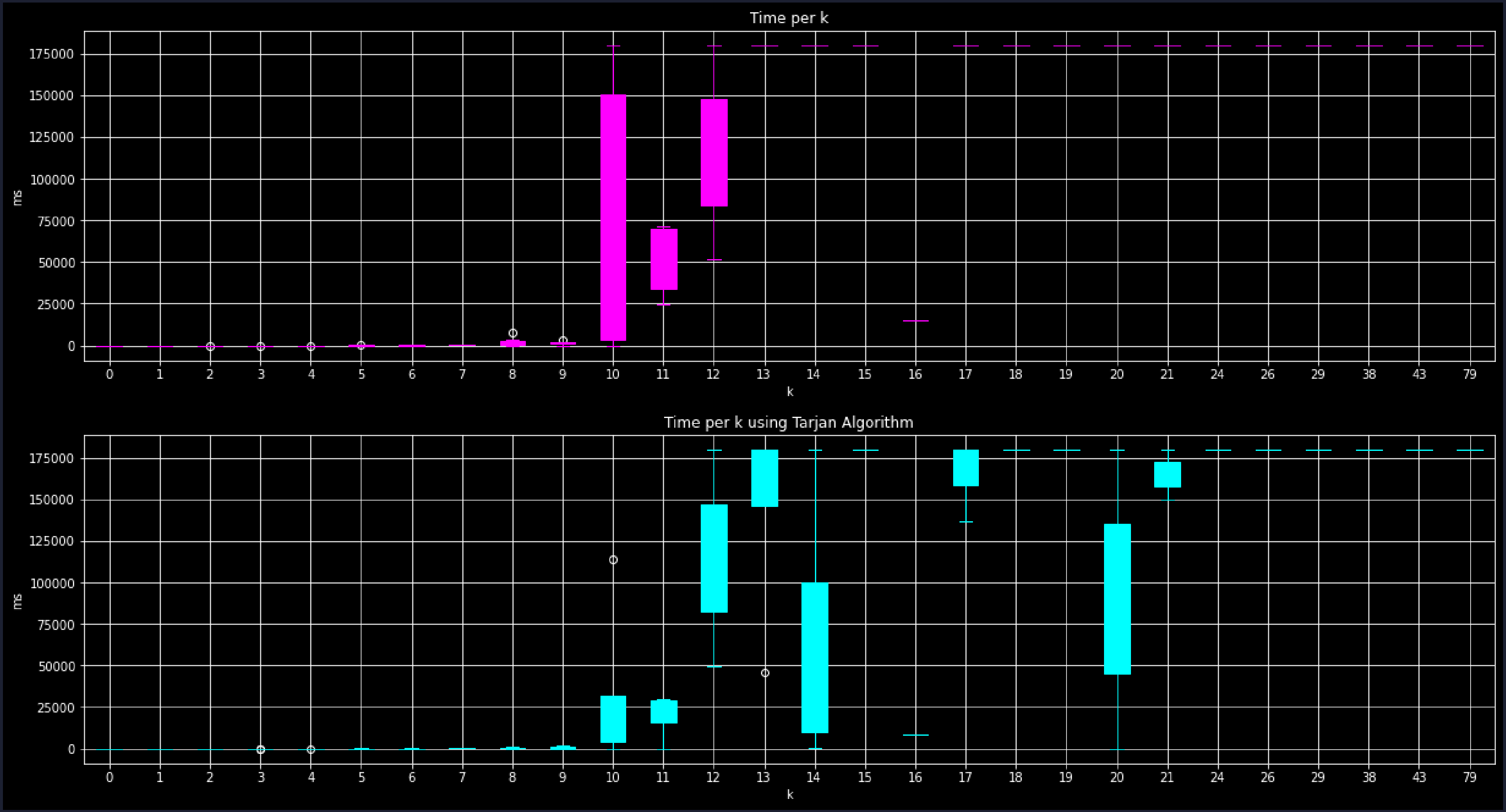
Tarjan's Algorithm - Performance



Tarjan's Algorithm - Synthetic



Tarjan's Algorithm - Complex



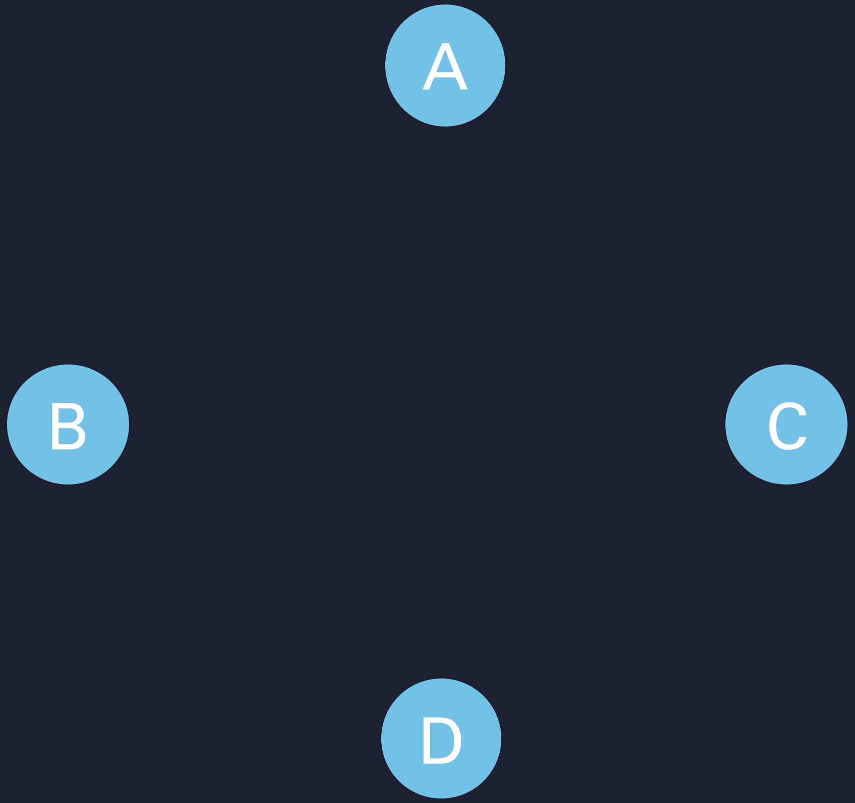
Calculation of min k

- What is the worst case graph?
- A fully connected graph
- In fact, a fully connected graph is not the worst case:

```
if(m == n * (n - 1))
{
        return k = n – 1;
}
```

