

Basics of Log:

Logb(a): Log means what should be the power of b to get the a.

log2(64) = 6: In this context what should be the power of 2 to get 64 which is 6.

Log3(27) = 3: what should be the power of 3 to get 27 which is 3.

Log5(25) = 2: what should be the power of 5 to get 25 which is 2.

Log2(32) = 5: what should be the power of 2 to get 32 which is 5.

Logb(a) = x 🡪 bx=a

Log2(10) => 3. something.

Log2(40) => 5. something.

Note:

**2k=N => log2N=k => Loga(aN) = N**

Log2(26) = 6

Log3(35) = 5



N=100

100 🡪 50 🡪 25 🡪12 🡪 6 🡪 3 🡪 1. 6 times.

N=324

324 🡪 162 🡪 81 🡪 40 🡪 20 🡪 10 🡪 5 🡪 2 🡪 1. 8 times.

Generically:

N 🡪 N/2 🡪 N/4 🡪 N/8 🡪 … 🡪 1.

N/20 🡪 N/21 🡪 N/22 🡪 N/23 🡪 ... 🡪 N/2k.

N/2k=1

N=2k

Log2N=k



ANS: Log2 N

O(LogN)

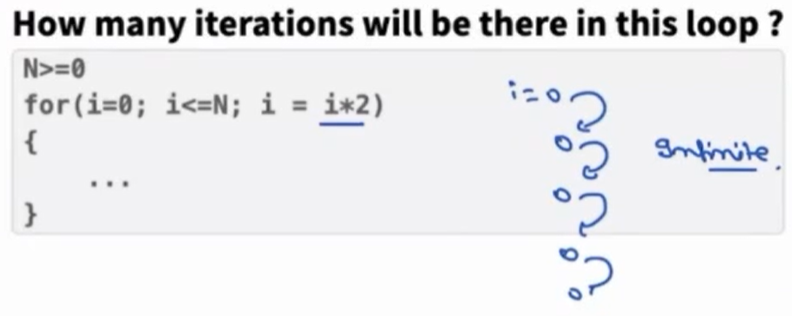
N=32 then 32🡪16🡪8🡪4🡪2🡪1.



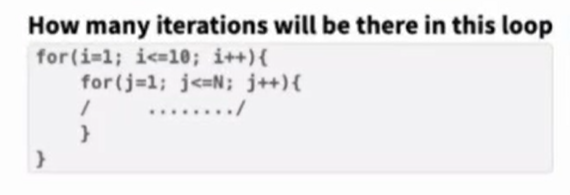
ANS: Log2 N

O(LogN)

N=32 then 1🡪2🡪4🡪8🡪16🡪32.



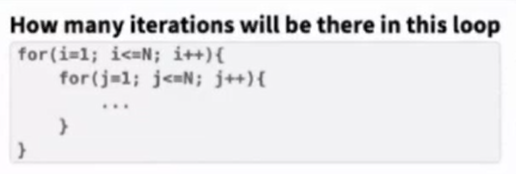
ANS: INFINITE



ANS: 10\*N

O(N)

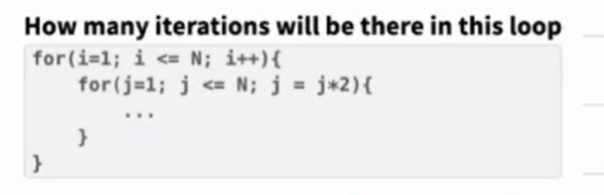
|  |  |  |
| --- | --- | --- |
| **i** | **j** | **Iteration** |
| 1 | [1, N] | N |
| 2 | [1, N] | N |
| 3 | [1, N] | N |
| 4 | [1, N] | N |
| 5 | [1, N] | N |
| 6 | [1, N] | N |
| 7 | [1, N] | N |
| 8 | [1, N] | N |
| 9 | [1, N] | N |
| 10 | [1, N] | N |
| Total Iterations 🡪 |  | 10N |



ANS: N\*N

O(N2)

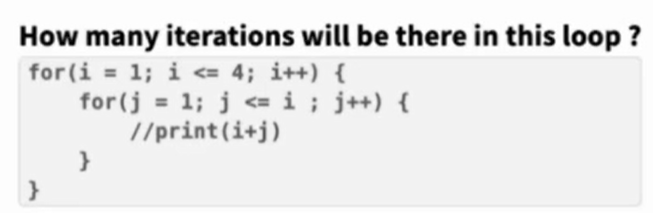
|  |  |  |
| --- | --- | --- |
| **i** | **j** | **Iteration** |
| 1 | [1, N] | N |
| 2 | [1, N] | N |
| 3 | [1, N] | N |
| 4 | [1, N] | N |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| N | [1, N] | N |
| Total Iterations 🡪 |  | N\*N |



ANS: N\*log(N)

O(N\*LogN)

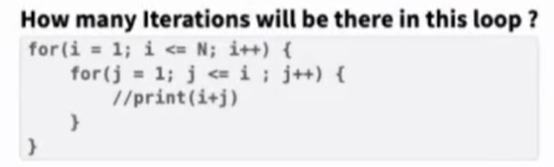
|  |  |  |
| --- | --- | --- |
| **i** | **j** | **Iteration** |
| 1 | [1, N] by multiplying 2 | Log2(N) |
| 2 | [1, N] by multiplying 2 | Log2(N) |
| 3 | [1, N] by multiplying 2 | Log2(N) |
| 4 | [1, N] by multiplying 2 | Log2(N) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| N | [1, N] by multiplying 2 | Log2(N) |
| Total Iterations 🡪 |  | N\* Log2(N) |



ANS: 10

O(1)

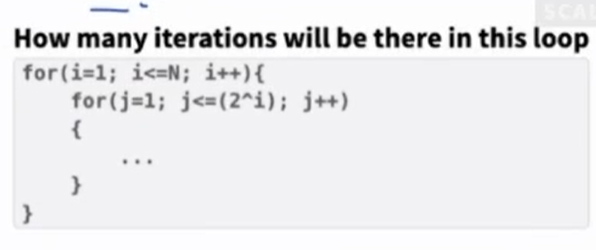
|  |  |  |
| --- | --- | --- |
| **i** | **j** | **Iteration** |
| 1 | [1, 1] | 1 |
| 2 | [1, 2] | 2 |
| 3 | [1, 3] | 3 |
| 4 | [1, 4] | 4 |
| Total Iterations 🡪 |  | 10 |



ANS: N(N+1)/2

O(N2)

|  |  |  |
| --- | --- | --- |
| **i** | **j** | **Iteration** |
| 1 | [1, 1] | 1 |
| 2 | [1, 2] | 2 |
| 3 | [1, 3] | 3 |
| 4 | [1, 4] | 4 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| N | [1, N] | N |
| Total Iterations 🡪 |  | N\* (N+1)/2 (GP) |



ANS: 2\*(2n-1)

O(2n)

|  |  |  |
| --- | --- | --- |
| **i** | **j** | **Iteration** |
| 1 | [1, 2] = [1, 21] | 21 |
| 2 | [1, 4] = [1, 22] | 4=22 |
| 3 | [1, 8] = [1, 23] | 8=23 |
| 4 | [1, 16] = [1, 24] | 16=24 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| N | [1, 2n] | 2N |
| Total Iterations 🡪 |  | 2\*( 2n-1) |

Sum= 21+22+23+24+…+2N a=2(1st value), r=2(Common ratio), Term =N

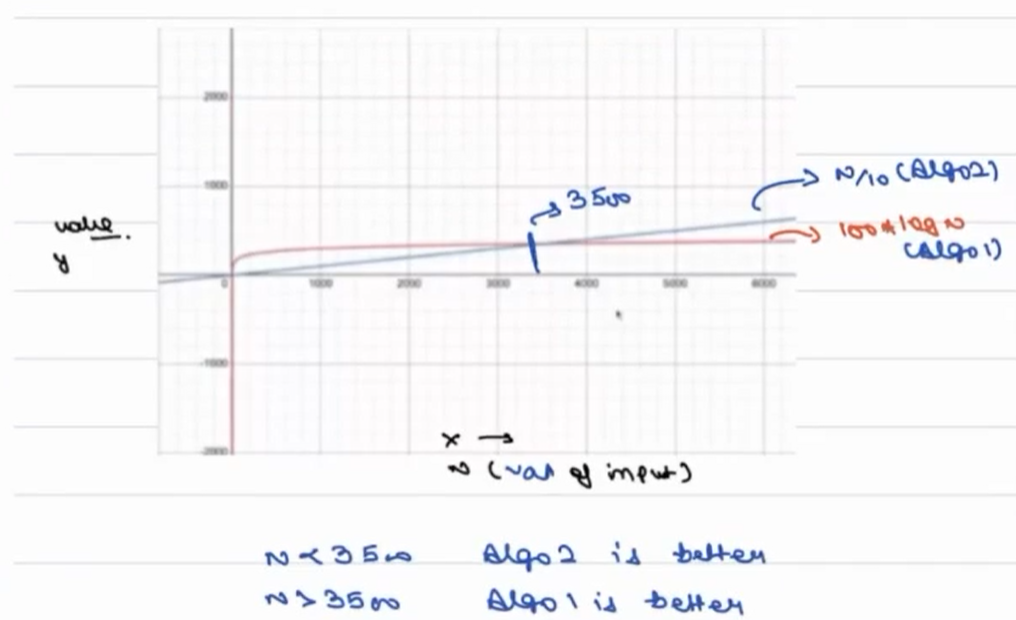
Sum= a(rN-1)/r-1 🡪 2(2N-1)/2-1 🡪 2(2n-1)

COMPARE TWO DIFFERENT ALGORITHMS:

Suppose:

Algo. 1 taking 100\*Log(N) iteration

Algo. 2 taking N/10 iteration

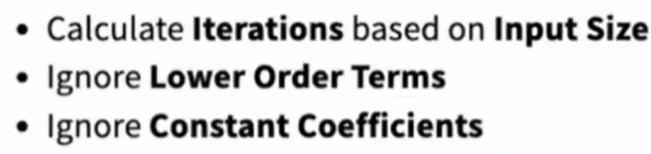


Conclusion: Algo. 1 is better because its taking less iterations for huge data.

