Efficient String Building — Coding Interview Notes (Light Theme)

General Pattern Template

```
# arr is a list of characters
def fn(arr):
    ans = []
    for c in arr:
        ans.append(c)

    return "".join(ans)
```

Concept:
The **Efficient String Building** pattern optimizes string concatenation in loops. In Python, strings are immutable, so repeated concatenation using += creates many intermediate copies. Instead, accumulate characters or substrings in a **list** and join them once at the end.

Time Complexity: O(n), where n is the number of characters.
Why efficient: Joining a list of strings runs in linear time, avoiding repeated reallocations.

Key Ideas

1. Avoid repeated string concatenation inside loops.
2. Use a list to collect pieces (characters, substrings).
3. Apply `''.join(list)` once at the end for optimal performance.
4. This pattern is vital when processing or constructing large strings.

Example 1: Building a String from Characters

Goal: Convert a list of characters into a single string efficiently.
Approach: Append characters to a list and use ''.join() to combine them.

```
def build_string_from_chars(chars):
    result = []
    for ch in chars:
        result.append(ch)
    return ''.join(result)

# Example
print(build_string_from_chars(['h','e','l','l','o']))  # Output: 'hello'
```

Example 2: Efficiently Building from Substrings

Goal: Construct a large string from multiple smaller substrings (e.g., lines of text).
Approach: Collect substrings in a list and join them once.

```
def build_from_substrings(lines):
    result = []
    for line in lines:
        result.append(line + "\n")
    return ''.join(result)

# Example
lines = ["line 1", "line 2", "line 3"]
print(build_from_substrings(lines))
# Output:
# line 1
# line 2
# line 3
```

Example 3: String Building with Conditions

Goal: Build a formatted string conditionally (e.g., join selected items).
Approach: Append only the valid parts and join at the end.

```
def conditional_join(words):
    result = []
    for w in words:
        if len(w) > 3:
            result.append(w.upper())
    return ', '.join(result)

# Example
print(conditional_join(['a', 'code', 'chat', 'ai']))  # Output: 'CODE, CHAT'
```

Summary Table

ConceptUse CaseComplexity List-based buildingCharacter or substring accumulationO(n) Naive string concatenationUsing += in loopO(n^2) Conditional buildingFilter & format selected itemsO(n)