Longest Common Prefix

Problem Statement: Given a set of strings, find the longest common prefix.
Examples:
Input: *{“geeksforgeeks”, “geeks”, “geek”, “geezer”}*
Output: *“gee”*
Input: *{“apple”, “ape”, “april”}*
Output: *“ap”*

Longest Common Prefix using Word by Word Matching

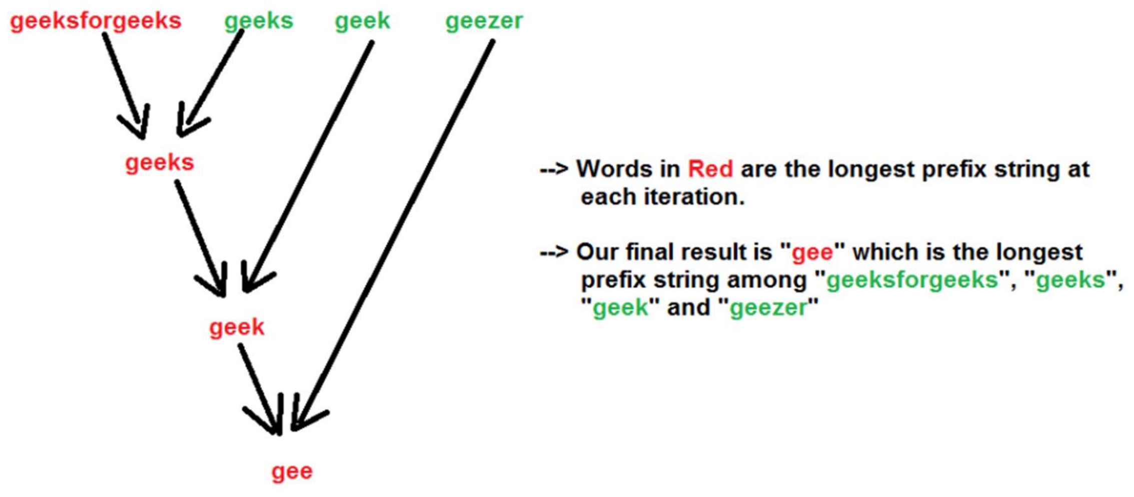
We start with an example. Suppose there are two strings- “geeksforgeeks” and “geeks”. What is the longest common prefix in both of them? It is “geeks”. Now let us introduce another word “geek”. So now what is the longest common prefix in these three words ? It is “geek”
We can see that the longest common prefix holds the associative property, i.e-

```
LCP(string1, string2, string3)
         = LCP (LCP (string1, string2), string3)


Like here


LCP (“geeksforgeeks”, “geeks”, “geek”)
     =  LCP (LCP (“geeksforgeeks”, “geeks”), “geek”)
     =  LCP (“geeks”, “geek”) = “geek”
```

So we can make use of the above associative property to find the LCP of the given strings. We one by one calculate the LCP of each of the given string with the LCP so far. The final result will be our longest common prefix of all the strings.
Note that it is possible that the given strings have no common prefix. This happens when the first character of all the strings are not same.
We show the algorithm with the input strings- “geeksforgeeks”, “geeks”, “geek”, “geezer” by the below figure.


geeksforgeeks
geeks
geek
geezer
geeks
geek
gee
--> Words in Red are the longest prefix string at each iteration.
--> Our final result is "gee" which is the longest prefix string among "geeksforgeeks", "geeks", "geek" and "geezer"

Below is the implementation of above approach:

C++

```
//  A C++ Program to find the longest common prefix
#include<bits/stdc++.h>
using namespace std;

// A Utility Function to find the common prefix between
// strings- str1 and str2
string commonPrefixUtil(string& str1, string& str2)
{
    string result = "";
    int len = min(str1.length(), str2.length());

    // Compare str1 and str2
    for (int i = 0; i < len; i++)
    {
        if (str1[i] != str2[i])
```

```
            break;
        result += str1[i];
    }

    return (result);
}

// A Function that returns the longest common prefix
// from the array of strings
string commonPrefix (string arr[], int n)
{
    string prefix =  arr[0];

    for (int i=1; i < n; i++)
        prefix = commonPrefixUtil(prefix, arr[i]);

    return (prefix);
}

// Driver program to test above function
int main()
{
    string arr[] = {"geeksforgeeks", "geeks",
                    "geek", "geezer"};
    int n = sizeof(arr) / sizeof(arr[0]);

    string ans = commonPrefix(arr, n);
    if (ans.length())
        printf ("The longest common prefix is - %s",
                 ans.c_str());
    else
        printf("There is no common prefix");

    return (0);
}
```

