

Time & Space Complexity

Ideal: \rightarrow Time \downarrow and Space \downarrow (most optimized)

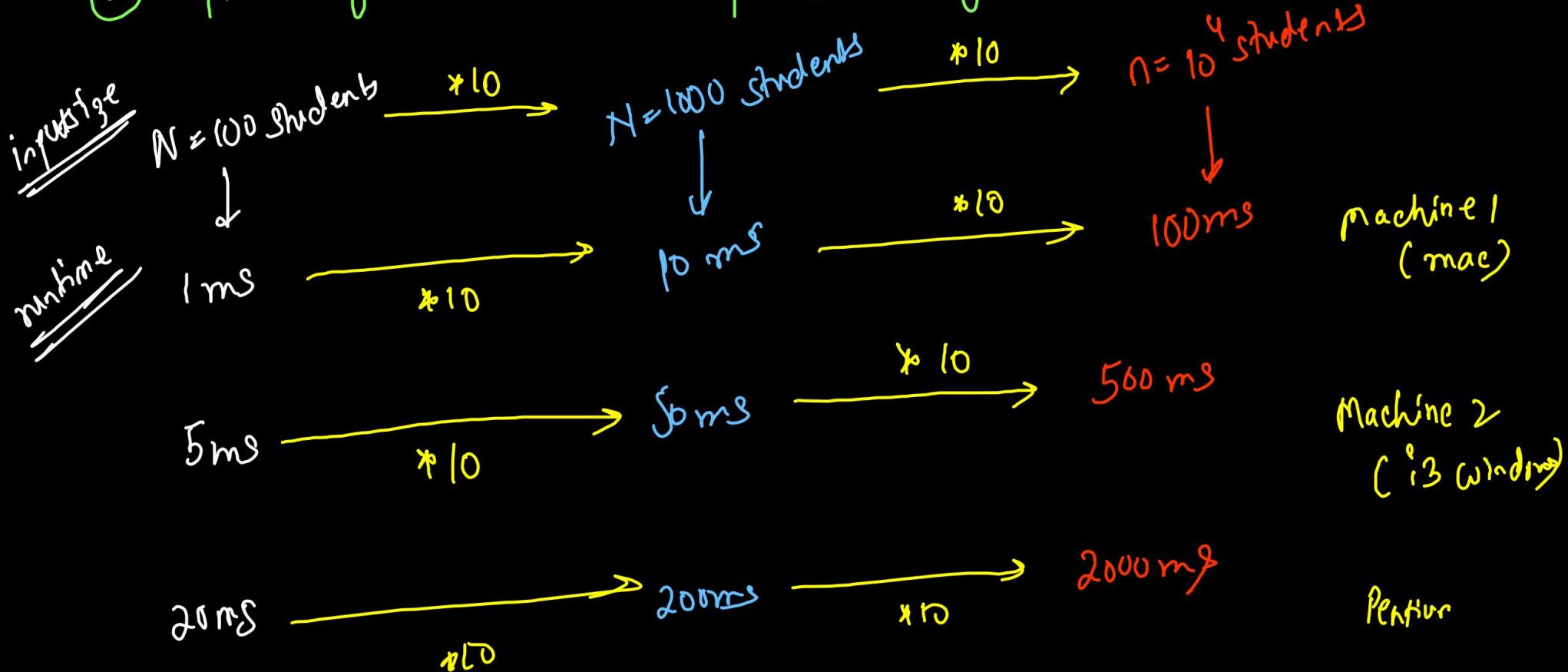
Time \uparrow Space \downarrow vs Time \downarrow Space \uparrow

Algo 1 Algo 2

Time Complexity

① Runtime \rightarrow Machine Dependent
eg. Leetcode \rightarrow Similar machines

② Growth of runtime over input size (Asymptotic Analysis)



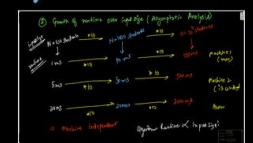
\Rightarrow Machine Independent

Algorithm Runtime \propto Input Size?

Time & Space Complexity	
Time & Space	Space & Time (not always)
Time Complexity	Time & Space
Time Complexity	Time & Space

Growth of runtime $\Rightarrow T(n) = 5n^2 + 3n + 6$

The graph illustrates the growth of runtime $T(n)$ as a function of input size n . The solid blue curve represents the actual runtime. The red dashed line represents the worst-case upper bound, labeled as $\text{bigO}(n) = n^2$. The yellow dashed line represents the best-case lower bound, labeled as $\text{omega}(n^2) = \Omega(n^2)$. The green dashed line represents the average/tight bound, labeled as $\text{theta}(n^2)$.



$$T(n) = 5n^2 + 3n + 6$$

ignore smaller terms

$$O(7n^2) \approx O(n^2)$$

(ignore constant terms)

$$7n^2 > 5n^2 + 3n + 6$$

$n=1$

$7 \not> 5+3+6$ ignore smaller terms

$n=2$

$7 \times 4 = 28 \not> 5 \times 4 + 3 \times 2 + 6$ ignore smaller terms

$$n=3$$

$$7 \times 3^2 > 5 \times 3^2 + 3 \times 3 + 6$$

$$= 63 \qquad \qquad = 60$$

$n \geq 3 \Rightarrow \text{Good!}$



Runtime $T(n)$

$$Q \backslash \begin{array}{c} n^3 + 100n + 200 \\ \swarrow \quad \searrow \end{array}$$