**To compare two different algorithms, we use asymptotic analysis technique.**

We care in this only when input is large

**Big’O**



100\*Log(N) 🡪 Big’O(Log(N)): here 100 is constant

N/10 🡪 O(N): here 10 is constant

Suppose for some algorithm we calculated below iterations:

4N2+3N+1 🡪 4N2 (we are ignoring lower order terms i.e., 3N+1) 🡪 O(N2) ignoring constant coefficients i.e., 4

So, for 4N2+3N+1 iteration O(N2)

Ordering:

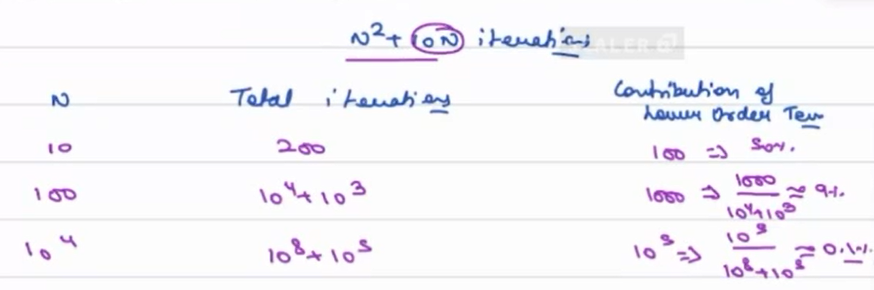
Log(N) **<** **<** N **<** N\*Log(N) **<** N **<** N2 **<** N3 **<** 2N **<** N! **<** NN

e.g.,



When iterations are only constant then Big’O value is 1. O (1)

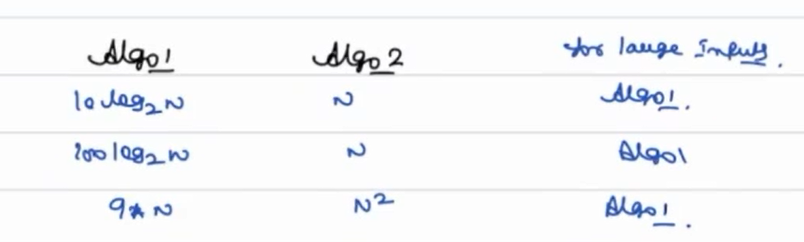
**Why do we neglect lower order terms?**



Conclusion: As input size increases, the contribution of lower order terms decreases.

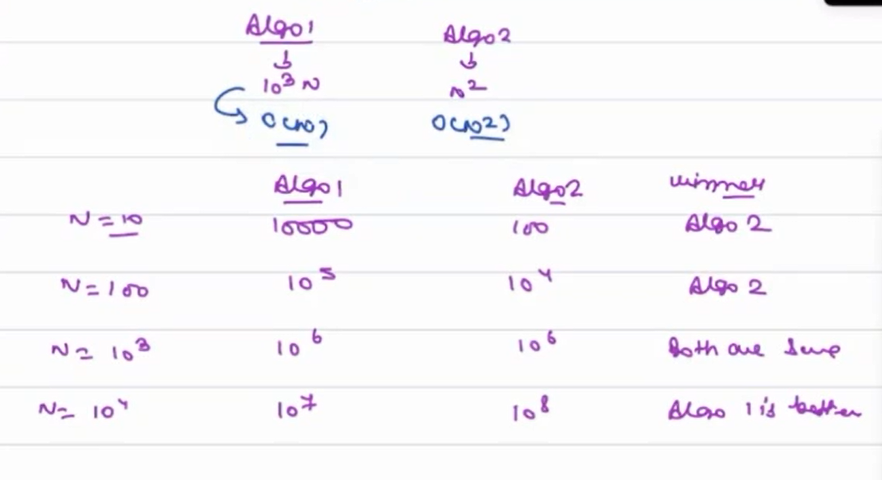
**Why do we neglect constant coefficient?**

**N=100**



Issues in Big’O:

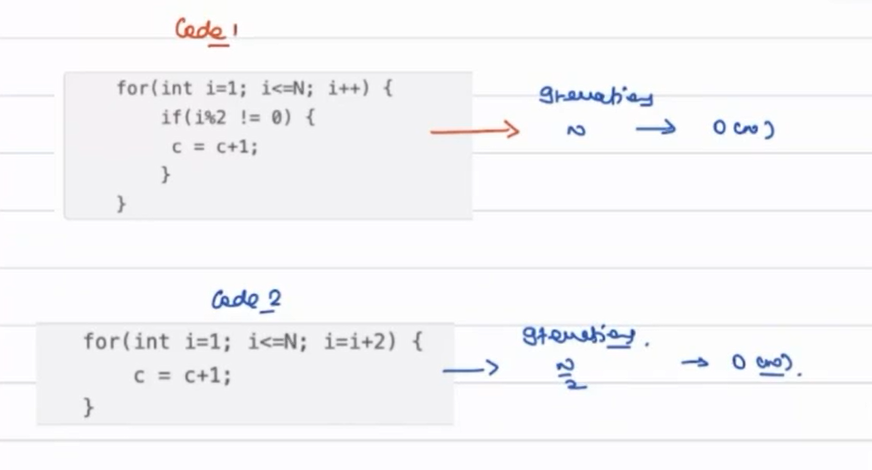
Issue 1:



Big O only give correct answer for larger input

Claim is for all large inputs >= 1000 algo1 will perform better else algo. 2

Issue 2:

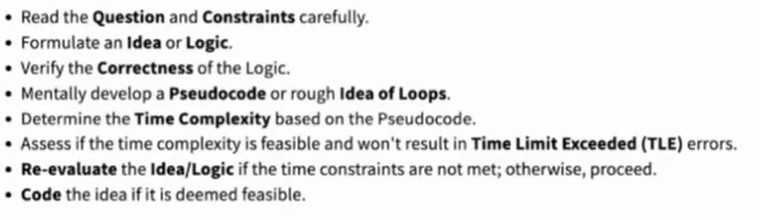


Code 2 is better but BigO is same.

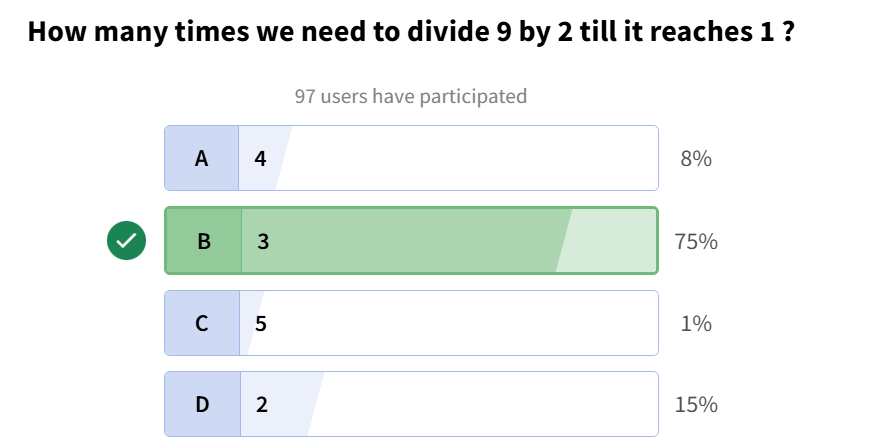
TIME LIMIT EXCEDED ERROR:

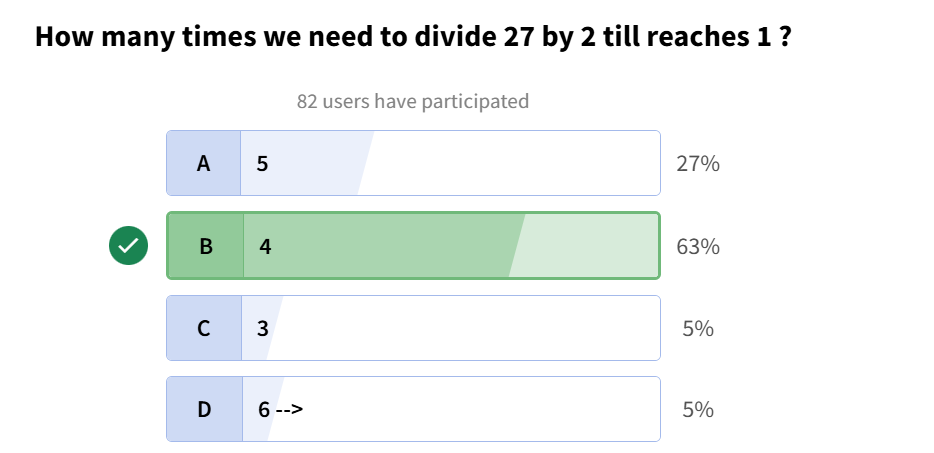
* All online code editors mutually decided to use same machine for code execution i.e., 1GHz machine
* 1GHz processors = 109 instruction per second
* For any code submission they gave 1 sec of time
* For any problem we are given 1 sec. time limit if its not work withing 1 sec. then we get TLE error
* This 1 GHz allocated to each person who using online editor
* Approximation 1: In a small code generally (not always), iteration= 10 instruction, so I can do in 1 sec. approx. 108 iterations.
* Approximation 2: In a big code iteration =100 instruction, so I can do in 1 sec. approx. 107 iterations.
* Based on Approximation 1 & 2 conclusion is our code can have 107 iterations to 108 iterations then only it will run in 1 sec.

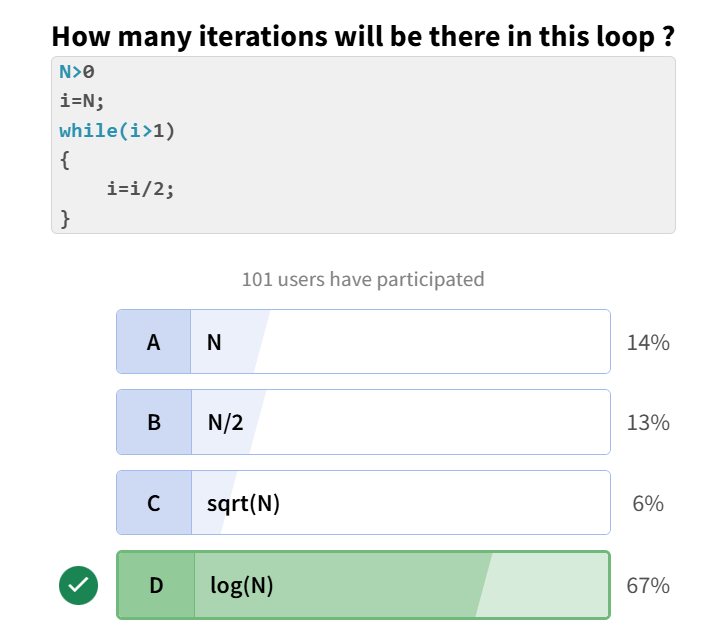
HOW TO APPROACH A PROBLEM:

