Evaluating leader election algorithms (LCR, HS)

1- Introduction

Leader election is a fundamental problem in distributed systems, especially in ring networks, where processes must coordinate to elect a single leader. This report evaluates two classical leader election algorithms:

- LCR (Le Lann, Chang, Roberts) Algorithm: A simple algorithm where nodes forward their unique IDs clockwise until the largest ID completes a full cycle.
- HS (Hirschberg-Sinclair) Algorithm: A more optimised algorithm where nodes initiate leader election in exponentially increasing probe distances (2^k).

The goal is to experimentally validate the correctness of these algorithms and compare their time complexity (rounds) and message complexity (total messages sent).

2- Implementation details

The simulator is designed to model a bidirectional ring network of n processors operating in synchronous rounds. Each processor has a unique ID chosen from the set {1, 2,…,αn} (with α = 3), and can communicate with its clockwise and counterclockwise neighbours. The implementation ensures both algorithms terminate, where all processors eventually know the elected leader (the processor with the maximum ID).

The following classes form the core of the simulator:

- **Message.java**: Represents messages exchanged between processors, supporting fields for message type, ID, direction (for HS), and hop count (for HS).
- **Processor.java**: Models a processor in the ring, maintaining state (e.g. ID, status, leader ID), tracking received messages, and interacting with neighbours via an algorithm-specific delegate.
- **RingNetwork.java**: Initializes and manages the ring topology, linking processors bidirectionally and generating IDs based on user input (ascending, descending, or random).
- **LeaderElectionAlgorithm.java**: An interface defining the contract for algorithm-specific behaviour (initialize, getMessagesToSend, isTerminated).
- **LCRAlgorithm.java**: Implements LCR, where each processor sends its own ID clockwise in round 1, then on subsequent rounds, it compares incoming IDs with its own.
- **HSAlgorithm.java**: Implements HS, by using phases (phaseMap) with distance 2^^{phase} stored as ‘hopCount’. Nodes that see a larger ID drop out (activeMap). Each “out” message decrements ‘hopCount’ each time it traverses an edge. If ‘hopCount’ hits 1 at a smaller node, it becomes an "in" message.
- **LeaderElectionSimulator.java**: Coordinates the simulation by handling user input, running the chosen algorithm, and tracking performance metrics (rounds, messages, correctness).

These classes work together to simulate synchronous rounds, ensuring messages are processed hop-by-hop, with each round following the pattern: reading messages, updating state, generating new messages, and transmitting them.

2.1- Termination

- LCR: Once the highest ID returns to its origin node, that node declares itself as the leader and broadcasts a termination message.

- HS: When a node's "out" probe completes a full cycle, the node declares itself as the leader and broadcasts a termination message.
- The simulator stops once all processors have a non-null 'leaderID'.

3- Experimental setup

We conducted 11 experiments for each ID assignment (ascending, descending, random) to test the two algorithms with varying ring sizes. The ring sizes start from 50,100,200 up to 1000.

For each experiment, the following were recorded:

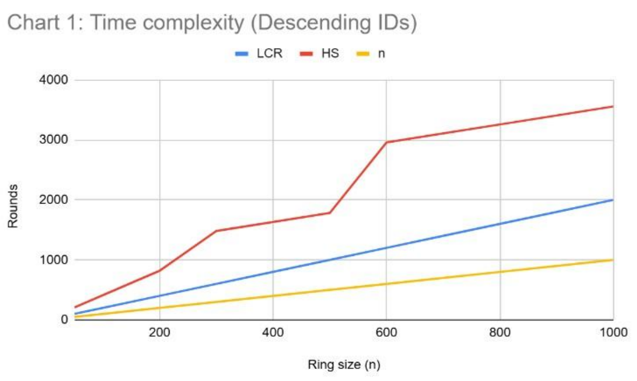
- Total Rounds: The number of synchronous steps required until all nodes recognize the leader.
- Total Messages: The number of messages exchanged throughout execution.
- Correctness Check: Ensures exactly one leader is elected and known by all processors.

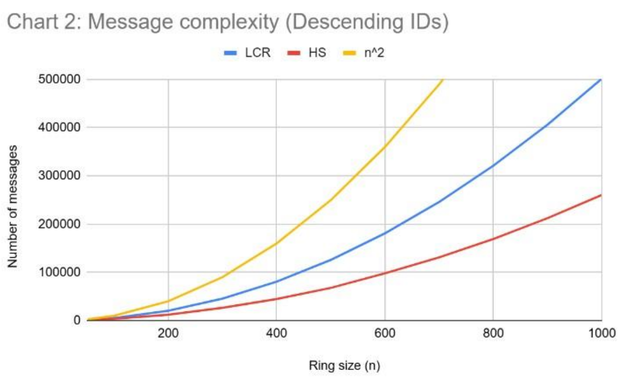
4- Results

• Descending ID assignment:

