HANDS ON-3

DESIGN ANALYSIS AND ALGORITHMS

UTA ID:1002233393

RASHMITHA RAMASANI

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

1.Find the runtime of the algorithm mathematically (I should see summations).

1. Nested loop: The code contains two nested loop,

both will iterate from 1 to n.

Outer loop: the outer loop runs at $i=1:n$, it means it runs n times.

Every time outer loop runs, the inner loop also runs at $i=1:n$, since it run times.

Inner loop: The inner loop statement is executed once per iteration of inner loop.

i.e, $x = x+1$

Total iterations: The inner loop runs n times for each iteration, the total iterations are

$$n^* n = n^2$$

Runtime:

The runtime will be $O(n^2)$

proof:

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

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

$$T(n) = 1 + n^* n$$

$$T(n) = 1 + n^2$$

Therefore, the runtime is $O(n^2)$.
Hence proved.

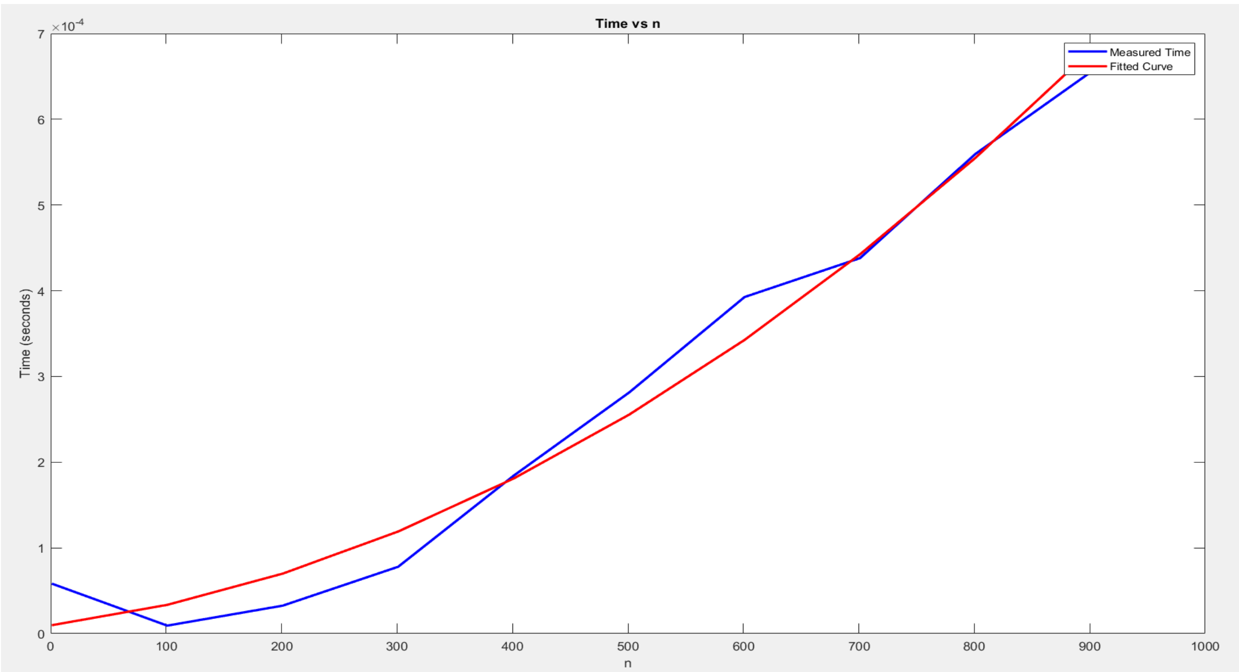
2.Time this function for various n e.g. n = 1,2,3.... You should have small values of n all the way up to large values. Plot "time" vs "n" (time on y-axis and n on x-axis). Also, fit a curve to your data, hint it's a polynomial.

The time vs n graph :Time on y axis

N on x-axis

Blue line: Represents the actual measured times.

Red line: Represents the fitted quadratic curve.



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.

3. Big O: A function $f(n)$ is said to be $O(g(n))$, if there exist positive constant c and n_0 such that $f(n) \le cg(n)$, for all $n \ge n_0$.