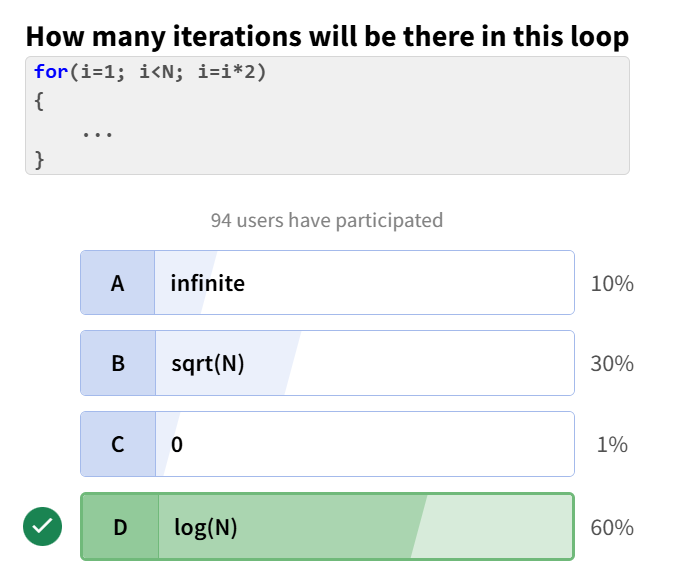


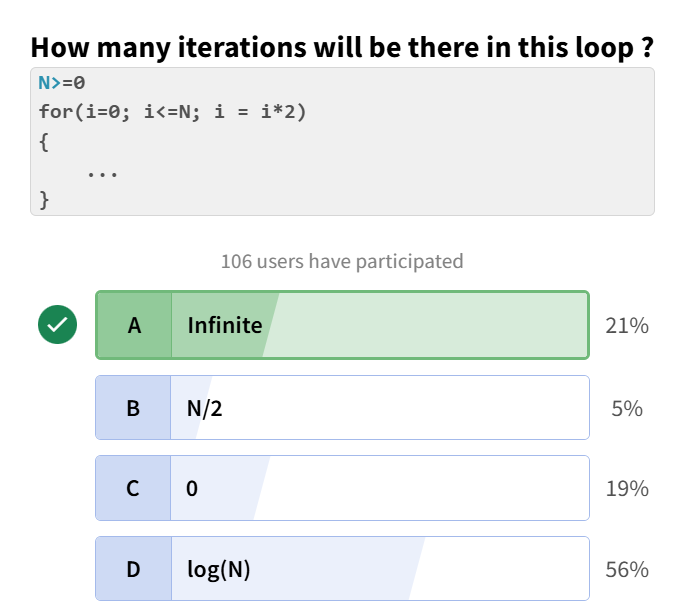
QUIZZES:

