Question 1: Why Suzuki-Kasami Algorithm satisfy ME1 to ME3.

ME1: Since there is at most one token (the queue) in the system, at most one process can be in the critical section at a given time.

- Suppose that Pi and Pj are in the critical section at the same time. It would mean that a process Pk has sent the token queue to both Pi and Pj. So it would mean that both Pi and Pj were the head of the queue \( \Rightarrow \) impossible \( \Rightarrow \) Pi and Pj can't be in the critical section at the same time.

ME2: If process Pj requested to enter in the critical section but could not, it means that Pj hasn't received the token queue, because Pj was not the head of the queue.

- The queue is not empty (because at least Pj is in it) and Pj is not the head of the queue \( \Rightarrow \) it exists a process Pk which was the head of the queue \( \Rightarrow \) Pk is currently in the critical section \( \Rightarrow \) the system is still lively, there can't be deadlock.

ME3: A process Pk that wishes to enter in the critical section broadcasts the request to all processes involved in the system. So the process Pj, which is currently in the critical section, receives it too. Pj will enqueue for sure the process Pi at a given position (= the processes which are before him are known) in the queue Q, when Pj's critical section finishes.

Since the token is given to successive heads of Q, and that processes are added at the end of Q (positions remain the same until dequeueing), Pi will be eventually head of Q and the token queue will be given to Pj \( \Rightarrow \) Pi will enter the critical section as fairly as the other processes.
Question 2. Why Suzuki-Kasami Algorithm leads to a messages complexity of \( N \) by CS (where \( N \) is the number of processes)

Each time a process \( P_i \), \( 0 \leq i < N \), wants to enter the critical section, it broadcasts the request to all the other processes involved in the system.

\( \Rightarrow \) \( N-1 \) messages are sent for the entering CS phase.

Then the process which has the token makes local decisions to update the queue and find out who is the next process to be elected, when it finishes executing the critical section.

\( \Rightarrow \) no more messages are needed to decide which process will be the next to enter the critical section.

Then the process \( P_i \) which has the token gives it to the next process \( P_j \) which should have it.

\( \Rightarrow \) only one message is required to send the token from \( P_i \) to \( P_j \) when exiting the critical section.

So we have:
- entering CS phase: \( N-1 \) messages
- decisional phase: 0 messages
- exiting CS phase: 1 message

\( \Rightarrow N-1 + 0 + 1 = N \) messages are required by critical section entering to fulfill the properties of mutual exclusion with the Suzuki-Kasami algorithm.

Question 3

We could improve the fairness of this solution by making comparisons on sequence numbers of the latest visit to CS when the processes request are added in the queue \( Q \). This would be like giving more priority to the processes which entered less in the CS.

Thus, a process which has never required to enter in CS will be placed in first position of the sub-queue which corresponds to the requests sent during a given execution of the CS.

2+ Yes, but this is not queue while
The fairness of Mutex be better