Ring size (n)	**Number of messages (LCR)**	**Number of messages (HS)**	**Number of rounds (LCR)**	**Number of rounds (HS)**	**LCR correctness**	**HS correctness**
50	1325	1198	101	209	Yes	Yes
100	5150	3654	201	413	Yes	Yes
200	20300	12316	401	820	Yes	Yes
300	45450	26490	601	1483	Yes	Yes
400	80600	44640	801	1633	Yes	Yes
500	125750	67790	1001	1783	Yes	Yes
600	180900	97988	1201	2958	Yes	Yes
700	246050	131138	1401	3108	Yes	Yes
800	321200	169288	1601	3258	Yes	Yes
900	406350	212438	1801	3408	Yes	Yes
1000	501500	260588	2001	3558	Yes	Yes

Table 1: Results for Descending Id assignments

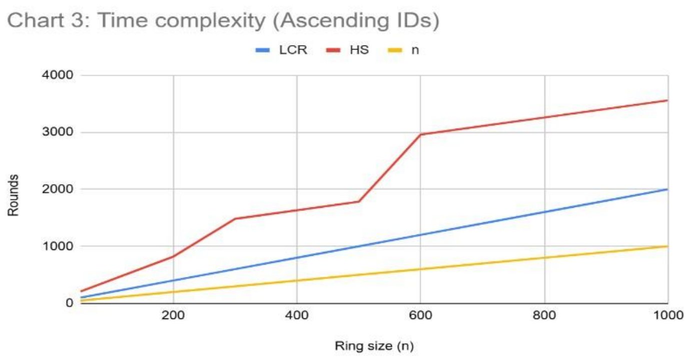


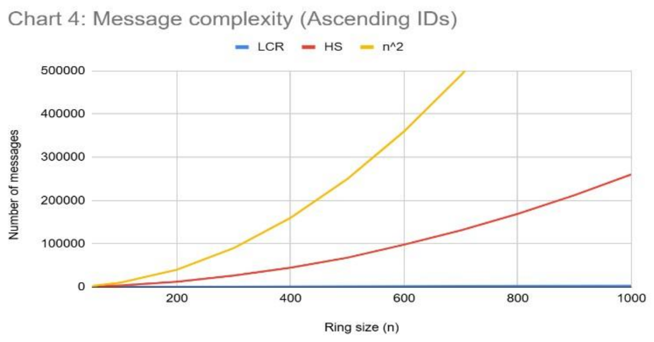


• **Ascending ID assignment:**

Ring size (n)	Number of messages (LCR)	Number of messages (HS)	Number of rounds (LCR)	Number of rounds (HS)	LCR correctness	HS correctness
50	149	1198	101	209	Yes	Yes
100	229	3654	201	413	Yes	Yes
200	599	12316	401	820	Yes	Yes
300	899	26490	601	1483	Yes	Yes
400	1199	44640	801	1633	Yes	Yes
500	1499	67790	1001	1783	Yes	Yes
600	1799	97988	1201	2958	Yes	Yes
700	2099	131138	1401	3108	Yes	Yes
800	2399	169288	1601	3258	Yes	Yes
900	2699	212438	1801	3408	Yes	Yes
1000	2999	260588	2001	3558	Yes	Yes

Table 2: Results for ascending Id assignments

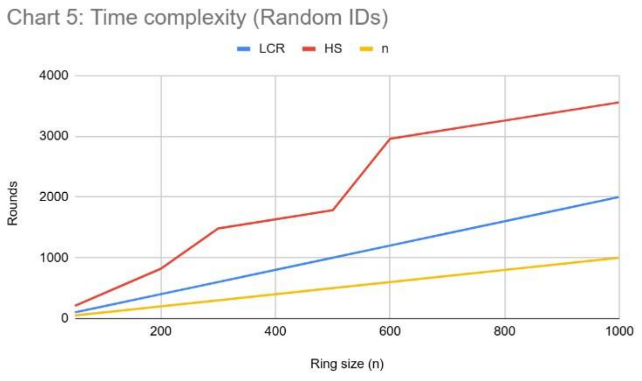


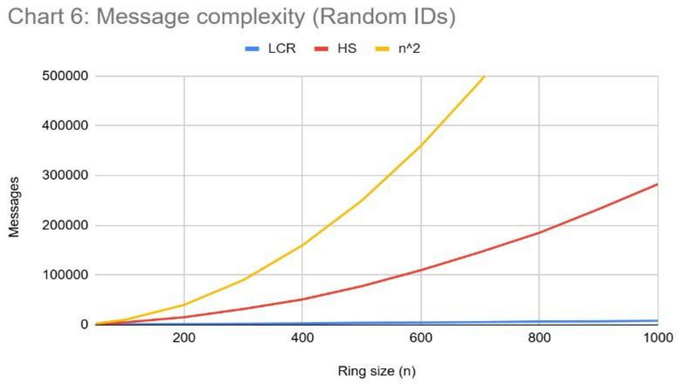


• **Random ID assignment:**

Ring size (n)	Number of messages (LCR)	Number of messages (HS)	Number of rounds (LCR)	Number of rounds (HS)	LCR correctness	HS correctness
50	267	1674	101	209	Yes	Yes
100	589	4935	201	413	Yes	Yes
200	1296	15321	401	820	Yes	Yes
300	2115	31921	601	1483	Yes	Yes
400	2923	51155	801	1633	Yes	Yes
500	4025	77804	1001	1783	Yes	Yes
600	4749	109972	1201	2958	Yes	Yes
700	5351	146169	1401	3108	Yes	Yes
800	6662	185165	1601	3258	Yes	Yes
900	7014	232746	1801	3408	Yes	Yes
1000	8399	282944	2001	3558	Yes	Yes