Output

```
The longest common prefix is - gee
```

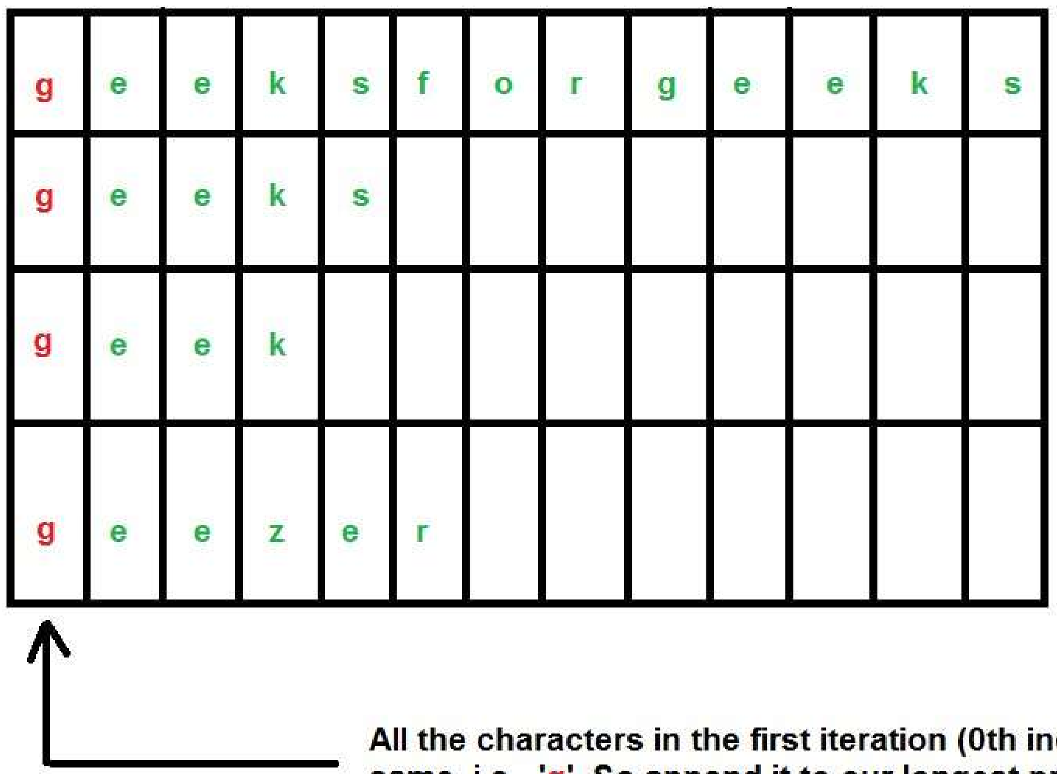
Time Complexity : Since we are iterating through all the strings and for each string we are iterating though each characters, so we can say that the time complexity is O(N M) where,

```
N = Number of strings
M = Length of the largest string
```

Auxiliary Space : To store the longest prefix string we are allocating space which is O(M).

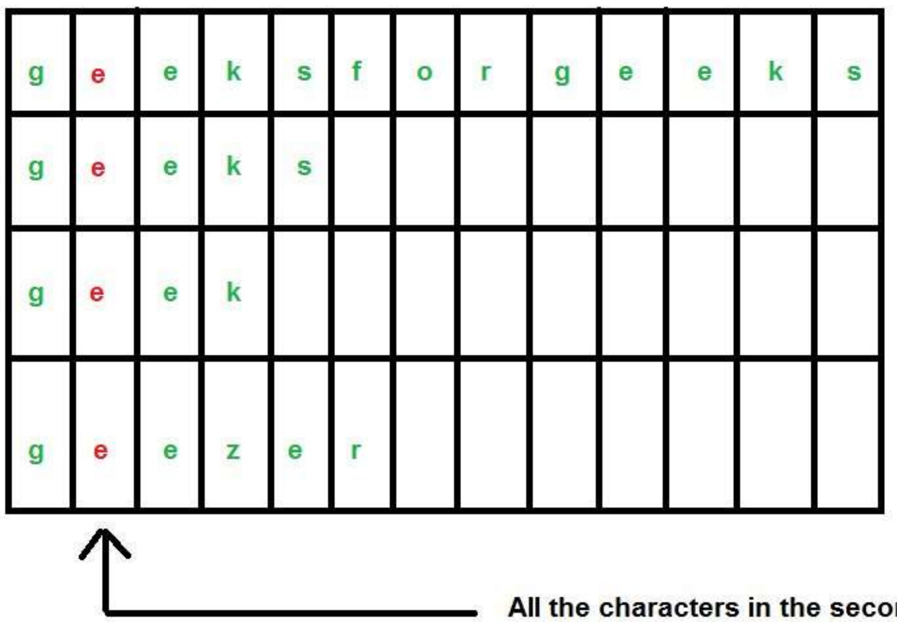
Longest Common Prefix using Character by Character Matching

