Leader Election Algorithm

- Primary function of leader election algorithms is to designate a single processor from a collection of processors to serve as a coordinator.
- If the selected coordinator experiences a failure for any reason, a new coordinator is chosen from the remaining processors.
- The process with the highest priority is chosen as the new coordinator.
- Consequently, when a coordinator failure occurs, this algorithm will select the active process with the highest priority number.

Bully Algorithm:

- The Bully Algorithm is a dynamic leader election technique in distributed computing.
- It ensures that a leader is always present in the system, even in the event of failures or recoveries.
- The provided code demonstrates the Bully Algorithm in a distributed system. It shows how the system can recover from a leader failure and elect a new leader.
- It is an example implementation that uses a Node class to represent a process and an Election class to manage the election procedure.
- Node: Each Node has a unique id and an isActive status indicating its state.

```
class Node {
    public int nodeId;
    public boolean isActive;

    public Node(int nodeId) {
        this.nodeId = nodeId;
        this.isActive = true;
    }
}
```

- Election: This class oversees the election process. It contains a Scanner object for input, an array of Node objects, and an integer nodeCount representing the total number of nodes.

```
public class Election {
    Scanner input;
    Node[] nodes;
    int nodeCount = 5;
```

Methods:

- setupNodes(): This method sets up the nodes in the system.

```
public void setupNodes() {
    nodes = new Node[nodeCount];
    for (int i = 0; i < nodeCount; i++) {
        nodes[i] = new Node(i);
    }
}
```

- startElection(): This method initiates the election process. It first simulates a node failure by making the highest ID node inactive.

```
public void startElection() {

    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    int startNodeId = 0;
    System.out.println("Node " +startNodeId + " sends message to node "+
        nodes[getHighest()].nodeId);
    System.out.println("Node no " + nodes[getHighest()].nodeId + " fails");
    nodes[getHighest()].isActive = false;
```

Then it starts the election process from the node with the lowest ID. If there are active nodes with higher IDs, the current node sends them an election message. If the higher ID nodes are active, they respond with an OK message, and the current node increments its ID. If there are no active nodes with higher IDs, the current node becomes the leader and sends a coordinator message to all lower ID nodes.

```
        boolean electionInProgress = true;
        while (electionInProgress) {

            boolean higherNodesExist = false;
            for (int i = startNodeId + 1; i < nodeCount; i++) {
                if (nodes[i].isActive) {
                    System.out.println("Node " + startNodeId + " passes
                        Election(" + startNodeId + ") message to Node " + i);
                    higherNodesExist = true;
                }
            }
```

```
            for (int i = startNodeId + 1; i < nodeCount; i++) {
                if (nodes[i].isActive) {
                    System.out.println("Node " + i + " passes Ok(" + i + ")
                        message to Node " + startNodeId);
                }
            }
            startNodeId++;

        } else {
            int leader = nodes[getHighest()].nodeId;
            System.out.println("Finally Node " + leader + " becomes Leader"
                );
            for (int i = leader - 1; i >= 0; i--) {
                if (nodes[i].isActive) {
                    System.out.println("Node " + leader + " passes
                        Coordinator(" + leader + ") message to Node " + i);
                }
            }

            System.out.println("End of Election");
            electionInProgress = false;
            break;
        }
    }
```

- getHighest(): This method returns the index of the active node with the highest ID.

```
public int getHighest() {
    int highestId = -99;
    int highestIdIndex = 0;
    for (int i = 0; i < nodes.length; i++) {
        if (nodes[i].isActive && nodes[i].nodeId > highestId) {
            highestId = nodes[i].nodeId;
            highestIdIndex = i;
        }
    }
    return highestIdIndex;
}
```

Output:

```
Node 0 sends message to node 4
Node no 4 fails
Node 0 passes Election(0) message to Node 1
Node 0 passes Election(0) message to Node 2
Node 0 passes Election(0) message to Node 3
Node 1 passes Ok(1) message to Node 0
Node 2 passes Ok(2) message to Node 0
Node 3 passes Ok(3) message to Node 0
Node 1 passes Election(1) message to Node 2
Node 1 passes Election(1) message to Node 3
Node 2 passes Ok(2) message to Node 1
Node 3 passes Ok(3) message to Node 1
Node 2 passes Election(2) message to Node 3
Node 3 passes Ok(3) message to Node 2
Finally Node 3 becomes Leader
Node 3 passes Coordinator(3) message to Node 2
Node 3 passes Coordinator(3) message to Node 1
Node 3 passes Coordinator(3) message to Node 0
End of Election
```

- First node 0 sends message to node 4. Now suppose node 4 fails.

- Node 0 sends election messages to all higher active nodes i.e. nodes 1, 2 and 3 which then responds with ok messages.
- Now all these nodes who received election messages, sends election messages to all higher active nodes as shown in output.
- Finally the node which does not receive response from any higher active node, becomes the leader i.e. node 3.
- Node 3 then informs all other nodes about the new co-ordinator.