From the upper and lower bounds, the time complexity is n^2,

Therefore upper bound is $O(n^2)$

~~Big theta~~: Big Omega: A $f(n)$ is said to be $\Omega(g(n))$ if there exist positive constant c, and n_0 such that $f(n) \ge cg(n)$, for all $n \ge n_0$ from the lower bound, the time complexity is n^2,

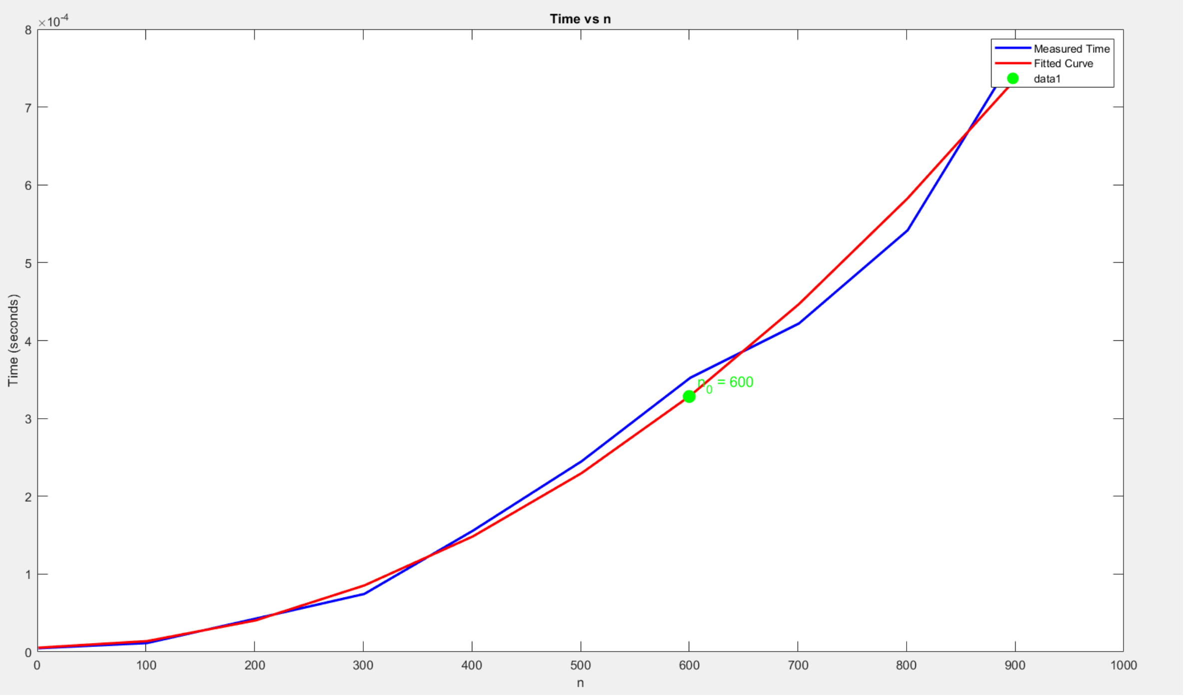
Therefore it is $\Omega(n^2)$.

Big Theta: A $f(n)$ is said to be $\Theta(g(n))$,

if there exist positive constant c_1, c_2 and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$

Since the upper and lower bounds are n^2, and then Big theta is $\Theta(n^2)$

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.



The time vs n graph :Time on y axis

N on x-axis

Blue line: Represents the actual measured times.

Red line: Represents the fitted quadratic curve

GREEN dot represents the data

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;
```

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

6

4. given function $x = f(n)$

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

Nested loop: The function has two nested loops.

Outer loop: The outerloop execute n times, when the outerloops runs from $i = 1:n$

Inner loop: for the each iteration in outerloop, the inner loop runs from $j = 1$ to n, this also executes n times.

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

$$= 2\sum_{i=1}^{n} n$$

$$\Rightarrow 2 * n * n$$

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

Time Complexity is $O(n^2)$.

Therefore modifying the function doesnot change the overall runtime complexity.

Finally it wont change the asymptotic complexity, polynomial growth rate and summation doesnot change, it still remains same.

5. Will it effect your results from #1?

5. No, Overall time complexity will remain same $O(n^2)$,

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

$$= n \sum_{i=1}^{n} 1$$

$$\Rightarrow n * n$$

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

In modified function, $y = i + j$ operation performed, but it does not affect time complexity.

The runtime complexity remains same $O(n^2)$,

Hence does not change, remains same.