Time and Space Complexity

Q1 Given an array of size n+1 consisting of integers from 1 to n. One of the elements is duplicate in the array. find the duplicate element.

Soln int arr [n+1]

$O(n^2)$ ←

Observations
① Time Consuming
② Space efficient
↳ O(1)

arr = | 6 | 1 | 2 | 4 | 3 | 2 | 7 | 5 |

Method I Brute force

arr = | 6 | 3 | 2 | 4 | 8 | 1 | 7 | 1 | 5 |
0 1 2 3 4 5 6 7

no of swaps. = 7 + 6 + 5 + 4 + 2 = 24 operations.

```
bool flag = false;
for (int i = 0 , i < n; i++) {
    for (int j = i+1, j < n; j++) {
        if (arr [i] == arr [j]) {
            cout << arr[i];
            flag = true;
            break;
        }
    }
    if (flag == true) break;
}
```

Problem :- We are using extra space. ↳ O(n)

Time efficient - O(n)

Method II arr =

0	1	2	3	4	5	6	7
6	3	2	4	1	7	1	5

int check[8] :

0	1	2	3	4	5	6	7
0	∅	∅	∅	∅	0	∅	∅
	1	1	1	1		1	1

Steps → 0 hai to 1 karo
1 hai to wahi duplicate
Only 7 operations.

* Good Method & it saves time.

Time Complexity → Hardware se koi lena dena nhi hai

Method I

↓

13th Gen i9

↓

faster

Method -2

↓

5th gen i3

↓

slower

• Method III ← using maths

$$1+2+3+4+\cdots+n = \frac{n(n+1)}{2}$$

0	1	2	3	4	5	6	7
6	3	2	4	1	7	1	5

arr =

```
int sum = 0;
for (int i=0; i<n; i++) {
    sum += arr[i];
}
```

Sum ⇒ (29) —

No of operations = 7 + 1

int s = n*(n+1)/2; s = (28) —

cout << sum - s;

Best possible Code

Observations :

1. Time efficient
2. Space efficient

Notations for different types of Time Complexity —

- **Big Oh Notation :** ↓ upper bound

$O(n)$
$O(n^2)$
$O(\log n)$
$O(n^3)$
$O(2^n)$
$O(1)$

where n → is usually the size of array / data structure

Q1 Calculate the TC for iterating in a loop.

```
for (int i = 0; i<n ; i++) {
        cout << "PhysicsWallah \n";
}
```

↳ n times loop chlega

Ans – O(n) Sorted

Q2 What if this time we increament the pointer by 2 ?

```
for (int i = 0; i<n ; i+= 2) {
        cout << "Garima \n";
}
```

⇒ ∴ n/2 iterations / rounds / operations.

∴ **Ans** $O\left(\frac{n}{2}\right) \approx O(n)$ Sorted

- $O(kn) \approx O(n)$ (k is constant) — using this formula.

Q3 for(int i=1; i<=n-7; i++) {
    cout << "GG\n";
}

Soln n-7 iterations

⇒ T.C @ O(n-7) ≃ O(n)

{ • O(n ± k) ≃ O(n) }

① O(5n³ + 3) = O(5n³) = O(n³)

② O(6n² - 8) = O(6n²) = O(n²)

③ O(n² + n) = O(n²)

{ • ~~O(k x^n ± k x^{n-1}~~

{ • O(k₁ n^m ± k₂ n^{m-1} ± k₃ n^{m-1}) ≃ O(n^m) }

(4) $O(11n^{13/2} + 7n^4 - 2n^3 + 6n) = O(n^{13/2})$

Q4 Calculate the time Complexity for traversing 2 arrays of size n and m.

```
int a[n], b[m];
for(int i=0; i<n; i++) { } → 'n' times → O(n)
    a[i]++;
}
for(int i=0; i<m; i++) { } → 'm' times → O(m)
    b[i]++;
}
```

Sol^n | T.C = $O(n+m)$

Q5 Calculate the TC of following code.

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

Sol^n No of iterations → $n \times n = n^2$

| T.C = $O(n^2)$ |

Q6 T.C —

```
for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
        cout << "44";
    }
}
```

→ (n-1) → (n-1)

Total ops = (n-1) * (n-1) = (n-1)²

↓

n² + 1 - 2n

O(n² - 2n + 1)

O(n²) Ans

(2)

Q7 T.C —

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j++) {
        cout << "44";
    }
}
```

n

Total ops = n * (n-1)

T.C = O(n²) Ans

Sol^n $i = 1 \rightarrow j = 1$ to $i = 1$

$i = 2 \rightarrow j = 1, 2 \rightarrow 2$

$i = 3 \rightarrow j = 1, 2, 3 = 3$

⋮

$i = n \rightarrow j = 1, 2, 3 \cdots n : n$

No of iterations $\rightarrow 1 + 2 + 3 + \cdots n$

$= \frac{n * (n+1)}{2} = \frac{n^2 + n}{2}$

$T.C = O\left(\frac{n^2}{2} + \frac{n}{2}\right) \approx O\left(\frac{n^2}{2}\right) \approx \boxed{O(n^2)}$

Q T.C —