We have discussed word by word matching algorithm in previous post.
In this algorithm, instead of going through the strings one by one, we will go through the characters one by one.
We consider our strings as – “geeksforgeeks”, “geeks”, “geek”, “geezer”.



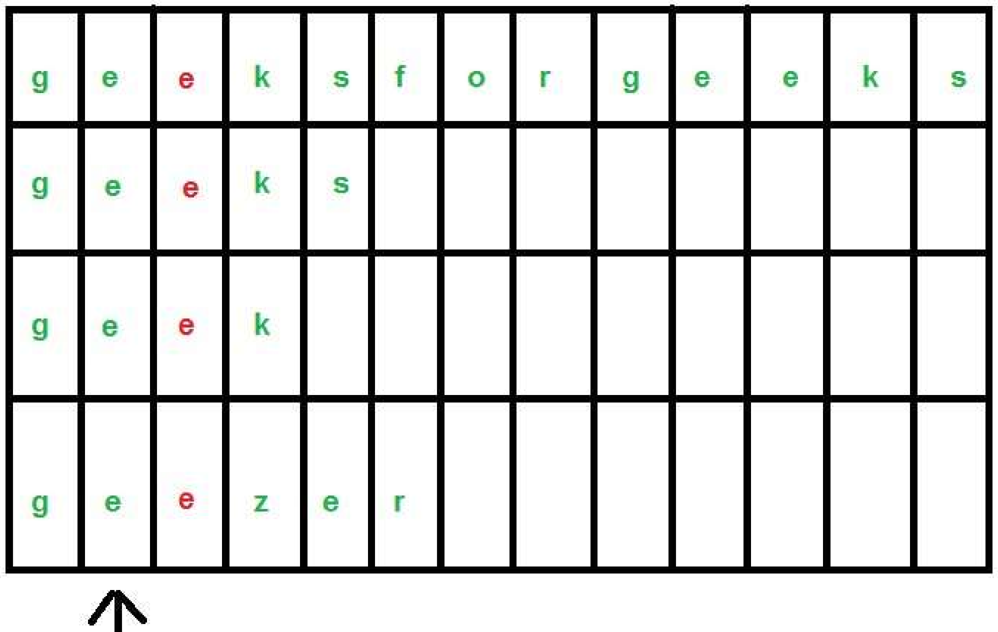
g	e	e	k	s	f	o	r	g	e	e	k	s
g	e	e	k	s								
g	e	e	k									
g	e	e	z	e	r							

All the characters in the first iteration (0th index) are same, i.e - 'g'. So append it to our longest prefix string

