Wahat is time complexity?

Time complexity tell us how long an algorithm takes to run as the size of the input grows. It helps us know if an algorithm is fast on slow.

Big O Notation:

We use big O notation to describe time complexity. It shows the worst-case scenario for an algorithm.

[how much time it will take if the input is at its large size]

- Here are some common type of time complexity:

1. O(1) – constant time → The time stays the same no matter how big the input is.
   Ex: arr[0];
2. O(log n) – Logarithmic time → The time increases slowly as the input size grows.
   Ex: Binary Search.
3. O(n) – Linear time → The time is directly in proportional increase to the input size.
   [Single Loop]
   Ex: Looping through an array to find specific item. Linear Search.

4. $O(n^2)$ - Quadratic time → Ex: Nested Loops

5. $O(2^n)$ — Exponential Time → time doubles with each additional input: its very slow for large inputs.

Ex: Recursive Algorithm like Fibonacci sequence

- $O(1)$ → super fast
- $O(n)$ → Time ↑ input size ↑
- $O(\log n)$ → Time increase slowly, even for big input
- $O(n^2)$ → Time increase very fast, Aboid this for a large input.

Time Complexity:

When,

- Assingment Operation. [a=10, b=20]
- Comparison Operation. [a>b]
- Matmatical Operation. [a+b]

} $O(1)$

- Function Call.
- Function এর ভেতরের Operation:

[Note: $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^2 \log n) < O(n^3) < O(2^n)$]

Ex: 01

```
#include <iostream>
using namespace std;
int main() {
    int a;          // 1
    cin>>a;         // 1
    cout<<a<<endl;  // 1
    return 0;
}
```

Output:
input : 10
output : 10

Time complexity = O(1+1+1)

General ↑ ⇒ $O(3) \cong O(1)$

Ex: 02

```
#include <iostream>
using namespace std;
int main() {
    // 1   11   10
    for(int i=0; i<=10; i++) {
        cout<<i<<endl;
    }
    return 0;
}
```

Here, Exact time complexity:
⇒ 1+11+10+10 = 32 times.

$O(32) \sim O(1)$

Ex: 03

```
#include <iostream>
using namespace std;
int main() {
    int n; // 1
    cin>>n; // 1
    for(int i=1; i<=n; i++) {
        cout<<i<<endl; // n
    }
    return 0;
}
```

Here,

No. Operation:

$1+1+1+n+1+n+n = 3n+4$

$\Rightarrow \boxed{O(3n+4) \sim O(n)}$

Ex: 04

```
#include <iostream>
using namespace std;
int main() {
    int i, n, even[101]; // 1
    for(i=0; i<101; i++) {
        even[i] = 0; //101
    }
    for(i=0; i<101; i+=2){
        even[i] = 1; // 51
    }
    scanf("%d", &n); // 1
```

1st Loop:

$n = 101$

2nd Loop:

Let $m = 51$

∴ time complexity is $n > m$ so,

$\boxed{O(n)}$ in General.

```
if(even[n]){
    printf ("Even");
}else{
    printf ("Odd");
}
return 0;
}
```

1

Ex:05

```
#include <iostream>
Using namespace std;
int main() {

    for (int i=01; i<=10; i++){
        for (int j=01; j<=10; j++){
            cout<< i <<endl;
        }
    }
    return 0;
}
```

Here, Exact steps:

Outer Loop:

for outer Loop

i=1 the inner Loop works = 1+11+10+10 = 32 times

then for i=10

⇒ 32×10 = 320 times

∴ Now, outer Loop

1+11+10 = 22 time

So, total Exact steps = 22+320 = 342 times;

But in General:

Outer: 10 and inner: 10

so; 10×10 = 100 times

$O(10^2)$

Ex: 6

```
#include <iostream>
using namespace std;
int main () {
    int n, m;
    cin >> n >> m;

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

        for (int j=1; j<=m; j++){
            cout << i << " " << j << endl;
        }
    }
```

Time complexity:

O(n*m)

Ex: 07

```
#include <iostream>
using namespace std;
int main () {
    int n, m;
    cin >> n >> m;
    for (int i=1; i<=n; i++){
        for (int j=1; j<=m; j++){
            for (int k=1; k<=n; k++){
                cout << i << " " << j << endl;
            }
        }
    }
```

Here,

O(n*m*n)

Time complexity.

```
    return 0;
}
```

Ex: 08

```
#include<iostream>
using namespace std;
int main() {
    int n, m;
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++) {
            cout<<i<<" "<<j<<endl;
        }
        for(int k=1; k<=n; k++){
            cout<<i<<" "<<j<<endl;
        }
    }
    return 0;
}
```

