MTech CSE – 1st Semester
Student ID: **A125023**
Student Name: **SRIJITA VERMA**

IIT Bhubaneswar
Imagine, Innovate, Inspire
(A University established by Government of Odisha)

Question

Solve the following recurrence relation arising from the LUP decomposition solve procedure:

$$T(n) = \sum_{i=1}^{n} \left[O(1) + \sum_{j=1}^{i-1} O(1) \right] + \sum_{i=1}^{n} \left[O(1) + \sum_{j=i+1}^{n} O(1) \right].$$

Answer

Step 1: Interpret the Recurrence

The recurrence consists of two main summations, each iterating over $i = 1$ to n.

Each part models the cost of:

- constant-time operations, and
- inner loops that perform a constant-time operation multiple times.

Our goal is to count the total number of constant-time operations and determine the overall asymptotic complexity.

Step 2: Simplify the First Summation

Consider the first term:

$$\sum_{i=1}^{n} \left[O(1) + \sum_{j=1}^{i-1} O(1) \right].$$

Inner Summation

The inner sum

$$\sum_{j=1}^{i-1} O(1)$$

executes $i - 1$ times. Hence:

$$\sum_{j=1}^{i-1} O(1) = O(i).$$

First Summation Becomes

$$\sum_{i=1}^{n} [O(1) + O(i)] = \sum_{i=1}^{n} O(i).$$

Step 3: Evaluate the First Summation

We know that:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Therefore:

$$\sum_{i=1}^{n} O(i) = O(n^2).$$

Step 4: Simplify the Second Summation

Now consider the second term:

$$\sum_{i=1}^{n} \left[O(1) + \sum_{j=i+1}^{n} O(1) \right].$$

Inner Summation

The inner sum

$$\sum_{j=i+1}^{n} O(1)$$

executes $n - i$ times. Hence:

$$\sum_{j=i+1}^{n} O(1) = O(n-i).$$

Second Summation Becomes

$$\sum_{i=1}^{n} [O(1) + O(n-i)] = \sum_{i=1}^{n} O(n-i).$$

Step 5: Evaluate the Second Summation

As i ranges from 1 to n, the quantity $n - i$ ranges from $n - 1$ down to 0. Thus:

$$\sum_{i=1}^{n} (n-i) = \sum_{k=0}^{n-1} k = \frac{n(n-1)}{2}.$$

Therefore:

$$\sum_{i=1}^{n} O(n-i) = O(n^2).$$

Step 6: Combine Both Parts

From the above steps:

- The first summation contributes $O(n^2)$,
- The second summation contributes $O(n^2)$.

Hence:

$$T(n) = O(n^2) + O(n^2) = O(n^2).$$

Final Answer

$$\boxed{T(n) = O(n^2)}$$

Comparison with LUP Decomposition Cost. While the LUP decomposition itself requires $O(n^3)$ time, the solve phase analyzed here runs in $O(n^2)$ time, making it efficient when solving linear systems with multiple right-hand sides once the factorization has been computed.

Relation to LUP Solve Algorithm (Pseudocode)

The recurrence corresponds to the solve phase of the LUP decomposition, which consists of forward and backward substitution.

Forward Substitution (Solving $Ly = Pb$)

```
for i = 1 to n:
    y[i] = b[i]
    for j = 1 to i-1:
        y[i] = y[i] - L[i][j] * y[j]
```

The outer loop runs n times, and the inner loop executes $i - 1$ constant-time operations, giving rise to the summation

$$\sum_{i=1}^{n} \left[O(1) + \sum_{j=1}^{i-1} O(1) \right].$$

Backward Substitution (Solving $Ux = y$)

```
for i = n down to 1:
    x[i] = y[i]
    for j = i+1 to n:
        x[i] = x[i] - U[i][j] * x[j]
```

Here, the inner loop executes $n - i$ constant-time operations, producing the summation

$$\sum_{i=1}^{n} \left[O(1) + \sum_{j=i+1}^{n} O(1) \right].$$

Together, these two procedures account for the full recurrence analyzed above.

Relation to LUP Solve Algorithm (Pseudocode Interpretation)

The recurrence solved earlier corresponds to the *solve phase* after LUP decomposition, not to the decomposition itself. This solve phase consists of two steps:

- Forward substitution to solve $Ly = Pb$,
- Backward substitution to solve $Ux = y$.

Below, we present the pseudocode-level interpretation and show how each loop maps directly to the recurrence relation.

Forward Substitution (Lower Triangular Solve)

Pseudocode

```
for i = 1 to n:
    y[i] = b[i]
    for j = 1 to i-1:
        y[i] = y[i] - L[i][j] * y[j]
```

Cost Analysis

- The outer loop runs n times.
- The inner loop runs $i - 1$ times for a given i.
- Each inner operation takes $O(1)$ time.

Thus, the total cost of forward substitution is:

$$\sum_{i=1}^{n} \left[O(1) + \sum_{j=1}^{i-1} O(1) \right],$$

which matches the first summation in the recurrence.

Backward Substitution (Upper Triangular Solve)

Pseudocode

```
for i = n down to 1:
    x[i] = y[i]
    for j = i+1 to n:
        x[i] = x[i] - U[i][j] * x[j]
```

Cost Analysis

- The outer loop runs n times.
- The inner loop runs $n - i$ times for a given i.
- Each inner operation takes $O(1)$ time.

Thus, the total cost of backward substitution is:

$$\sum_{i=1}^{n} \left[O(1) + \sum_{j=i+1}^{n} O(1) \right],$$

which matches the second summation in the recurrence.

Combined Interpretation

Combining both phases, the total running time of the LUP solve procedure is:

$$T(n) = \sum_{i=1}^{n} \left[O(1) + \sum_{j=1}^{i-1} O(1) \right] + \sum_{i=1}^{n} \left[O(1) + \sum_{j=i+1}^{n} O(1) \right].$$

This recurrence therefore represents exactly the combined cost of:

- Forward substitution, and
- Backward substitution.

As shown earlier, this simplifies to:

$$T(n) = O(n^2).$$

Result

The recurrence arises directly from the nested-loop structure of forward and backward substitution in the LUP solve procedure. This confirms that, while the LUP decomposition itself requires $O(n^3)$ time, each subsequent solve step runs in quadratic time.

Practical Efficiency. Both forward and backward substitution access matrix entries in a largely sequential manner, which leads to good cache locality in practice. This memory access pattern further contributes to their efficiency beyond the quadratic asymptotic time complexity.

Interpretation in the Context of LUP Decomposition

This recurrence corresponds to the forward substitution and backward substitution steps used when solving:

$$Ax = b \quad \text{using} \quad PA = LU.$$

The key insight is:

- LUP decomposition itself takes $O(n^3)$ time,
- Once L and U are available, each solve step (forward and backward substitution) takes $O(n^2)$ time.

This confirms that the solve phase is quadratic, not cubic.

Conclusion

The given recurrence simplifies to a quadratic time complexity. This aligns with the theoretical expectation for forward and backward substitution in LUP-based linear system solvers, where nested loops execute a linear number of constant-time operations.