Worst Case
 $\Rightarrow O(n^3) = \text{cubic}$

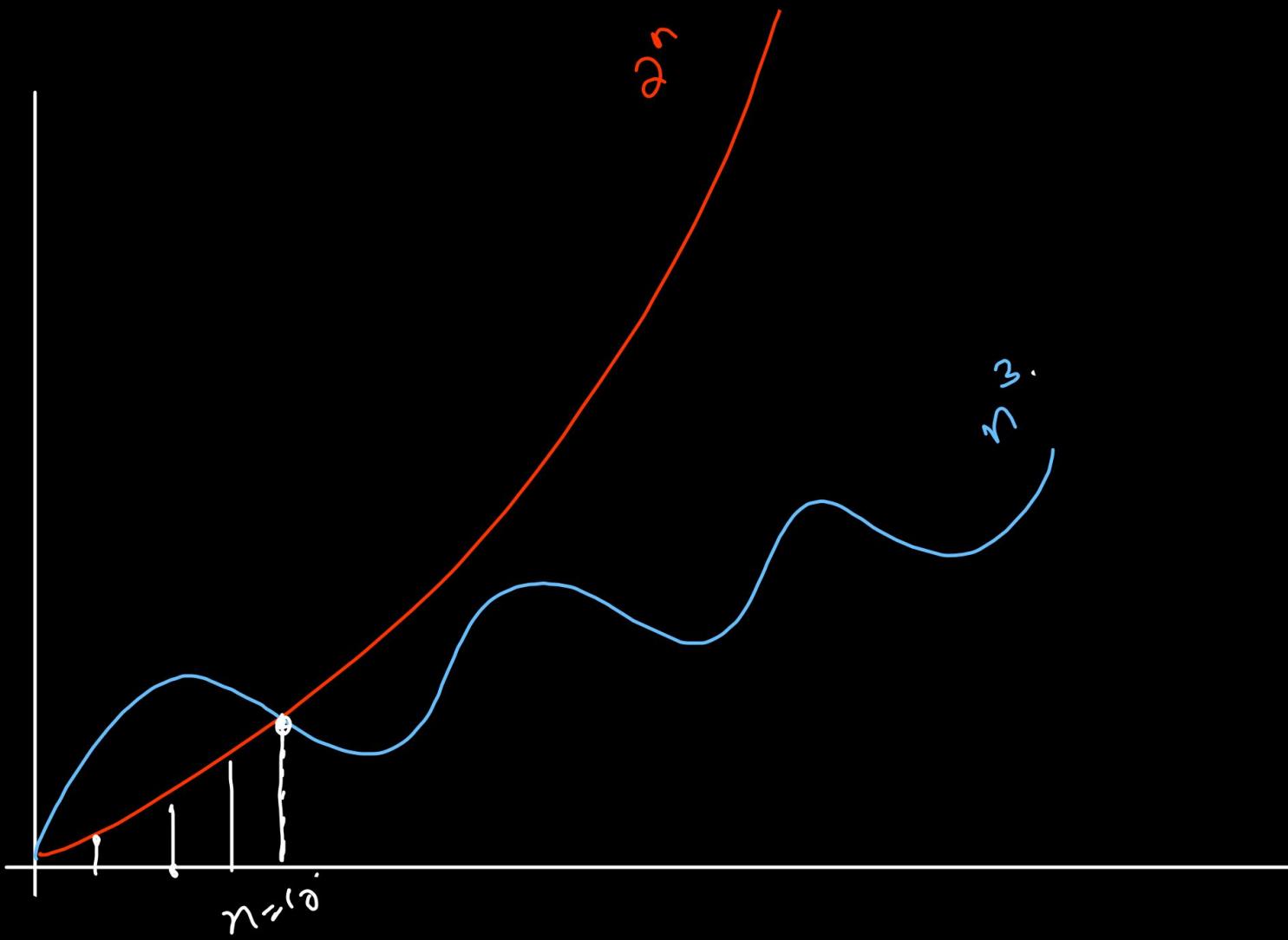
$$Q \backslash \begin{array}{c} 100n + 200n + 300n \\ = 600n \\ \searrow \end{array} \Rightarrow O(n) = \text{linear}$$

$$Q \backslash \begin{array}{c} 200 + 100 + 400 \\ = 700 \\ \searrow \end{array} \Rightarrow O(1) = \text{constant}$$

$$Q \backslash \begin{array}{c} n^5 + n^3 + n^1 + n^5 \\ \searrow \quad \searrow \quad \searrow \quad \searrow \end{array} \Rightarrow O(n^5)$$

$$Q \backslash \begin{array}{c} 2^n + n^{10} + n^9 + n^8 \\ \searrow \quad \searrow \quad \searrow \quad \searrow \end{array} \Rightarrow O(2^n) \Rightarrow \text{exponential}$$

$T(n) = 100 + 10n^2$ $O(n^2) = O(n^2)$ $2n^2 \geq 10n^2$ $n^2 \leq 10n^2$ $n^2 \geq 10$ $n \geq \sqrt{10}$



$\int_0^1 (x^2 + y^2)^{1/2} dx$	area under
$\int_0^1 (x^2 + y^2)^{1/2} dx$	$y = (x^2 + 1)^{1/2}$
$\int_0^1 (x^2 + y^2)^{1/2} dx$	$= (\sqrt{x^2 + 1}) \cdot dx$
$\int_0^1 (x^2 + y^2)^{1/2} dx$	$= (x^2 + 1)^{1/2} \cdot dx$
$\int_0^1 (x^2 + y^2)^{1/2} dx$	$= (x^2 + 1)^{1/2}$