Here,

Time complexity
$= O(n*(m+n))$

But, if value of
$n <= 100$
$m <= 50$
then time complexity is $O(n^2)$

Ex: 9

```
#include<iostream>
using namespace std;
int main() {
    int n,m,o;
    cin>>n>>m>>o;
    for(int i=1; i<=n; i++){
        cout<<i<<endl;
    }
    for (int j=1; j<=m; j++){
        cout<<j<<endl;
    }
    for (int k=1; k<=o; k++){
        cout<<k<<endl;
    }
    return 0;
}
```

Time complexity:
O(n+m+o)

[Note: যদি n, m, o এর value দেয়া থাকে তাহলে যার value বেশি সেটা total, time complexity.
example:
Let n=100, m=10, o=50;
O(n)]

Ex: 10

```
#include <iostream>
using name space std;
int main () {
    int n1, n2, result;
    n1 = 10;
    n2 = 20;
    result = n1 + n2;
    return 0;
}
```

Here,

3 টা assingment Op:

```
n1 = 10;
n2 = 20;
result =
```

and

1 Addition Op:

```
n1+n2;
```

total = 4 operation

O(4) ✗

Time Coplexity is: O(1) ✓

[Note: Operation এর সাথে input এর কোনো সম্পর্ক নাই, যদি n এর value 10 হয় 4টা operation হবে even n এর value যদি 10000 ও হয় same 4টা Op. হবে; input এর মান যাই হওটেকনা কেন এই 4টা op. হবে so constant O(1).]

Ex: 11

```
#include <iostream>
using namespace std;
int main() {
    int n, result;
    cin>>n;
    result = n * (n+1) / 2;
    cout<<result<<endl;
    return 0;
}
```

No. of Operation:

(=) assingment op = 1 টা

(*) Mathmatical op. 3 টা

Total op = 4 টা

[Constant]

Time Complexity:

O(1)

Ex: 12

```
#include <iostream>
using namspace std;
int main() {
    int n, result;
    cin>>n;
    result=0;
    for(int i=1; i<=n; i++){
        result=result+i;
    }
    cout<<result<<endl;
    return 0;
}
```

n এর মান এর সাপেক্ষে Linearly বাড়তেছে. ∝ সমানুপাতিক আকারে.

n এর value যত টা যদি 10 হয় তাহলে 10 বার Loop চলবে এবং 10 বার তার ভিতরের operation গুলো হবে,

for this: Time complexity

O(n).

Ex: 13 Binary Search

```
#include <iostream>
using namespace std;
int main(){
    int arr[]={1,2,3,4,5,6,7,8,9,10};
    int left=0; int x=7;
    int right=9;
    while(left<=right){
        int mid=(left+right)/2;
        if(arr[mid]==n){
            cout<<"Found"<<endl;
        }
        if(arr[mid]<x){
            left=mid+1;
        }else{
            right=mid-1;
        }
    }
    return 0;
}
```

for worst case: time compexity $O(\log n)$

No. of iteration: (4)

Min. NO. comparision → Best Case

Max. No comparision → Worst Case

(i) Best Case:

Let, Search element = 5

$mid = \frac{L+R}{2} = \frac{0+9}{2} = 4$

mid = 4

if search element is

Here, 1 comparision needed.

so this is best case

time complexity is O(1).

(ii) Worst Case:

Let, Search element = 10

$mid = \frac{0+9}{2} = 4 \rightarrow 1$

$s \neq mid$

$mid = \frac{5+9}{2} = 67 \rightarrow 2$

$s \neq mid$

$mid = \frac{8+9}{2} = 8 \rightarrow 3$

$s \neq mid$

$mid = \frac{9+9}{2} = 9 \rightarrow 4$

$s \neq mid$

$mid = \frac{8+9}{2} =$

$mid = \frac{9+9}{2} = 9$

$mid = s = mid$

Here, in worst case

NO. of comparision 4

Let, annay length, $n = 10$.

(i) 1st ite $= \frac{n}{2}$

(ii) 2nd i $= \frac{\frac{n}{2}}{2}$

(iii) 3rd i $= \frac{n}{2^3}$

(iv) 4th i $= \frac{n}{2^4}$

$\vdots$

$= \frac{n}{2^k} = 1$

(ii) $\Rightarrow n = 2^k$

$\Rightarrow \log_2 n = k \log_2 2$

$\Rightarrow \boxed{k = \log_2 n}$

$k = \log_2 10$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$] ceiling নিতে হবে

(4)

(iii) Average Case:

Time complexity $= O(\log n)$

Ex: 14 Bubble Sort:

0	1	2	3	4
22	14	12	18	9

$n = 5$ ← size

No. of iteration = 5 − 1 = 4

[number of iteration = n − 1

And NO. of. comparision = n − 1 − i]