- Removing a single edge (a,b) => k = n – 2
- Can we rule out even more cases with this approach?
- Idea for min k: add as many edges as possible to a graph without creating a cycle

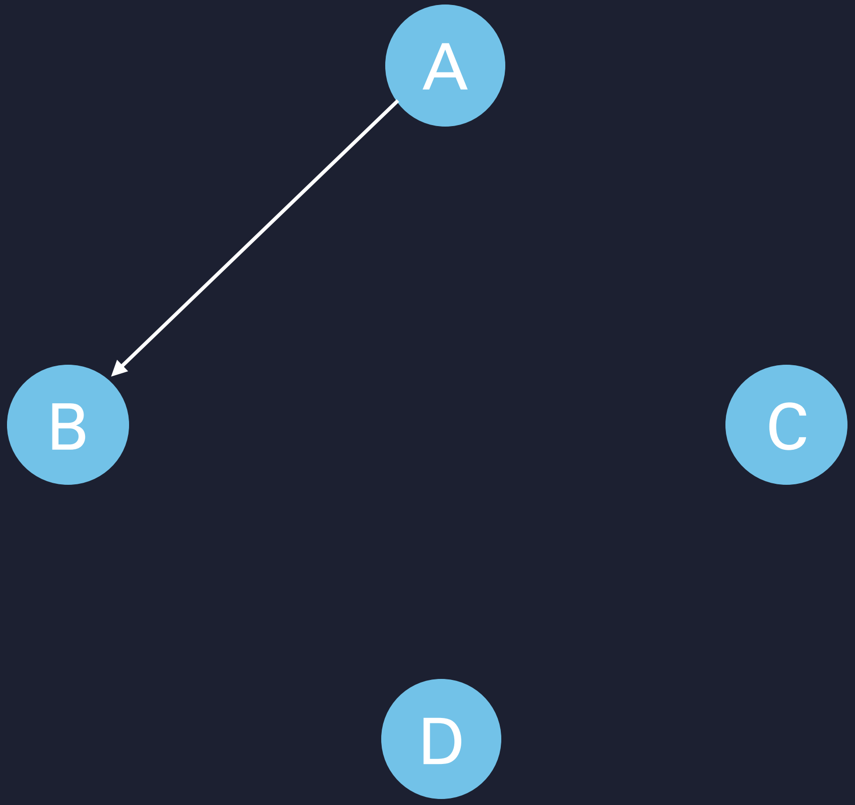
Calculation of min k



|V| = 4
|E| = 0
k = 0

|E|	k
0	0
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

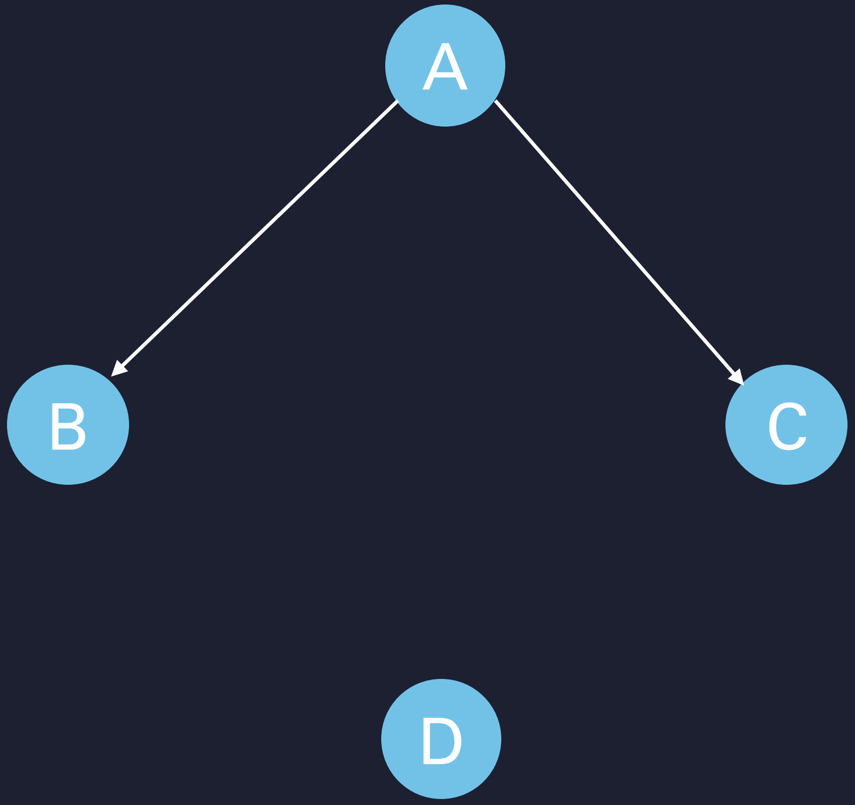
Calculation of min k



