

Q What 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 or slow.

Q 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 proportional to the input size.

[Single Loop]

Ex: Looping through an array to

$O(n^2) > O(n) > O(\log n) > O(m) > O(\sqrt{n})$ find specific item. Linear Search.

$O(n^2) > O(n) >$

4. $O(n^2)$ - Quadratic Time \rightarrow Ex: Nested Loops

5. $O(2^n)$ - Exponential Time \rightarrow time doubles with each additional input; it's very slow for large inputs.

Ex: Recursive Algorithm

Like Fibonacci sequence

$O(1)$ \rightarrow super fast of all others O(1)

$O(n)$ \rightarrow Time \uparrow input size \uparrow

$O(\log n)$ \rightarrow Time increase slowly, even for big input

$O(n^2)$ \rightarrow Time increase very fast, Avoid this for

large input.

Time Complexity: \leftarrow Smallest terms - (1) O(1)

When,

• Assignment Operation. [$a=10, b=20$] \leftarrow Smallest term - (1) O(1)

• Comparison Operation. [$a < b$] \leftarrow Smallest term - (1) O(1)

• Mathematical Operation. [$a+b$] \leftarrow Smallest term - (1) O(1)

• Function Call. \leftarrow Smallest term - (1) O(1)

• Function এর ভেত্তার Operation: [good sign]

of course result won't original

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

Ex: 01

```
#include <iostream>
using namespace std;

int main() {
    int a; // 1
    cin >> a; // 1
    cout << a << endl; // 1
    return 0;
}
```

Ex: 01

Output:

input: 10 are given

output: 10

Time complexity = $O(1+1+1)$

General $\Rightarrow O(3) \approx O(1)$

Ex: 02

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

Hence, Exact time complexity:

$\Rightarrow 1 + 11 + 10 + 10 = 32$ times.

$O(32) \approx O(1)$

Ex: 03

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

Ex: 03

Hence, total statements:

No. Operations: $n + 1 + 1 + n + 1 + n + n = 3n + 4$

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

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

: O(n)

/ bres > n > bres

O(n)

{

Ex: 03

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; // 50
    }
    cout << endl; // 1
}
```

1st Loop: statements

1st Loop: statements

$$n = 101 \rightarrow 101 + 101$$

2nd Loop:

$$\Rightarrow \text{Let } m = 51$$

; time complexity

Is $n > m$ so,

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

```

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

```

Ex: 05

```
#include <iostream>
Using namespace std;
```

```
int main() {
```

```

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

```

```
return 0;
```

3.2.7
<main> abnennbar
die eingesetzte gru
} () rufen frei
aus der fkt
mehrere mal

} (i++ i <= i < i frei) not

} (i++ i <= i < i frei) not

Hence, } Exact steps:

Outer Loop:

↳ for outer Loop

i=1 the inner Loop

works = 1+11+10+10

= 32 times

then for i=10

= $32 \times 10 = 320$ 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

$O(10^2)$ so, $10 \times 10 = 100$ times

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;
        }
    }
}
```

} (inner loop)

{ (outer loop) bring

Time complexity:

$O(n*m)$

Ans: 24

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++) {
            cout << i << " " << j << endl;
        }
    }
}
```

} (inner loop)

Here,

$O(n*m)$

Time complexity,

```
return 0;
```

```
}
```

Ex: 08 (Output) 0

```
#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

$$n \leq 100$$

$$m \leq 50$$

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: n, m, o এর
 value মান আছে
 - অগুলো মান value বেশি
 সেটা total time
 complexity].

example:
 Let $n = 100, m = 10, o = 50;$

$O(n)$

Ex: 10

```
#include <iostream>
using namespace std;

int main () {
    int n1, n2, result;
    n1 = 10;
    n2 = 20;
    result = n1 + n2;
    return 0;
}
```

(1) O

Hence, this is constant

3 to assignment op:

n1 = 10;

n2 = 20;

1 result = tri

and

1 Addition Op:

n1 + n2;

total = 4 operation

O(4) X

Time Complexity is: O(1) ✓

[Note: Operation एवं सार्व
input एवं दोनों समान होते,
मान दोनों value 10 ही 4 तो
operation होते even यह
value मान 10000 3 ही same
4 तो OP. होते input एवं
मान मान उत्तेकता करते ही 4 तो
OP. होते so constant O(1).]