Algorithm:

iteration-1 i = 0

22 14 12 18 9

14 22 12 18 9

14 12 22 18 9

14 12 18 22 9

14 12 18 9 **22**

iteration-2 i = 1

14 12 18 9 22

12 14 18 9 22

12 14 18 9 22

12 14 9 **18 22**

iteration-3

12 14 9 18 22

12 9 14 18 22

iteration-3 i = 2

12 14 9 18 22

12 14 9 18 22

12 9 **14 18 22**

iteration-4 i = 3

12 9 14 18 22

9 12 14 18 22

No. of Comparision for : $n-1-i$

$i=0$, if $n=5$

$\Rightarrow 5-1-0 = 4$ comparison

$i=1$

$= 5-1-1 = 3$ compairison

$i=2$

$= 5-1-2 = 2$ compairison

$i=3$

$= 5-1-3 = 1$ Compairison

Example: 14 Bubble Sort

```
#include <iostream>
using namespace std;
int main () {
    int ann[5] = {22, 14, 12, 18, 9};
    int n=5;

    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-1-i; j++){
            int temp
            if (ann[j] > ann[j+1]){
              int temp = ann[j];
              ann[j] = ann[j+1];
              ann[j+1] = temp;
            }
        }
    }
    for (int i=0; i<n; i++){
        cout<<i<<endl;
    }
    return 0;
}
```

Here,

No. of iteration = 4

and No. of compairison

$\Rightarrow 4+3+2+1$

$\Rightarrow 1+2+3+\dots n-1$

$\Rightarrow \frac{n(n-1)}{2}$

$\Rightarrow \frac{n^2-n}{2}$

Worst case: $O(n^2)$ time complexity.

best case: $O(n)$

Example: 15

ⓘ

```
Void main()
{
  int i, j, n;
  scanf("%d", &n);
  for(i=n; i>=1; i/=2){
    for(j=n/2; j>=1; j--){
      printf("Be simple\n");
    }
  }
  for(i=1; i*i<=n; i++){
    printf("think high.\n");
  }
}
```

For Loop: 1 (outer)

1st iteration: $i = n$

2nd " : $i = \frac{n}{2}$

3rd " : $i = \frac{n}{4}$

so on...

until $i = 1$

$\log_2(n)$

$\therefore O(\log n)$

For Loop: 2 (inner)

Loop run, $\frac{n}{2}$ times

We can write,

$\therefore O(n)$

For Loop: 3

Loop run, $i*i <= n$ ~~means~~

means, $n^{\frac{1}{2}}$ times.

$\therefore O(n^{\frac{1}{2}})$

Here, Loop 1 and 2

$O(n \log n)$

Overall Time complexity is, $O(n \log n)$

becaues it is faster.

(ii)

```
Void fun(int n) {
    int i, j, k;
    for(i=1; i<=n; i++){
        for(j=1; j<=i; j++){
        }
    }
    for(k=1; k<=100; k++){
        printf("Nice Day!");
    }
}
```

(iii)

Loop : 1 (outer)

$i <= n$, so no. of iteration is n time.

$O(n)$

Loop : 2 (inner)

Here, $j = i$ it means, no. of iteration increase as i increase.

$\therefore O(n)$

Loop 1 and 2 number of ~~iteration~~ time complexity is $\therefore O(n^2)$

Loop : 3

Loop is running $k = 100$ time,

$O(1)$

$\therefore$ Total Time complexity is $O(n^2) > O(1)$

$\therefore O(n^2)$

(iii)

```
void fun(){
  int i, j, k;
  for (i=n/2; i<=n; i++){
    for(j=1; j<=n/2; j++){
      for(k=1; k<=n; k*=2){
      }
    }
  }
```

Loop: 1

Loop run, $i = \frac{n}{2}$ times

$O\left(\frac{n}{2}\right) \sim O(n)$

Loop: 2

Loop run, $j = \frac{n}{2}$ times

$O\left(\frac{n}{2}\right) \sim O(n)$

Loop: 3 if $n=8$

Loop run, ~~k=~~

1st iteration: $k=1$

2nd " : $k=2$

3rd " : $k=4$

4th " : $k=8$

it means, $\frac{n}{2}$

$\frac{n}{4}$

$\therefore O(\log_2 n) \sim O(\log n)$

$\therefore$ Total time complexity is $= O(n) \times O(n) \times O(\log n)$

$= \boxed{O(n^2 \log n)}$

function (i), (ii), (iii) out of three function; the fasten one is,

→ $O(n\log n) < O(n^2) < O(n^2\log n)$

Why:

The logarithmic growth is $O(n\log n)$ ensures that it grows more slowly than quardratic growth in $O(n^2)$.

$O(n\log n)$: is fasten of the three because it grows slowen that both $O(n^2)$ and $O(n^2\log n)$.

$O(n^2)$: is ~~slowen~~ fasten than $O(n^2\log n)$ because it grows slowen than $O(n^2\log n)$.

$O(n^2\log n)$: is the slowest of the three. it has quardic term n^2 combine with a logarithmic temm $\log n$.