|V| = 4
|E| = 1
k = 0

|E|	k
0	0
1	0
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

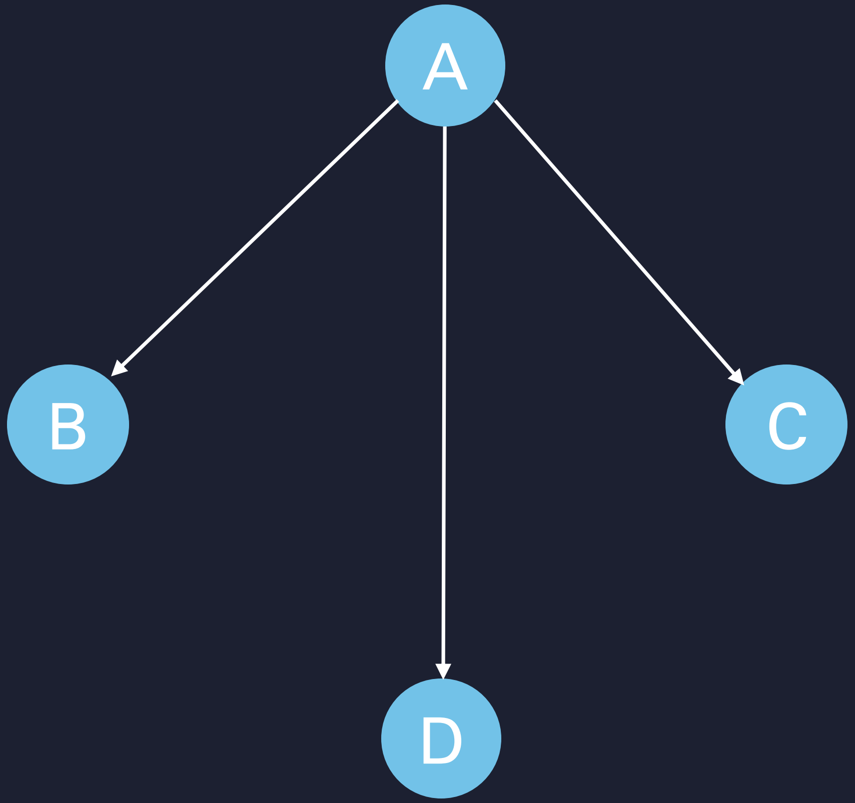
Calculation of min k



|V| = 4
|E| = 2
k = 0

|E|	k
0	0
1	0
2	0
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

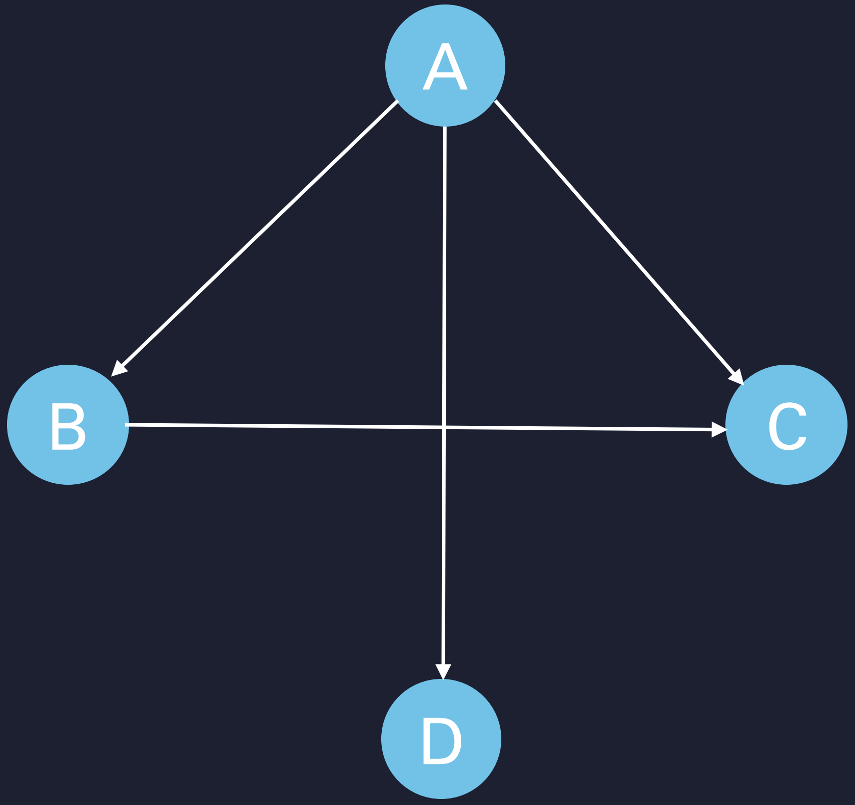
Calculation of min k



|V| = 4
|E| = 3
k = 0

|E|	k
0	0
1	0
2	0
3	0
4	
5	
6	
7	
8	
9	
10	
11	
12	

