Manish Sakariya

Follow 8 Followers About

Time Complexity Examples

Manish Sakariya Apr 29, 2019 · 4 min read

> *Example 1: O(n) Simple Loop*

```java
    public static void main(String[] args) {
        int n=100;
        for(int i=0;i<n;i++) {
            System.out.println(i);
        }

    }
```

> *Example 2: O(n²) Nested Loop*

```java
    public static void main(String[] args) {
        int n=100;
        for(int i=0;i<n;i++) {
            for(int j=0;j<n;j++) {
                System.out.println(i);
            }
        }
    }
```

> *Example 3: O(n²) Consecutive Statements.*

```java
    public static void main(String[] args) {
        int n=100;
        for(int m=0;m<n;m++) {
            System.out.println(m);
        }
        for(int i=0;i<n;i++) {
            for(int j=0;j<n;j++) {
                System.out.println(i);
            }
        }
    }
}
```

> *Example 4: O(n) with if-else loop.*

time complexity of if statement is O(1) and else is O(n). as O(n)>O(1) time complexity of this program is O(n)

```java
    public static void main(String[] args) {
        int n = 100;
        int length = 10;
        if (length < 10) {
            System.out.println("not enougth lenth");
        } else {

            for (int m = 0; m < n; m++) {
                System.out.println(m);
            }
        }
    }
}
```

> *Example 5: O(logn) logarithmic complexity*

```java
    public static void main(String[] args) {
```

```
        }
    }
```

Iteration	Value of M
1	n
2	n/2
3	n/2^2
4	n/2^3
.	.
.	.
k	n/2^k

loop will run k times so that m>1.

m>1 ; putting value of m for as n/2^k

n/2^k >1

2^k > n

k > logn

> *Example 6: O(√n)*

```
    public static void main(String[] args) {
        int n=100;
        int i=1,s=1;
        while(s<=n) {
            i++;
            s=s+i;
            System.out.println("*");
        }
    }

}
```

```
+-----------+--------------------+---------------+
|   1       | 1                  | 1+1           |
|   2       | 2                  | 1+1+2         |
|   3       | 3                  | 1+1+2+3       |
|   4       | 4                  | 1+1+2+3+4     |
|   .       |   .                | .             |
|   .       |   .                | .             |
|   k       |   k                | 1+1+2+3+4+..+k|
+-----------+--------------------+---------------+
```

number of time the loop will run is k

so 1+2+3+..+k>n (that's when loop breaks)

$k(k+1)/2>n$

$k^2+k>n$

$k^2>n$ (ignoring k as $k^2>k$)

$k>\sqrt{n}$

so time complexity of program is $\sqrt{n}$

> *Example 7: O(√n)*

```java
    public static void main(String[] args) {
        int n=100;
        for(int i=1;i*i<=n;i++) {
            System.out.println("*");
        }
    }
```

let's say loop runs k times

so $k^2>n$ (to break the condition)

$k>\sqrt{n}$

Example 8: O(n logn)

```java
    public static void main(String[] args) {
        int n = 100;
        for (int i = n / 2; i <= n; i++) {
            for (int j = 1; j + n / 2 < n; j++) {
                for (int k = 1; k <= n; k = k * 2) {
                    System.out.println("*");
                }
            }
        }
    }

}
```

first loop will run n/2 times

second loop will run n/2 times as j+n/2< n is the condition. so n/2 is already added to the iterator so only n/2 times loop will run.

third loop : 2^k >n (iteration wise 2^0, 2^1,2^2,…2^k)

k>logn

so time complexity is n/2*n/2*logn

so n^2logn is the time complexity.

