



Solution Review: List Sort Using Trie

This review provides a detailed analysis of the solution to the Array Sort Using Tries Challenge.

We'll cover the following

- Solution: Pre-Order Traversal
- Time Complexity

Solution: Pre-Order Traversal

main.py

Trie.py

TrieNode.py

```
1 from Trie import Trie
2 from TrieNode import TrieNode
3 # Recursive Function to generate all words in alphabetic order
4
5
6 def get_words(root, result, level, word):
7     # Leaf denotes end of a word
8     if (root.is_end_word):
9         # current word is stored till the 'level' in the character array
10        temp = ""
11        for x in range(level):
12            temp += word[x]
13        result.append(temp)
14
15    for i in range(26):
16        if (root.children[i] is not None):
```

```

17         # Non-null child, so add that index to the list
18         word[level] = chr(i + ord('a'))
19         get_words(root.children[i], result, level + 1, word)
20
21
22 def sort_list(arr):
23     result = []
24
25     # Creating Trie and Inserting words from array
26     trie = Trie()
27     for word in arr:
28         trie.insert(word)

```

This exercise is very similar to **Challenge 2**, except the fact that you have to create the trie yourself.

Since the `children` list for each node stores letters in alphabetical order, the tree itself is ordered from top to bottom. All we need to do is make a pre-order traversal (think of `a` as the left most child and `z` as the right most child) and store the words in a list just like we did in the previous challenge.

Time Complexity

We first insert the nodes into the graph and then traverse all the existing nodes. Hence, the bottleneck worst case time complexity is $O(n)$.

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#) ⓘ




Challenge 3: List Sort Using Trie

Challenge 4: Word Search



 Completed

 Report an Issue

