**1.For the code given below we should find the summations  
function x = f(n)**

**x = 1;**

**for i = 1:n**

**for j = 1:n**

**x = x + 1;**

Ans:

This code looks like the it shows the output as (n2)+1

0. The first step takes the time c to execute

1.The outer loop itrates form 1 to n so let the time taken be n

2.Inner loop also iterates from 1 to n so time taken is n2

3.and in the inner loop x=x+1 ,X will be incremented by 1 in each iteration so this caluculates the square root but initially x=1 so the output will be ((square root of n )+1)

The time taken is n(n+c)

4.the output is (n2)+1

The time Complexity is T(n)=c+n+n(n+c)+n^2

=c+n+2n2+nc

=2n2+n(1+c)+c

So the runtime complexity is O(n2)

**2. Time this function for various n e.g. n = 1,2,3.... You should have small values of n all the way up to large values. Plot "time" vs "n"**

For the sake of values assume c=1 so the equation will be 2n2+2n+1

The given Below is the graph for

Time vs N

A graph with a line

Description automatically generated

A graph with a line

Description automatically generated

|  |  |
| --- | --- |
| n | T(n) |
| 1 | 5 |
| 2 | 15 |
| 3 | 29 |
| 4 | 49 |
| 5 | 75 |
| 6 | 107 |
| 7 | 145 |
| 8 | 189 |
| 9 | 239 |
| 10 | 295 |
| 20 | 841 |
| 30 | 1791 |
| 40 | 3149 |
| 50 | 4905 |

**3. Find polynomials that are upper and lower bounds on your curve from #2. From this specify a big-O, a big-Omega, and what big-theta is.**

A graph with a line

Description automatically generated

Let us assume that the upper bound is 10n2+2n+1

Lower bound is (1/4)n2+2n+1

A graph with lines and numbers

Description automatically generated

The Big O Big, Big omega and Big Tetha will be O(n2)

Big O: will be worst case

Big Omega: will be the best case

Big tetha: is the Average case

**4. Find the approximate (eye ball it) location of "n\_0" . Do this by zooming in on your plot and indicating on the plot where n\_0 is and why you picked this value. Hint: I should see data that does not follow the trend of the polynomial you determined in #2.**

A graph with lines and numbers

Description automatically generated

The n\_0 is 0 as the graph intersects at 0

**5.**

**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 + j;**

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

Yes the run time increases

1. The first step takes the time c to execute

2.The outer loop itrates form 1 to n so let the time taken be n

3.Inner loop also iterates from 1 to n so time taken is n2

4.and in the inner loop x=x+1 ,X will be incremented by 1 in each iteration so this caluculates the square root but initially x=1 so the output will be ((square root of n )+1) The time taken is n(n+c)

5.The time taken to execute y=i+j is n(n+c)

4.the output is (n2)+1

The time Complexity is T(n)=c+n+2n(n+c)+n^2

=c+n+3n2+2nc

=3n2+n(2+c)+c

=3n2+3n+1

So the runtime complexity is O(n2)

**6. Will it effect your results from #1?**

Yes the values will change according to new T(n)

But the run time complexity does not change

i.e

|  |  |
| --- | --- |
| **n** | **T(n)** |
| 1 | 7 |
| 2 | 31 |
| 3 | 91 |
| 4 | 205 |
| 5 | 391 |
| 6 | 667 |
| 7 | 1051 |
| 8 | 1561 |
| 9 | 2215 |
| 10 | 3031 |
| 20 | 24061 |
| 30 | 81091 |
| 40 | 192121 |
| 50 | 375151 |