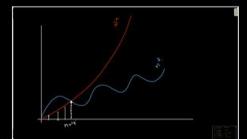
Time Complexity Chart

$O(1) < O(\log_2 n) << O(\sqrt{n}) < O(n) < O(n \log n)$

constant logarithmic square root linear *poly*

$< O(n^2) < O(n^3) <<< O(2^n) < O(3^n) < O(n!)$

quadratic cubic *poly* exponential



O(1) Constant Time Complexity

① System.out.print() *Printing*

② Scanner scnr = new Scanner(Systems);
int monos = scnr.nextInt(); { *Inputs for variable*

③ logical Operators, Comparison operators → arithmetic operation
(&&, ||, !) (==, !=, >, <, !=, <=) (+, -, /, %, *)

④ Assignment operation (=)

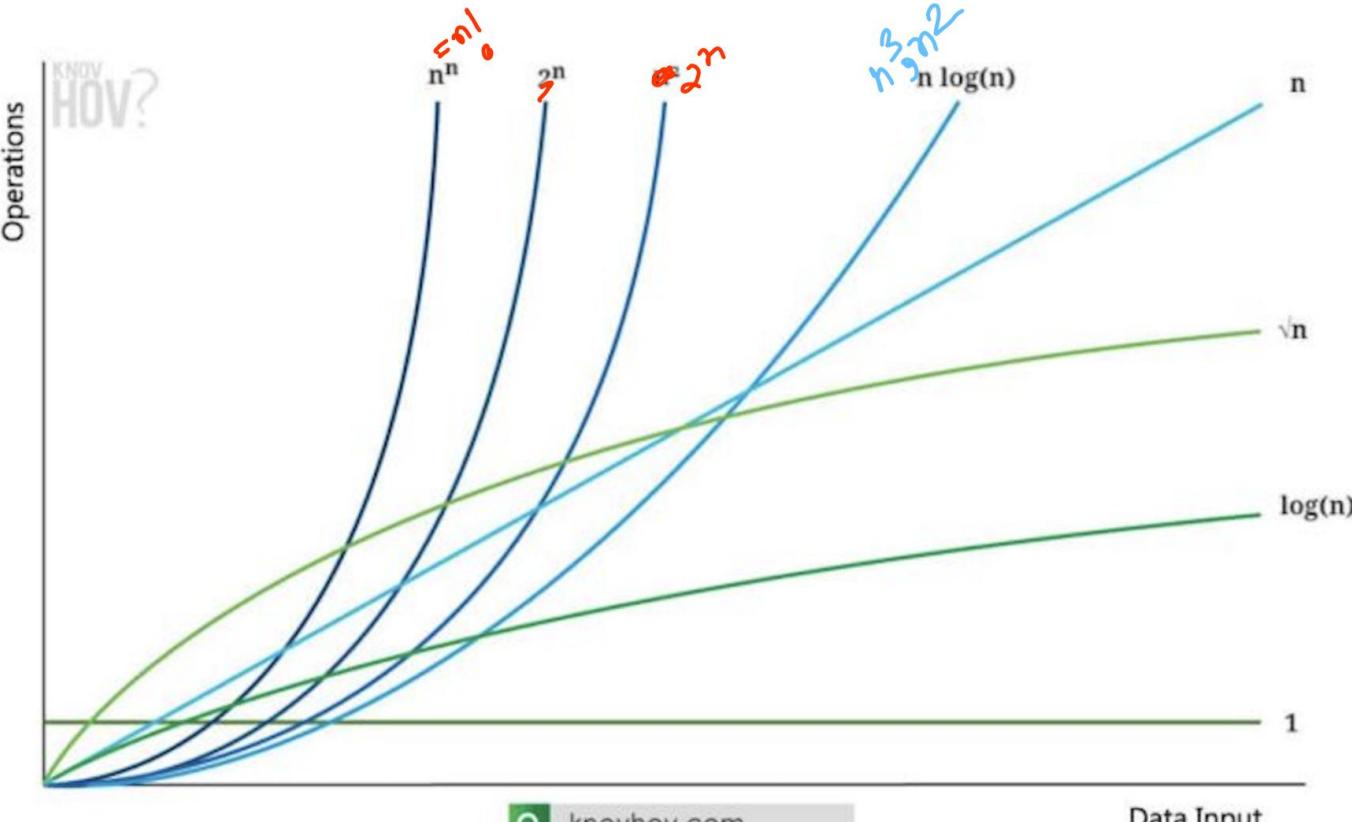
⑦ Array indexing (arr[i])

⑧ Bitwise operators (&, |, ^, !, <<, >>)

⑤ if-else conditions / switch

⑥ Function call
↳ passing parameter
↳ return statement

Time Complexity (Best)	
$O(1)$	$O(n)$
$O(n)$	$O(n^2)$
$O(n^2)$	$O(n^3)$
$O(n^3)$	$O(n^4)$



knovhov.com

Data Input

BIG O' NOTATION TIME COMPLEXITY

- | | |
|--------------------------------------|--------------------------|
| ① Constant Time Complexity | ② Logarithmic Complexity |
| ③ Linear Time Complexity | ④ Quadratic Complexity |
| ⑤ Exponential Complexity | ⑥ Cubic Complexity |
| ⑦ Higher-order Polynomial Complexity | ⑧ Factorial Complexity |
| ⑨ Space Complexity | ⑩ Time Complexity |
| ⑪ Time Complexity | ⑫ Space Complexity |
| ⑬ Time Complexity | ⑭ Space Complexity |
| ⑮ Time Complexity | ⑯ Space Complexity |
| ⑰ Time Complexity | ⑱ Space Complexity |
| ⑲ Time Complexity | ⑳ Space Complexity |
| ⑳ Time Complexity | ⑴ Space Complexity |



for loops time complexity.

