Feature

⇒ Return Search Results.

	Rahul (10^5)	Raj (10^3)
Execution time	15 sec	10 sec
	↓ windows XP	↓ macbook pro m2 chip
	8 sec	10sec
	↓ C++	↓ python
	8 sec	5 sec
	↓ volcano	↓ antarctica,

Execution time is not the right metric to compare algorithms.

Akhil   Mayank.

$100\log_{10} n$ $N/10$

$n \leq 3500$, $N/10$ (Mayank) is better.
$n > 3500$, $100\log_{10} n$ (Akhil) is better.

Ind vs Pak : 1.8 million ⇒ 18 lakh.
Generally > 1cr.

Youtube video : usually > 1 million.

⇒ Algo 1, Algo 2, Algo 3, Algo 4.

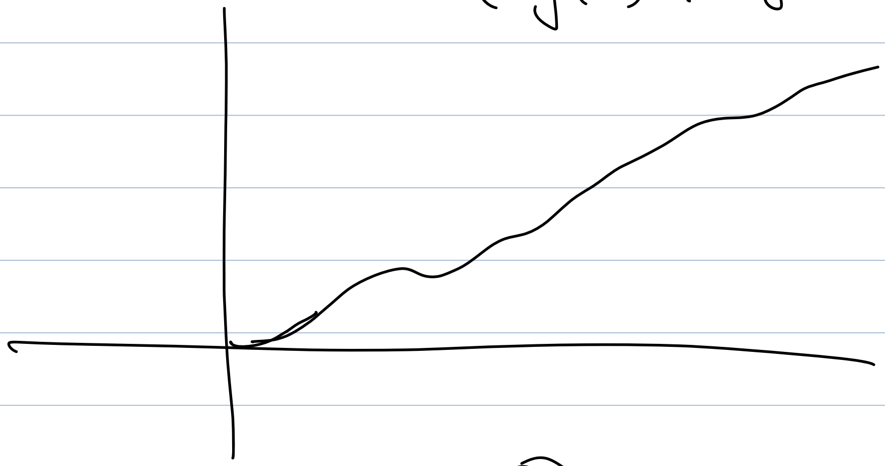
<u>Asymtotic Analysis of Alogorithms</u> → Analysing algorithms for large input.

Bigo   ~~Theta~~   ~~Omega.~~

Iteration ⇒ $f(n) =$ —

$n^2 +$ Big ⇒ $g(n) = (\ )$

$c\, g(n) > f(n)$



arr[] , k

n

```
for (int i=0 ; i<n ; i++) {
    if (arr[i] == k)
        return true;
}
    return false;
```

Calculate Big O

1) Find the number of iteration
2) Ignore lower order terms,

⇒ no of iterations $= 100n + n^2$

n	value	Contribution of $100n$
10	$1000 + 100$	90%
100	$10000 + 10000$	50%
10^5	$10^7 + 10^{10}$	$< 0.1\%$

Ignor Constants

Manoj	Pooja	For large input
n	$100 \log n$	Pooja -
$10n$	$\frac{n^2}{100}$	Manoj
$N \log N$	$1000 \log N$	Pooja.
$10^{10} \log N$	$\frac{n}{10}$	Manoj

Issues in Big O

$n = 10$

Bhappu   Srinivas

$10^3 n$ n^2

According to Big O $\Rightarrow$ Bhapp is better.

$\Rightarrow$ Bhappu algorithm is better for all input sizes $\Rightarrow$ WRONG CLAIM.

n	$10^3 n$	n^2
10	10^4	100
100	10^5	10^4
1000	10^6	10^6

For $n \geq 1000$, Bhappu algorithm is better $\Rightarrow$ CORRECT CLAIM.

Compare $10 \log N$ $10^5 \log N$.

Big O $\Rightarrow$ Both are same.

↓

Check the number of iterations.

Space Complexity

```
void func() {

    int x = 4;      ⇒ 4 bytes
    int y = 5;      ⇒ 4 bytes
    long z = 2L;    ⇒ 8 bytes.

}
```

⇒ 16 bytes. ⇒ O(1)

SC: O(1)

```
void func(int n) {

    int arr[n];     ⇒ 4n bytes.
    int y = 5;      ⇒ 4 bytes
    long z = 2L;    ⇒ 8 bytes.

}
```

⇒ $4n + 12$

SC: O(n)

```
void func ( int n) {

    int arr [n]          = 4n bytes.
    int arr_2 [n][n]     = 4n^2 bytes
    int x, y;       =   8bytes

}
```

$$\Rightarrow \quad 4n^2 + 4n + 8$$

$$SC : O(n^2)$$

```
void func ( int arr[n])

    int x ;
    inty ;

}
```

$$\underset{\substack{\downarrow \\ \text{Input} \\ \text{Space}}}{4n} + \underset{\substack{\downarrow \\ \text{Space} \\ \text{created} \\ \text{by algom.}}}{8}$$

$$\text{Total Space} \Rightarrow \text{Input Space} + \underset{\substack{\Downarrow \\ \text{Space Complexity}}}{\text{Space created by algo}}$$

```
int   max_num  (int arr [], int n) {

    int  max_ans  = -1;

    for (int i=0; i<n ; i++) {

        if (max_ans > arr[i])
                max_ans = arr (i);
    }

    return  max_ans;
}
```

TLE ⇒ Time Limit Exceeded.

Execution time limit ⇒ 1 sec.

Processor Specs ⇒ 1 Ghz

⇒ 10^9 instructions per second.

Q Print all odd numbers till n.

```
void printOdd (int n) {
    for (int i=0 ; i<n ; i++) {
        if (i%2 != 0)
            print (i);
    }
}
```

iterations ⇒ n.

instruction per iteration ⇒ 5.

Total instructions ⇒ 5n.

Case 1 :

1 iterations $\simeq$ 10 instruction.

n iterations $\simeq$ 10n instructions,

Max number of iterations $\Rightarrow$ 10^8

$10n \le 10^9 \Rightarrow \boxed{n \le 10^8}$

Case 2

1 iteration $\Rightarrow$ 100

Max number of iterations $\Rightarrow$ 10^7

In general our code should have at max $[10^7, 10^8]$ iterations

Constraints

n^3

n^2

$n\sqrt{n}$

$n\log 10$

n

$\log n$

$\sqrt{n}$

$$1 \leq n \leq 10^5$$

Sol1 : $O(n^2)$: $\simeq \boxed{10^{10}}$

TLE

Sol2 : $O(n)$ $\simeq 10^5$

Accepted

How to approach a problems

1) Read Question.
2) Read Constraints
3) Guess the Expected time Complexity.
4) Think in that direction.

⇒ Space $\underline{\underline{[10^6, 10^7]}}$ bytes

$n = 10^5$ arr[n][n] ✗

$n = 10^3$ arr[n][n] ✓

```
for (int i = 1 , j=1 ; j<=n ; i++) {
        print (i+j) ;
        if (i%n == 0)
              j++ ;
}
```

$n = 10$

$i = 1, 2, 3, 4 \ldots$

$\ldots \; 10$

j	i	Totale
1	n	n
2	n	n
		n
3	n	n
⋮	n	n
⋮	n	n
n	1	n

n^2

int arr[n]. = O(n).

$\Rightarrow \boxed{\text{logn}}$

SC: O(n)

TC: O(logn)