If I modified the function to be:

```
x = f(n)
  x = 1;
  y = 1;
  for i = 1:n
      for j = 1:n
          x = x + 1;
           y = i +
```

. Will this increate how long it takes the algorithm to run (e.x. you are timing the function like in #2)?

⇨ Solution:

⇨ Step 1: Understanding the Modification

```
function x = f(n)
      x = 1;
      y = 1;
      for i = 1:n
         for j = 1:n
            x = x + 1;
            y = i + j;
```

⇨ Now includes an **additional assignment operation** inside the inner loop:

$$y = i + j$$

Step 2: Analyzing the Complexity

- The added statement $y = i + j$ is a simple assignment.
- The **loop structure remains the same** $(O(n^2))$.
- The total number of operations per iteration of the inner loop **increases from 1 to 2**.

Thus, the new number of operations is:

$$T(n)=\sum_{i=1}^{n}\sum_{j=1}^{n} 2 = 2n^3$$

Step 3: Will the Algorithm Take Longer?

Yes, but **only by a constant factor**.

- The **growth rate** remains $O(n^2)$, so the asymptotic complexity does **not** change.

- However, **in practice**, execution time **will** increase because each inner loop iteration now performs **two** operations instead of one.
- The difference would be **small** but noticeable when timing large n.

Step 4: Conclusion

- **Theoretical Complexity: $O(n^2)$ remains unchanged.**
- **Actual Execution Time: Increases by a constant factor (~2x).**
- **Impact on Big-O Analysis:** No effect. The function still grows quadratically.
- **Empirical Measurement:** If timed, the modified function would take roughly **twice as long** for large nn

🚀 **Final Answer:** Yes, the function will take longer to run, but its complexity remains $O(n^2)$