```
for (int i = 1; i <= n; i++) {
    for (int j = i, j <= n; j++) {
        cout << "44";
    }
}
```

$i = 1 \rightarrow j = 1$ to n

$i = 2 \rightarrow j = 2$ to n

$i = 3 \rightarrow j = 3$ to n

⋮

$i = n \rightarrow j = n$ to n

No of iter^n $= (1 + 2 + 3 \cdots n) + (2 + 3 + 4 \cdots n) \cdots + (n - n)$

$= 1 + 2 + 3 + \cdots + (n-1) + n$

$$\frac{n \cdot (n+1)}{2}$$

$$O\left(\frac{n^2}{2} + \frac{n}{2}\right) \simeq O(n^2)$$

Q8 T.C —

```
for (int i = 0; i<n; i++) {
    for (int j = 0; j<m; j++) {
        cout << "GG";
    }
}
```

Sol^n

i = 0 → j = 0, 1, 2, ... m-1
i = 1 → j = 0, 1, 2, ... m-1
n

i = n-1 → j = 0, 1, 2 ... m-1

No of iteration = m + m + m ..... m
= n * m

T.C $O(n * m)$ Sol^n

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

Q9 Calculate T.C —

```
int c = 0                          (k < < n)   (k ≠ 1)  k, 2, 3, 4, 5, ——
for (int i = 1 ; i ≤ n ; i *= k) {
    c++;
}
```

Soln eg. $n = 100$, $k = 2$ → no of ops. = 7

$i = 1, 2, 4, 8, 16, 32, 64, 128$ khatam

$i = 1, k, k^2, k^3, k^4 \ldots\ldots K^x$

→ This loop will end when $k^x > n$

[no of iterations → no of times 'i' is changing]

$= k^x = n$

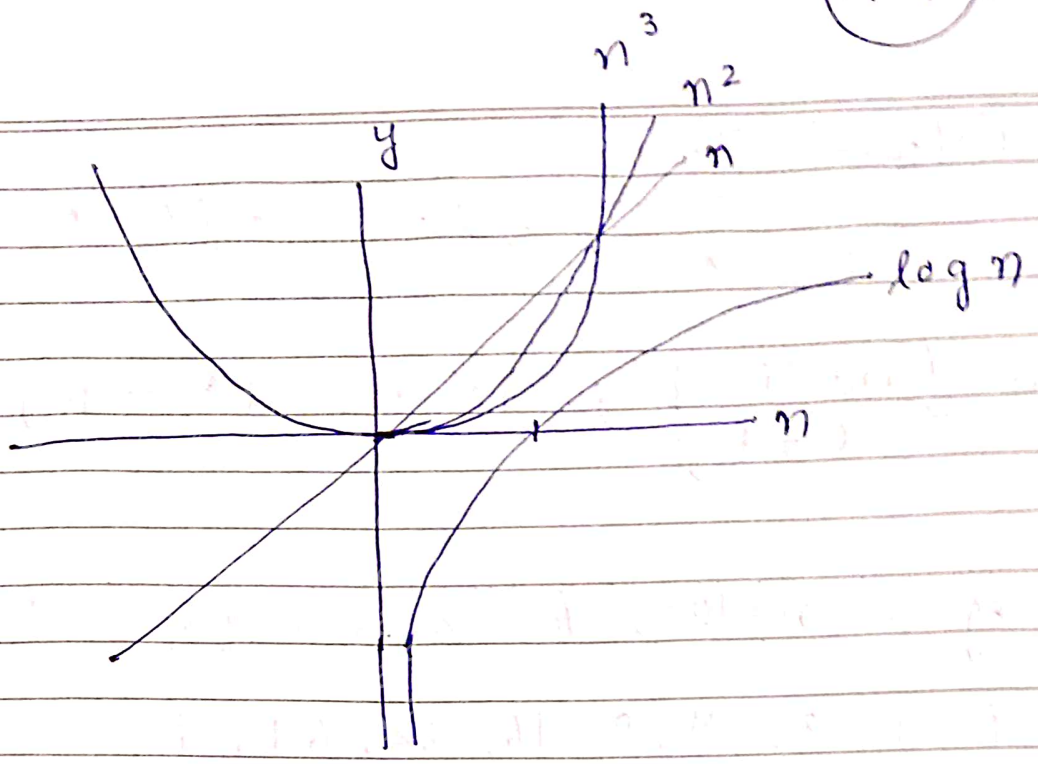
* $a^b = c$
  $\Rightarrow \log_a c = b$

$= \log_k n = x$

$$T.C = O(\log_k n)$$

$$T.C = O(\log n)$$

$(n>1)$



Q10 Calculate T.C —