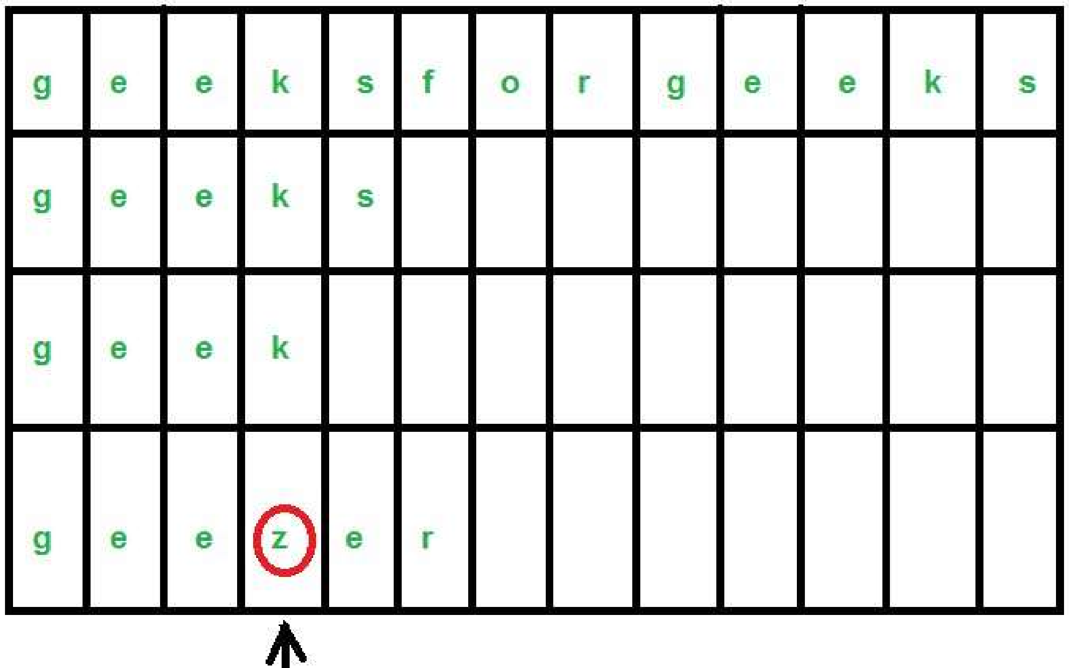


g	e	e	k	s	f	o	r	g	e	e	k	s
g	e	e	k	s								
g	e	e	k									
g	e	e	z	e	r							

All the characters in the second iteration (1st index) are same, i.e- 'e'. So append it to our longest prefix string



All the characters in the third iteration (2nd index) are same, i.e- 'e'. So append it to our longest prefix string



In the fourth iteration (3rd index) all the characters are not same, so we stop our iteration here and our longest prefix string is "gee"

Below is the implementation of this approach.

- C++
- Java
- Python 3
- C#
- Javascript

```
//  A C++ Program to find the longest common prefix
#include<bits/stdc++.h>
using namespace std;

// A Function to find the string having the minimum
// length and returns that length
int findMinLength(string arr[], int n)
{
    int min = arr[0].length();

    for (int i=1; i<n; i++)
        if (arr[i].length() < min)
            min = arr[i].length();

    return(min);
}

// A Function that returns the longest common prefix
// from the array of strings
string commonPrefix(string arr[], int n)
{
    int minlen = findMinLength(arr, n);

    string result; // Our resultant string
    char current;  // The current character

    for (int i=0; i<minlen; i++)
    {
        // Current character (must be same
        // in all strings to be a part of
        // result)
        current = arr[0][i];
        for (int j=1 ; j<n; j++)
            if (arr[j][i] != current)
                return result;

        // Append to result
        result.push_back(current);
    }

    return (result);
}

// Driver program to test above function
int main()
{
    string arr[] = {"geeksforgeeks", "geeks",
                    "geek", "geezer"};
    int n = sizeof (arr) / sizeof (arr[0]);

    string ans = commonPrefix (arr, n);

    if (ans.length())
        cout << "The longest common prefix is "
             << ans;
    else
        cout << "There is no common prefix";
    return (0);
}
```

Time Complexity : Since we are iterating through all the characters of all the strings, so we can say that the time complexity is O(N M) where,

```
N = Number of strings
```

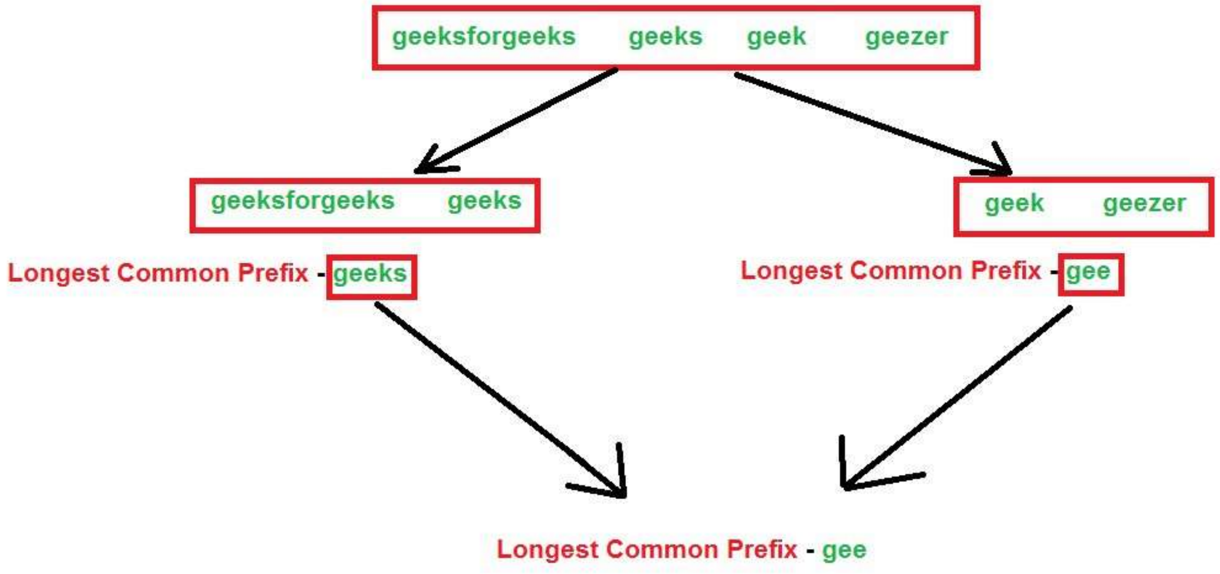
```
M = Length of the Smallest string
```

