**1. Find the Runtime Mathematically**

The given function:

```python

function x = f(n)

x = 1;

for i = 1:n

for j = 1:n

x = x + 1;

```

can be analyzed as follows:

To find the runtime of the given algorithm mathematically, we need to analyze the nested loops and how they contribute to the overall number of operations performed by the algorithm.

The given algorithm has a nested loop structure where both the outer loop and the inner loop run from `1` to `n`. The operation inside the innermost loop is `x = x + 1;`, which is executed each time the inner loop runs.

Let's break it down:

- The outer loop runs `n` times.

- For each iteration of the outer loop, the inner loop also runs `n` times.

Since the innermost operation (`x = x + 1;`) is executed once for each iteration of the inner loop, the total number of times this operation is executed is equal to the product of the number of iterations of the outer loop and the inner loop.

Mathematically, we can represent the total number of operations performed (T(n)) as a double summation:

Since the inner summation simply adds 1 for each value of \(j\) from 1 to \(n\), it evaluates to \(n\) for each iteration of \(i\). Therefore, the summation simplifies as follows:

This is because for each iteration of \(i\), the inner loop runs \(n\) times. The summation \(\sum\_{i=1}^{n} n\) is then equivalent to \(n\) times \(n\), as the sum repeats \(n\) for \(n\) times:

Therefore, the runtime of the algorithm is \(O(n^2)\), which indicates that the time complexity is quadratic in relation to the size of the input \(n\). This means that if the input size doubles, the runtime will quadruple, reflecting the quadratic nature of the algorithm's performance.

2. Time the Function for Various ‘ n ‘

Let's implement the function in Python, time it for various values of ‘ n ‘ , and plot the results.

A screen shot of a computer screen

Description automatically generated

**3. Find Polynomial Bounds**

**Upper Bound Polynomial**: It would be a polynomial that lies above all the noisy data points for all n. This could be obtained by adding a constant to the fitted polynomial, a'(n^2) + b’n + c’ , where a’>=a, b’>=b and c’>=c+k, and K is a constant that ensures the polynomial is above all data points.

**Lower Bound Polynomial**: It would be a polynomial that lies below all the noisy data points for all n. Similarly, it could be obtained by subtracting a constant from the fitted polynomial a’'(n^2) + b’’n + c’ , where a’<=a, b’<=b and c’<=c-k ensuring it is below all data points.  
  
big-O : O(n^2)

big-Omega : Omega(n^2)

big-theta : theta(n^2)

**4. Modification Effect on Runtime**

Because there is an additional assignment operation in the loop, adding the operation y = i + j; will cause the runtime to increase slightly. Nevertheless, the algorithm's total time complexity remains unchanged as this increase is constant for every iteration. The nested loops continue to be the main factor, determining (n^2)

Growth rate is : n^2

**5. Effect of Modification on Results from #1**

The algorithm's time complexity, which is still 2 in the mathematical analysis, is unaffected by the modification n^2. The extra operations added to the loops only result in a fixed amount of work being added each iteration; the order of growth is unaffected.

**6. Implement Merge Sort**

Find the code on github Link