$i + i : n \Rightarrow i + (n - i) \text{ nos}$

$i + H_{n-1} = H_n \text{ nos}$

$H_n > H_{n-1} > \dots$

O(n)

(n) O

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:

⇒ assignment op = 1 ⚡

⇒ Mathematical op. 3 ⚡

total op = 4 ⚡

[Constant]

Time Complexity:

$O(1)$

Ex: 12

```
#include <iostream>
using namespace 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 এবং সাথে

Linearly যাবে তাই \propto হিসাবে আলোকে,

n এর value এর পাদ

10 টা তাই 10 বাব Loop

চলবে এবং 10 বাব তার

ভিত্তির operation গুলি 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] == x) {
```

```
            cout << "Found" << endl;
```

```
}
```

```
        if (arr[mid] < x) {
```

```
            left = mid + 1;
```

```
} else {
```

```
            right = mid - 1;
```

```
}
```

```
    return 0;
```

(real) O = ~~ct(xlgm)~~ ~~amT~~

```
}
```

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

No. of iteration: 4

Min. No. comparison \rightarrow Best Case

Max. No. comparison \rightarrow Worst Case

i) Best Case:

Let, Search element = 5

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

$$\text{mid} = 4$$

if search element is 5
Here, 1 comparison needed.
so this is best case

time complexity is $O(1)$,

ii) Worst Case:

Let, Search element = 10

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

$$S \neq \text{mid}$$

$$\text{mid} = \frac{5+9}{2} = 7 \rightarrow 2$$

$$S \neq \text{mid}$$

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

$$S \neq \text{mid}$$

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

$$S \neq \text{mid} \quad S \neq \text{mid}$$

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

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

$$\text{mid} = \boxed{S = \text{mid}}$$

Here, in worst case

No. of comparison 54

Let, array length, $n = 10$

i) 1st ite

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

ii) 2nd ite i)

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

iii) 3rd ite i bin

$$= \frac{\frac{n}{2^2}}{2} \quad \text{for } 3\text{-bit tri}$$

iv) 4th ite i bin

$$= \frac{n}{2^4} \quad \begin{cases} \text{for } 4\text{-bit tri} \\ \text{or } 2\text{-bit tri} \end{cases}$$

$$\vdots = \frac{n}{2^k} = 1 \quad \begin{cases} (\text{if } k=3 \Rightarrow 2^3=8) \text{ of } 4\text{-bit tri} \\ (\text{if } k=4 \Rightarrow 2^4=16) \text{ of } 2\text{-bit tri} \end{cases}$$

(1) O(2^k) \approx 2^k fixings until

seen below (ii)

or = elements doesn't fit

$$2^k - n = \frac{2^k - n}{2^k} = \text{bin} \quad \begin{cases} \text{bin} \neq 2^k \end{cases}$$

$$2^k - n = \frac{2^k - n}{2^k + 1} = \text{bin} \quad \begin{cases} \text{bin} \neq 2^k + 1 \end{cases}$$

$$2^k - n = \frac{2^k - n}{2^k + 2} = \text{bin} \quad \begin{cases} \text{bin} \neq 2^k + 2 \end{cases}$$

etc. etc. etc. etc. etc. etc. etc. etc.

iii) Average Case:

Time = E_{avg}

Time complexity = $O(\log n)$

Time = E_{avg}

(avg) O(fixings until seen below)

Time = E_{avg} until small
Hence minimum to 100%

(ii) reduction to 0.01

Ex: 14 Bubble Sort: [not maximized to 100]

Iteration $i=0$

| | | | | | |
|----|----|----|----|---|-------------|
| 0 | 1 | 2 | 3 | 4 | $n=5$ ksize |
| 22 | 14 | 12 | 18 | 9 | |

$$\text{No. of iteration} = 5-1 = 4 \quad \boxed{\text{number of iteration}} = n-1$$

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 | 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 |
| 9 | 12 | 14 | 18 | 22 |

No. of Comparison for: $[n-i]$ odd n=5

i=0, n= if n=5

$$\Rightarrow 5-1-0 = 4 \text{ comparison}$$

i=1

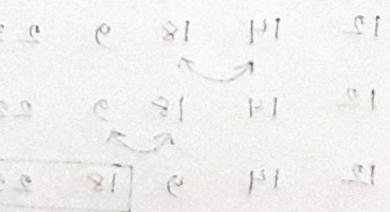
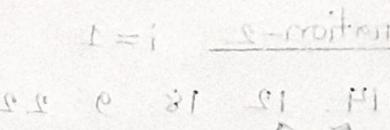
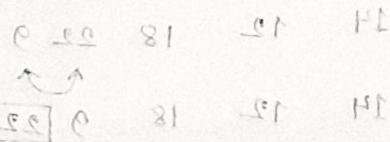
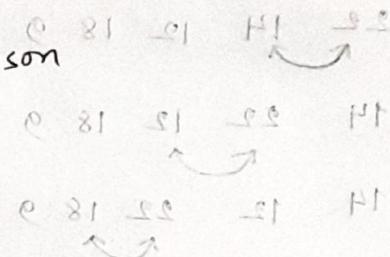
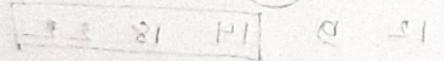
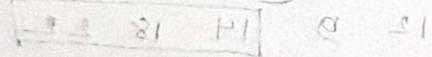
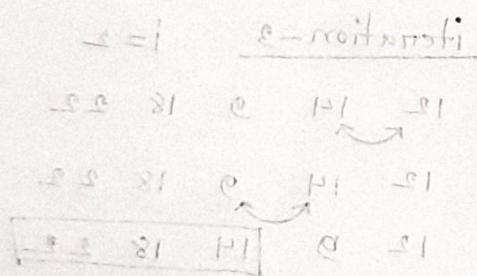
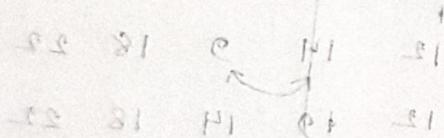
$$= 5-1-1 = 3 \text{ comparison}$$