Auxiliary Space : To store the longest prefix string we are allocating space which is O(M).

Longest Common Prefix using Divide and Conquer Algorithm

We have discussed word by word matching and character by character matching algorithms.

In this algorithm, a divide and conquer approach is discussed. We first divide the arrays of string into two parts. Then we do the same for left part and after that for the right part. We will do it until and unless all the strings become of length 1. Now after that, we will start conquering by returning the common prefix of the left and the right strings.
The algorithm will be clear using the below illustration. We consider our strings as – “geeksforgeeks”, “geeks”, “geek”, “geezer”.


geeksforgeeks geeks geek geezer
geeksforgeeks geeks
geek geezer
Longest Common Prefix - geeks
Longest Common Prefix - gee
Longest Common Prefix - gee

Below is the implementation.

C++

```
//  A C++ Program to find the longest common prefix
#include<bits/stdc++.h>
using namespace std;

// A Utility Function to find the common prefix between
// strings- str1 and str2
string commonPrefixUtil(string str1, string str2)
{
    string result;
    int n1 = str1.length(), n2 = str2.length();
    for (int i=0, j=0; i<=n1-1&&j<=n2-1; i++,j++)
    {
        if (str1[i] != str2[j])
            break;
        result.push_back(str1[i]);
    }
    return (result);
}

// A Divide and Conquer based function to find the
// longest common prefix. This is similar to the
// merge sort technique
string commonPrefix(string arr[], int low, int high)
{
    if (low == high)
        return (arr[low]);

    if (high > low)
    {
        // Same as (low + high)/2, but avoids overflow for
        // large low and high
        int mid = low + (high - low) / 2;

        string str1 = commonPrefix(arr, low, mid);
        string str2 = commonPrefix(arr, mid+1, high);
        return (commonPrefixUtil(str1, str2));
    }
}

// Driver program to test above function
int main()
{
    string arr[] = {"geeksforgeeks", "geeks",
                    "geek", "geezer"};
    int n = sizeof (arr) / sizeof (arr[0]);

    string ans = commonPrefix(arr, 0, n-1);

    if (ans.length())
        cout << "The longest common prefix is "
             << ans;
    else
        cout << "There is no common prefix";
    return (0);
}
```

Output

```
The longest common prefix is gee
```

Time Complexity: Since we are iterating through all the characters of all the strings, so we can say that the time complexity is O(N M) where,

```
N = Number of strings
```

```
M = Length of the largest string string
```

Auxiliary Space: To store the longest prefix string we are allocating space which is O(M Log N).

Longest Common Prefix using Binary Search

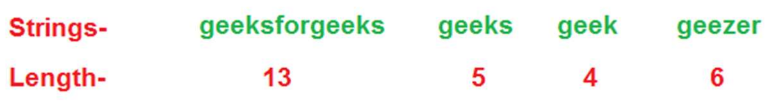
Longest Common Prefix using Binary Search

The idea is based on following observations:

