



# Practice Set: Loop Complexities





Takeaway from class-

$$O(n) (i = i + c)$$
  
 $O(\log n) (i = 2 * i)$   
 $O(\log \log n) (i = i * i)$ 

#### Question 1: What will be the time complexity of following nested loop?

Consider the following program fragment:

Which one of the following statements about the runtime R(N) is true?

$$\boxed{\mathbf{A}} \ R(N) = \Theta(\log N)$$

$$lacksquare$$
  $R(N) = \Theta(\sqrt{N})$ 

$$\boxed{\mathbf{C}} \ R(N) = \Theta(N)$$



#### Answer:

C Note that the inner loop will be executed  $\sqrt{N}\sqrt{N}$  times.





```
for( int i = n; i > 0; i /= 2 ) {
   for( int j = 1; j < n; j *= 2 ) {
      for( int k = 0; k < n; k += 2 ) {
            ... // constant number of operations
      }
   }
}</pre>
```



Answer:  $\theta(n(logn)^2)$ 





```
for ( int k = n; k >0; k /= 3 ) {
  for ( int i = 0; i < n; i += 2 ) {
     // constant number C of elementary operations
  }
  for ( int j = 2; j < n; j = (j*j)) {
     // constant number C of elementary operations
  }
}</pre>
```







Answer:  $\theta(nlogn)$ 







```
int i=1;
while (i<= n) {
   int j = i;
   while (j > 0) {
        j = j/2;
    }
   i++;
}
```







Answer:  $\theta(nlogn)$ 





```
for (k = 1; k <= n; k += 1)
{
    for (i = 1; i <= n; i *= 3)
    {
        j = i;
        while (j > 1)
        {
            sum += 1;
            j /= 3;
        }
    }
}
```



# Answer: $\theta(n (log n)^2)$

Inner most while loop will run  $\log i$  times.

For i = 1, inner most loop will run log 1 times

For i =3, inner loop will run log 3 times

For  $i = 3^2$ , inner loop will run log  $3^2$  times

For  $i = 3^3$ , inner loop will run log  $3^3$  times

For  $i = 3^k = n$ , inner loop will run log  $3^k$  times (here  $k = \log n$ )

(We will consider outermost for loop after a while, lets focus on inner for loop) Inner for loop will run log1 + log3 + log  $3^2$  + log  $3^3$  + ....+ log  $3^k$  = log  $1.3.3^2.3^3.3...3^k = log <math>3^{1+2+3+...k}$  = log  $3^{k^2} = k^2 = (logn)^2$ 







```
for(int i = 0; i < N*N; i++) {
  for(int j = 0; j < i; j++) {
    //something O(1)
  }
}</pre>
```



Answer:  $\theta(n^4)$ 







```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n * n; j++) {
        for (int k = 0; k < j; k++) {
            sum++;
        }
    }
}</pre>
```





# Answer: $\theta(n^5)$

```
The most inner loop is executed in quadratic time (not constant), hence it should be 0(n) * 0(n^2) * 0(n^2) = 0(n^5).
```

Here are all the costs:

- Most outer loop 0(n)
- The second loop 0(n^2) for each element for the outer loop
- The most inner loop 0(n^2) for each element for the second loop

```
for (int i = 0; i < n; i++) -> runs n times.
  for (int j = 0; j < n * n; j++) -> runs n² times.
  for (int k = 0; k < j; k++) -> runs n² times (k == j == n²)

n * n² * n² = n^5.
```



```
i = 1;
k = 1;
while(k<n){
    k = k+ i;
    i = i + 1;
}</pre>
```



Answer:  $\theta(\sqrt{n})$ 

m	k	i
0	1	1
1	1+1	2
2	1+1+2	3
3	1+1+2+3	4
4	1+1+2+3+4	5

So we see that k is  $1 + \sum_{i=1}^{m} i$ 

This sum is a triangular number, and so:

$$k = 1 + m(m+1)/2$$

And if we fix m to the total number of iterations, then:

$$1 + m(m-1)/2 < n \le 1 + m(m+1)/2$$
.

So *n* is  $O(m^2)$ , and thus *m* is  $O(\sqrt{n})$ .

The number of iterations m is a measure of the complexity, so it is  $O(\sqrt{n})$ .





```
for(int i =1; i<=n;i++)
    {
    for(int j=i; j<=n; j+=i*2);
    }</pre>
```



#### Answer: $\theta(nlogn)$

Inner loop will run  $\frac{n}{2i}$  times.

For i = 1, inner loop will run n/2 times For i = 2, inner loop will run n/(2x2) times For i = 3, inner loop will run n/(2x3) times

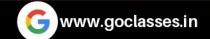
For i =4, inner loop will run n/(2x4) times

For i =n, inner loop will run n/(2xn) times

#### Time Complexity:

$$= \frac{n}{2} + \frac{n}{2 \times 2} + \frac{n}{2 \times 3} + \frac{n}{2 \times 4} + \frac{n}{2 \times 5} + \dots + \frac{n}{2 \times n} =$$

$$= n/2 \left[ 1/2 + 1/3 + 1/4 + \dots \cdot 1/n \right] = n \log n$$







```
p = 0
for( i=1; i<n; i=i*2 ) {
    p++
}

for( j=1; j<p; j=j*2 ) {
    some_statement
}</pre>
```





Answer:  $\theta(logn)$ 

Because they are not independent: The first computes  $p = O(\log n)$  and the second depends on it. The time complexity of the second loop would be  $O(\log p) = O(\log \log n)$ .

