Question 1

(i)

The processors will select maximum **4** different values.

In the scenario where the *consensus(id, 1, 3)* algorithm is executed and up to 3 processors can experience receiving failures, considering a system with n processors (p_1, p_2, p_3, …, p_n) where each processor p_i selects a value v_i:

Let's assume 3 processors - p_1, p_2, and p_3 have receiving failures, in the worst-case scenario, all 3 processors did not receive all messages sent by other processors during this round. Consequently, p_1, p_2, and p_3 end up receiving only their own initial values v_1, v_2, and v_3 due to these receiving failures. Other processors unaffected by these failures receive all values, including v_1, v_2, and v_3.

At the conclusion of the algorithm's execution:

- P_1 will select $\{v_1\}$.
- P_2 will select $\{v_2\}$.
- P_3 will select $\{v_3\}$.
- Processors unaffected by failures will select $v_k = min\{v_1, v_2, v_3, \ldots, v_n\}$.

So, if $v_k < v_1, v_2, v_3$, then the processors will select 4 different values which is v_1, v_2, v_3, and v_k.

(ii)

Proof.

To prove that invoking the algorithm consensus$(\mathrm{id}, \frac{\mathrm{f}}{4} + 1, \mathrm{f})$ on all processors solves the quarter-consensus problem when at most f processors can exhibit receiving failures, we need to show both correctness and termination:

Correctness: At least one fourth of the processors will select the same value.

Termination: The algorithm terminates in a finite number of rounds.

Proof of Correctness - Correct Output

Prove that value selected by the processor and at least one fourth of the processors select the same value.

The algorithm sends the current value V_{id} to all neighbors in each round and updates its $V_{selected}$ based on the minimum received value. There are n processors, and at most f processors can fail. That means there must be some round r such that $1 \leq r \leq f + 1$, at which no process fails. Therefore, all processors must agree on the same value.

In our case, we invoke the algorithm with the number of rounds $r = \frac{f}{4} + 1$, since at most f processors can fail and f is a multiple of 4, this means at least $\frac{f}{4}$ processors are guaranteed not to fail.

During each round, each processor sends its value to all neighbors. If a processor is not experiencing a receiving failure, it will receive values from all of its neighbors. Since at least $\frac{f}{4}$ processors are guaranteed not to fail, all processes that haven't yet failed will receive identical set of values. This ensures that consensus is achieved. Therefore, in each round, there is a subset of processors that can fully communicate, and they will select the minimum value among them. As we are running the algorithm for $\frac{f}{4} + 1$ rounds, at least one fourth of the processors will have to agree on the same minimum value.

Therefore, the algorithm satisfies the validity property for quarter-consensus.

Proof of Correctness – Termination

The algorithm runs for a fixed number of rounds ($r = \frac{f}{4} + 1$)

Therefore, it terminates in a finite number of rounds.

Question 2

The algorithm is unable to guarantee consensus among non-failing processors within r rounds due to one Byzantine-failing processor.

In the initial $r - 1$ rounds, the failing processor can send the correct values to all nodes to maintain the temporary consensus. However, in the last (rth) round, it can send different values (v_1 and v_2) from set U to non-failing processors.

Assume $U = \{v_1, v_2, v_3\}$ and $v_1 < v_2 < v_3$, if v_2 is the correct value from the previous round, since $v_1 \in U$ and $v_1 < v_2$, some non-failing processors will pick v_1 according to algorithm logic. Therefore, upon completion of the algorithm, non-failing nodes will exhibit disagreement.

Question 3

(i) In the system there are 10000 processors distributed over a ring with 10^6 rings of identifiers (keys). A consistent hashing algorithm allocates all processors at intervals being

100 distance apart. In order to make sure all the keys can be found; the finger tables of all processors should contain the successor of each processor as the first entries.

To minimize the number of messages needed to find a key, we should reduce the communication complexity. The formular of the communication complexity is

$k + \frac{n}{k} = k + \frac{10000}{k}$, when k =100 we get the minimum communication complexity.

In Chord each processor knows the addresses of at most m processors, where 2^m is the size of the ring of identifiers: $2^m = 10^6$

Let p_i be the process id, h_p be the hash function. Assume that each processor has a finger table that allows it to store its own address plus the addresses of two other processors. The following addresses must be stored in the finger table of each processor:

$$fingleTable[0] = successor\left(\left(h_p(p_i) + 1\right) mod\ 2^m\right)$$

$$= successor\left(\left(h_p(p_i) + 1\right) mod\ 10^6\right)$$

$$fingleTable[1] = successor\left(\left(h_p(p_i) + 10000\right) mod\ 2^m\right)$$

$$= successor\left(\left(h_p(p_i) + 10000\right) mod\ 10^6\right)$$

$$fingleTable[2] = successor\left(\left(h_p(p_i) + 1000000\right) mod\ 2^m\right)$$

$$= successor\left(\left(h_p(p_i) + 1000000\right) mod\ 10^6\right)$$

(ii) Since the successor is added to the first entry of every finger table, each processor can use finger table as a lookup table to find any keys that are stored in the system.

The maximum number of messages is the same as the communication complexity we calculated in section (i) which is $k + \frac{10000}{k}\ and\ k = 100$

Therefore, the maximum messages $= 100 + \frac{10000}{100} = 200$ messages.

Question 4

A synchronous distributed algorithm to find a given set of keys in a Chord P2P system by searching finger tables is implemented by **Find.java**