Time complexity

It tells us how long an algo takes to run as the input increases.

Ways to determine
- → Experimental Analysis
- → Assymptotic Notation

Experimental Analysis

measure actual runtime using system clock.

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

        long start = System.currentTimeMillis();
        for(int i=0; i<100000; i++){
            // System.out.println("Hello Akarsh!");
        }
        long end = System.currentTimeMillis();

        System.out.println(end-start);
    }
}
```

Assymptotic Notation

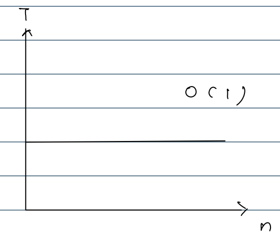
Dependent on user input, not on system.

" No. of times a repetitive statement is running ".

Asymptotic
- → Worst case (O)
- → Best case (Ω)
- → Average case (Θ)

Time Complexity → It describes the growth rate of an algo's running times, but not actual times in seconds.

Constant time complexity $O(1)$

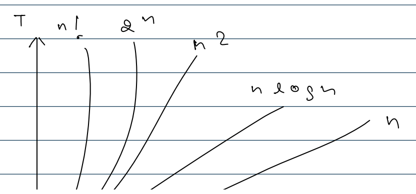


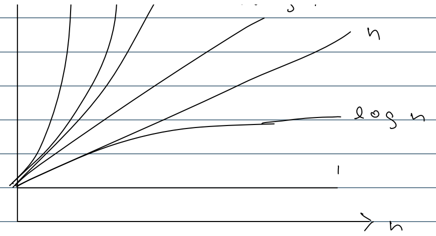
mathematical expressions, logical operators, relations operators, bitwise operators, variable declaration & assignment.

Example

$f(n) = 5n^2 + 2n^5 + 3n + 2 \Rightarrow \text{Time} = O(n^5)$

$f(n) = 2^n + 3n^2 + 2 \Rightarrow \text{Time} = O(2^n)$






```java
System.out.println("Hello Akarsh");
System.out.println("Hello Akarsh");
System.out.println("Hello Akarsh");
System.out.println("Hello Akarsh");
System.out.println("Hello Akarsh");
System.out.println("Hello Akarsh");
```

Time => O(6) => O(1)

Best

worst

```java
// Linear Search
public static int linearSearch(int[] arr, int item) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == item) {
            return i;
        }
    }
    return -1;
}
```

Worst => O(n)

Best => O(1)

```java
// Maximum value in an array
public static int maxValue2(int[] arr) {
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```

Best, worst => O(n)

```java
// Reverse printing an array
public static void reversePrint(int[] arr) {
    for (int i = arr.length - 1; i >= 0; i--) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();
}
```

Best, worst => O(n)

```java
// Reversing an array
public static void reverseArray(int[] arr) {
    int i = 0;
    int j = arr.length-1;
    while (i < j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        i++;
        j--;
    }
}
```

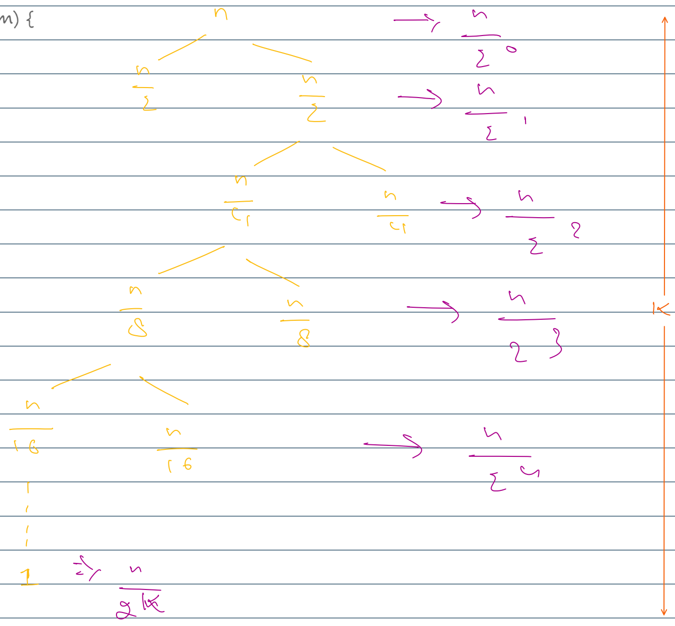
8	1	5	7	2	3	4	9

=> $\frac{n}{2}$ times

Best, worst => O(n)

```java
// Binary Search
```

```java
// Binary Search
public static int binarySearch(int[] arr, int item) {
    int n = arr.length;
    int low = 0;
    int high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == item) {
            return mid;
        } else if (arr[mid] > item) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return -1;
}
```



