```java

class Program {
  // O(j + d) time | O(j + d) space
  public static List<Integer> topologicalSort(List<Integer> jobs, List<Integer[]> deps) {
    JobGraph jobGraph = createJobGraph(jobs, deps);
    return getOrderedJobs(jobGraph);
  }

  public static JobGraph createJobGraph(List<Integer> jobs, List<Integer[]> deps) {
    JobGraph graph = new JobGraph(jobs);
    for (Integer[] dep : deps) {
      graph.addPrereq(dep[1], dep[0]);
    }
    return graph;
  }

  public static List<Integer> getOrderedJobs(JobGraph graph) {
    List<Integer> orderedJobs = new ArrayList<Integer>();
    List<JobNode> nodes = new ArrayList<JobNode>(graph.nodes);
    while (nodes.size() > 0) {
      JobNode node = nodes.get(nodes.size() - 1);
      nodes.remove(nodes.size() - 1);
      boolean containsCycle = depthFirstTraverse(node, orderedJobs);
      if (containsCycle) return new ArrayList<Integer>();
    }
    return orderedJobs;
  }

  public static boolean depthFirstTraverse(JobNode node, List<Integer> orderedJobs) {
    if (node.visited) return false;
    if (node.visiting) return true;
    node.visiting = true;
    for (JobNode prereqNode : node.prereqs) {
      boolean containsCycle = depthFirstTraverse(prereqNode, orderedJobs);
      if (containsCycle) return true;
    }
    node.visited = true;
    node.visiting = false;
    orderedJobs.add(node.job);
    return false;
  }

  static class JobGraph {
    public List<JobNode> nodes;
    public Map<Integer, JobNode> graph;

    public JobGraph(List<Integer> jobs) {
      nodes = new ArrayList<JobNode>();
      graph = new HashMap<Integer, JobNode>();
      for (Integer job : jobs) {
        addNode(job);
      }
    }

    public void addPrereq(Integer job, Integer prereq) {
      JobNode jobNode = getNode(job);
      JobNode prereqNode = getNode(prereq);
      jobNode.prereqs.add(prereqNode);
    }

    public void addNode(Integer job) {
      graph.put(job, new JobNode(job));
      nodes.add(graph.get(job));
    }

    public JobNode getNode(Integer job) {
      if (!graph.containsKey(job)) addNode(job);
      return graph.get(job);
    }
  }

  static class JobNode {
    public Integer job;
    public List<JobNode> prereqs;
    public boolean visited;
    public boolean visiting;

    public JobNode(Integer job) {
      this.job = job;
      prereqs = new ArrayList<JobNode>();
      visited = false;
      visiting = false;
    }
  }
}

```