```swift
        }
    }

    class JobGraph {
        var nodes: [JobNode]
        var graph: [Int: JobNode]

        init(jobs: [Int]) {
            nodes = [JobNode]()
            graph = [Int: JobNode]()
            for job in jobs {
                addNode(job: job)
            }
        }

        func addNode(job: Int) {
            let jobNode = JobNode(job: job)

            nodes.append(jobNode)
            graph[job] = jobNode
        }

        func addDependencyToJob(job: Int, dependency: Int) {
            let jobNode = getNode(job: job)
            let dependencyNode = getNode(job: dependency)
            jobNode.dependencies.append(dependencyNode)
            dependencyNode.numberOfPrerequisites += 1
        }

        func getNode(job: Int) -> JobNode {
            if let node = graph[job] {
                return node
            } else {
                graph[job] = JobNode(job: job)
                return graph[job]!
            }
        }
    }

    // O(j + d) time | O(j + d) space
    func topologicalSort(jobs: [Int], dependencies: [[Int]]) -> [Int] {
        let jobGraph = createJobGraph(jobs: jobs, dependencies: dependencies)
        return getOrderedJobs(jobGraph: jobGraph)
    }

    func createJobGraph(jobs: [Int], dependencies: [[Int]]) -> JobGraph {
        let jobGraph = JobGraph(jobs: jobs)

        for dependency in dependencies {
            let job = dependency[0]
            let dep = dependency[1]
            jobGraph.addDependencyToJob(job: job, dependency: dep)
        }

        return jobGraph
    }

    func getOrderedJobs(jobGraph: JobGraph) -> [Int] {
        var orderedJobs = [Int]()
        var nodesWithNoPrerequisites = jobGraph.nodes.filter { $0.numberOfPrerequisites == 0 }

        while nodesWithNoPrerequisites.count > 0 {
            if let jobNode = nodesWithNoPrerequisites.popLast() {
                orderedJobs.append(jobNode.job)
                removeDependencies(jobNode: jobNode, nodesWithNoPrerequisites: &nodesWithNoPrerequisites)
            }
        }

        let graphHasEdges = jobGraph.nodes.filter { $0.numberOfPrerequisites > 0 }.count > 0

        return graphHasEdges ? [] : orderedJobs
    }

    func removeDependencies(jobNode: JobNode, nodesWithNoPrerequisites: inout [JobNode]) {
        while jobNode.dependencies.count > 0 {
            if let dependency = jobNode.dependencies.popLast() {
                dependency.numberOfPrerequisites -= 1

                if dependency.numberOfPrerequisites == 0 {
                    nodesWithNoPrerequisites.append(dependency)
                }
            }
        }
    }
}
```