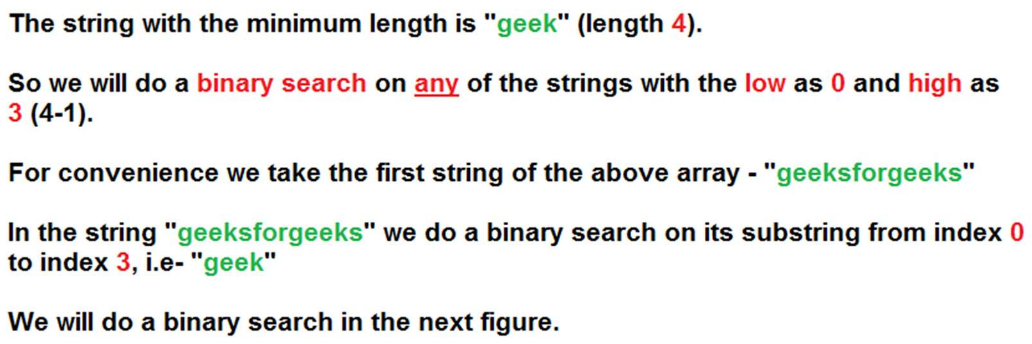
1. The length of common prefix cannot be greater than the smallest string in given set of strings
2. Instead of find each character from start if present in all strings, we can take a possible prefix (most probably the smallest string) and check for common prefix by dividing in half.

Illustration:

- *Consider strings as {"geeksforgeeks", "geeks", "geek", "geezer"}*
- *Smallest string given = "geek", so the Longest common prefix possible is "geek"*



Strings-	geeksforgeeks	geeks	geek	geezer
Length-	13	5	4	6



The string with the minimum length is "geek" (length 4).

So we will do a binary search on any of the strings with the low as 0 and high as 3 (4-1).

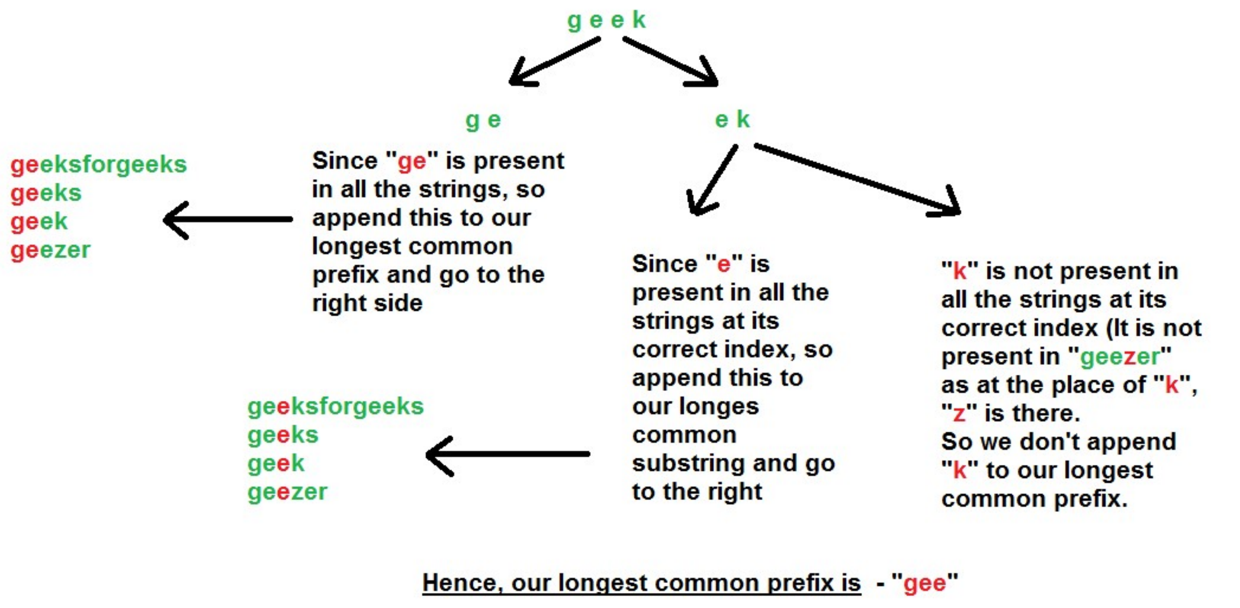
For convenience we take the first string of the above array - "geeksforgeeks"

In the string "geeksforgeeks" we do a binary search on its substring from index 0 to index 3, i.e- "geek"

We will do a binary search in the next figure.