Table 3: Results for random Id assignments





5- Discussion

5.1- Correctness:

As illustrated from Tables 1,2,3 Both LCR and HS algorithms successfully elected a single leader (the processor with the maximum ID) in all tested scenarios, spanning ring sizes from 0 to 1,000 and ID arrangements (clockwise ordered, counterclockwise ordered, and random). No instances of multiple leaders or election failures occurred, affirming the implementations' correctness within our synchronous simulation framework.

5.2- Time complexity

- **LCR:**

In theory, the time complexity (number of rounds) for the LCR algorithm is O(n), and for the terminating version it requires another O(n). Thus, O(n) is the overall complexity. Charts 1, 3, and 5 show that the number of rounds in our implementation is the same for all ID assignments. The algorithm requires exactly 2n + 1 rounds, which aligns perfectly with the theoretical time complexity for the LCR algorithm.

- **HS:**

According to Hirschberg-Sinclair's paper, the theoretical time complexity for the HS algorithm should be O(log n), and another O(n) for termination. So, the overall time complexity is O(n) in the worst case. In our implementation, the number of rounds is still the same for all ID assignments, and it is approximately 4n. This is still linear, so it aligns with the theoretical time complexity.

We can notice from charts 1,3,5 that the number of rounds is always higher than the LCR algorithm. This does not mean that the HS algorithm is less efficient, but its due to the way we implemented the algorithm (refer to section 6).

5.3- Message complexity

LCR:

The theoretical message complexity depends on IDs ordering.

- Ascending IDs (Clockwise): the algorithm transmits a maximum of O(n) messages. This is actually the best case, as the maximum ID is only transmitted (n) times. This is confirmed by Table 2, which shows that the number of messages is nearly 3n (e.g. 149 for n=50, 2999 for n=1,000).

- Descending IDs (Counterclockwise): This is the worst-case scenario because the largest ID starts at the ring end. The number of messages approach a maximum of O(n^2). From chart 2, when n = 1000, messages reach 500,500 (via n^2 / 2 + n), reflecting each hop's transmission as the maximum ID traverses the ring.

- Random IDs: similar to descending IDs, messages average O(n^2), with Table 3 indicating 267 for n=50 and 8399 for n=1,000, reflecting higher communication due to unpredictable ID distributions.

HS:

The theoretical message complexity according to the paper introduced by Hirschberg and Sinclair (1980) Is O(n log n), with another O(n) for termination. In our implementation:

- Ascending IDs: messages range from O(n log n) to $O(n^2)$, with Table 2 showing 1198 for n=50 and 260,588 for n=1,000. The bidirectional ring significantly increases message counts compared to LCR.
- Descending IDs: the same as ordered IDs.
- Random IDs: messages consistently hit $O(n^2)$, as seen in Table 3 (1674 for n=50, 282,944 for n=1,000), due to dense communication across phases.

Our implementation is clearly much higher than the theoretical message complexity, which will be explain in the next section.

6- Implementation Discrepancy: HS time and message complexity

Hirschberg-Sinclair's paper treats a probe traveling 2^k edges as one "atomic" step, counting that as a single message and round. However, in our simulation messages move hop-by-hop, where each edge traversal is a new message and round. Therefore, a single "out" probe intended to move 2^k edges contributes 2^k messages and rounds, rather than just one. This inflates HS's message count to near $O(n^2)$ and time complexity (rounds) to O(n log n) , compared to the theoretical O(n log n)messages and O(log n) rounds in a multi-hop model.

7- Conclusion

Both LCR and HS reliably elected a single leader across all tests, confirming their correctness. LCR is simple and efficient for ordered IDs, requiring ~2n rounds and between O(n) to $O(n^2)$ messages, but scales poorly in worst cases (e.g. 501,500 messages for n=1,000 descending IDs). HS provides faster convergence with ~3,558 rounds for n=1,000, however, because of our hop-by-hop model, its message count (e.g. 282,944 for n=1,000 random IDs) approaches $O(n^2)$, exceeding the theoretical O(n log n) messages.

HS suits large networks prioritising rounds, while LCR is better for small, ordered rings.

Future Improvements: A multi-hop approach for the HS algorithm would more closely match HirschbergSinclair paper, producing near O(n log n) messages and O(log n) rounds in real runs.