Question 1: Suppose a protocol can tolerate (i.e. works when confronted with) byzantine failures. Will the same protocol tolerate (the same number of) crash failures?

Question 2: How many messages does the consensus protocol for crash failures exchange if there are no failures?
   Can you somehow optimize this?

Question 3: Consider an asynchronous system of \( n \) processes, \( f \) of which may fail by crashing (only). Let each process \( p \) have an input value \( C[p].in \in \{0,1\} \). Consider the following protocol for process \( p \).

\[
\begin{align*}
\text{forall } q \text{ (including } p) & \quad \text{send } C[p].in \text{ to } q. \\
\text{receive } n - f \text{ values and store them in the multiset } V. \\
\text{decide on } C[p].\text{decision} = \text{majority}(V)
\end{align*}
\]

(where \( \text{majority}(V) \) computes the majority of values in the multiset \( V \), returning 1 if there is a tie). Now answer the following questions.

a) Why can the algorithm only consider \( n - f \) received values (and no more) to compute the majority, even if no processes crashed?

b) Why can different processes decide on different values using this protocol?

c) How many 0 (or 1) valued inputs should there be initially, to guarantee that all correct processors decide on the same value?