- *Now we start breaking this hypothesis using binary search:*
  - *Break "geek" by finding mid.*
    - *first half = "ge", which is present is all given strings. So add "ge" to the actual longest common substring.*

            - *second half = “ek”, which is not present in last string “geezer”. Hence we need to repeat the process for this string.*
        - *Break “ek” by finding mid.*
            - *first half = “e”, which is present is all given strings. So append “e” to the actual longest common substring.*
            - *second half = “k”, which is not present in last string “geezer”. Hence we need to repeat the process for this string.*
        - *Break “k” by finding mid.*
            - *mid = “k”, which is not present in last string “geezer”. Hence we discard this.*
        - *Now no more string is present to match.*
- *Therefore, Longest Common Prefix using Binary Search = “gee”*


g e e k
g e
e k
geeksforgeeks
geeks
geek
geezer
Since "ge" is present in all the strings, so append this to our longest common prefix and go to the right side
Since "e" is present in all the strings at its correct index, so append this to our longes common substring and go to the right
"k" is not present in all the strings at its correct index (It is not present in "geezer" as at the place of "k", "z" is there. So we don't append "k" to our longest common prefix.
geeksforgeeks
geeks
geek
geezer
Hence, our longest common prefix is - "gee"

Below is the implementation of above approach.

- C++
- Java
- Python3
- C#
- Javascript

```
//  A C++ Program to find the longest common prefix
#include <bits/stdc++.h>
using namespace std;

// A Function to find the string having the minimum
// length and returns that length
int findMinLength(string arr[], int n)
{
    int min = INT_MAX;

    for (int i = 0; i <= n - 1; i++)
        if (arr[i].length() < min)
            min = arr[i].length();
    return (min);
}

bool allContainsPrefix(string arr[], int n, string str,
                       int start, int end)
{
    for (int i = 0; i <= n - 1; i++)
        for (int j = start; j <= end; j++)
            if (arr[i][j] != str[j])
                return (false);
    return (true);
}

// A Function that returns the longest common prefix
// from the array of strings
string commonPrefix(string arr[], int n)
{
    int index = findMinLength(arr, n);
    string prefix; // Our resultant string

    // We will do an in-place binary search on the
    // first string of the array in the range 0 to
    // index
    int low = 0, high = index;

    while (low <= high) {
        // Same as (low + high)/2, but avoids overflow
        // for large low and high
        int mid = low + (high - low) / 2;

        if (allContainsPrefix(arr, n, arr[0], low, mid)) {
            // If all the strings in the input array
            // contains this prefix then append this
            // substring to our answer
            prefix = prefix
                     + arr[0].substr(low, mid - low + 1);

            // And then go for the right part
            low = mid + 1;
        }
        else // Go for the left part
            high = mid - 1;
    }

    return (prefix);
}

// Driver program to test above function
int main()
{
    string arr[]
        = { "geeksforgeeks", "geeks", "geek", "geezer" };
    int n = sizeof(arr) / sizeof(arr[0]);

    string ans = commonPrefix(arr, n);

    if (ans.length())
        cout << "The longest common prefix is " << ans;
    else
        cout << "There is no common prefix";
    return (0);
}
```

Learn Data Structures & Algorithms with GeeksforGeeks

Output

```
The longest common prefix is gee
```

Time Complexity : O(NM log M)