①

```
for(int idx = 1; idx <= n; idx++){
    System.out.print(idx + " ");
}
```

$$n=5$$

idx=1

$$k \text{ ms} = O(1)$$

idx=2

$$k \text{ ms} = O(1)$$

idx=3

$$k \text{ ms} = O(1)$$

idx=4

$$k \text{ ms} = O(1)$$

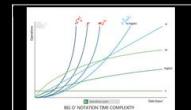
idx=5

$$\underline{k \text{ ms} = O(1)}$$

$$k+k+k+k+k = 5k$$

$$k+k+\dots+k \text{ times} = nk$$

$$O(n) = \text{linear}$$





```
int n = scn.nextInt();  
  
for(int idx = 1; idx <= n; idx++){  
    System.out.print(idx + " ");  
}  
  
int m = scn.nextInt();  
for(int idx = 1; idx <= m; idx++){  
    System.out.print(idx + " ");  
}
```

②

Two loops
which are
independent,
Time complexity
will add up!



idn = 1 k ms
idn = 2 k ms
idn = 3 k ms
idn = 4 k ms
idn = 5 k ms

}
= nk ms

+

idn = 1 k ms
idn = 2 k ms
idn = 3 k ms

}
= mk ms

$(n+m) \times k \text{ ms}$
 $\Rightarrow O(n+m)$
linear

③ Nested loops

```

int n = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = 1; j <= n; j++){
        System.out.print(i + " " + j);
    }
}

```

Time Complexity
Will multiply if loops
var are dependent
 $\Rightarrow O(1)$

$$n = 4$$

$i=1$ $j=1$ kms $i=3$ $j=1$ kms
 $j=2$ kms
 $j=3$ kms
 $j=4$ kms

$i=2$ $j=1$ kms $i=4$ $j=1$ kms
 $j=2$ kms
 $j=3$ kms
 $j=4$ kms

$$\begin{aligned}
 & n \times k + nk + nk + nk \\
 & = 4nk = n \times n \times k \stackrel{m}{\leq} \\
 & \Rightarrow O(n^2) \text{ quadratic}
 \end{aligned}$$



$$n=5, m=3$$

(4)

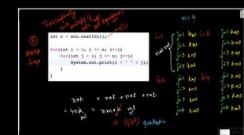
```
int n = scn.nextInt();
int m = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = 1; j <= m; j++){
        System.out.print(i + " " + j);
    }
}
```

$i=1 \quad j=1, j=2, j=3 \quad mk$
+
 $i=2 \quad j=1, j=2, j=3 \quad mk$
+
 $i=3 \quad j=1, j=2, j=3 \quad mk$
+
 $i=4 \quad j=1, j=2, j=3 \quad mk$
+
 $i=5 \quad j=1, j=2, j=3 \quad mk$

$$= nxm \times k$$

$$O(nxm)$$





5

```

int n = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = 1; j <= i; j++){
        System.out.print(i + " " + j);
    }
}

```

$n=5$

$i=1 \quad j=1 \quad k \text{ ms} = O(1) \Rightarrow 1 \times k \text{ ms}$

$i=2 \quad j=1 \quad k \text{ ms}$
 $j=2 \quad k \text{ ms}$
 $\} \Rightarrow 2 \times k \text{ ms}$

$i=3 \quad j=1 \quad k \text{ ms}$
 $j=2 \quad k \text{ ms}$
 $j=3 \quad k \text{ ms}$
 $\} \Rightarrow 3 \times k \text{ ms}$

$i=4 \quad j=1, 2, 3, 4 \quad 4 \times k \text{ ms}$

$i=5 \quad j=1, 2, 3, 4, 5 \quad 5 \times k \text{ ms}$

$$1k + 2k + 3k + 4k + 5k \\ = k(1 + 2 + 3 + \dots + n)$$

$$= k \times \left(\frac{n(n+1)}{2} \right) = \cancel{\frac{k \cdot n^2}{2}} + \cancel{\frac{k \cdot n}{2}} \\ \Rightarrow O(n^2)$$

```

int n = scn.nextInt();
for(int i = 1; i <= n; i++){
    for(int j = 1; j <= i; j++){
        System.out.print(i + " " + j);
    }
}

```

6

```
int n = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = i; j <= n; j++){
        System.out.print(i + " " + j);
    }
}

}
```

$$n \approx 5$$

$i = 1$	$j = 1, 2, 3, 4, 5$	$5k$
$i = 2$	$j = 2, 3, 4, 5$	$4k$
$i = 3$	$j = 3, 4, 5$	$3k$
$i = 4$	$j = 4, 5$	$2k$
$i = 5$	$j = 5$	$1k$

$$\frac{n^{\infty}(n+1)}{2} \times k$$

$$\Rightarrow \underline{\underline{O(n^2)}}$$



⑦

```
for(int i = 1; i < n; i++){
    System.out.print(i + " " + j);
}
```

→ 1, 2, 3, 4, ..., n-1 ⇒ $n-1$ times
 $nk - k \Rightarrow O(n)$
=

⑧

```
for(int i = 1; i <= n / 2; i++){
    System.out.print(i + " " + j);
}
```

→ 1, 2, 3, 4, ..., n/2 ⇒ $n/2$ times
 $\frac{n}{2} \cdot k \Rightarrow O(n)$
=



int n = scn.nextInt();

