2053. Kth Distinct String in an Array

A **distinct string** is a string that is present only **once** in an array.

Given an array of strings `arr`, and an integer `k`, return *the* `k`th ***distinct string*** *present in* `arr`. If there are **fewer** than `k` distinct strings, return *an* ***empty string*** `""`.

Note that the strings are considered in the **order in which they appear** in the array.

Example 1:

```
Input: arr = ["d","b","c","b","c","a"], k = 2
Output: "a"
Explanation:
The only distinct strings in arr are "d" and "a".
"d" appears 1st, so it is the 1st distinct string.
"a" appears 2nd, so it is the 2nd distinct string.
Since k == 2, "a" is returned.
```

Example 2:

```
Input: arr = ["aaa","aa","a"], k = 1
Output: "aaa"
Explanation:
All strings in arr are distinct, so the 1st string "aaa" is returned.
```

Example 3:

```
Input: arr = ["a","b","a"], k = 3
Output: ""
Explanation:
The only distinct string is "b". Since there are fewer than 3 distinct strings, we
return an empty string "".
```

Constraints:

- `1 <= k <= arr.length <= 1000`
- `1 <= arr[i].length <= 5`
- `arr[i]` consists of lowercase English letters.

2053. Kth Distinct String in an Array

```
/*
    brute force
    Time complexity: O(n^2)
    Space complexity: O(1)
*/
class Solution {
public:
    std::string kthDistinct(std::vector<std::string>& arr, int k) {
        int n=arr.size();
        for(int i=0;i<n;++i){
         bool ok=true;
         int j=0;
         while(j<n&&ok){
             if(i!=j&&arr[i]==arr[j]) ok=false;
             j++;
         }
         if(ok) k--;
         if(k==0) return arr[i];
        }
        return "";
    }
};
```

2053. Kth Distinct String in an Array

```
/*
   Countig+Map
   Time complexity: O(nlogn)
   Space complexity: O(n)
*/
class Solution {
public:
   std::string kthDistinct(std::vector<std::string>& arr, int k) {
      std::map<std::string,int> freq;
      for(auto& s: arr) freq[s]++;
      for(auto& s: arr){
         if(freq[s]==1) k--;
         if(k==0) return s;
      }
      return "";
   }
};
```

2053. Kth Distinct String in an Array

```
/*
   sorting+Binary search
   Time complexity: O(nlogn)
   Space complexity: O(1)
*/
class Solution {
public:
   static string kthDistinct(vector<string>& arr, int k) {
      const int n=arr.size();
      vector<string> sorted(arr.begin(), arr.end());
      sort(sorted.begin(), sorted.end());

      for(string& s: arr){
         int i=lower_bound(sorted.begin(), sorted.end(), s)-sorted.begin();
         if (i==n-1 ||sorted[i]!=sorted[i+1]) k--;
         if (k==0) return s;
      }
      return "";
   }
};
```

2053. Kth Distinct String in an Array

```
/*
   Trie
   Time complexity: O(nlogn)
   Space complexity: O(n+logn)
*/

// struct Trie for N alphabets
const int N=26;
struct Trie {
   Trie* next[N];
   int count=0;

   Trie() {
   //    cout<<"Create a trie object!\n";
      memset(next, 0, sizeof(next));
   }

   void insert(string& s) {
      Trie* Node=this;
      for(char c: s){
         int i=c-'a';
         if (Node->next[i]==NULL) Node->next[i]=new Trie();
         Node=Node->next[i];
      }
      Node->count++;
   }

   bool findcnt1(string& s) {
      Trie* Node=this;
      for(char c: s){
         int i=c-'a';
         if (Node->next[i]==NULL) return 0;
         Node=Node->next[i];
      }
      return Node->count==1;
   }
};
```

```
class Solution {
public:
    static string kthDistinct(vector<string>& arr, int k) {
        Trie trie;
        for(string& s: arr)
            trie.insert(s);

        for(string& s: arr){
            if (trie.findcnt1(s)==1) k--;
            if (k==0)  return s;
        }
        return "";
    }
};
```