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

$$n = 2^k$$

$$\log_2 n = \log_2 2^k$$

$$\log_2 n = k \cdot \log_2 2$$

$$\log_2 n = k \cdot 1$$

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

Time complexity

$$\boxed{\begin{aligned} \log_a a &= 1 \\ \log_b a^k &= k \cdot \log_b a \end{aligned}}$$

Log mathematical formulas

```java
int n = 566789;
int i = 0;

while (i < n) {
    // System.out.println("Hello Akarsh");
    i++;
}
```

O(n)

```java
i = 1;
while (i <= n) {
    // System.out.println("Hello Akarsh");
    i *= 2;
}
```

k

1 2 4 8 16 32 . . . . n

2^0 2^1 2^2 2^3 2^4 2^5 2^k

$$2^k <= n \Rightarrow 2^k = n$$

$$\log_2 2^k = \log_2 n$$

$$k \cdot \log_2 2 = \log_2 n$$

$$k \cdot 1 = \log_2 n$$

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

Time

```java
while (n > 0) {
    // System.out.println("Hello Akarsh");
    n /= 2;
}
```

$n = 1024$

$\xleftrightarrow{\quad k \quad}$

1024 512 256 128 64 32 ---- 1

2^{10} 2^9 2^8 2^7 2^6 2^5 2^0

n $\frac{n}{2}$ $\frac{n}{4}$ $\frac{n}{8}$ $\frac{n}{16}$

$\frac{n}{2^0}$ $\frac{n}{2^1}$ $\frac{n}{2^2}$ $\frac{n}{2^3}$ $\frac{n}{2^4}$ ---- $\frac{n}{2^{10}}$

$\boxed{\frac{n}{2^k} = 1} \Rightarrow k = \log_2 n$

```
while (i <= n) {
    // System.out.println("Hello Akarsh");
    i += 2;
    i += 3;
}
```

$i:0$ $i:5$ $i:10$ $i:15$ $i:20$ $i:25$ $i:30$

$n: 30, k: 6$

Time $\Rightarrow O\left(\frac{n}{5}\right) = O(n)$

```
while (i <= n) {
    // System.out.println("Hello Akarsh");
    i *= 2;
    i *= 3;
}
```

$\xleftrightarrow{\quad k \quad}$

1 6 36 216 ----- ?

6^0 6^1 6^2 6^3 ----- 6^k

$6^k <= n \Rightarrow 6^k = n$

$\log_6 6^k = \log_6 n$

$k \cdot \log_6 6 = \log_6 n$

$k \cdot 1 = \log_6 n$

$\boxed{k = \log_6 n}$ → Time

```
while (n > 0) {
    // System.out.println("Hello Akarsh");
    n /= 2;
    n /= 3;
}
```

n $\frac{n}{6}$ $\frac{n}{36}$ $\frac{n}{216}$ ---- ?

$\frac{n}{6^0}$ $\frac{n}{6^1}$ $\frac{n}{6^2}$ $\frac{n}{6^3}$ ---- $\frac{n}{6^k}$

Time

$\frac{n}{6^k} > 0 \Rightarrow \frac{n}{6^k} = 1 \Rightarrow n = 6^k \Rightarrow \boxed{k = \log_6 n}$

```java
int k = 2;
while (i <= n) {
    // System.out.println("Hello Akarsh");
    i += k;
}
```

while (i <= n) {
sop
i = i + 2 ;
}

$\Rightarrow O(\frac{n}{2})$

$\Downarrow$

$O(n)$

$O(\frac{n}{k})$

$\Downarrow$

$O(n)$

```java
while (i <= n) {
    // System.out.println("Hello Akarsh");
    i *= k;
}
```