However, the first loop is still  $O(\log n)$ , which indeed makes the time complexity of the entire program  $O(\log n + \log \log n) = O(\log n)$ , as you say.







```
for (int j = 2; j < N; j++) {
   for (int k = 2*j; k <= N; k += j) {
      some_statement
   }
}</pre>
```





# Answer: $\theta(nlogn)$

Inner loop will run 
$$\frac{n-2j}{j} = \frac{n}{j} - 2 = \frac{n}{j}$$
 times.

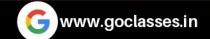
For j = 1, inner loop will run n/1 times For j = 2, inner loop will run n/2 times For j = 3, inner loop will run n/3 times For j = 4, inner loop will run n/4 times

.

For j = n, inner loop will run n/n = 1 time

#### Time Complexity:

$$= \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \frac{n}{5} + \dots + \frac{n}{n}$$
$$= n \left[ 1 + 1/2 + 1/3 + 1/4 + \dots \cdot 1/n \right] = n \log n$$







```
int sum = 0;
for (int i = 1; i < n; i++) {
    for (int j = 0; j < n/i; j++) {
        sum++;
    }
}</pre>
```

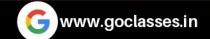
Answer:  $\theta(nlogn)$ 







```
for (i=1;i<=n;i*=2){
  for (j=1;j<=i;j++) {
    // some O(1) operation
  }
}</pre>
```





### Answer: $\theta(n)$

```
For i = 1, inner most loop will run 1 times
```

For i =2, inner loop will run log 2 times

For  $i = 2^2$ , inner loop will run  $2^2$  times

For  $i = 2^3$ , inner loop will run  $2^3$ times

For  $i = 2^k = n$ , inner loop will run  $2^k$  times (here  $k = \log n$ )



```
for (int i = 1; i < n; i*=2)
  for (int j = 0; j < i; j +=2)
  {
    // some contstant time operations
}</pre>
```

Answer:  $\theta(n)$ 









```
for ( int i = 1; i < n*n*n; i *= n ) {
    for ( int j = 0; j < n; j += 2 ) {
        for ( int k = 1; k < n; k *= 3 ) {
            // some contstant time operations
        }
    }
}</pre>
```





Answer:  $\theta(n \log n)$ 









```
for(int i = 0; i < n^3; i++){
    if(i % 3 == 0){
        break;
    }
    else{
        print ":D"
    }
}</pre>
```





Answer:  $\theta(1)$ 







What will be the time complexity of following loop?

```
for(i = 1; i < n; i = i * 2) {
    for(j = 1; j < i; j++) {
        sum++;
    }
}</pre>
```

https://courses.cs.washington.edu/courses/cse332/21au/exams/oldExams/cse332-midterm-12wi-soln.pdf



Answer:  $\theta(n)$ 









# Question 18 What will be the time complexity of following nested loop?

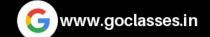
```
int n;
int sum;
for (int i = 1; i < n; i++)
  for (int j = 0; j < i*i; j++)
      if (j % i == 0)
          for (int k = 0; k < j; k++)
              sum++;
```





# Answer: $\theta(n^4)$

The if condition will be true when j is a multiple of i; this happens i times as j goes from 0 to i \* i, so the third for loop runs only i times. The overall complexity is  $O(n^4)$ .







# Answer: $\theta((logn)^2)$

```
for (i = 1; i \le N; i = i*2)
    for (j = 1; j \le i^2; j = j * 2)
                           109 12 + 109 22 + 109 24 + 109 26 + 109 8+
           10912
i = l
           10922
1=2
                                     - - 1692<sup>2</sup>K
           109(212 , 24
i = 2^2
            109(23)2 > 26
\tilde{\iota} = 2^3
                            = 21092 + 41092 + 61092+ 2klog2
         109(2K)2 2K
                              = 2+4+6+ - 2K
                               = 2(1+2+3+--k) = O(k^2)
 2K= N J K=1092
                                                       = \mathcal{Q}((\log n)^2)
```



#### Question 20

```
for ( i=1; i < n; i *= 2 )
for ( j = n; j > 0; j /= 2 )
for ( k = j; k < n; k += 2 ) {
sum += (i + j * k);
}</pre>
```





# Answer: $\theta(n(logn)^2)$



Running time of the inner, middle, and outer loop is proportional to n,  $\log n$ , and  $\log n$ , respectively. Thus the overall Big-Oh complexity is  $O(n(\log n)^2)$ 

More detailed optional analysis gives the same value. Let  $n=2^k$ . Then the outer loop is executed k times, the middle loop is executed k+1 times, and for each value  $j=2^k,2^{k-1},\ldots,2,1$ , the inner loop has different execution times:

j	Inner iterations	
$2^k$	1	
$2^{k-1}$	$(2^k-2^{k-1})^{\frac{1}{2}}$	
$2^{k-2}$	$(2^k - 2^{k-2})^{\frac{7}{2}}$	
- 1		
$ 2^1 $	$(2^k-2^1)\frac{1}{2}$	
$2^0$	$(2^k-2^0)\frac{1}{2}$	

In total, the number of inner/middle steps is

$$1 + k \cdot 2^{k-1} - (1 + 2 + \dots + 2^{k-1}) \frac{1}{2} = 1 + k \cdot 2^{k-1} - (2^k - 1) \frac{1}{2}$$
$$= 1.5 + (k-1) \cdot 2^{k-1} \equiv (\log_2 n - 1) \frac{n}{2}$$
$$= O(n \log n)$$

Thus, the total complexity is  $O(n(\log n)^2)$ .