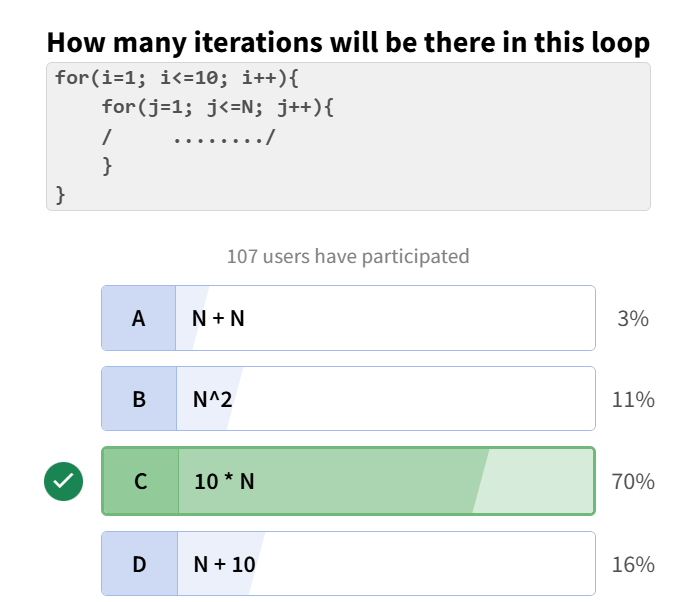


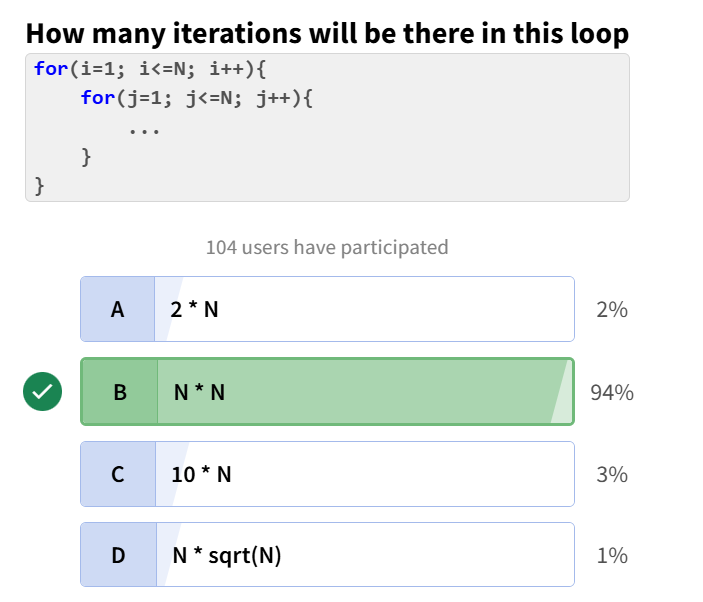


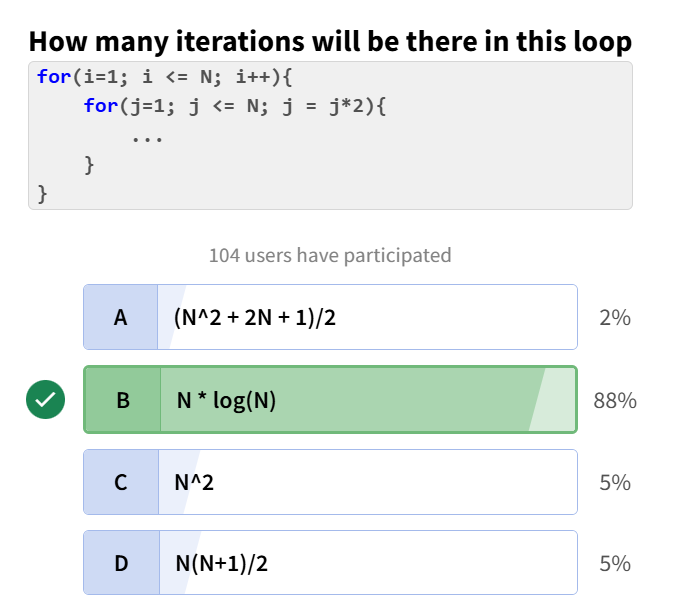


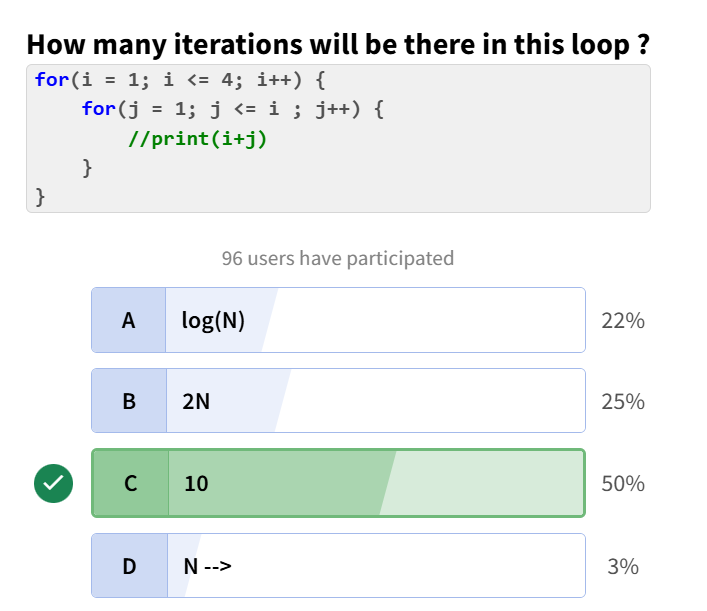


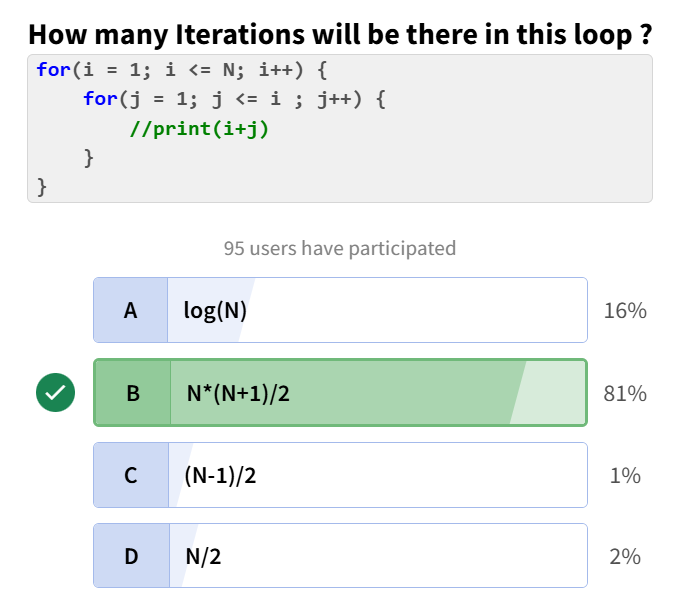


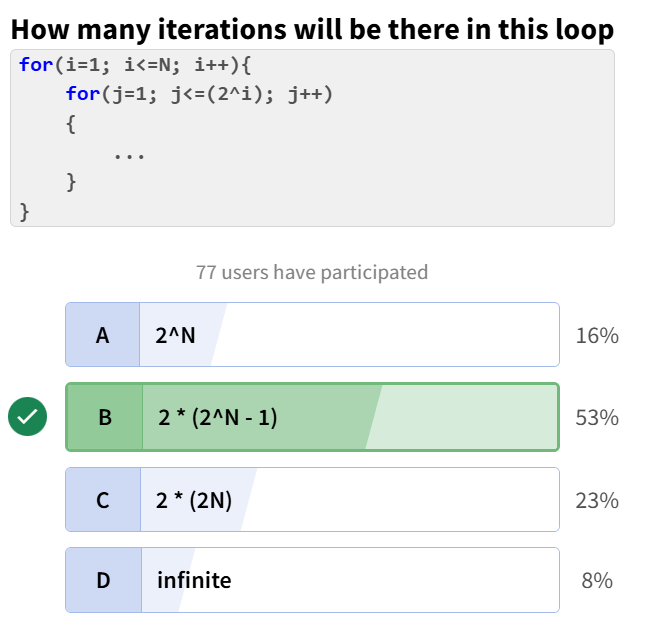


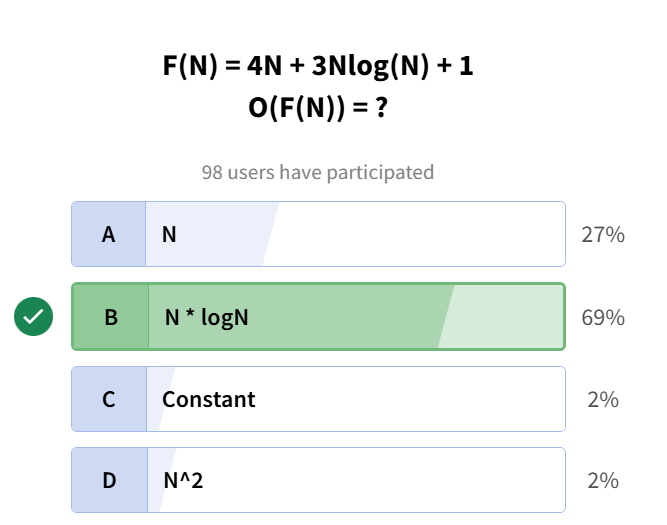


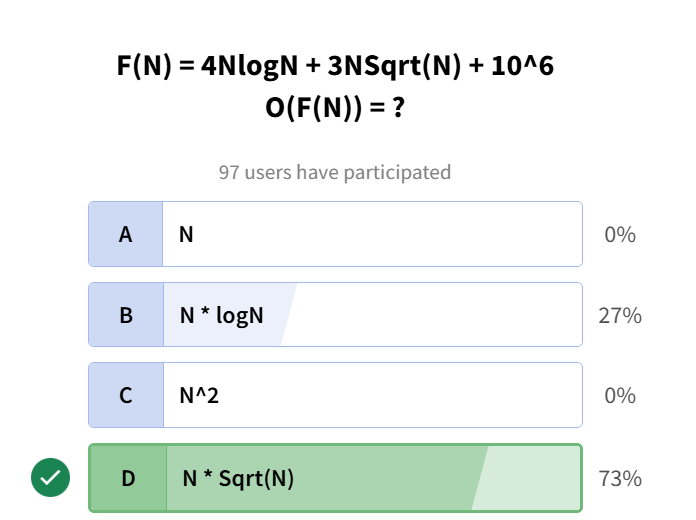








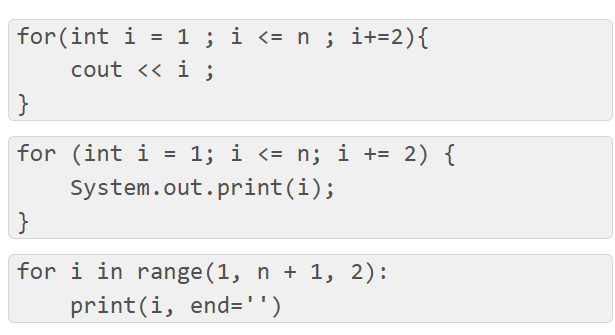


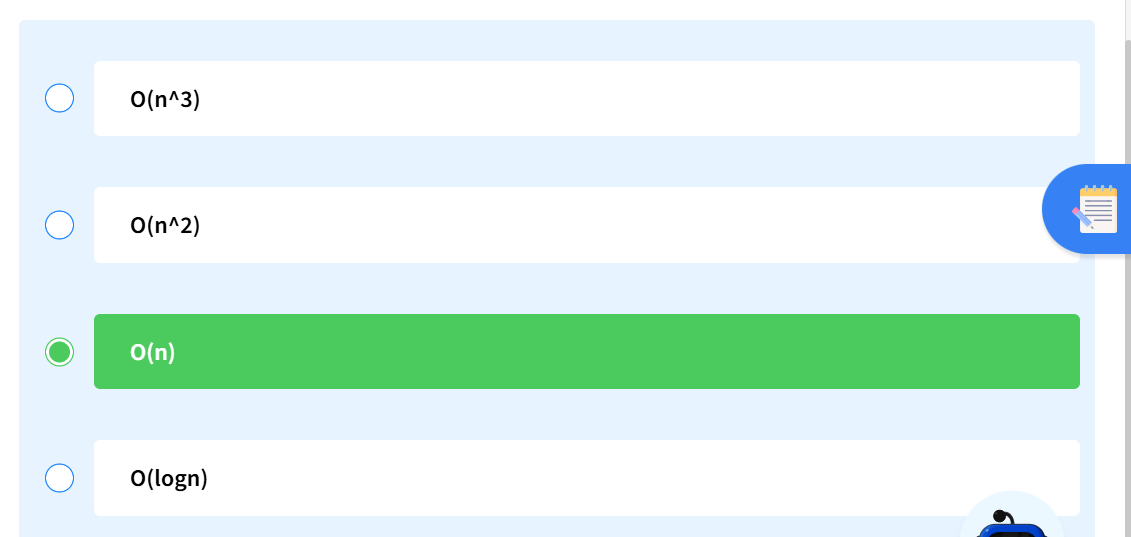


ASSIGNMENTS:

**Q1. Loop\_Time Complexity**

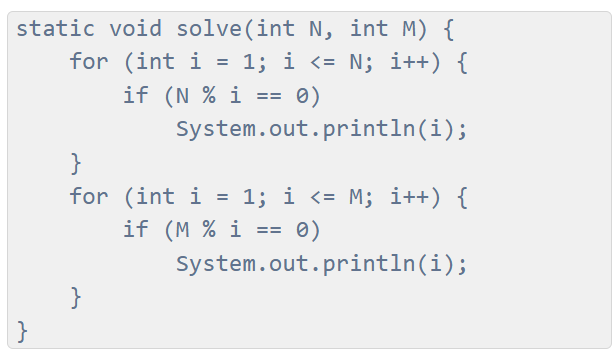


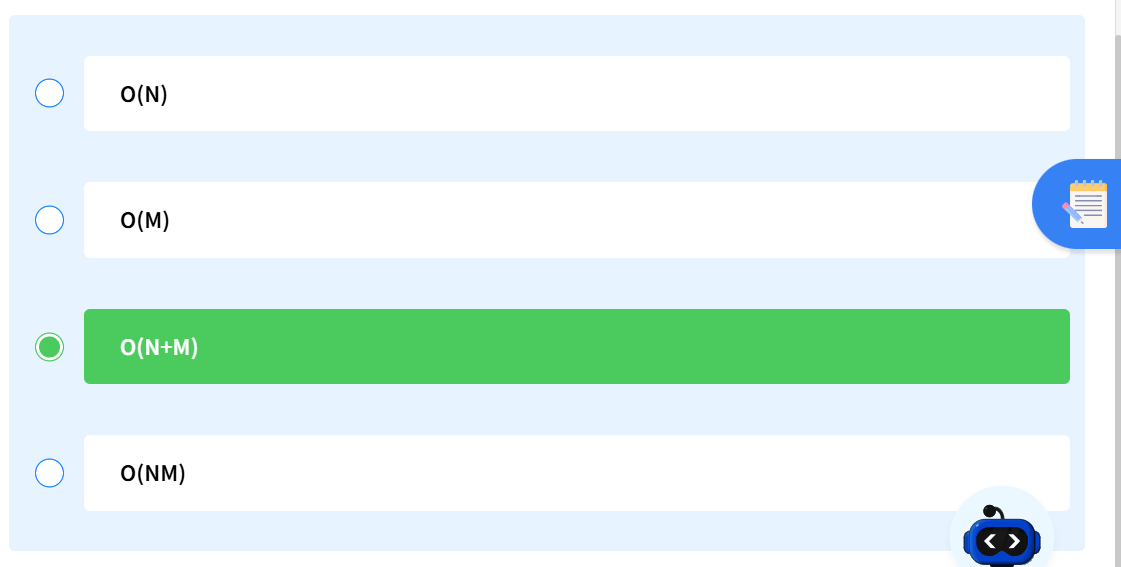




**Q2. Find Time Complexity – 2**

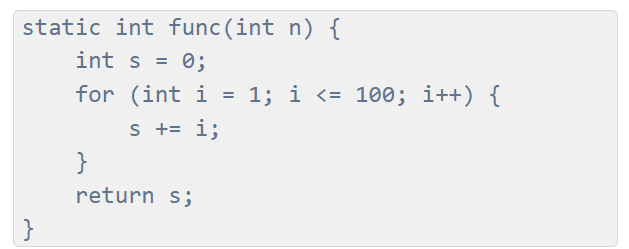
What is the time complexity of the following code snippet

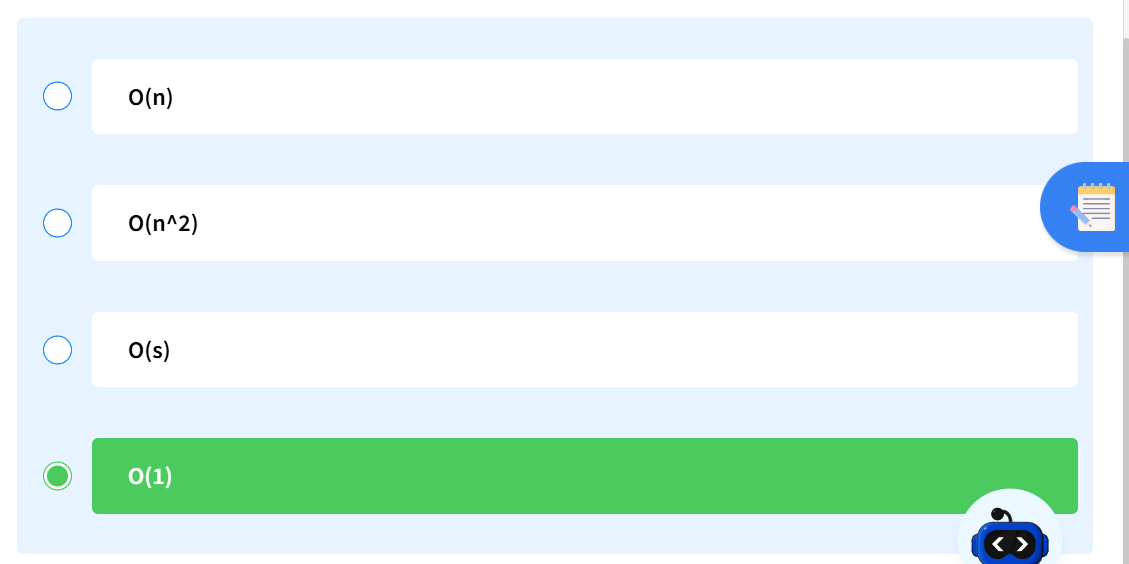




**Q3. Linear Loop Time Complexity**

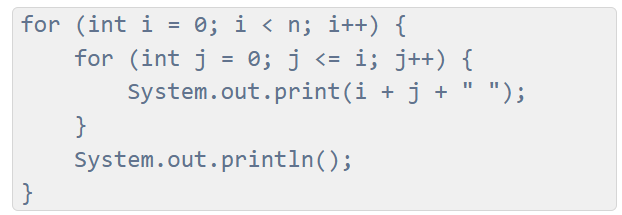
What is the time complexity of the following code :





