# Problem

From n nodes we want to choose one that will be the leader of the rest.
Conditions:

1. Each node don't know the number of members.
2. Unordered ring.
3. Synchronized rounds.
4. "left" processor may not mean the same to all processors, so their lists of neighbours can be not in logical order.

# Overview

This algorithm, provided by Hirschberg and Sinclair [1], is an efficient way two solw Leader problem, requiring $\mathcal{O}(n \log n)$ messages. Previous successes belong to:

- LeLann [2] algorithm, wich requires $O(n^2)$ messages.
- Chang and Roberts [3]: average - O(n log n), but worst case still $\mathcal{O}(n^2)$

# Algorithm

A naive algorithm, which achieves $\mathcal{O}(n^2)$, involves simply passing our $Id$ in different directions. However, this quadratic time complexity can be reduced to $\mathcal{O}(n)$ rounds. Our algorithm takes steps of length $2^i$, during which the number of active processes is reduced. Those that fall within the range $2^i$ from a more possible candidate for the role of leader will become inactive, resulting in a significant reduction in the number of messages sent.
"Must have" to be in head:

1. *sendpass* - sending message in same direction as recieved was going.
2. *sendecho* - returns message back to sender of recieved message.

Lets take a look on pseudocode from [1].

---

```
The Algorithm

/// if after coming back process's messages it still has
/// "candidate" status, it means, that it is the biggest in
/// range 2^i around him
run_candidate:
    status <--"candidate"
    maxnum <-- 1
    WHILE status = "candidate"
        sendboth ("from", myvalue, 0, maxnum)
        await both replies (but react to other messages)
        IF either reply is "no" THEN status <-- "lost"
        maxnum <-- 2*maxnum


/// the function thanks to wich candidates eleminate number of possible canditates
/// and have abbility to get info, that they have won
receive_message ("from", value, num, maxnum):
    IF value < myvalue
        sendecho ("no", value)
    IF value > myvalue
        status <-- "lost"
        num <- num + 1
```

```
        IF num < maxnum
            sendpass ("from", value, num, maxnum)
        ELSE
            sendeeho ("ok", value)
    IF value = myvalue
        status <- "won"


///besides accepting his own message, which came back to him,
///the processor can play the role of a transit, inactive node
receive_message ("no", value) or ("ok", value)
    IF value != myvalue
        senpass the message
    ELSE
        this is a reply the processor was awaiting

///one from me, just for simle and correct ending
///of this election, eleminating all possible unread messages
receive_message ("end", value):
    inform next processor of end
    stop process
```

---

This pseudocode is more suitable for a solution based on maintaining references to neighboring processes. However, in the implementation I provided, solution stuck to to passing information through channels, requiring a slight modification in the code structure. Nevertheless the given pseudocode effectively reflects the algorithm's flow, as much, as it reduces the number of messages).

# Correctness

There is not much to say:

1. if process is "lost", then it is lesser than some other process, so it is OK, that it become inactive
2. if process is still "candidate", then it is bigger, than met by him.
3. if "from" message comes back to original sender - than it passed all the processes with success, so original sender is greater than each one of the rest.
4. when $maxnum >= n$ then the algo would end, because somebody would cover all the processes by "from" message.
5. if process is "won", then it received message after it visited all the processes

# Proof of complexity

## The worst case

Note that each active processor by the start of a round $i$(starting with 0) can initiate at most $4 * 2^i$ messages to be sent.
Another observation gives us on start of round $i$:

$$\textbf{number of "candidate" processes} \leq \left\lceil \frac{n}{2^{i-1}+1} \right\rceil$$

It follows from the fact, that if we separate all nodes on blocks of length $2^{i-1}+1$, then after round $i-1$ only one of nodes(processes) in block could stay active.

It all provides us to the fact that total number of sent messages is bounded from above by:

$$4 \cdot (\sum_{0}^{\lceil \log n \rceil} 2^i \cdot \left\lceil \frac{n}{2^{i-1}+1} \right\rceil) \quad \leq \quad 4 \cdot (\sum_{0}^{\lceil \log n \rceil} 2n) \quad \approx \quad 8\, n\, \log n$$

### The "better can't imagine" case

In case we would have instance of size $2^k$ which look like this:

$$\cdots p_{n-1} < p_0 > p_1 > p_2 \cdots$$

After first round the is possibility, that could happen, that every one but the biggest process ($p_0$), would get "lost" status, what provide us to comlexity:

$$4 \cdot n + \sum_{0}^{\lceil \log n \rceil} 2^i \cdot 1 \;=\; 4 \cdot n + \sum_{0}^{k} 2^i \;<=\; 4 \cdot n + 2 \cdot n \;=\; \mathcal{O}(n)$$

## References


[1] Hirschberg, D.S., Sinclair, J.B.: Decentralized extrema-finding in circular configurations of processes. Commun. ACM 23, 627–628 (1980)

[2] LeLann, G. Distributed systems–Towards a formal approach. Inform. Proc. 77, North-Holland Pub. Co., 1977, Amsterdam, pp. 155-160.

[3] Chang, E., and Roberts, R. An improved algorithm for decentralized extrema-fmding in circularly configuraiions of processes. Comm. ACM 22, 5 (May 1979), 281-283.