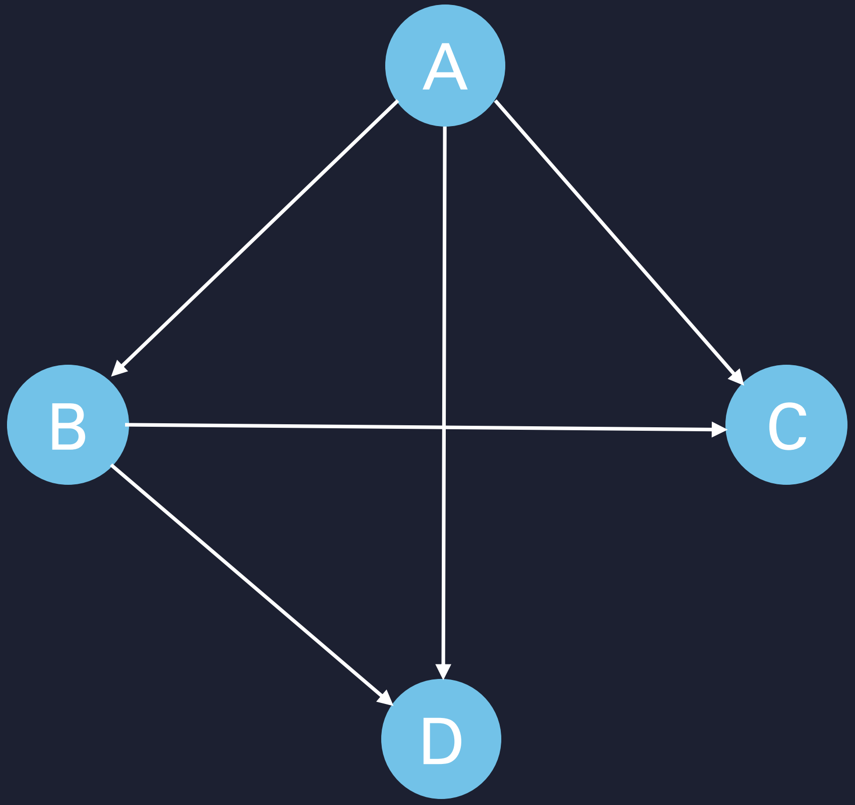
Calculation of min k



|V| = 4
|E| = 4
k = 0

|E|	k
0	0
1	0
2	0
3	0
4	0
5	
6	
7	
8	
9	
10	
11	
12	

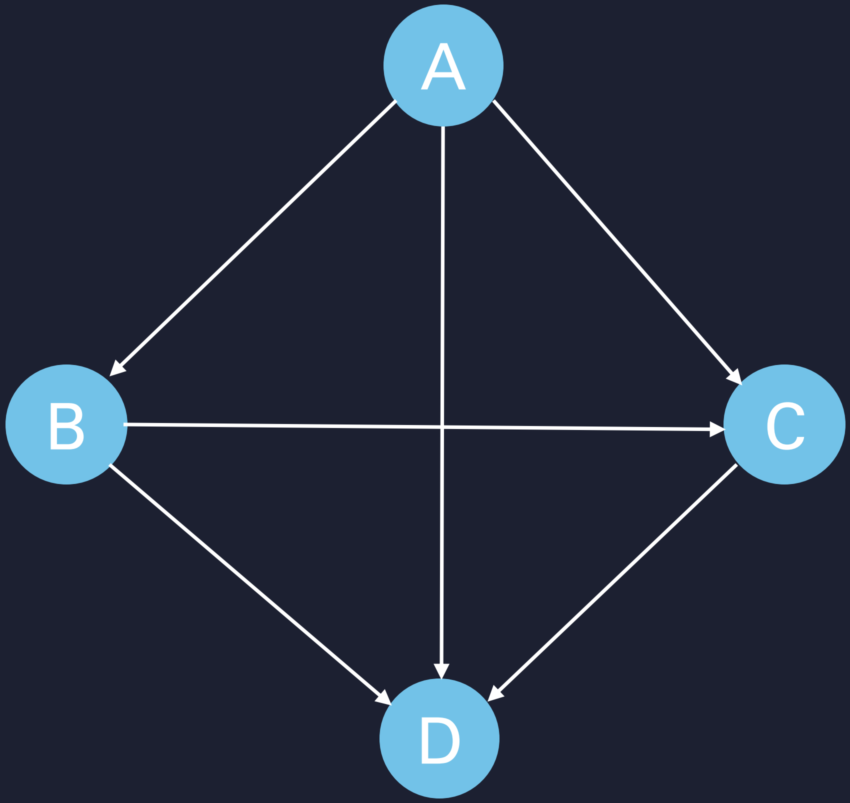
Calculation of min k



|V| = 4
|E| = 5
k = 0

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	
7	
8	
9	
10	
11	
12	

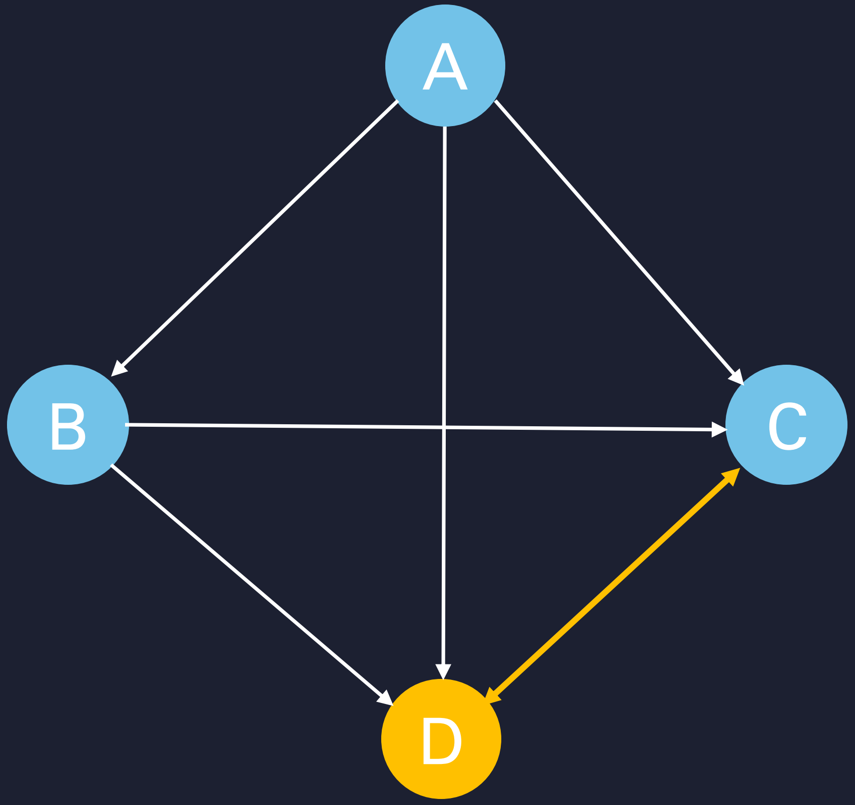
Calculation of min k



|V| = 4
|E| = 6
k = 0

