Operations Count (N)	`string` ($O(N^2)$)	`StringBuilder` ($O(N)$)
1,000	10 ms	1 ms
10,000	1 s	10 ms
1,000,000	30 min (Unusable)	50 ms

Approaches Compared

1. string (Immutable)

Approach:

Each concatenation creates a new string object, copying all existing characters.

Time Complexity:

Overall: $O(N^2)$ for N concatenations

Thread Safety: Yes (immutable)

Performance: Very poor for large operations

2. StringBuilder (Mutable)

Approach:

Uses a resizable buffer to append characters without creating new objects.

Time Complexity:

Overall: $O(N)$

Thread Safety: No (not synchronized)

Performance: Fastest for single-threaded scenarios

3. StringBuffer (Mutable & Thread-Safe)

Approach:

Similar to StringBuilder but synchronized for multi-threading.

Time Complexity:

Overall: $O(N)$

Thread Safety: Yes

Performance: Slightly slower than StringBuilder due to locking

Key Observations

string

Each concatenation copies the entire previous content.

Time increases quadratically.

Becomes unusable for large concatenations.

StringBuilder

Appends in constant amortized time.

Scales linearly even for millions of operations.

Ideal for loops and large string construction.

StringBuffer

Safe for multi-threaded use.

Slightly slower than StringBuilder due to synchronization overhead.