# Recursive vs Iterative: Letter Case Permutation

## 1. Time Complexity

Both recursive and iterative solutions generate all possible combinations.  
  
\* Each letter doubles the possibilities (lowercase/uppercase).  
\* If string length = n, with k letters, number of results = 2^k.  
  
So:  
  
T(n) = O(2^k · n)  
  
(where n is for copying/appending characters into strings).  
  
👉 Recursive = Iterative in time complexity.

## 2. Space Complexity

Recursive version:

\* Uses call stack frames for recursion depth = O(n).  
\* Output storage = O(2^k · n).  
\* Total: O(2^k·n + n).

Iterative version:

\* Uses explicit stack or queue.  
\* Stack/queue size at most O(2^k).  
\* Output storage same = O(2^k · n).  
\* Total: O(2^k·n + 2^k).

👉 In practice:  
  
\* Recursive → smaller code, but risks stack overflow if input string is very long.

**Performance in Python**

* Recursive has **function call overhead**.
* Iterative avoids recursion limit (default ~1000).

\* Iterative → avoids recursion limit, slightly more predictable memory usage.

## 3. Performance in Python

\* Python function calls (recursion) are slower than while loops because of function call overhead.  
\* For small inputs (say length ≤ 15), recursive and iterative run almost the same.  
\* For larger inputs (say length > 20), iterative usually outperforms recursion in Python.

## 4. Ease of Understanding

\* Recursive version is easier to explain (backtracking naturally maps to recursion).  
\* Iterative version is more robust for production systems.

## ✅ Bottom line:

\* Both have same theoretical efficiency.  
\* Recursive: more elegant, better for teaching.  
\* Iterative: faster in Python for big inputs, safer against recursion depth errors.