i=2

$$= 5-1-2 = 2 \text{ comparison}$$

i=3

$$= 5-1-3 = 1 \text{ comparison}$$



Example: 14 Bubble Sort

```
#include <iostream>
using namespace std;

int main() {
    int arr[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 (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }

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

Here,

$$\begin{aligned} \text{No. of iteration} &= 4 \\ \text{and No. of comparison} & \\ \Rightarrow & 4 + 3 + 2 + 1 \\ \Rightarrow & 1 + 2 + 3 + \dots + n-1 \\ \Rightarrow & \frac{n(n-1)}{2} \\ \Rightarrow & \frac{n^2-n}{2} \end{aligned}$$

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

best case: $O(n)$

Example: 15

①

```

int i, j, n;
scanf("xd", &n);
for (i=n; i>=1; i--) {
    for (j=n/2; j>=1; j--) {
        printf("Be simple\n");
    }
    printf("think high.\n");
}
}

```

For Loop: 1 (outer)

1st iteration: $i = n$
 2nd $i = \frac{n}{2}$
 3rd $i = \frac{n}{4}$
 ...
 until $i = 1$

$$\log_2(n)$$

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

For Loop: 2 (inner)

Loop num - $\frac{n}{2}$ times,

We can write,

$$\therefore O(n)$$

For Loop: 3

Loop num, $i * i <= n$ means

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

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

Hence, Loop 1 and 2

$$O(n \log n)$$

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

because 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!");
    }
}