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	
8	
9	
10	
11	
12	

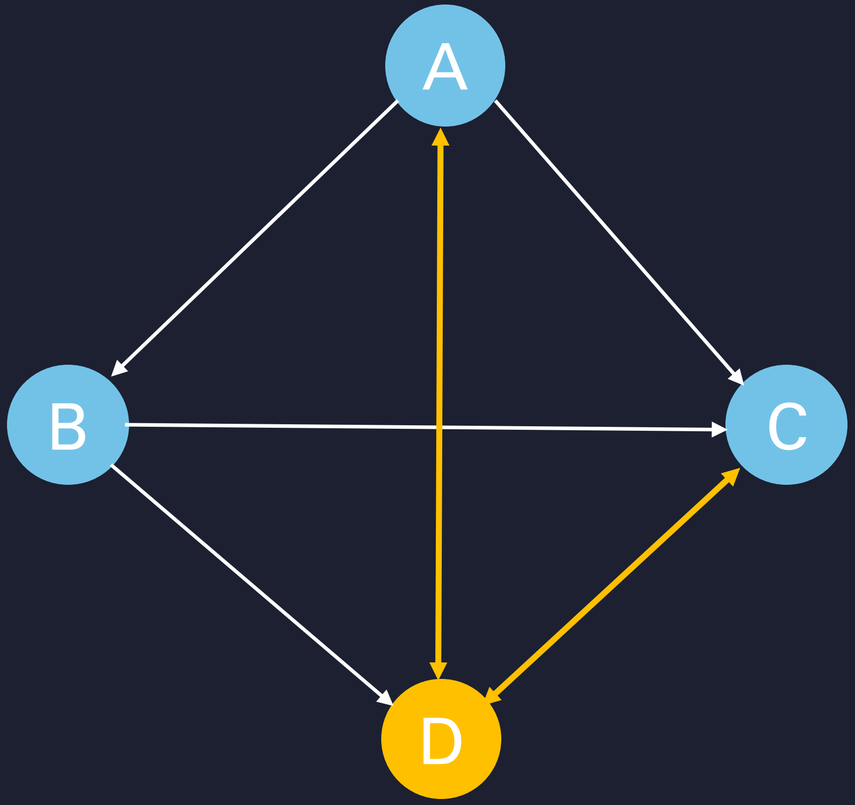
Calculation of min k



|V| = 4
|E| = 7
k = 1

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	
9	
10	
11	
12	

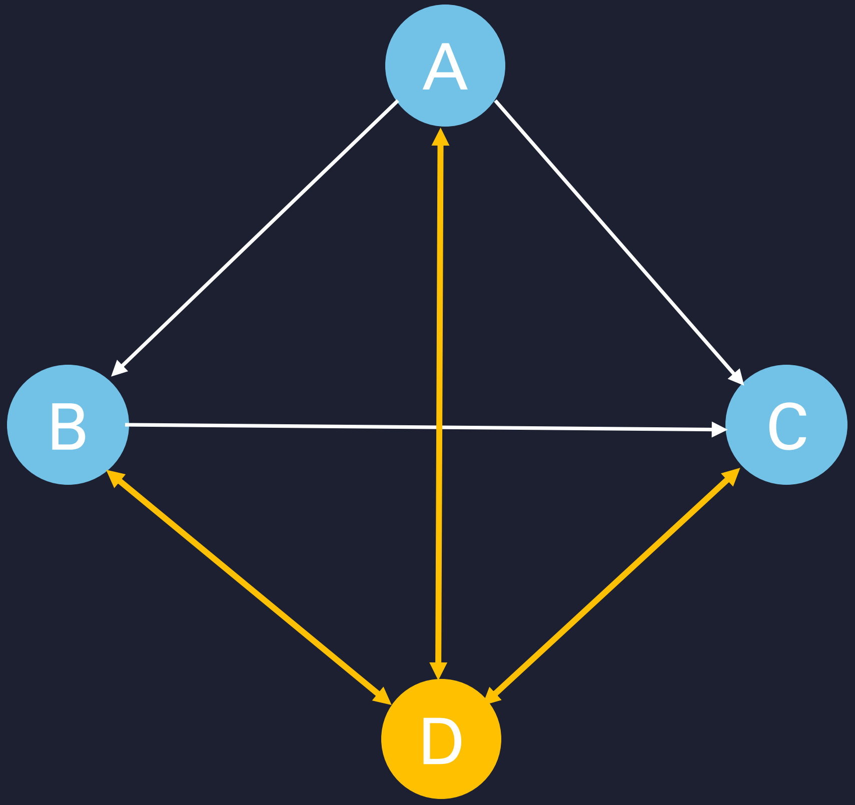
Calculation of min k



|V| = 4
|E| = 8
k = 1

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	
10	
11	
12	

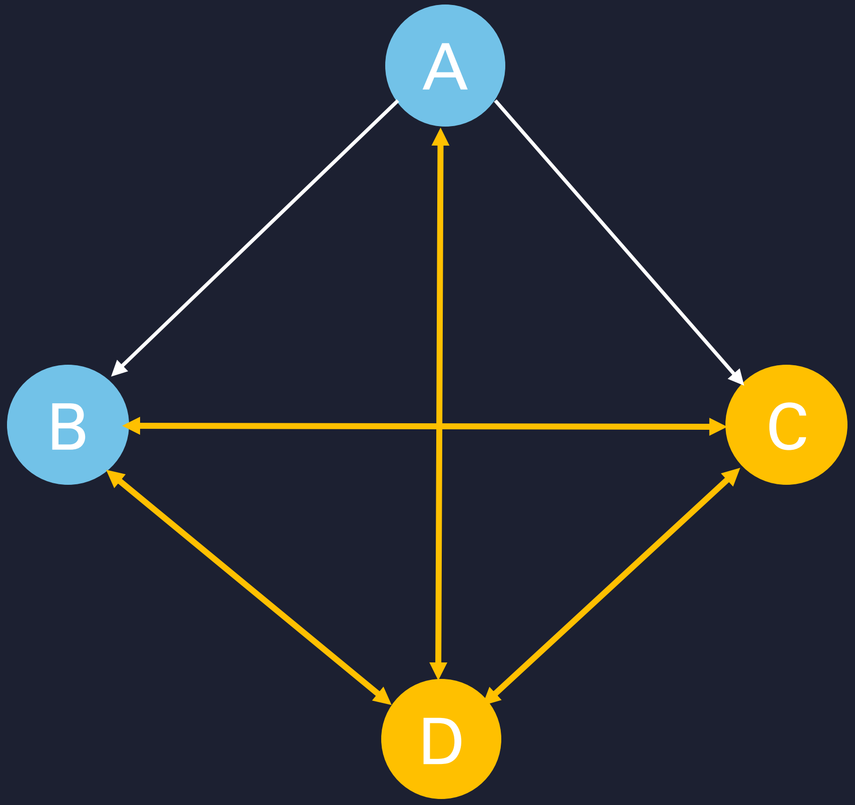