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

for(int idx = n; idx >= 0; idx++) {
 System.out.println(idx);
}

n is +ve
integer

O iteration
loop

$$n = 5$$

$$idx = 5$$

$$idx \leq 0 \quad 5 \leq 0 \text{ false}$$

k ms

k ms

$O(1)$
Constant

$$n = 5$$

$$idx = 5 \quad 5 \geq 0$$

k ms

$$6 \quad 6 \geq 0$$

k ms

$$7 \quad 7 \geq 0$$

k ms

$$8 \quad 8 \geq 0$$

k ms

$$9$$

\Rightarrow infinite time
non terminating

infinite loop

```

11
int n = scn.nextInt();

for(int idx = 1; idx * idx <= n; idx++){
    System.out.println(idx);
}

```

06

```

int root = (int)Math.sqrt(n);
for(int idx = 1; idx <= root; idx++){
    System.out.println(idx);
}

```

$$n=4 \Rightarrow 2$$

$$\text{id}_n=1 \quad 1*1 \leq 4 \quad \checkmark$$

$$\text{id}_n=2 \quad 2*2 \leq 4 \quad \checkmark$$

$$n=9 \Rightarrow 3$$

$$\text{id}_n=1 \quad 1*1 \leq 9 \quad \checkmark$$

$$\text{id}_n=2 \quad 2*2 \leq 9 \quad \checkmark$$

$$\text{id}_n=3 \quad 3*3 \leq 9 \quad \checkmark$$

$$n=16 \Rightarrow 4$$

$$\text{id}_n=1 \quad 1*1 \leq 16$$

$$\text{id}_n=2 \quad 2*2 \leq 16$$

$$\text{id}_n=3 \quad 3*3 \leq 16$$

$$\text{id}_n=4 \quad 4*4 \leq 16$$

$$n=25 \Rightarrow 5$$

$$\text{id}_n=1 \quad 1^2 \leq 25$$

$$\text{id}_n=2 \quad 2^2 \leq 25$$

$$\text{id}_n=3 \quad 3^2 \leq 25$$

$$\text{id}_n=4 \quad 4^2 \leq 25$$

$$\text{id}_n=5 \quad 5^2 \leq 25$$

$$\Rightarrow O(\sqrt{n} + N)$$

$$= O(\sqrt{N})$$

$$= O(N^{1/2})$$



(12)

```

int n = scn.nextInt();

for(int idx = 1; idx <= n; idx = idx * 2){
    System.out.println(idx);
}

```

$$n = 4 = 2^2$$

$$\text{id} n = 1 \leq 4$$

$$\text{id} n = 2 \leq 4$$

$$\text{id} n = 4 \leq 4$$

3 times

$$n = 8 = 2^3$$

$$\text{id} n = 1 \leq 8$$

$$\text{id} n = 2 \leq 8$$

$$\text{id} n = 4 \leq 8$$

$$\text{id} n = 8 \leq 8$$

4 times

$$n = 16 = 2^4$$

$$\text{id} n = 1 \leq 16$$

$$\text{id} n = 2 \leq 16$$

$$\text{id} n = 4 \leq 16$$

$$\text{id} n = 8 \leq 16$$

$$\text{id} n = 16 \leq 16$$

5 times

$$n = 2^n \Rightarrow \log_2 n = x$$

loop is running n times

$$= O(\log_2 n)$$

$$n = 32 = 2^5 \quad 5 = \log_2 32$$

6 times

$$n = 64 = 2^6 \quad 6 = \log_2 64$$

7 times

$$n = 128 = 2^7 \quad 7 = \log_2 128$$

8 times

Value	Count
00000000	1
00000001	1
00000010	1
00000011	1
00000100	1
00000101	1
00000110	1
00000111	1
00001000	1
00001001	1
00001010	1
00001011	1
00001100	1
00001101	1
00001110	1
00001111	1
00010000	1
00010001	1
00010010	1
00010011	1
00010100	1
00010101	1
00010110	1
00010111	1
00011000	1
00011001	1
00011010	1
00011011	1
00011100	1
00011101	1
00011110	1
00011111	1
00100000	1
00100001	1
00100010	1
00100011	1
00100100	1
00100101	1
00100110	1
00100111	1
00101000	1
00101001	1
00101010	1
00101011	1
00101100	1
00101101	1
00101110	1
00101111	1
00110000	1
00110001	1
00110010	1
00110011	1
00110100	1
00110101	1
00110110	1
00110111	1
00111000	1
00111001	1
00111010	1
00111011	1
00111100	1
00111101	1
00111110	1
00111111	1
01000000	1
01000001	1
01000010	1
01000011	1
01000100	1
01000101	1
01000110	1
01000111	1
01001000	1
01001001	1
01001010	1
01001011	1
01001100	1
01001101	1
01001110	1
01001111	1
01010000	1
01010001	1
01010010	1
01010011	1
01010100	1
01010101	1
01010110	1
01010111	1
01011000	1
01011001	1
01011010	1
01011011	1
01011100	1
01011101	1
01011110	1
01011111	1
01100000	1
01100001	1
01100010	1
01100011	1
01100100	1
01100101	1
01100110	1
01100111	1
01101000	1
01101001	1
01101010	1
01101011	1
01101100	1
01101101	1
01101110	1
01101111	1
01110000	1
01110001	1
01110010	1
01110011	1
01110100	1
01110101	1
01110110	1
01110111	1
01111000	1
01111001	1
01111010	1
01111011	1
01111100	1
01111101	1
01111110	1
01111111	1