**Q4. Double Loop Analysis**

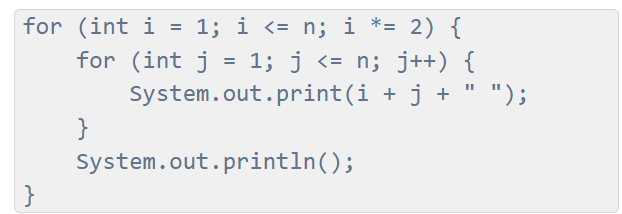
What is the time complexity of the following code :

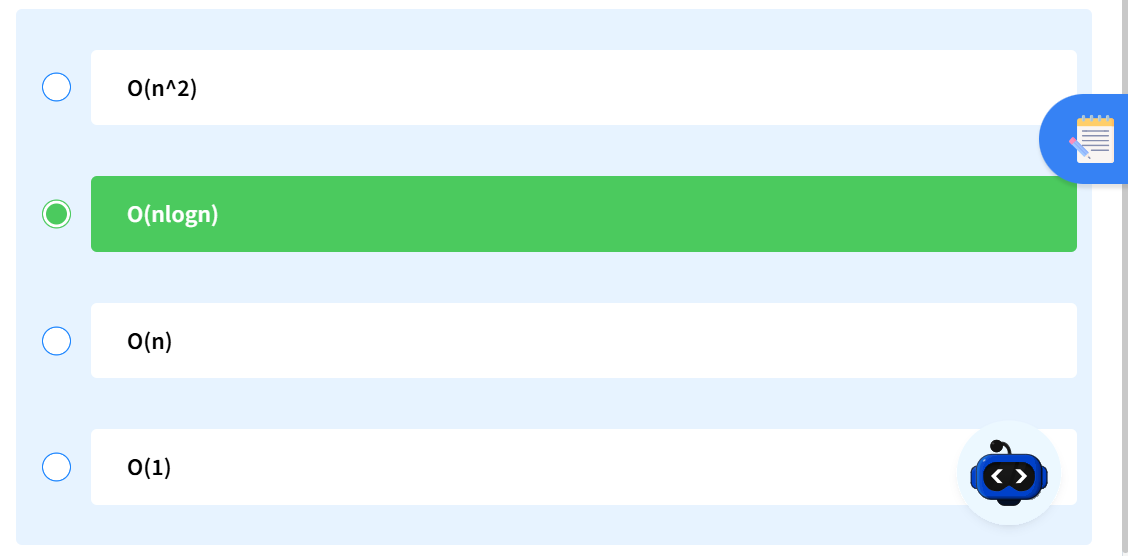




**Q5. Find Time Complexity – 8**

What is the time complexity of the following code :

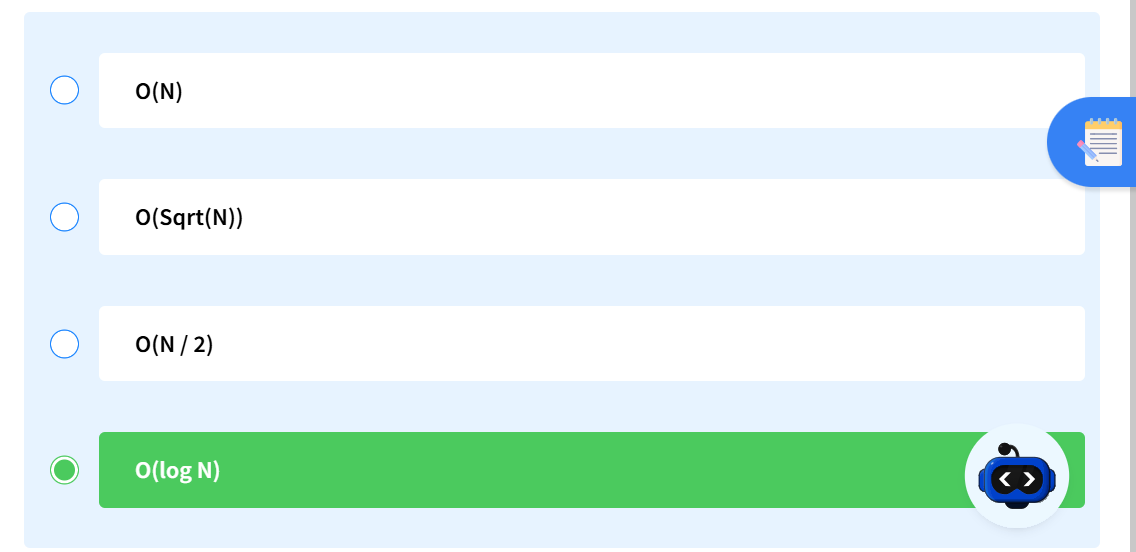




**Q6. Time-Complexity-5**

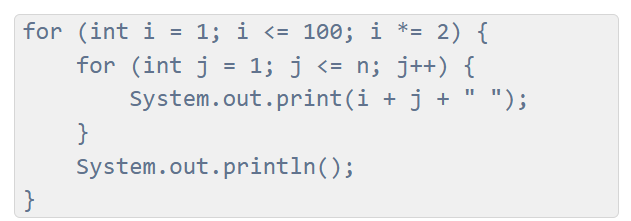
What is the time complexity of the following code :

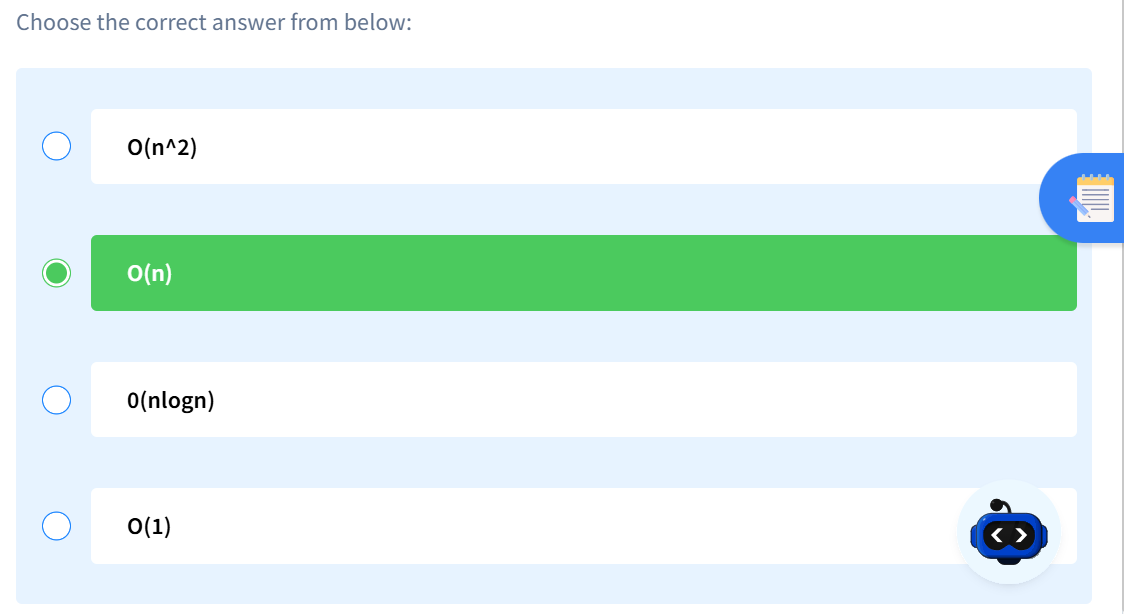




**Q7. Nested Loop with Doubling**

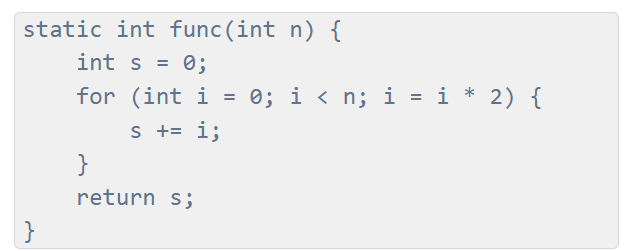
What is the time complexity of the following code :

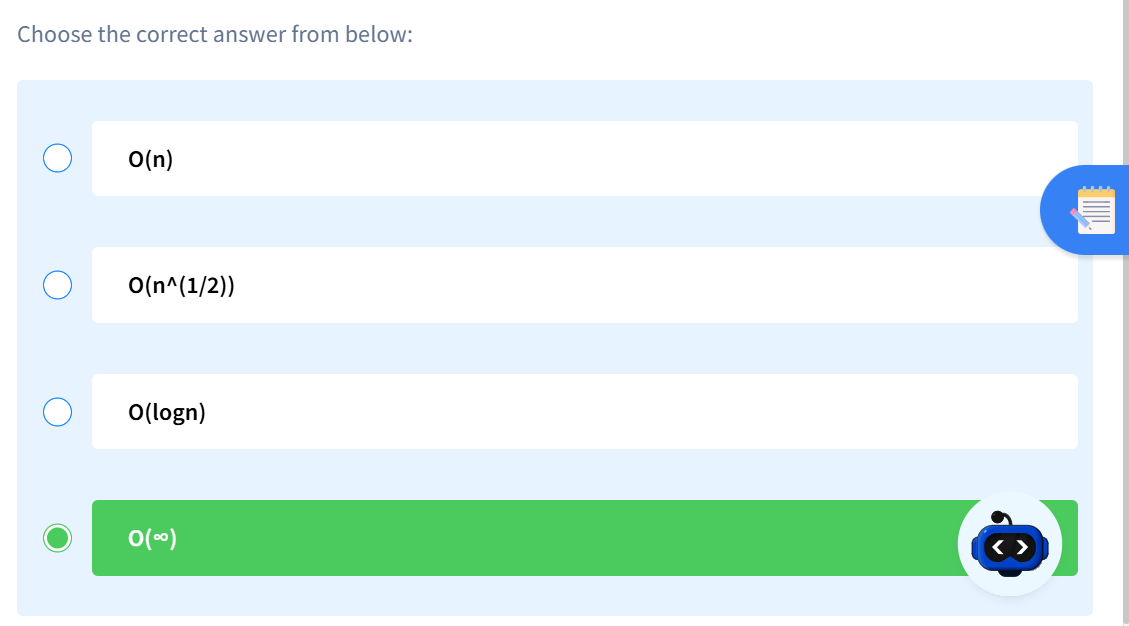




**Q8. Time Complexity with Condition**

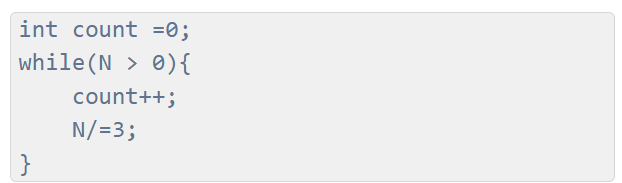
What is the time complexity of the following code :