Calculation of min k



|V| = 4
|E| = 9
k = 1

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	1
10	
11	
12	

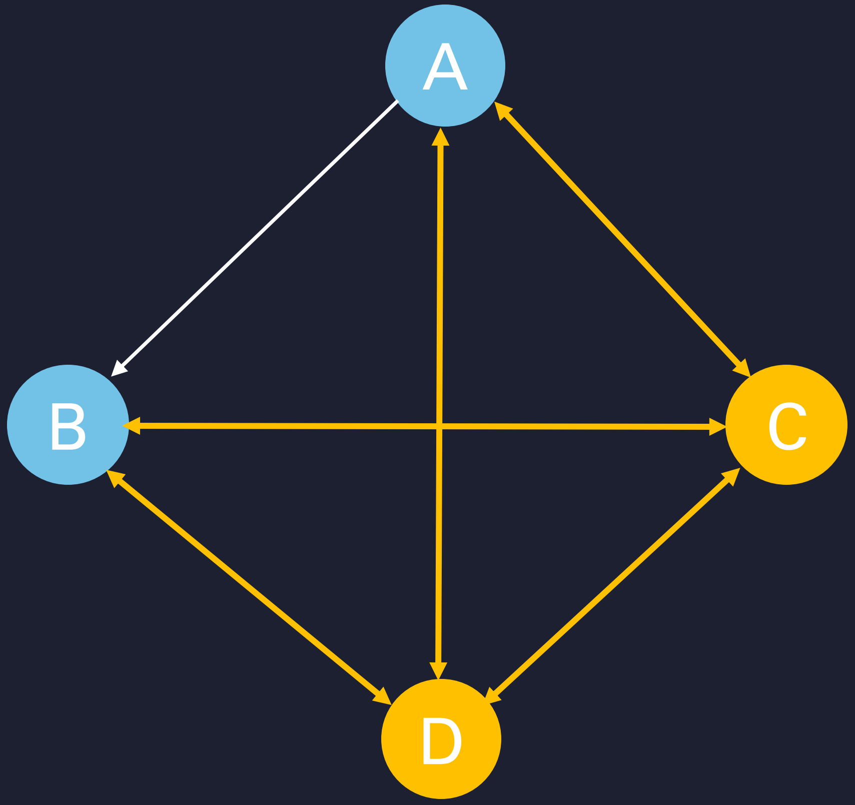
Calculation of min k



|V| = 4
|E| = 10
k = 2

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	1
10	2
11	
12	

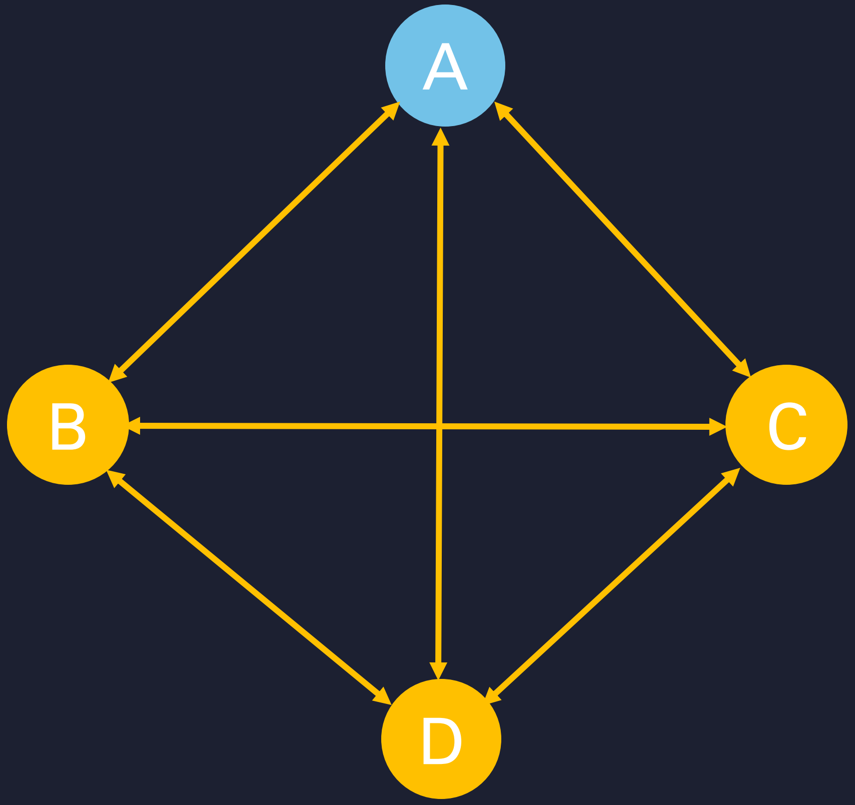
Calculation of min k



|V| = 4
|E| = 11
k = 2

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	1
