Below is a strategy tree for sorting 3 numbers \(a_1, a_2, a_3\) using comparisons. In the tree, a square is used to denote a leave and each node is a comparison (we assume all the three numbers are distinct). Clearly, you have many other strategies to sort the 3 numbers. For this particular strategy, one needs 3 comparisons (the maximal number of comparisons on the paths in the tree, i.e., the height). To show the strategy is indeed a strategy (i.e., is able to sort any 3 numbers), you need only verify the following. Recall that \(a_1, a_2, a_3\) can be given as a permutation of 1,2,3. Each permutation corresponds to a path on the strategy tree. For instance, when \(a_1 = 1, a_2 = 2, a_3 = 3\), the path is \(yy\). When \(a_1 = 2, a_2 = 3, a_3 = 1\), the path is \(ynn\), etc. The thing we need to verify is that two permutations can not correspond to the same path on the tree.

\[
\begin{array}{c}
\text{a1}<\text{a2} \\
\text{y} \\
\text{a2}<\text{a3} & \text{a1}<\text{a3} \\
\text{y} & \text{y} \\
\text{a1}<\text{a3} & \text{a2}<\text{a3} \\
\text{y} & \text{y} & \text{y} & \text{y} \\
\end{array}
\]

Hence, to calculate \(M(n)\) in general, you need to find the number \(M(n)\) such that
1. there is a strategy tree with height \(M(n)\), for sorting \(n\) numbers,
2. there is no strategy tree with height \(M(n) - 1\), for sorting \(n\) numbers.

My thoughts:

- \textit{bag} – a variable represents a set of permutations of \(1234\cdots n\).
- \textit{split}(\textit{bag}, i, j), a function that returns a pair of \textit{bag}_1 and \textit{bag}_2. For each permutation \(P\) in \textit{bag}, if \(P[i] < P[j]\), then we put \(P\) in \textit{bag}_1; if \(P[i] > P[j]\), then we put \(P\) in \textit{bag}_2. For convenience, we use \textit{split}_y(\textit{bag}, i, j) to denote \textit{bag}_1 and use \textit{split}_n(\textit{bag}, i, j) to denote \textit{bag}_2. (this corresponds to Y/N branches in a strategy tree).
First understand the following function.
   //in below, returning yes means “no strategy”;
   //in below, returning no means ”have strategy”.
   Boolean NoStrategy(bag, k)
   {
      return no when bag contains at most one element.
      return yes when k = 0.
      You need check for all 1 ≤ i ≠ j ≤ n, this is true: NoStrategy(split_Y(bag, i, j), k – 1) or NoStrategy(split_N(bag, i, j), k – 1).
      You return yes (i.e., no strategy) when the checking is ok.
      Otherwise, you return no.
   }
Then, understand below:
   Boolean NoStrategy(n, k){
      bag = the set of all permutations of 1234⋯n;
      return NoStrategy(bag, k);
   }
Finally, understand our algorithm,
   int M(n){
      k = log n!;
      while NoStrategy(n, k)
         k ++;
      return k;
   }
Your job:
   1. to see if my thoughts are right;
   2. how to select data structures for bag (you can not explicitly store all permutations in bag since the set may be too large).