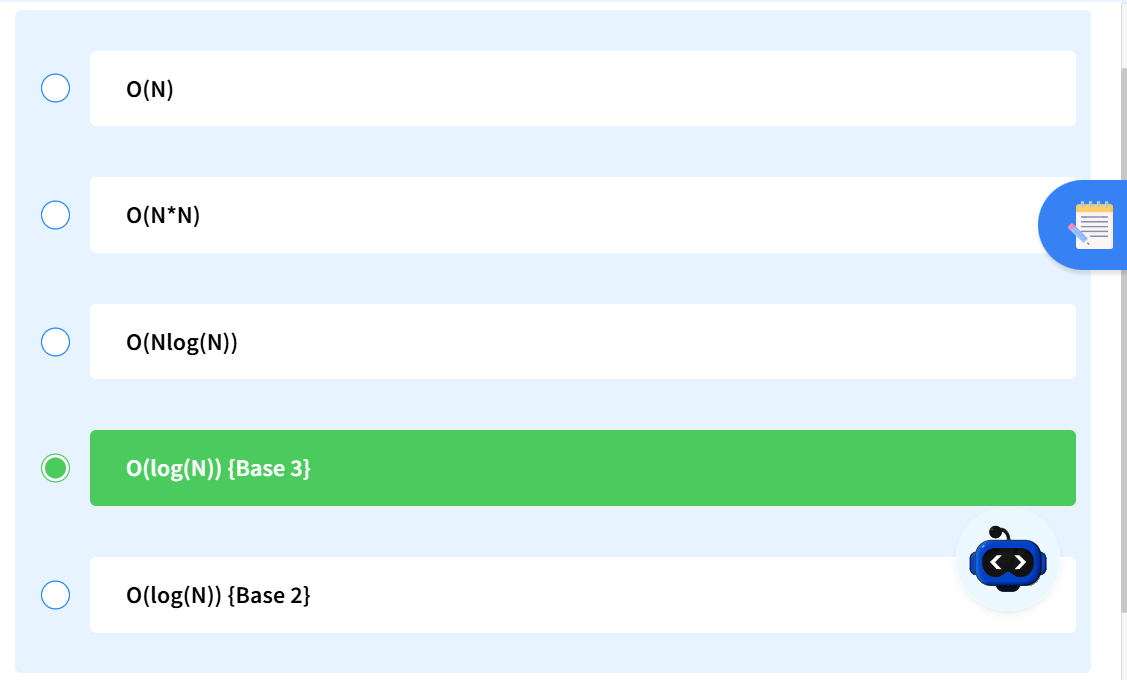




**Q9. Time Complexity Easy 01**

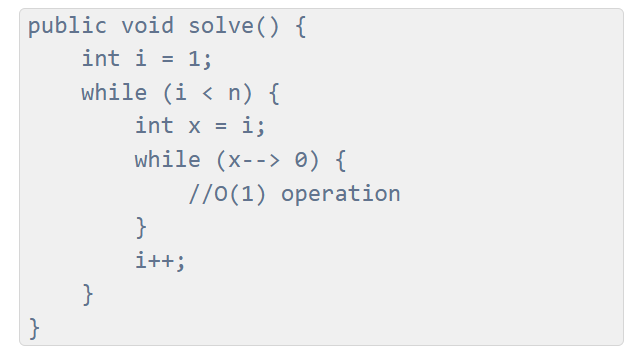
What is the Time Complexity of following snippet ?





**Q10. Time Complexity - 3B**

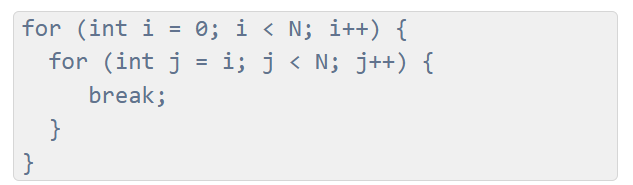
What will be the Time Complexity of the given code?

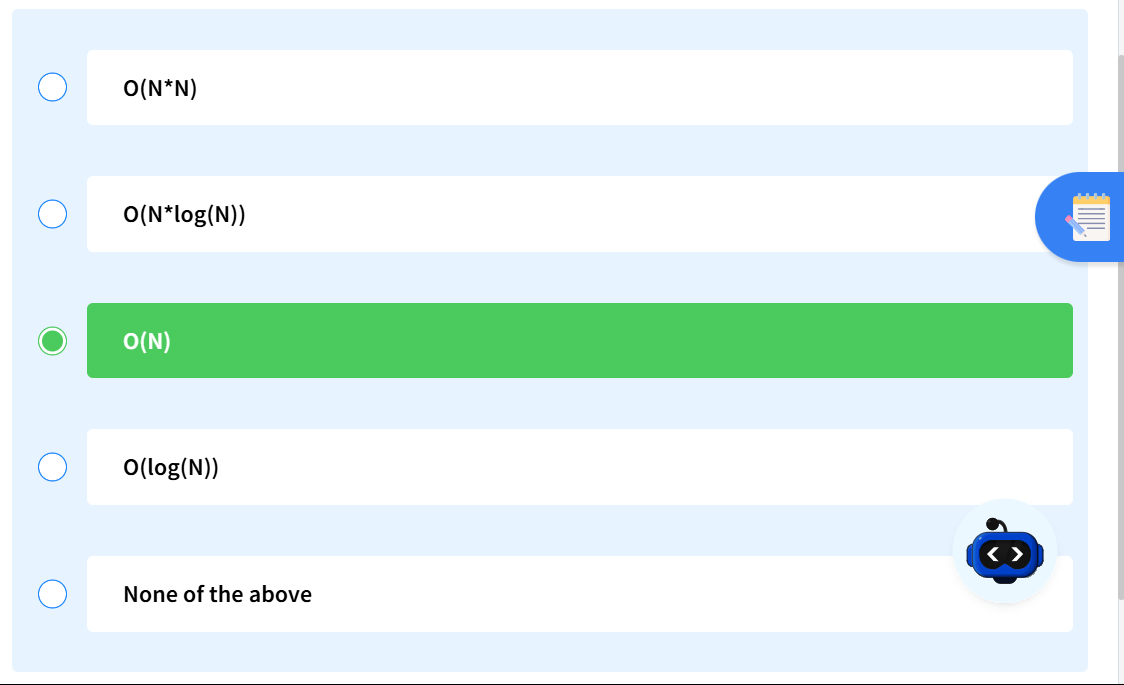




**Q11. Time Complexity Easy 02**

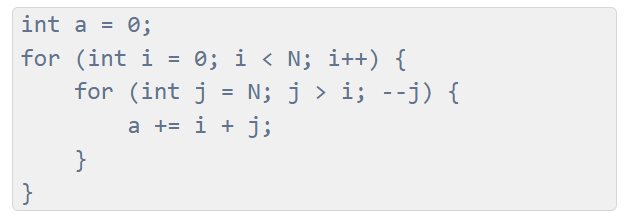
What is the Time Complexity of following snippet ?

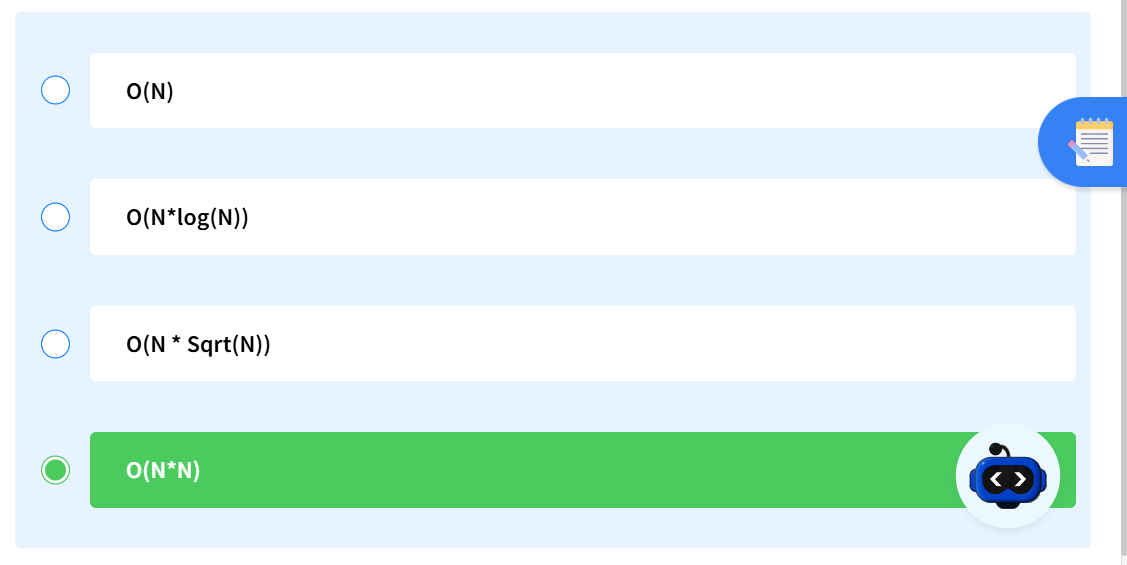




**Q12. NESTED\_CMPL2**

What is the time complexity of the following code :





ADDITIONAL ASSIGNMENT:

**Q1. Find Time Complexity – 4**

What is the time complexity of the following code snippet

int func(int n){

int s = 0;

for(int i = 1 ; i\*i\*i <= n ; i++){

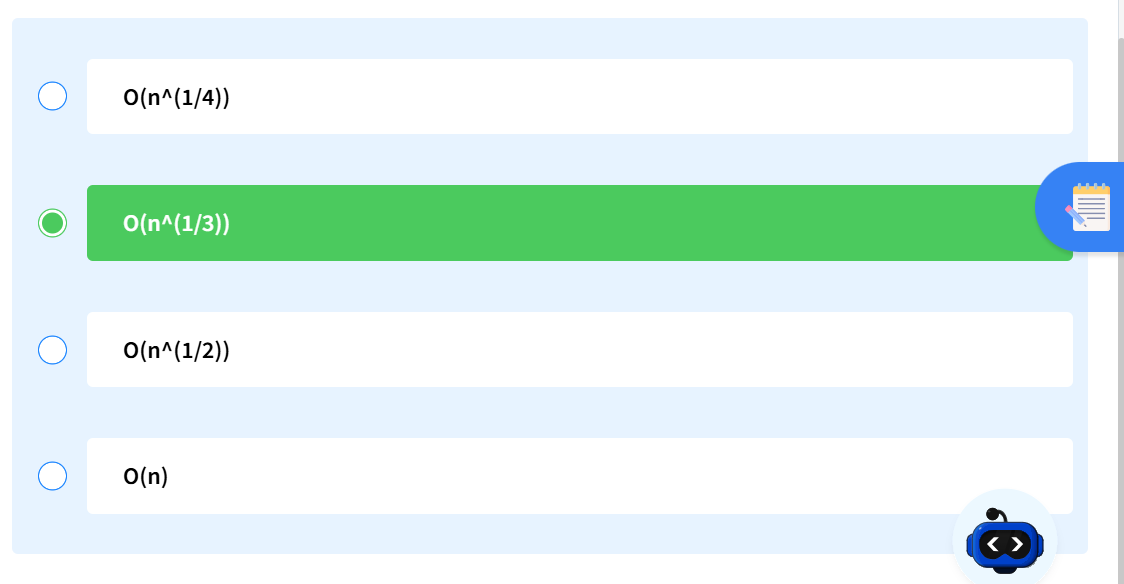
s = s + i;