10	2
11	2
12	

Calculation of min k



|V| = 4
|E| = 12
k = 3

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	1
10	2
11	2
12	3

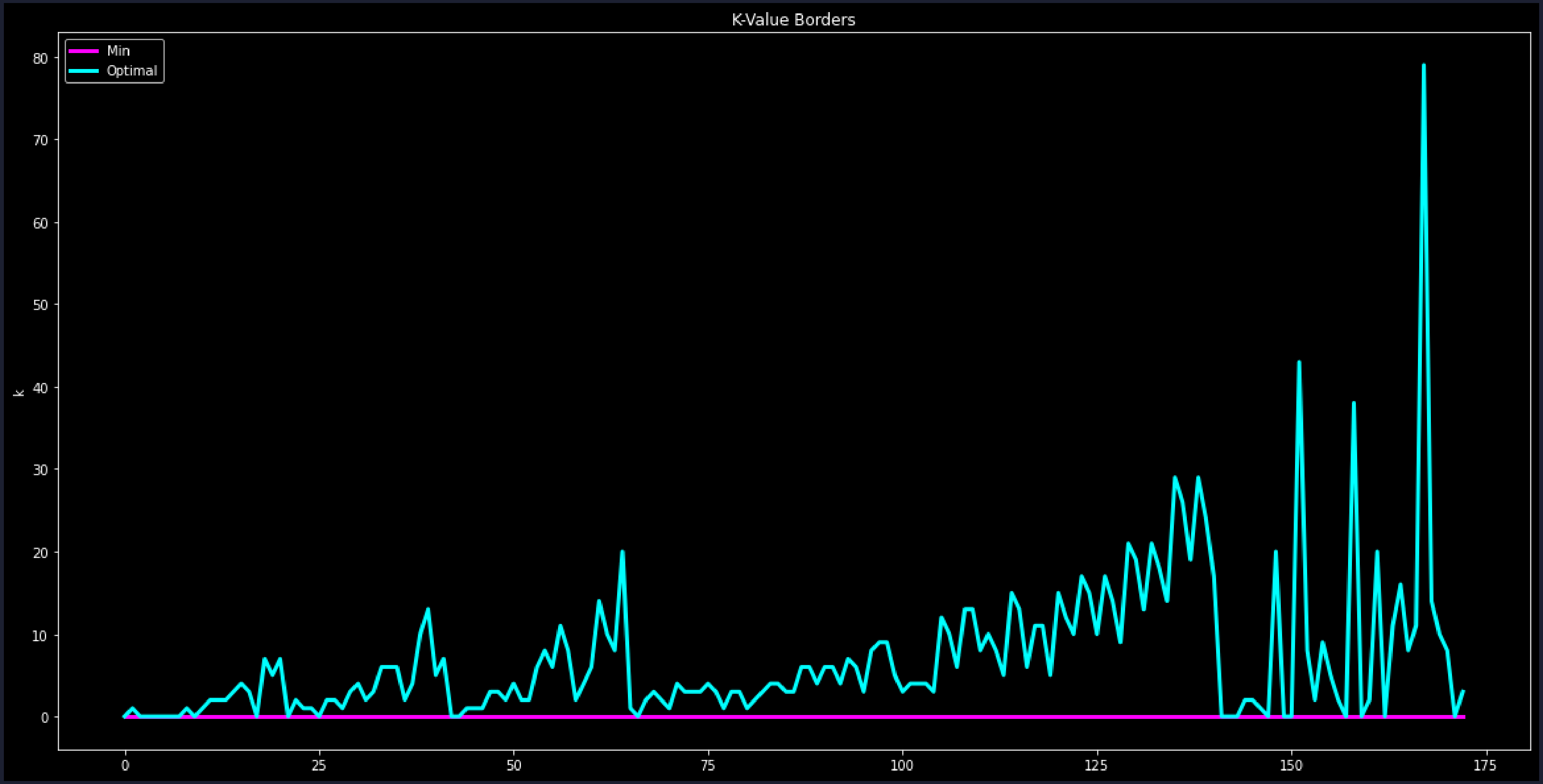
Calculation of min k

1. max $m = n * (n - 1)$

2. min $k = 0$ if $m \leq n * (n - 1) / 2$

|E|	k
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	1
10	2
11	2
12	3

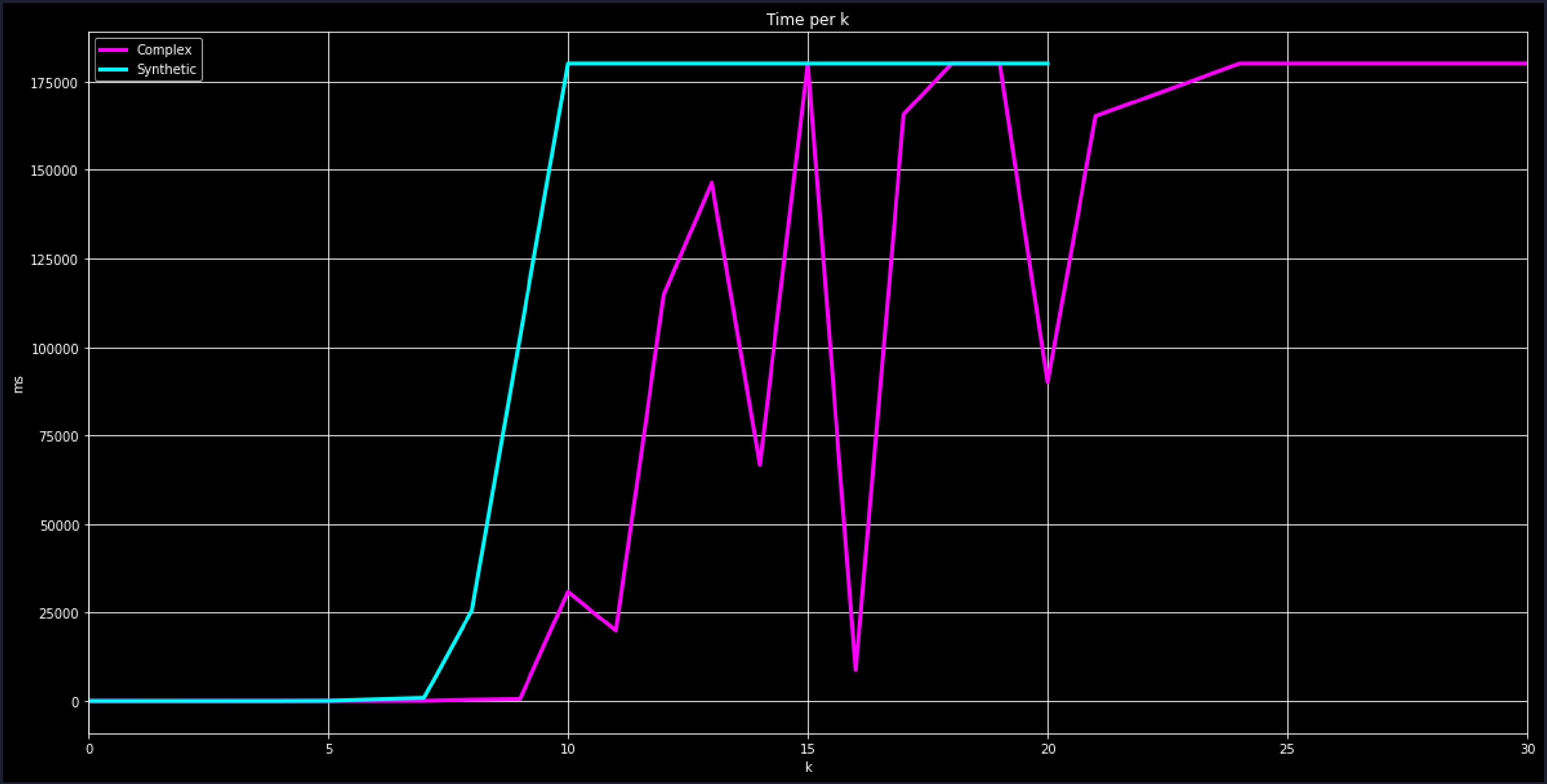