```
int c = 0;
for (int i = 0; i<n; i++) {
    for (int j = i+1; j<m; j++) {
        c++;
    }
}
```

$m-1$ ⇒ $i = 0 \rightarrow j = 1, 2, 3 \dots m-1$

$m-2$ ⇒ $i = 1 \rightarrow j = 2, 3, 4 \dots m-1$

$m-3$ ⇒ $i = 2 \rightarrow j = 3, 4, 5 \dots m-1$

$\vdots$ $\quad$ $\vdots$

$m-n$ ⇒ $i = n-1 \rightarrow j = n, n+1, \dots m-1$

Total no of iter^n = $(m-1) + (m-2) + (m-3) - \dots - \dots$
$\dots + [m-(n+1)]$

$S = \frac{N}{2}[a_1 + a_2]$ $\quad$ ↳ $(n+1)$ terms

$$S = \frac{N}{2}\left[m-1+m-n-1\right]$$

$$S = \frac{n}{2}\left[2m-n-2\right]$$

$$\Rightarrow O\left[\frac{n}{2}\left[2m-n-2\right]\right]$$

$$\Rightarrow O(n*m-n^2) \approx O(m*n)$$

2nd Method

$i=0 \rightarrow j = 1, 2, 3 \ldots\ldots m-1 : m-1$

$i=1 \rightarrow j = 2, 3, 4 \ldots\ldots m-1 : m-2$

$\vdots$

$i=n-1 \rightarrow j = n, n+1, n+2 \ldots\ldots m-1 : m-n$

Total it^n $= (m-1)+(m-2)+(m-3) \ldots\ldots +(m-n)$

$= \underbrace{(m-1)+(m-2)+\ldots(m-n)}_{n \text{ terms}} + \underbrace{(m-n-1)+\ldots 2+1}_{(m-n-1) \text{ terms}}$

$$-\left(\overbrace{(m-n-1)+\ldots 2+1}^{m-1}\right)$$

$$S = \frac{(m-1)(m-1+1)}{2} - \frac{(m-n-1)(m-n-1+1)}{2} \approx m*n$$

Space Complexity: Study of all extra space used in terms of given 'n', 'm'

Q1 Calculate SC —

```
int a[n];
for (int i = 0; i<n; i++) {
    a[i]++;
}
```

Soln → No. of ops = n

T.C = O(n)

S.C = O(n)

Q2 What will be the space complexity if we just traverse without creating any array?

```
int c = 0;
for (int i = 0; i<n; i++) {        ↘ No extra space
    c++;
}
```

T.C → O(n)

S.C → O(1)

Q3 Calculate SC of nested loop

```
vector<int> a;
vector<int> b;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        a[i]++; b[j]++;
    }
}
```

```
a.push_back(10);
b.push_back(5);
```

TC : $O(n*m)$

SC : $O(n*m)$

Q4

```
int a[n];
int b[m];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        a[i] = i;
        b[j] = j;
    }
}
```

or

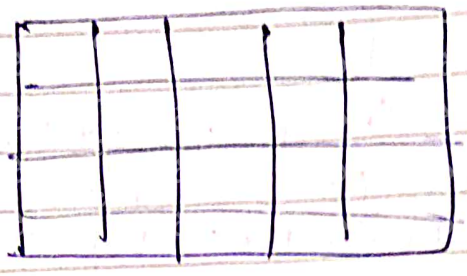
```
vector<int> a(n);
vector<int> b(m);
```

TC = $O(n*m)$

SC = $O(n+m)$

Space Complexity of creating a 2d matrix —

int arr [m][n];
↳ rows ↳ col



cells = m * n

SC : O(m * n)

What will be the space complexity if we create 3 arrays of the same size?

```
int a[n], b[n], c[n];
for(int i=0; i<n; i++) {
    c++;
}
```

T.C : O(n)
S.C = O(n+n+n)
= O(3n)
S.C = ≈ O(n)

Q. Calculate TC & SC

```
int a[n][n/2];
for(int i=1; i<n; i*=2) {
    for(int j=0; j<n/2; j++) {
        a[i][j]++;
    }
}
```

Space use : $n * \frac{n}{2} \Rightarrow \frac{n^2}{2}$

S.C $\rightarrow O\left(\frac{n^2}{2}\right) \approx O(n^2)$ ✓

T.C $\rightarrow O(\log_2 n * \frac{n}{2})$

$\boxed{T.C = O(n * \log n)}$ ✓