Example 9: O(nlog^{2}n)

```java
    public static void main(String[] args) {
        int n = 100;
        for (int i = n / 2; i <= n; i++) {
            for (int j = 1; j < n; j=j*2) {
                for (int k = 1; k <= n; k = k * 2) {
                    System.out.println("*");
                }
            }
        }
```

first loop will run n/2 times

second and third loop as per above example will run logn times

so time complexity = n/2*logn*logn = O(nLog^{2}n)

> *Example 10: O(n) with break statement*

```
    public static void main(String[] args) {
        int n = 100;
        for (int i =1 ; i <= n; i++) {
            for (int j = 1; j < n; j++) {
                System.out.println("*");
                break;
            }
        }
    }
```

first loop will run N times

second will break out after every first iteration. so it will run 1 time

so time complexity is O(n) instead of O(n^2)

> *Example 11: O(nLogn)*

```
    public static void main(String[] args) {
        int n = 100;
        for (int i =1 ; i <= n; i++) {
            for (int j = 1; j < n; j=j+i) {
                System.out.println("*");
                break;
            }
        }
    }
```

i=1	i=2	i=3	.....	i=n

j=2	j=3	j=4		
j=3	j=5	j=7		
j=4	j=7	j=10		
.				
.				
n	n/2	n/3	....	1

so n+n/2+n/3+..+1 times total loop will run

n(1+1/2+1/3+1/4) = nlogn

so time complexity is nlogn.

> *Example 12: $O(n^2)$*

```
    public static void main(String[] args) {
        int n = 100;
        for (int i =1 ; i <= n/3; i++) {
            for (int j = 1; j < n; j=j+4) {
                System.out.println("*");
                break;
            }
        }
    }
```

outer loop will run n/3 times

inner loop will run n/4 times

so total time complexity is $(n/3)*(n/4)=n^2/12=O(n^2)$

> *Example 13: $O(\log^2 n)$*

```
    public static void main(String[] args) {
```

```java
            j = n;
            while (j > 0) {
                j = j / 2;
            }
            i = i * 2;
        }

    }
```

outer loop will run *logn* times [2^k>n]

inner loop will run *logn* times [n/2^k>1]

so total time complexity=logn*logn=$\log^2 n$

> *Example 14: O(n^5)*

```java
    public static void main(String[] args) {
        int n = 100;
        for (int i = 0; i < n; i++) {
            for (int j = 1; j < i * i; j++) {
                if (i % j == 0) {
                    for (int k = 0; k < j; k++) {
                        System.out.println("*");
                    }
                }
            }
        }

    }
```

first loop will run n times

second loop will run n^2 times [$1, 2^2, 3^2, \ldots, n^2$]

third loop will run n^2 time [1+2+3+4+…+n}

total time complexity = $n^*n^{2*}n^2$ = $O(n^5)$

```
int count = 0;
for (int i = N; i > 0; i /= 2) {
    for (int j = 0; j < i; j++) {
        count += 1;
    }
}
```

loop will run N+N/2+N/4+..N/2^n

=N(1+1/2+1/4+…+1/2^n)

=N(1+1)

=2N = 0(n)

sum of the series for reference

[S_n= $1/2+1/4+\ldots+1/2^n$

$2S_n=2/2+2/4+2/8+\ldots+2/2^n$

$2S_n=1+1/2+1/4+..+1/2^{n-1}$

$2S_n=1+S_n-1/2^n$

$S_n=1–1/2^n$

$S_n=1$ (when $n=\infty$)]

> *Example 16: O(logn)*

```
int gcd(int n, int m) {
    if (n%m ==0) return m;
```

```
            n = n%m;
            swap(n, m);
        }
        return n;
}
```

Fibonacci : 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,..

lets take gcd of 8 and 5 (it is two Fibonacci numbers)

gcd(8,5) => gcd (5,8%5)=>gcd(5,3)=>gcd(3,5%3)=>

gcd(3,2)==>gcd(2,1)=>gcd(1,1)=>gcd(1,0)

look at the second argument in the gcd function.

it is 5,3,2,1,1,0 in recursive calls.

it is following Fibonacci series pattern.

why we have taken gcd of Fibonacci numbers?

because it takes maximum number of steps.

so time complexity of gcd algo==time complexity of fib series

hence time complexity is O(logn)

Algorithms Time Complexity Examples