Prove that you cannot do less than in floor (1.5n)-2 comparisons for finding the minimum and maximum elements in an array simultaneously.

Solution:

Let us say we have n elements in an array, out of which we need to find the maximum and minimum elements simultaneously.

- Suppose we maintain two lists of probable min and max which are having the same array elements in the beginning
  Example: A = [20, 4, 5, 1000, 70, 65]
  minList = [20, 4, 5, 1000, 70, 65] and maxList = [20, 4, 5, 1000, 70, 65]

- So, when we start comparing elements from the sets, say
  if **a < b**, **a** can be removed from the maximum list, and **b** can be removed from the minimum list.

  In this example, if we compare 20 and 4, 4 < 20 so 4 is removed from the max list, and 20 is removed from the list

- the final min and max list will be like
  minList = [4, 5, 65]
  maxList = [20, 1000, 70]

  and the minList and maxList are two disjoint sets which are formed in (n/2) pairwise comparisons

- Now we compare elements with the lists
  minList = [4, 5, 65] -> [4, 65] -> [4]
  maxList = [20, 1000, 70] -> [1000, 70] -> [1000]

  each time we reduce the size of the list by 1
  so, it is (n/2 - 1) comparisons for minList + (n/2 - 1) for maxList = n - 2 comparisons

total no of comparisons = n/2 comparisons + (n/2 - 1) + (n/2 - 1) = 3n/2 - 2 comparisons.

3n/2 – 2 comparisons are is the best way to decide min and max elements because, we are not making any duplicate comparisons, thus eliminating any forms of redundancy which can affect our program performance