}

return s;

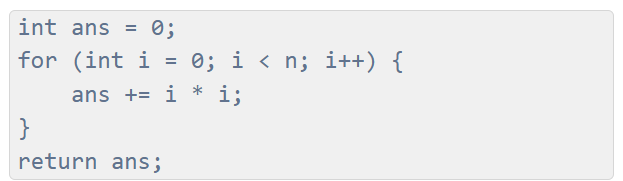
}

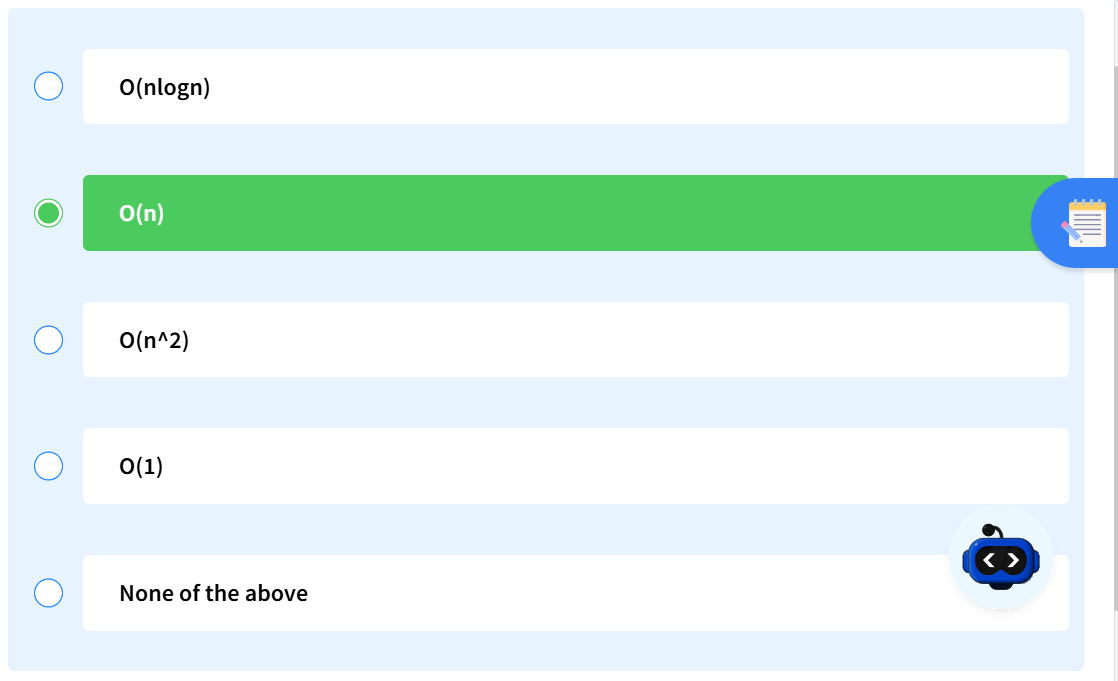
Cube root of N i.e. n1/3



**Q2. Complexity of Single Loop**

What is the complexity of the following code snippet ?





**Q3. Find Time Complexity – 9**

What is the time complexity of the following code snippet

for(int i = 1 ; i <= n ; i++){

for(int j = 1 ; j <= 3^i ; j++){

print(i + j);

}

}



Total work=31+32+33+⋯+3n

First term a=3

Common ratio r=3

n terms.

Sum🡪 a(rn-1)/(r-1) where r! = 1

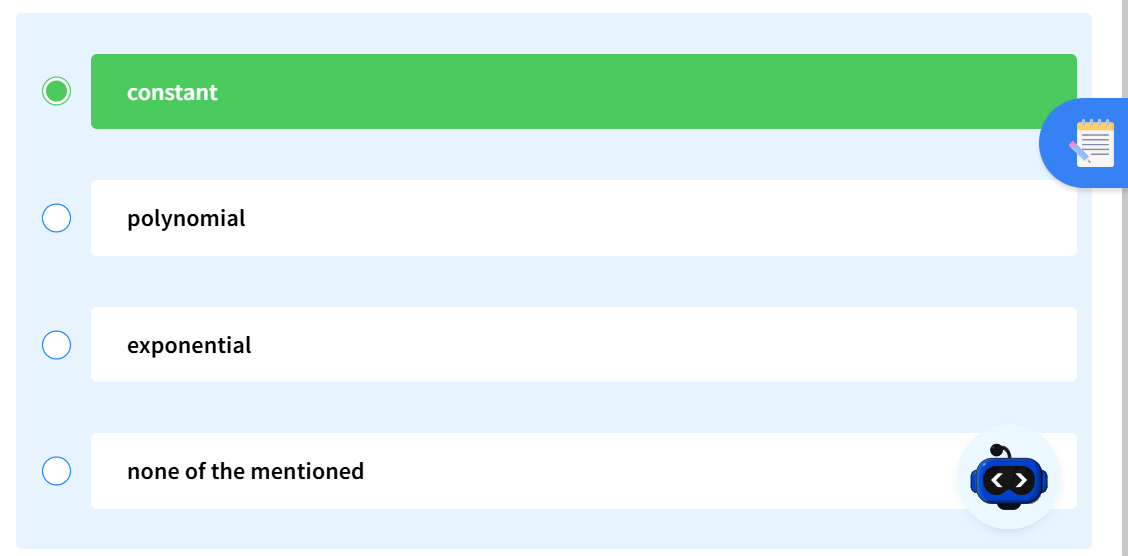
3(3n-1)/3-1

3/2 \* 3n-3/2

O(3n)

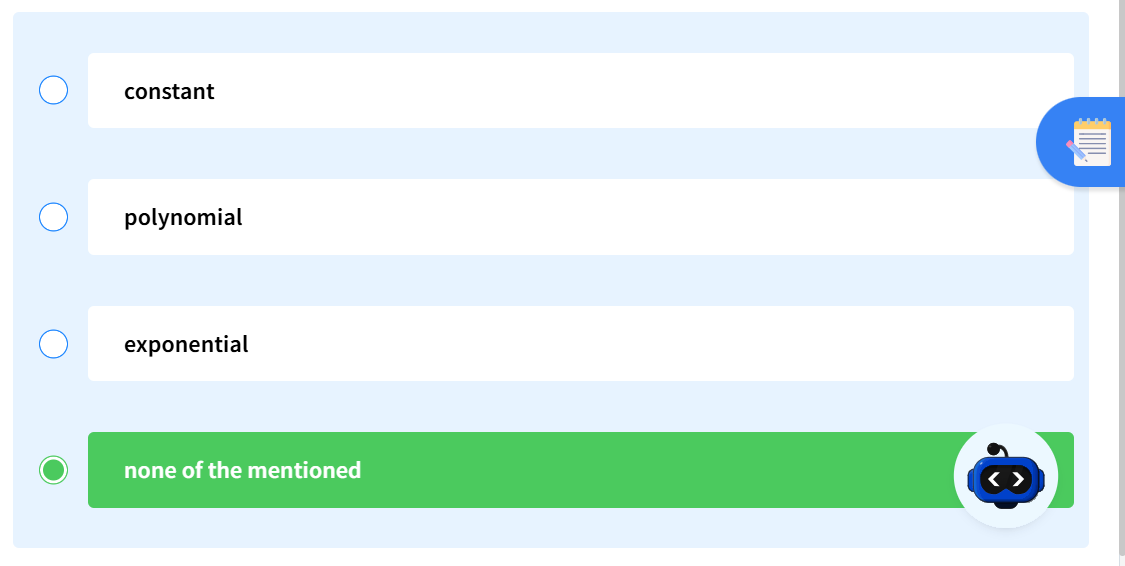
**Q4. Identifying O(1) Complexity**

If an algorithm has a time complexity of O(1), then the complexity of it is ?



**Q5. Identifying O(log n) Complexity**

If for an algorithm time complexity is given by O(log2n) then complexity will:



Answer is logarithmic

**Polynomial:**

An algorithm has **polynomial time complexity** if its running time can be expressed as:

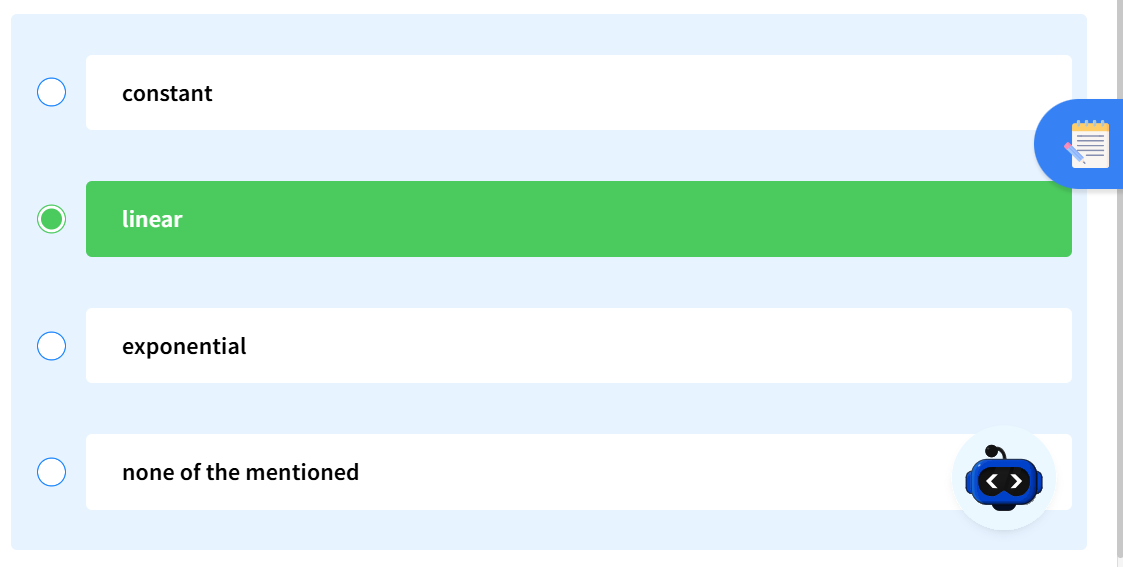
T(n)=O(nk)

where:

* **n** = size of the input
* **k** = some constant (1, 2, 3, …)

**Q6. Algorithm Complexity Classification**

If an algorithm has a time complexity of O(n), then the complexity of it is ?



