# Trie Time Complexity

- Let N be the number of keys in the trie
- Let $L_{key}$ be the length of a key

Let's analyze contains…

Independent of N!

Constant time!

```
contains(T,key):
O(1)    if T is null:
            return false (for Set) or null (for Map)

O(1)    if key is empty string:
            return T.terminates (for Set) or T.value (for Map)

        return contains(T.children.get(key[0]),rest(key))
```

$O(L_{key})$ or $O(1)$ $O(L_{key})$ or $O(1)$

With proper optimization, each instance is O(1). How many times does it run in the worst case? $O(L_{key})$.

# Trie Time Complexity, Summary

With a max key length, worst case time complexities:

| | Contains | Insert | Delete |
|---|---|---|---|
| BST | O(N) | O(N) | O(N) |
| AVL | O(logN) | O(logN) | O(logN) |
| Trie | O(1) | O(1) | O(1) |

# Time Complexity, a Fraud Revealed?

So why *ever* use an AVL tree over a trie?

- If B is the size of the alphabet, and L is the max length of a key, there are $B^L$ unique keys.
  - A maximum size before the tree is full!
- Asymptotic runtime complexity studies behavior as $N \to \infty$.
- For an arbitrary N, we need $L=O(\log_B N)$
  - Now O(logN), not O(1) recursive calls
- Aside: a similar result is true for Radix Sort:
  - Runtime is O(NL)
  - Don't want each key to appear on average ∞ times?
    then it's really O(NlogN)

*Now that you know the truth…*

**We'll pretend we don't.**

Trie operations? O(1)

Radix sort? O(N)

Tries and radix sort can be good options when the keys are convenient and naturally bounded.

They are not magic bullets.

Asymptotic analysis doesn't tell the full story.