Most Significant Digit (MSD)

- **Primary Idea:**
  Instead of comparing entire strings, the algorithm sorts them one character (or digit) at a time. It begins with the most significant digit and works its way to the least significant, partitioning the array into groups that share the same character at the current position.
- **Applicability:**
  The method is particularly efficient when many strings share common prefixes because it only needs to compare the differing parts of the strings.

Step-by-Step Process

1. **Initialization:**
    - **Auxiliary Storage:**
      An additional array is used to temporarily store strings during the distribution phase.
    - **Cutoff for Small Arrays:**
      For very small segments (subarrays), a simpler sorting method (like insertion sort) may be used instead of recursive distribution, to reduce overhead.
2. **Frequency Counting:**
    - **Examine Current Character:**
      For each string in the current segment, look at the character at the current position (say, position *d*). If the string is shorter than *d*, a special marker (like a sentinel value) is used to indicate that no character exists.
    - **Count Occurrences:**
      Build a frequency count of each possible character (or digit) that appears at the current position.
3. **Index Computation:**
    - **Transform Counts:**
      Convert the frequency counts into cumulative counts, which indicate the starting index for each character group in the auxiliary array.
    - **Bucket Boundaries:**
      These indices serve as boundaries or “buckets” where strings with the same current character will be placed.
4. **Distribution:**
    - **Place Strings into Buckets:**

Iterate through the current segment of strings, and place each string into its appropriate bucket in the auxiliary array based on its current character.
    - **Copy Back:**
      Once all strings are placed in the auxiliary array, copy them back into the original array in the order defined by the buckets.

5. **Recursive Sorting:**

    - **Process Each Bucket:**
      For each bucket (each group of strings with the same current character), recursively apply the same sorting process, but move to the next character position (*d + 1*).
        - **Note:** If a bucket contains only strings that have reached their end (i.e., no more characters), that group is already sorted.
    - **End Condition:**
      The recursion stops when either the subarray is small enough (thus using the simpler sort) or when there are no more characters to compare.

Key Points to Note

- **Language Agnostic:**
  The steps outlined here focus on the conceptual process. They do not rely on specific programming constructs or language-specific features, making the algorithm applicable in any programming environment.

- **Efficiency with Common Prefixes:**
  Because the algorithm deals with characters position by position, it is particularly well-suited for sorting strings where many items share the same initial segments. Only the differing parts are examined deeply.

- **Handling Varied Lengths:**
  When a string has fewer characters than the current position being examined, it is treated in a special way (usually by considering it to be “smaller” than any string that has a character at that position), ensuring correct ordering.

- **Divide and Conquer Nature:**
  By partitioning the array into buckets based on the current character and then recursively sorting these buckets, the MSD sort efficiently reduces the overall problem size with each recursive call.