13

```
for(int idx = n; idx >= 1; idx = idx / 2){  
    System.out.println(idx);  
}  
  
=> O(log2n)
```

exponential \rightarrow as N increases, δ^* grows rapidly

$$N=10 \rightarrow 20 \rightarrow 30 \rightarrow 40$$

$$2^{10} \rightarrow 2^{20} \rightarrow 2^{30} \rightarrow 2^{40}$$

$$1024 \rightarrow 1048576 \rightarrow 1073741824 \rightarrow 1.09 \times 10^{12}$$

logarithmic \rightarrow as N grows, $\log_2 N$ grows very slowly
 $N = 2^1 \rightarrow 2^{10} \rightarrow 2^{100} \rightarrow 2^{1000}$

$\log n$ is $\underbrace{10 \rightarrow 100 \rightarrow 1000}$  almost constant



14)

```
for(int idx = 0; idx * idx <= n; idx++){
    System.out.println(idx);  $O(\sqrt{n})$ 
}
```

$\text{idx} = 0, 1, 2, 3, \dots, \text{infinite}$

15)

```
for(int idx = 0; idx <= n; idx = idx * 2){
    System.out.println(idx);  $\text{runtime error}$ 
}
```

16)

```
for(int idx = 1; idx <= n; idx = idx * 3){
    System.out.println(idx);  $O(\log_3 n)$ 
}
```

17)

```
for(int idx = 1; idx <= n; idx = idx * 10){
    System.out.println(idx);  $\log_{10} n$ 
}
```

18)

```
for(int idx = 1; idx <= n; idx = idx + 2){
    System.out.println(idx);
}
```

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

19)

```
for(int idx = 1; idx <= n; idx = idx + 3){
    System.out.println(idx);
}
```

$\frac{n}{3} = O(n)$

20)

```
for(int idx = 1; idx <= n; idx++){
    for(int j = 0; j < 5; j++){
        System.out.println(idx);  $j \rightarrow O(5) = O(1)$ 
    }
}
```

$\rightarrow n \cdot O(1) = O(n)$

21)

```
for(int idx = 1; idx <= n; idx++){
    for(int j = 1; j <= n; j = j * 2){
        |
    }
}
```

$\log_2 n$

$= O(n \log_2 n)$

$n = 10$
 $\text{id}n = 1, 3, 5, 7, 9$

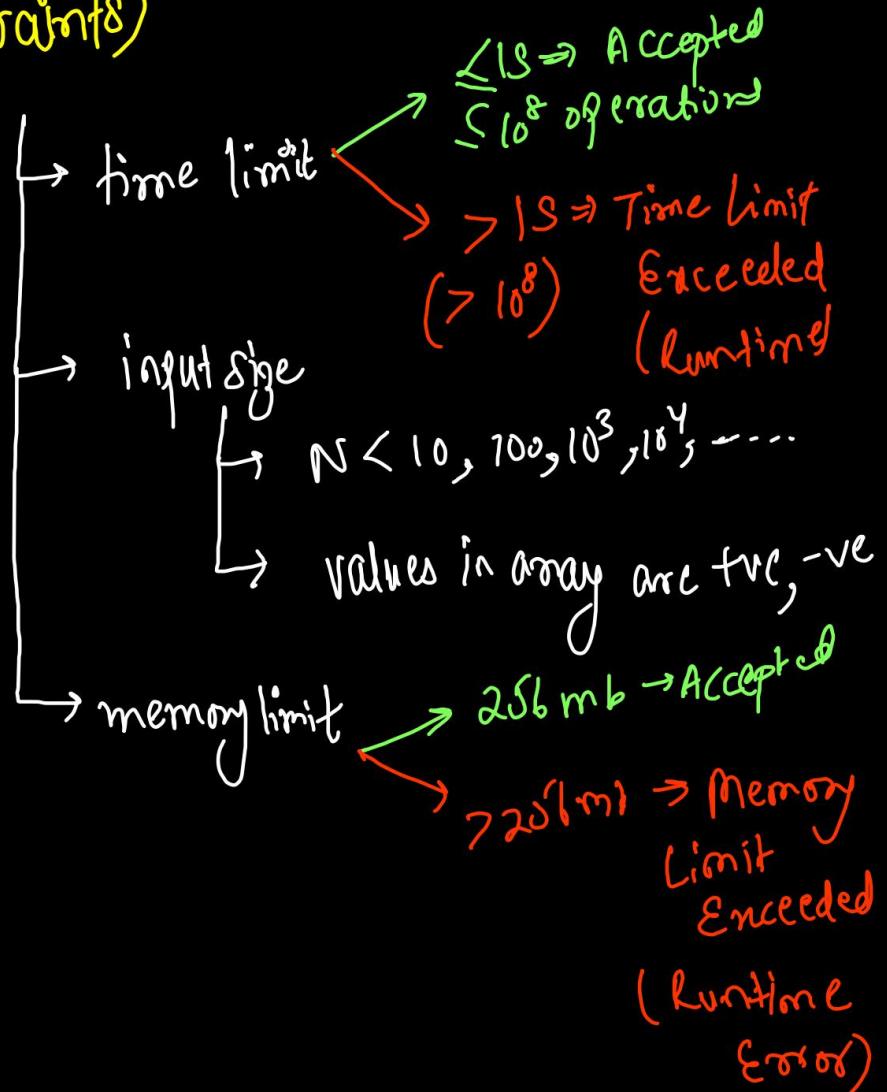
$n = 20$
 $\text{id}n = 1, 3, 5, 7, 9, 11, 13, 15, 17, 19$

$n/2 \text{ times}$
 $\Rightarrow O(n)$



Competitive Programming || Chart (Constraints)

