Distributed Algorithms

1) To prove that the given **k-set consensus algorithm** is **f-resilient to crash failures**, we must demonstrate two key properties:

1. **Validity**: Every output y_i is one of the initial inputs $\{x_1, x_2, \dots, x_n\}$
2. **k-Agreement**: The set of all outputs contains at most kkk distinct values.

At each round, processors merge their V sets by taking the union of their own V with the sets received from other processors.
At the final round (r), each processor outputs y=min(V).
Validity is easy to show because all he output is the min v and because we assume only faulty nodes the values must have been recive by an input.
k-Agreement is harder to prove, but it is close to what we saw during the lecture. Because there are $\frac{f}{k} + 1$ rounds, there is at least one round where there are fewer than k faulty nodes. Because of this, the maximum number of different values can be k at this round. The reason for this is that all the $n - f$ good nodes will have values between one and k values, either receiving at most kk−1 values in the worst case, with the other nodes adding one more value.

1. **The Critical Round with Fewer Faulty Nodes:**
   - Since the algorithm runs $\frac{f}{k} + 1$ rounds, there must be at least **one round where fewer than k faulty nodes influence the communication**. This is because the failures are spread across f faulty nodes, and the rounds allow enough time for information from non-faulty nodes to propagate.
2. **Bounding the Number of Values:**
   - In the worst case, a set V at any node can include at most k distinct values during this critical round. Here's why:
     - All non-faulty nodes collectively merge their V sets by receiving values from each other, and the number of distinct values in V cannot increase beyond the initial contributions from non-faulty nodes and the influence of k−1 faulty nodes.
     - Even if the k−1 faulty nodes attempt to introduce new values, the propagation ensures that no more than k distinct values can exist across all V sets by the end of the critical round.

2)

a) To demonstrate that no consensus algorithm satisfies non-faulty-node input validity, we need to show that the trust between nodes can be exploited, leading to a violation of the validity condition. Assume the nodes must agree on an output value based on their inputs, which belong to the set xxx (the set of non-faulty node inputs).

A Byzantine node can send a fabricated value that is not in xxx, thereby disrupting the agreement process. For instance, if the consensus is designed to output the input value of a particular node iii, transforming node iii into a Byzantine node results in an output value that no longer satisfies the non-faulty-node input validity condition.

Thus, the presence of a single Byzantine node can render the consensus invalid because it undermines the guarantee that the agreed-upon value comes from the set of inputs provided by non-faulty nodes.

b) To prove this, we need to show that for a given algorithm A, in all cases, non-faulty validity equals all-same validity.

We will analyze each case, assuming nnn non-faulty nodes:

1. If all nodes have the same value (0 or 1), non-faulty validity means the output must match the input of one of the nodes. This implies the output will be either 0 or 1 accordingly (if all nodes have 0, the output is 0; if all nodes have 1, the output is 1). Similarly, all-same validity means that if all nodes have the same value, the output is the same as their value (e.g., if all have 0, the output is 0, and vice versa).
2. Assume kkk nodes have the value 0 and n− k nodes have the value 1, where 0<k<n. Non-faulty validity means the output must be either 0 or 1 (matching one of the non-faulty inputs). However, all-same validity has no restriction on the output because not all non-faulty nodes have the same value.

This shows the equivalence of non-faulty validity and all-same validity in cases where all nodes agree, and the divergence when they do not.

c) Improvments: **Increase Wireless Transmission Bandwidth**

- **Rationale**: If the N1 implant could transmit data at higher rates (e.g., 5 Mbps instead of 1 Mbps), the compression challenge would become significantly less complex. Increasing the transmission bandwidth would allow for less aggressive compression, reducing potential data loss and computational demands.
- **Implementation**: Explore advancements in wireless communication technologies, such as 5G or ultra-wideband (UWB) systems, that are both high-speed and low-power. Additionally, optimize antenna designs and signal processing techniques to improve throughput without compromising the implant's energy efficiency.

Research in Information Theory

- **Rationale**: Information theory provides a theoretical foundation for understanding data redundancy, entropy, and compression limits. By applying advanced principles like Shannon's entropy or modern research in channel coding, Neuralink can develop compression algorithms tailored to the neural signal domain.
- **Implementation**: Invest in interdisciplinary research combining neuroscience, information theory, and signal processing. Collaborate with academic institutions or hire domain experts to identify innovative methods that maximize data compression while preserving neural signal integrity.

3) Given a graph where $n \gg 10$, the algorithm works as follows:

1. **Run the MIS (Maximal Independent Set) Algorithm**:
    - The first step is to execute the MIS algorithm, which runs in $O(\log^* n)$ time as we learned in class.
2. **Initial Circle Formation**:
    - After running the MIS algorithm, we obtain a circle structure, as illustrated in **Figure 1**.
3. **Group Formation**:
    - Each non-MIS node checks its neighbors to identify a connection to at least one MIS node. This is guaranteed because of the properties of MIS.
    - This step runs in $O(1)$ time.