```

$i = 1$: $\sim 1 \text{ ms}$
 $i = 2$: $\sim 2 \text{ ms}$
 $i = 3$: $\sim 3 \text{ ms}$
 $i = 4$: $\sim 4 \text{ ms}$
 $i = 5$: $\sim 5 \text{ ms}$
 $i = 6$: $\sim 6 \text{ ms}$
 $i = 7$: $\sim 7 \text{ ms}$
 $i = 8$: $\sim 8 \text{ ms}$
 $i = 9$: $\sim 9 \text{ ms}$
 $i = 10$: $\sim 10 \text{ ms}$
 $i = 11$: $\sim 11 \text{ ms}$
 $i = 12$: $\sim 12 \text{ ms}$
 $i = 13$: $\sim 13 \text{ ms}$
 $i = 14$: $\sim 14 \text{ ms}$
 $i = 15$: $\sim 15 \text{ ms}$
 $i = 16$: $\sim 16 \text{ ms}$
 $i = 17$: $\sim 17 \text{ ms}$
 $i = 18$: $\sim 18 \text{ ms}$
 $i = 19$: $\sim 19 \text{ ms}$
 $i = 20$: $\sim 20 \text{ ms}$
 $i = 21$: $\sim 21 \text{ ms}$
 $i = 22$: $\sim 22 \text{ ms}$
 $i = 23$: $\sim 23 \text{ ms}$
 $i = 24$: $\sim 24 \text{ ms}$
 $i = 25$: $\sim 25 \text{ ms}$
 $i = 26$: $\sim 26 \text{ ms}$
 $i = 27$: $\sim 27 \text{ ms}$
 $i = 28$: $\sim 28 \text{ ms}$
 $i = 29$: $\sim 29 \text{ ms}$
 $i = 30$: $\sim 30 \text{ ms}$
 $i = 31$: $\sim 31 \text{ ms}$
 $i = 32$: $\sim 32 \text{ ms}$
 $i = 33$: $\sim 33 \text{ ms}$
 $i = 34$: $\sim 34 \text{ ms}$
 $i = 35$: $\sim 35 \text{ ms}$
 $i = 36$: $\sim 36 \text{ ms}$
 $i = 37$: $\sim 37 \text{ ms}$
 $i = 38$: $\sim 38 \text{ ms}$
 $i = 39$: $\sim 39 \text{ ms}$
 $i = 40$: $\sim 40 \text{ ms}$
 $i = 41$: $\sim 41 \text{ ms}$
 $i = 42$: $\sim 42 \text{ ms}$
 $i = 43$: $\sim 43 \text{ ms}$
 $i = 44$: $\sim 44 \text{ ms}$
 $i = 45$: $\sim 45 \text{ ms}$
 $i = 46$: $\sim 46 \text{ ms}$
 $i = 47$: $\sim 47 \text{ ms}$
 $i = 48$: $\sim 48 \text{ ms}$
 $i = 49$: $\sim 49 \text{ ms}$
 $i = 50$: $\sim 50 \text{ ms}$
 $i = 51$: $\sim 51 \text{ ms}$
 $i = 52$: $\sim 52 \text{ ms}$
 $i = 53$: $\sim 53 \text{ ms}$
 $i = 54$: $\sim 54 \text{ ms}$
 $i = 55$: $\sim 55 \text{ ms}$
 $i = 56$: $\sim 56 \text{ ms}$
 $i = 57$: $\sim 57 \text{ ms}$
 $i = 58$: $\sim 58 \text{ ms}$
 $i = 59$: $\sim 59 \text{ ms}$
 $i = 60$: $\sim 60 \text{ ms}$
 $i = 61$: $\sim 61 \text{ ms}$
 $i = 62$: $\sim 62 \text{ ms}$
 $i = 63$: $\sim 63 \text{ ms}$
 $i = 64$: $\sim 64 \text{ ms}$
 $i = 65$: $\sim 65 \text{ ms}$
 $i = 66$: $\sim 66 \text{ ms}$
 $i = 67$: $\sim 67 \text{ ms}$
 $i = 68$: $\sim 68 \text{ ms}$
 $i = 69$: $\sim 69 \text{ ms}$
 $i = 70$: $\sim 70 \text{ ms}$
 $i = 71$: $\sim 71 \text{ ms}$
 $i = 72$: $\sim 72 \text{ ms}$
 $i = 73$: $\sim 73 \text{ ms}$
 $i = 74$: $\sim 74 \text{ ms}$
 $i = 75$: $\sim 75 \text{ ms}$
 $i = 76$: $\sim 76 \text{ ms}$
 $i = 77$: $\sim 77 \text{ ms}$
 $i = 78$: $\sim 78 \text{ ms}$
 $i = 79$: $\sim 79 \text{ ms}$
 $i = 80$: $\sim 80 \text{ ms}$
 $i = 81$: $\sim 81 \text{ ms}$
 $i = 82$: $\sim 82 \text{ ms}$
 $i = 83$: $\sim 83 \text{ ms}$
 $i = 84$: $\sim 84 \text{ ms}$
 $i = 85$: $\sim 85 \text{ ms}$
 $i = 86$: $\sim 86 \text{ ms}$
 $i = 87$: $\sim 87 \text{ ms}$
 $i = 88$: $\sim 88 \text{ ms}$
 $i = 89$: $\sim 89 \text{ ms}$
 $i = 90$: $\sim 90 \text{ ms}$
 $i = 91$: $\sim 91 \text{ ms}$
 $i = 92$: $\sim 92 \text{ ms}$
 $i = 93$: $\sim 93 \text{ ms}$
 $i = 94$: $\sim 94 \text{ ms}$
 $i = 95$: $\sim 95 \text{ ms}$
 $i = 96$: $\sim 96 \text{ ms}$
 $i = 97$: $\sim 97 \text{ ms}$
 $i = 98$: $\sim 98 \text{ ms}$
 $i = 99$: $\sim 99 \text{ ms}$
 $i = 100$: $\sim 100 \text{ ms}$

iii

Loop: 1 (outer) b/w

$i < n$, so no. of iteration
is n time. not
 $O(n)$ $\rightarrow O(n^2)$.

Loop: 2 (inner)

Here, $j \neq i$ 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 $\Rightarrow k=100$ time,

$O(1)$

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

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)
}
}
}

```

(n) O

O : good

\leq Total time complexity

is $\approx O(n) \times O(n) \times O(\log n)$

$$= O(n^2 \log n)$$

ii

3 (extra) code box

Loop : 1

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

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

Loop : 2

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

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

Loop : 3

if $n=8$

Loop runs, $k =$

1st iteration: $k=1$

2nd \sim : $k=2$

3rd \sim : $k=4$

4th \sim : $k=8$

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

$\therefore \frac{n}{4}$

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

function ①, ②, ③ out of three function, the faster one is,

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

Why :

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

$O(n \log n)$: is faster of the three because it grows slower than both $O(n^2)$ and $O(n^2 \log n)$.

$O(n^2)$: is slower than $O(n^2 \log n)$ because it grows faster than $O(n^2 \log n)$.

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