$1S = 10^8$ operations per second





Time Complexity Chart

1 second = 10^8 operations
time limit = 1s

N_{max}	Time Complexity	Examples
18	$O(2^n)$ or $O(r!)$	Recursion & Backtracking
22	$O(2^n \cdot n^2)$ {exponential}	Travelling Salesman (DP with bitmasking)
4×10^2	$O(N^3)$ cubic	3 nested loops (print all subsequences) Matrix Multiplication
10^4	$O(N^2)$ quadratic	Nested loops (Bubble/Insertion/selection), matrix ^{DP} ^{2D DP}
10^6	$O(N \log N)$	Arrays Sort (MergeSort / QuickSort / HeapSort) BS on Answer
10^8	$O(n)$ Linear	Linear search, max/min of array, merge 2 sorted arrays
10^{16}	$O(\sqrt{N})$	Checking prime.
10^{18} or 2^{63} long.max.value	$O(1)$ or $O(\log_2 N)$ constant-logarithmic	Digital Transversal, Binary Nos, Binary Search

① Print Hello world $\rightarrow O(1)$ Constant time complexity \Rightarrow never give TLE
 \Rightarrow always run
Check odd or even, print a to z, print table of 4

② Linear loop $\rightarrow O(n)$

Linear time complexity

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int n = scn.nextInt();  
  
    for(int idx=0; idx<=n; idx++){  
        System.out.println(idx);  
    }  
}
```

$$\Rightarrow N_{max} = 10^8$$

Integer.MAX_VALUE
 $= 2^{31} \times 10^9$
↑ TLE

Array printing, N^{th} fibonacci

Operation	Time Complexity	Description
Accessing an element	$O(1)$	Constant time complexity
Traversing an array	$O(n)$	Linear time complexity
Swapping two elements	$O(1)$	Constant time complexity
Inserting an element	$O(n)$	Linear time complexity
Deleting an element	$O(n)$	Linear time complexity
Sorting an array	$O(n \log n)$	Logarithmic time complexity
Merge sort	$O(n \log n)$	Logarithmic time complexity
Quicksort	$O(n^2)$	Quadratic time complexity
Binary search	$O(\log n)$	Logarithmic time complexity
Hash search	$O(1)$	Constant time complexity

③ Logarithmic Loop $\Rightarrow O(\log N)$ \Rightarrow It will never give TLE

```
Scanner scn = new Scanner(System.in);
int n = scn.nextInt();

for(int i = n; i > 0; i = i/3){
    System.out.println(i);
}
```

$\left\{ \log_3 N \right.$

$$N_{\max} = no \text{ (ints)}$$

$$\log N = 10^8$$

$$N = 3^{10^8} = \text{very big value} = +\infty$$

$$N = 10^8 \Rightarrow \underbrace{\text{Digits}}_{\text{long}} = 18$$

$\left\{ \text{Point Digits of a No} \Rightarrow O(\log_{10} N) \right.$

```
for(int i = n; i > 0; i = i/10){
    System.out.println(i % 10);
}
```

```
for(int idx=n; idx>=1; idx=idx/2){
    step=step+1;
}
```

$\left\{ \log_2 N \right.$



④ Quadratic Time (nested loops)

```
int n = sc.nextInt();
for (int i = 0; i<n; i++){
    for (int j = 0; j<n; j++){
        System.out.print("*");
    }
    System.out.println();
}
```

```
for(int i=0 ; i<n ; i++){
    for(int j=0 ; j<=i ; j++){
        System.out.print("* ");
    }
    System.out.println();
}
```

Printing Upper/lower Triangles

Grid/Bon patterns, Bubble sort,
insert Sort, selectn sort, Point Park

$$\Rightarrow O(n^2)$$

$$N_{\max} = 10^4$$

$$O(n^2) = 10^4 \times 10^4$$

$$= 10^8 \times 10^8$$

Accepted

$$N = 10^8$$

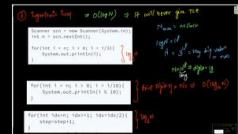
$$O(n^2) = O(10^8 \times 10^8)$$
$$= O(10^{16})$$

TLE $> 10^8$

$$N = 10^2$$

$$O(n^2) = 10^2 \times 10^2$$
$$= 10^4$$
$$> 10^8$$

TLE



⑤ GCD (Euclid Algorithm) Long Division

```
public static int gcd(int a, int b){  
    int numerator = a, denominator = b;  
  
    while(numerator % denominator != 0){  
        int remainder = numerator % denominator;  
  
        numerator = denominator;  
        denominator = remainder;  
    }  
  
    return denominator;  
}
```

} $O(\log(\min(a, b)))$
 $= \log N$



Check No is Prime



Approach ①

```
public static boolean isPrime1(int n){  
    if(n == 1 || n == 2) return true; // 1 and 2 are prime  
  
    for(int idx = 2; idx <= n - 1; idx++){ → O(n)  
        // is idx a factor of n OR is n a multiple of idx  
        if(n % idx == 0) return false;  
    }  
  
    return true;  
}
```

linear
 $N_{max} = 10^8$ (int)
brute force

$\leftarrow N = 10^9, 10^{11}, \dots, 10^{16}$
time limit exceeded

Approach ②