Calculation of min k



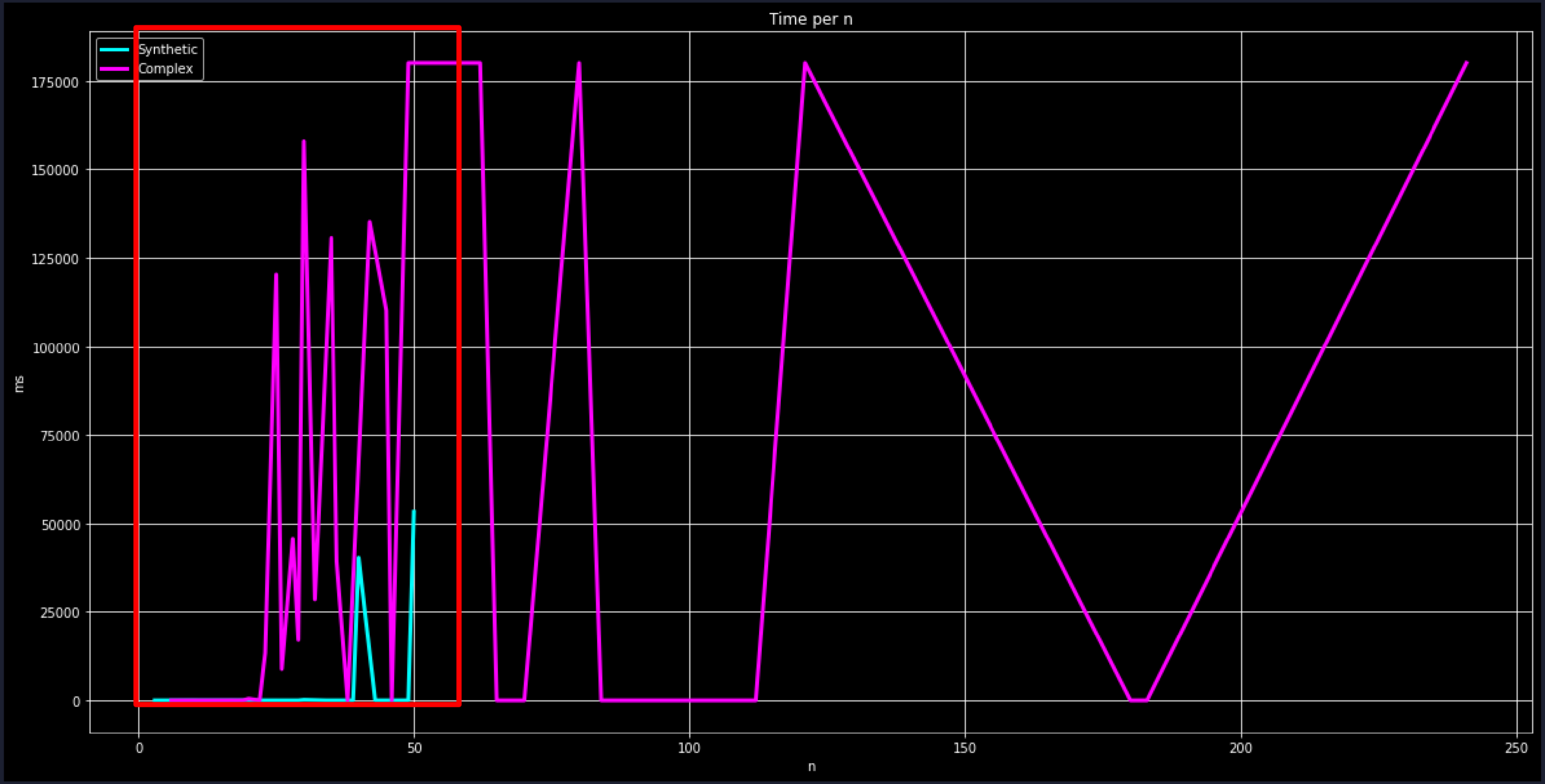
Performance

- → We set a timeout after 3 minutes for the plots
- → The best we could solve was k = 21
- → The algorithm was executed on an i9-9900K
- → Next time we will show the server performance too

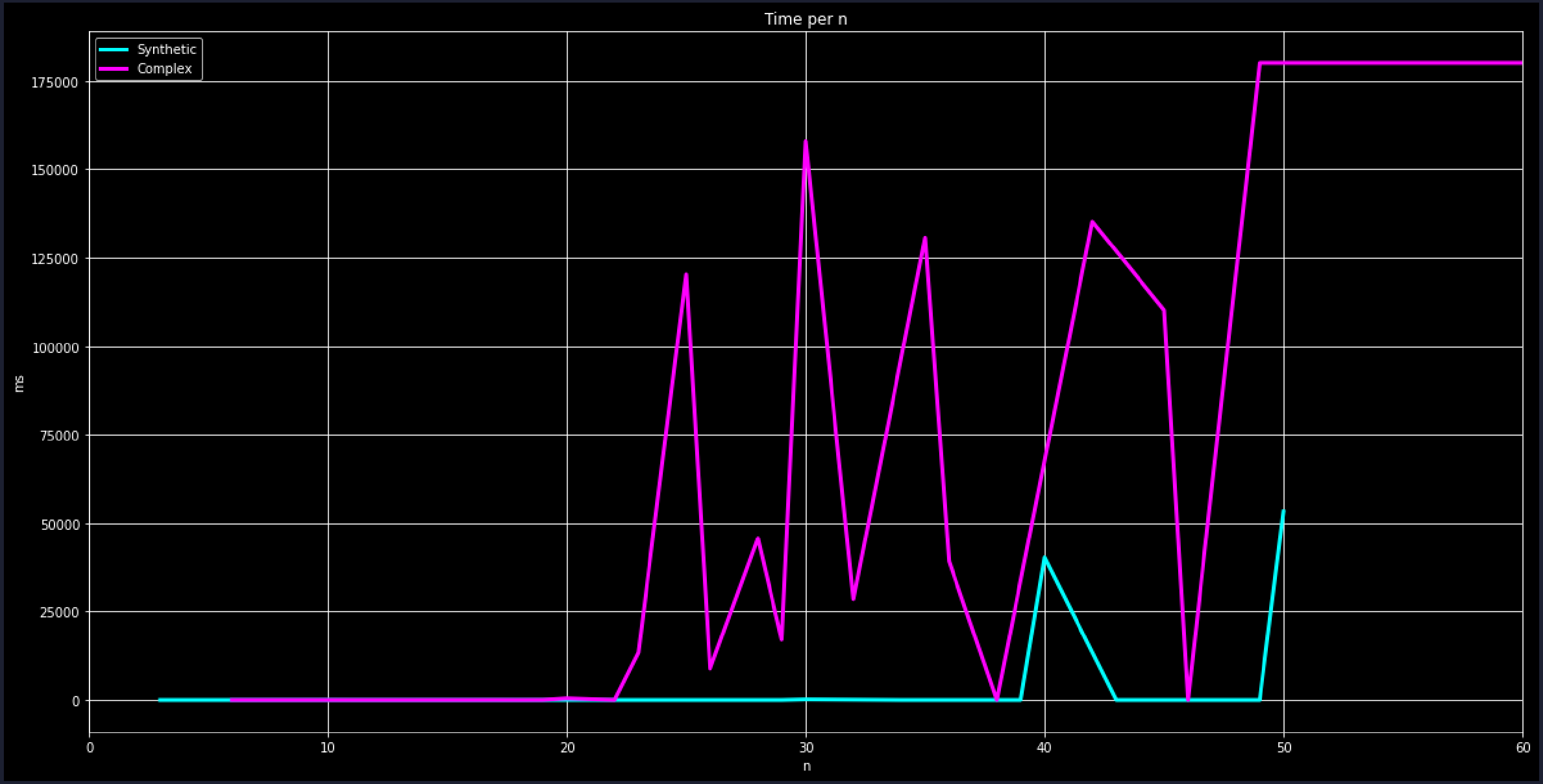
Performance – per k



Performance – per n



Performance – per n



Do you have any questions?