- Groups are formed as follows:
  - If a node has neighbors in both directions that belong to the MIS (e.g., n.leftn.leftn.left in MIS and n.rightn.rightn.right in MIS), it joins the group of the MIS node with the smallest ID: select(min(n.left.id,n.right.id))
  - Otherwise, the node joins the group of the single MIS node it is connected to.
- At this stage, we obtain groups of size 1 to 3, as illustrated in **Figure 2**.

4. **First Round of Joining**:
   - Nodes that belong to an MIS group but do not have additional members (i.e., groups consisting of a single MIS node) join their right neighbor. The directions "right" and "left" are arbitrary but consistent throughout the algorithm.
   - After this step, every group will have between 2 and 4 nodes.
   - This operation runs in O(1).
5. **Reapply the MIS Algorithm**:
   - The MIS algorithm is applied again, this time to the groups formed in the previous step.
   - The resulting groups will have sizes between 2 and 12, and the subunits within each group will range between 1 and 3.
   - This step runs in O(log∗n).
6. **Second Round of Joining**:
   - Groups with only one subunit join their neighbor, resulting in groups with sizes between 4 and 14.
   - This operation runs in O(1).
7. **Apply MIS Again**:
   - The MIS algorithm is executed once more on the updated groups.
   - This results in groups of size between 4 and 42.
   - This operation runs in O(log∗n).
8. **Third Round of Joining**:
   - Groups with a single subunit join their neighbor again, resulting in groups of size between 8 and 46.
   - This step runs in O(1).
9. **Final Grouping**:
   - At this stage, any group with a size between 8 and 10 is finalized as a segment.
   - Larger groups are split into smaller groups to meet the size constraints.
   - Since the maximum group size is bounded by n=46, this step runs in O(1).

Runtime Analysis:

The total runtime is the sum of all the steps:

$O(\log^* n)+O(1)+O(\log^* n)+O(1)+O(\log^* n)+O(1)=O(\log^* n)$

Conclusion:

This algorithm ensures that all groups adhere to the required size constraints while maintaining a total runtime of $O(\log^* n)$. Each step is efficient and respects the properties of the MIS algorithm and group formation process.

b) Assume we can solve the problem using algorithm FFF. This would mean that we can divide the ring into segments of size between 5 and 10 nodes. By repeatedly running this algorithm, we could eventually combine all segments into a single large segment. To do so, we would choose the leftmost segment in each iteration and designate a node within it as the leader.

However, it is well-known that leader election in a ring without unique IDs is impossible, as shown in class. In a ring topology with no unique identifiers, nodes are indistinguishable, and therefore no deterministic algorithm can elect a leader. Since the process of defining segments inherently requires distinguishing between nodes to assign them to specific segments and potentially elect a leader, this contradiction implies that algorithm FFF cannot solve the original problem.

Thus, the assumption that FFF can solve the problem leads to a contradiction, proving that without unique IDs, it is impossible to deterministically solve this task in a synchronous ring network.

~~Draft :~~

~~Given a graph where we have $n \gg 10$ our algorithm will work as follows,~~
~~1) we will first run MIS, with run in O(log * n) which we learned in the class,~~

~~2) we get a circle with as an example in the fig1,~~

~~3) we will create groups that each point not in the MIS check if he has a neighbor which is not is the MIS (he have to have a least one because of the MIS). O(1)~~

~~4) join into a groups as follows~~

~~If n.left in MIS and n.right in MIS. select(min(n.left.id, n.right.id))~~

~~Else choose the side where we have a MIS node,~~

~~We get groups of 1 to 3 (see example fig2)~~

~~5) We wait a round and left all sources MIS which have only themselves as root, to join there right neighbored (right and left are arbitrary) O(1)~~

~~6) After this Join of the round each node have between 2 to 4 nodes.~~

~~7) We will reapply MIS on the groups, which will get us groups between (2-12) but the different units are between (1-3) because each group is one unit O(log * n)~~

~~8) Then we do the same as (5) wait a round and joined the items, and we will get a groups of (4-14), and the new units are between (2-4) - O(1)~~

~~9) Then we activate MIS again and we get new groups of size (4-42) O(log *n)~~

~~10) Then we do the same as (5) wait a round and joined the items, and we get groups of (8-46)~~

~~11) each group that have between 8 to 10 is a group and the rest is splitting and create groups which is O(1) because the network size is bounded by n=46,~~

$$run\ time = O(\log(*\,n)) + O(1) + O(\log(*\,n)) + O(1) + O(\log(*\,n)) + O(1) = O(\log(*\,n))$$

~~b)~~

~~Assume we can solve the problom with algorithm F, that mean we will have segments of 5 to 10, and then we will rerun that the algorithm, up until we get one big segment, we will take the most left side segment, and we will set this node as the leader.~~

~~Beccuase we can not solve leader election in a cicle where there is no id's (we showed in the class) that mean F can not solve our problom which contradict.~~