```
public static boolean isPrime2(int n){  
    if(n == 1 || n == 2) return true; // 1 and 2 are prime  
  
    int sqrt = (int)Math.sqrt(n);  
    ↓ iteration  
    for(int idx = 2; idx <= sqrt; idx++){ → O(√N) = O(N^1/2)  
        // is idx a factor of n OR is n a multiple of idx  
        if(n % idx == 0) return false;  
    }  
  
    return true;  
}
```

$O(\sqrt{N}) = O(N^{1/2})$
 $N_{max} = (10^8)^2 = 10^{16}$ long
optimized approach

Linear Search Algorithm

```
public static String linearSearch(int[] arr, int target){  
    for(int idx = 0; idx < arr.length; idx++){  
        if(arr[idx] == target) {  
            return "True";  
        }  
    }  
  
    return "False";  
}
```

$$N_{\max} = 10^8$$

Worst case $\rightarrow \Theta(n)$ linear

Average case $\rightarrow \Theta(n)$ linear

Best case $\rightarrow \Theta(1)$ constant

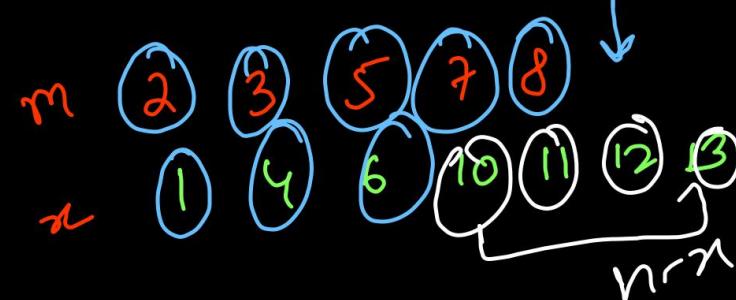


Merge 2 Sorted Arrays

```
public static int[] merge(int[] arr1, int[] arr2){  
    int first = 0, second = 0, third = 0;  
    int[] res = new int[arr1.length + arr2.length];  
  
    while(first < arr1.length && second < arr2.length){  
        if(arr1[first] <= arr2[second]){  
            res[third] = arr1[first];  
            first++;  
            third++;  
        } else {  
            res[third] = arr2[second];  
            second++;  
            third++;  
        }  
    }  
  
    while(first < arr1.length){  
        res[third] = arr1[first];  
        first++;  
        third++;  
    }  
  
    while(second < arr2.length){  
        res[third] = arr2[second];  
        second++;  
        third++;  
    }  
  
    return res;  
}
```

$$m = \text{correlation}$$

$$n = \text{array.length}$$



O(m+n)



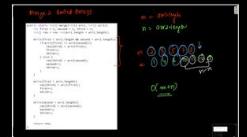
Print all subarrays

```
public static void printSubarrays(int[] arr){  
    for(int start = 0; start < arr.length; start++){  
        for(int end = start; end < arr.length; end++){  
            for(int idx = start; idx <= end; idx++){  
                System.out.print(arr[idx] + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

$$n \times n \times n = O(n^3)$$

ubic

$$N_{max} = 500$$
$$= 10^2 - 10^3$$



Target sum Subarray

```
public static boolean targetSumSubarray(int[] arr, int target){  
    for(int st = 0; st < arr.length; st++){  
        int sum = 0;  
        for(int end = st; end < arr.length; end++){  
            sum = sum + arr[end];  
            if(sum == target) return true;  
        }  
    }  
    return false;  
}
```

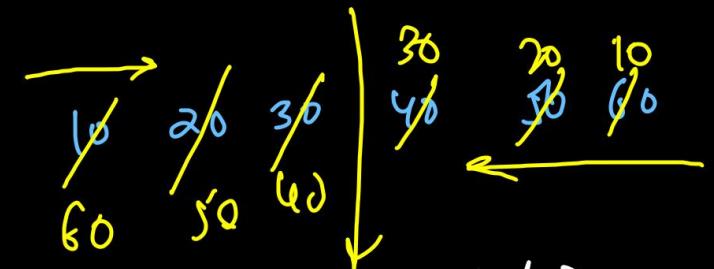
$O(n^2)$ quadratic

$$N_{\max} = 10^4$$



Reverse Array

```
public static void reverseArray(int[] arr){  
    int left = 0, right = arr.length - 1;  
  
    while(left < right){  
        int temp = arr[left];  
        arr[left] = arr[right];  
        arr[right] = temp;  
  
        left++; right--;  
    }  
}
```



$$O(n/2) + O(n/2)$$

$\approx O(n)$ linear

$$N_{\max} = 10^8$$



Sort 01

```
public static void sortBinary(int[] arr){  
    int left = 0, right = 0;  
    while(right < arr.length){  
        if(arr[right] == 1){  
            right++;  
        } else {  
            int temp = arr[left];  
            arr[left] = arr[right];  
            arr[right] = temp;  
  
            left++; right++;  
        }  
    }  
}
```

$O(n)$

Not optimized

Two pointer
most optimized

Sort 012

```
public static void sort012(int[] arr){  
    int left = 0, mid = 0, right = arr.length-1;  
    while(mid <= right){  
        if(arr[mid] == 0){  
            swap(arr, left, mid);  
            left++;  
            mid++;  
        }  
        else if(arr[mid] == 1){  
            mid++;  
        }  
        else{  
            swap(arr, right, mid);  
            right--;  
        }  
    }  
}
```

$O(n)$



Space Complexity

- ① Input Space Complexity (Parameters)
- ② Output Space Complexity (return type)
- ③ Recursion call stack space (Recursion)
- * ④ Extra or Auxiliary Space (Data Structures to perform Algo)
 - ↳ no extra space soln if in-place algorithm

O(1) space

↓
normal primitive
variables

O(n) space

↓
1D Array / String

O(n²) space

↓
2D Array / matrix