The recurrence relation is T(M) = T(M/2) + O(MN), where

- *N = Number of strings*

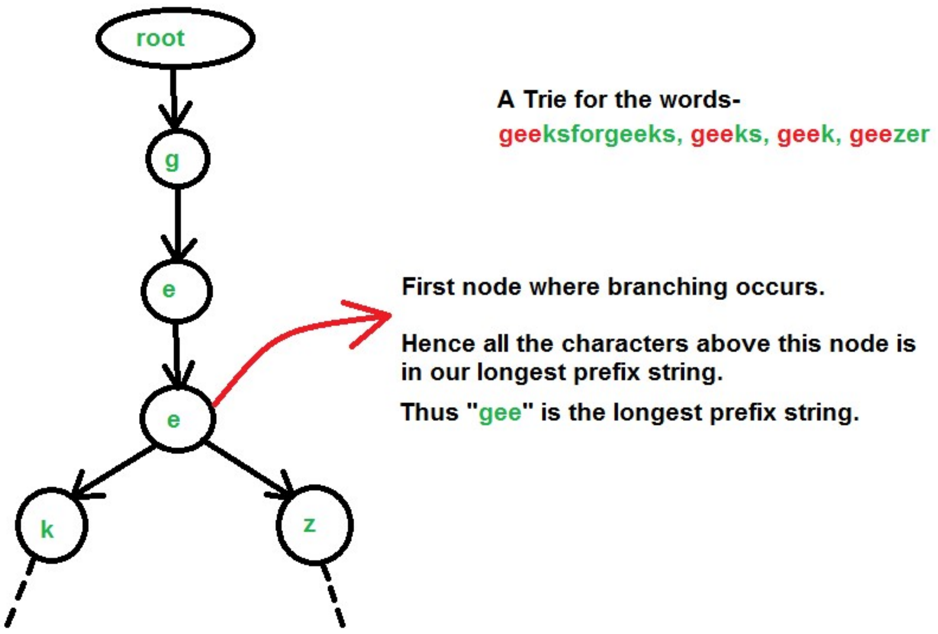
- *M = Length of the largest string*

So we can say that the time complexity is O(NM log M)

Auxiliary Space: To store the longest prefix string we are allocating space which is **O(N)** where, N = length of the largest string among all the strings

----

Algorithm Illustration considering strings as – “geeksforgeeks”, “geeks”, “geek”, “geezer”


root
g
e
e
k
z
A Trie for the words-
geeksforgeeks, geeks, geek, geezer
First node where branching occurs.
Hence all the characters above this node is in our longest prefix string.
Thus "gee" is the longest prefix string.

C++

```
// A Program to find the longest common

// prefix of the given words


#include<bits/stdc++.h>

using namespace std;


// Alphabet size (# of symbols)

#define ALPHABET_SIZE (26)


// Converts key current character into index
// use only 'a' through 'z' and lower case
#define CHAR_TO_INDEX(c) ((int)c - (int)'a')

// Trie node
struct TrieNode
{
    struct TrieNode *children[ALPHABET_SIZE];

    // isLeaf is true if the node represents
    // end of a word
    bool isLeaf;
};

// Returns new trie node (initialized to NULLs)
struct TrieNode *getNode(void)
{
    struct TrieNode *pNode = new TrieNode;

    if (pNode)
    {
        int i;

        pNode->isLeaf = false;

        for (i = 0; i < ALPHABET_SIZE; i++)
            pNode->children[i] = NULL;
    }
    return pNode;
}

// If not present, inserts the key into the trie
// If the key is a prefix of trie node, just marks leaf node
void insert(struct TrieNode *root, string key)
{
    int length = key.length();
    int index;

    struct TrieNode *pCrawl = root;

    for (int level = 0; level < length; level++)
    {
        index = CHAR_TO_INDEX(key[level]);
        if (!pCrawl->children[index])
            pCrawl->children[index] = getNode();

        pCrawl = pCrawl->children[index];
    }

    // mark last node as leaf
    pCrawl->isLeaf = true;
}

// Counts and returns the number of children of the
// current node
int countChildren(struct TrieNode *node, int *index)
{
    int count = 0;
    for (int i=0; i<ALPHABET_SIZE; i++)
    {
        if (node->children[i] != NULL)
        {
            count++;
            *index = i;
        }
    }
    return (count);
}

// Perform a walk on the trie and return the
// longest common prefix string
string walkTrie(struct TrieNode *root)
{
    struct TrieNode *pCrawl = root;
    int index;
    string prefix;

    while (countChildren(pCrawl, &index) == 1 &&
            pCrawl->isLeaf == false)
    {
        pCrawl = pCrawl->children[index];
        prefix.push_back('a'+index);
    }
    return (prefix);
}

// A Function to construct trie
void constructTrie(string arr[], int n, struct TrieNode *root)
{
    for (int i = 0; i < n; i++)
        insert (root, arr[i]);
    return;
}

// A Function that returns the longest common prefix
// from the array of strings
string commonPrefix(string arr[], int n)
{
    struct TrieNode *root = getNode();
    constructTrie(arr, n, root);

    // Perform a walk on the trie
    return walkTrie(root);
}

// Driver program to test above function
int main()
{
    string arr[] = {"geeksforgeeks", "geeks",
                    "geek", "geezer"};
    int n = sizeof (arr) / sizeof (arr[0]);

    string ans = commonPrefix(arr, n);

    if (ans.length())
        cout << "The longest common prefix is "
             << ans;
    else
        cout << "There is no common prefix";
    return (0);
}
```

Output :

```
The longest common prefix is gee
```

Time Complexity : Inserting all the words in the trie takes O(MN) time and performing a walk on the trie takes O(M) time, where-

```
N = Number of strings
M = Length of the largest string
```

Auxiliary Space: To store all the strings we need to allocate O(26*M*N) ~ O(MN) space for the Trie.