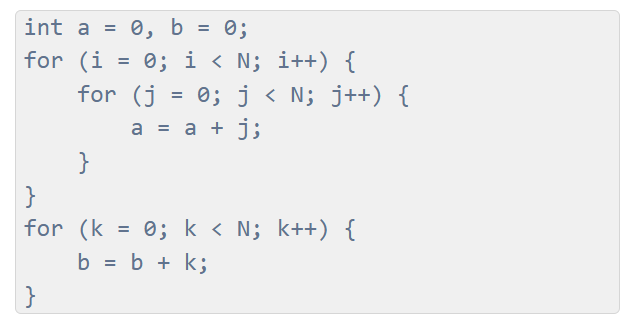
**Q7. Time-Complexity-12**

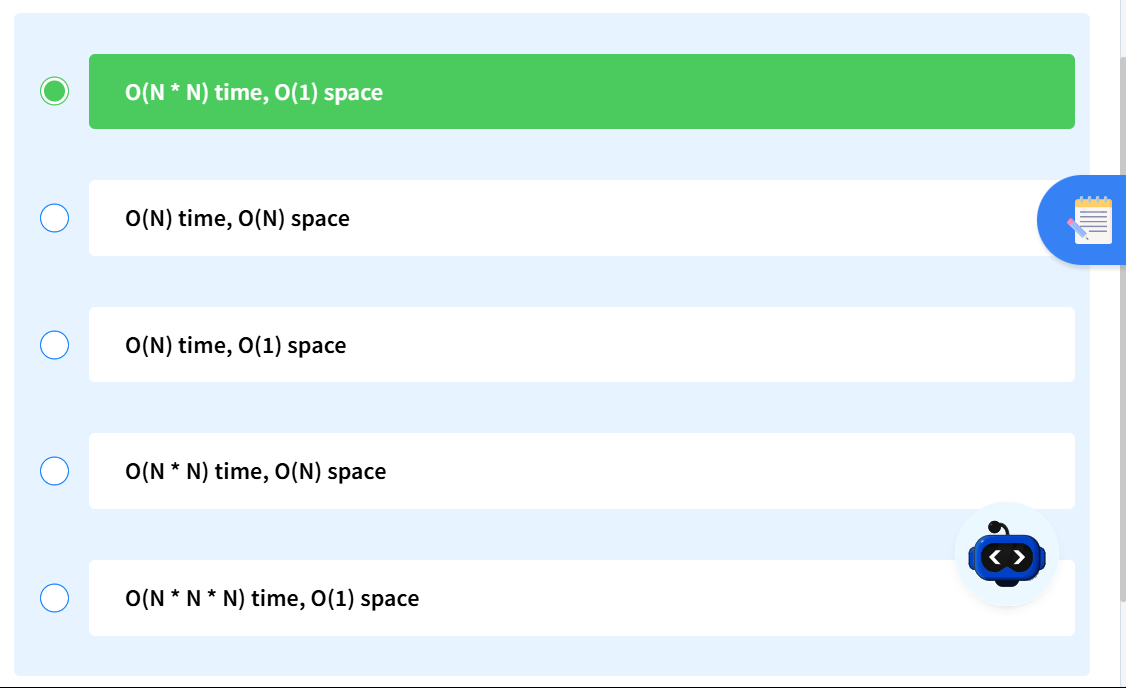
If for an algorithm time complexity is given by O((3/2)^n) then complexity will:



**Q8. NESTED\_CMPL**

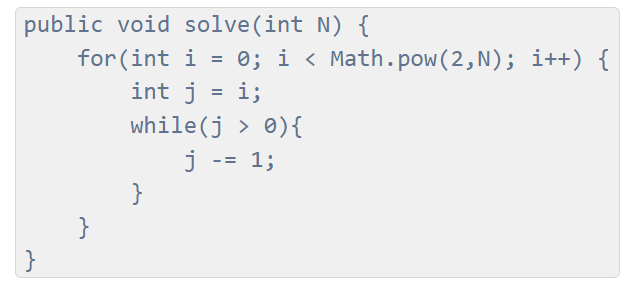
What is the time, space complexity of following code :

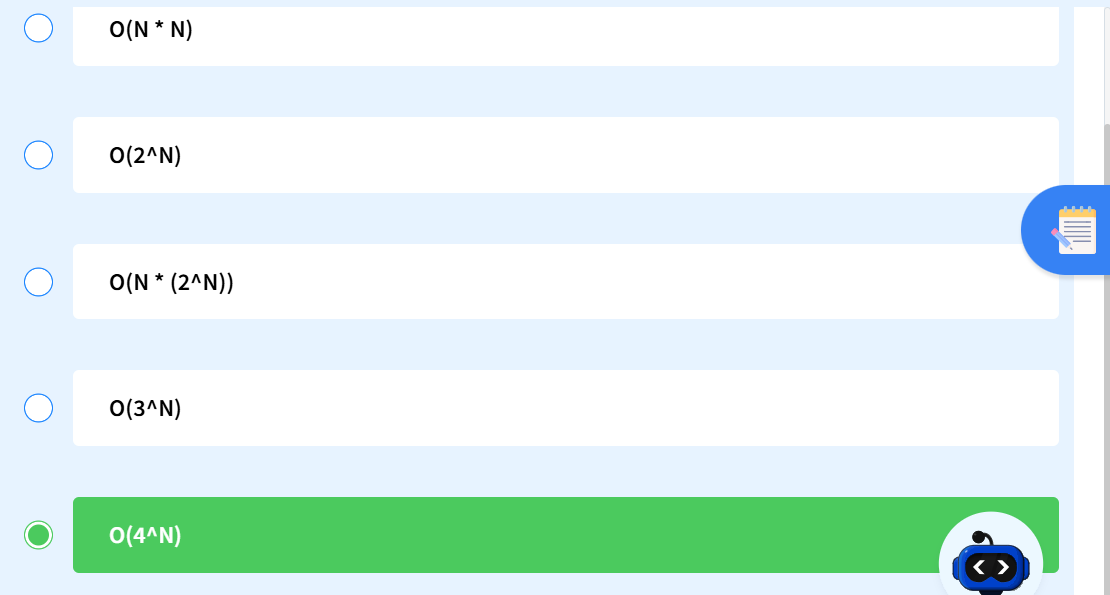




**Q9. Time Complexity - M4**

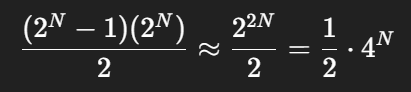
Find the Time Complexity of the following function solve :





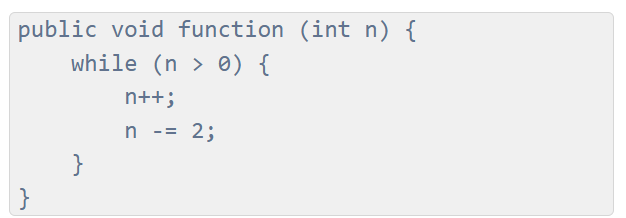
S=((N+1) \* N)/2

0+1+2+3+⋯+(2N−1)



**Q10. Find Time Complexity**

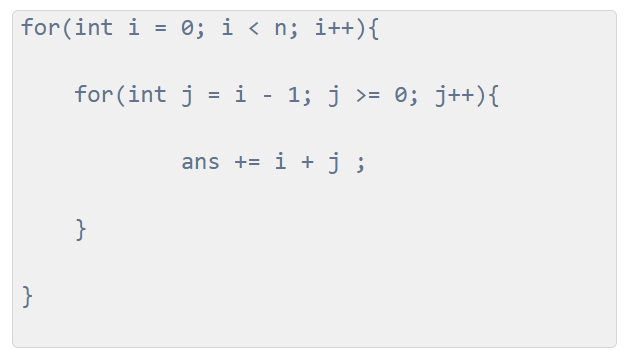
What will be the time complexity of the above function where n is a positive integer?

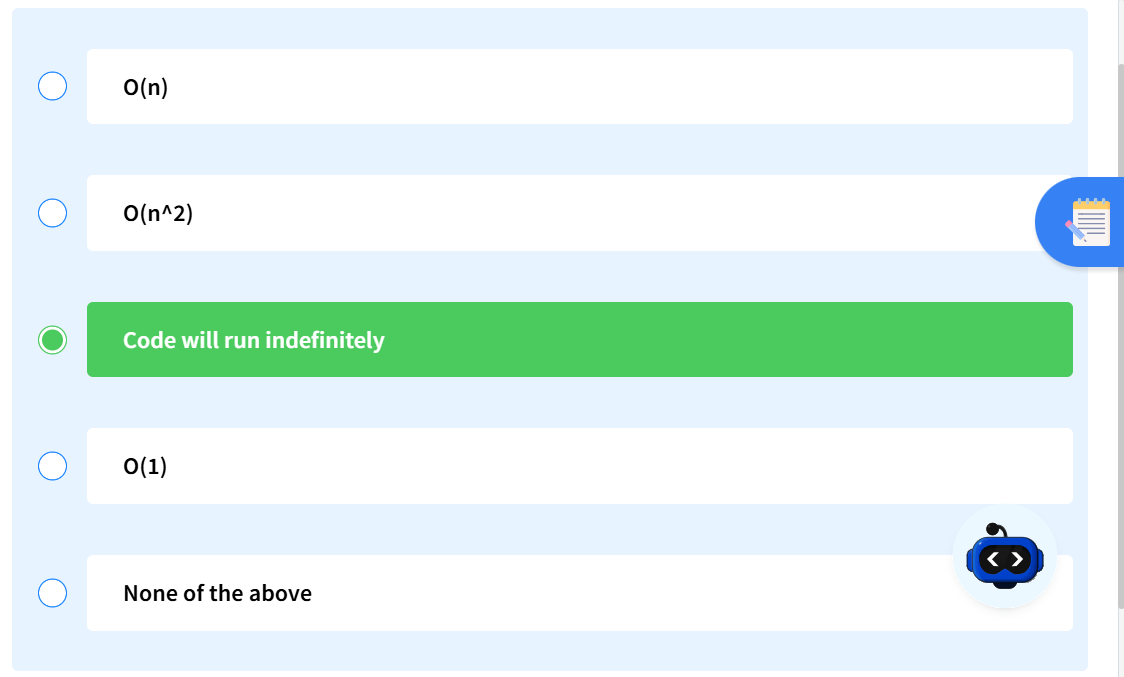




**Q11. Time Complexity-iii**

What is the time complexity of the following code snippet?





**Q12. Time Complexity – 8**

What is the time complexity of the following code snippet?