$\Rightarrow O(\log_k n)$

$\boxed{n = 5}$

```java
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        // System.out.println("Hello Akarsh");
    }
}
```

i = 1   i = 2   i = 3   i = 4   — — — i = n

n   n   n   n   n

$\Rightarrow$ n is n times

$\Rightarrow 5 + 5 + 5 + 5 + 5 = 5 \cdot 5 = 5^2$

$\Rightarrow n + n + ---- n$ times $= n \cdot n = n^2$

Time $\Rightarrow O(n^2)$

Nested loops → Dependent
Nested loops → Independent

```java
for (i = 1; i * i <= n; i++) {
    // System.out.println("Hello Akarsh");
}
```

$i * i < n$

$i^2 < n \Rightarrow i < n^{1/2}$

$i < \sqrt{n}$

$\Downarrow$

$\boxed{\text{Time} = \sqrt{n}}$

```java
for (i = 1; i <= n; i++) {
    for (int j = 1; j <= i * i; j++) {
        for (k = 1; k <= n / 2; k++) {
            // System.out.println("Hello Akarsh");
        }
```

```
    }
}
```

$i = 1 \quad i = 2 \quad i = 3 \quad i = 4 \quad - - - -$

$$1 * \frac{n}{2} + 2^2 * \frac{n}{2} + 3^2 \times \frac{n}{2} + 4^2 \times \frac{n}{2}$$

$$\frac{n}{2}\left[1^2 + 2^2 + 3^2 + 4^2 - - - - n\right]$$

$$\frac{n}{2}\left[\frac{n(n+1)(2n+1)}{6}\right]$$

$$\frac{n}{2} * n^3 = n^4 \Rightarrow Time = O(n^4)$$

$$n.(n+1).(2n+1) = (n^2 + n).(2n+1)$$
$$= 2n^3 + n^2 + 2n^2 + n$$
$$= 2n^3 + 3n^2 + n$$

```
for (i = 1; i <= n; i *= 2) {
    // System.out.println("Hello Akarsh");
}
```

$Time \Rightarrow O(\log_2 n)$

```
for (i = n / 2; i <= n; i++) {
    for (int j = 1; j <= n / 2; j++) {
        for (k = 1; k <= n; k = k * 2) {
            // System.out.println("Hello Akarsh");
        }
    }
}
```

$\rightarrow n/2$

$\rightarrow n/2$

$\rightarrow \log_2 n$

$$\frac{n}{2} * \frac{n}{2} * \log_2 n$$
$$\Downarrow$$
$$\frac{n^2}{4} * \log_2 n$$
$$\boxed{n^2 . \log_2 n}$$

```
for (i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j += i) {
        // System.out.println("Hello Akarsh");
    }
}
```

```
for( int j = 1 ; j<=n ; j = j + 5 ) {
    SOP( );
}
```

$i = 1 \quad i = 2 \quad i = 3 \quad i = 4 \quad i = 5 \quad - - - - i = n$

$n \quad \frac{n}{2} \quad \frac{n}{3} \quad \frac{n}{4} \quad \frac{n}{5} \quad - - - - n$

$$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} - - - -$$

$$n\left[\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + - - - + n\right]$$

$n . \log n$

Space complexity

Extra space we use in program.

It refers to how much memory (RAM) an algo uses w.r.t input size.

Example

```
int[] arr = {1, 2, 3, 4};
for(int i=0; i< arr.length; i++) {
        SOP(arr[i]);
}
```

=> $O(1)$

```
int[] a = {1, 2, 3};
int[] b = new int[a.length];
for(int i=0; i< a.length; i++) {
        b[i] = a[i];
}
```

=> $O(n)$

How to decide, if my solⁿ will work?

Time → 1s ⇒ 10^8 instructions/statements/operations

$n = 10^4$ => $n^2 = 10^8$ → ✓
=> $n^3 = 10^{12}$ → ✗

$n = 10^5$ => $n^2 = 10^{10}$ → ✗
$\log n$ = ✓
$n \cdot \log n$ = ✓

$n = 100$ => $n^4 = 10^8$ → ✓
$n^3 = 10^6$ → ✓
$n^5 = 10^{10}$ → ✗