Time and Space Complexity

Lecture- 19

Raghav Garg

Ques : Given an array of size n+1 consisting of integers from 1 to n. One of the elements is duplicate in the array. Find that duplicate element.

int arr[n+1] Ex: arr =

6	1	2	4	3	2	7	5

M-I: Brute Force

Method-1 (Brute Force) arr =

6	3	2	4	1	7	1	5

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

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

Observations

1) Time Consuming → $O(n^2)$

2) Space Efficient → $O(1)$

Another method for previous problem

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	1	1	1	1	0	1	1

Steps →
- 0 nai to 1 karo
- 1 nai to wahi duplicate
- Only 7 operations

Good Method ∆ it saves time

Problem : we are using extra space

Observations
1) Time Efficient : O(n)
2) Space Consuming → O(n)

* Time Complexity → Hardware se koi lena dena nahi hai

Method - 1

↓

13th Gen i9

↓

faster

Method - 2

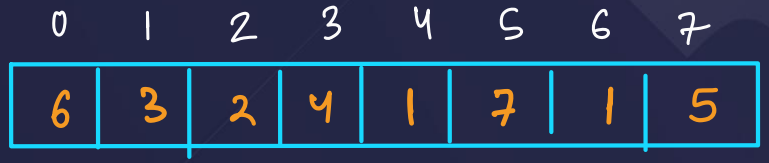
↓

5th gen i3

↓

slower

Solving the previous problem using maths




```
int sum = 0;
int n = arr.size() - 1;
for(int i= 0; i<= n ; i++){
|    sum += arr[i];
3
int S = n*(n+1)/2 ;
cout << sum - S ;
```

No. of ops = 7 + 1

Sum ⇒ 29
S = 28

29 - 28 = 1

Observations :

1) Time Efficient : O(n)

2) Space Efficient : O(1)

NOTE : Time Complexity can only be calculated on same programming device

Notations for different types of Time Complexity → Jyada dhyaan nahi denge → Omega, Theta of

Big Oh Notation : $O(n)$

$O(n^2)$

$O(\log n)$

$O(n^3)$

$O(2^n)$

$O(1)$

↓

upper bound

where → n is usually size of array / data structure

Ques : Calculate the time complexity for iterating in a loop.

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

5 operations ?

'n' times loop chalega

Ans : O(n)

n = 5

PW
PW
PW
PW
PW

What if this time we increment the pointer by 2?

$n = 10$

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

$i = 0$ 2 4 6 8 10

PW
PW
PW
PW
PW

$\Rightarrow \frac{n}{2}$ iterations / rounds / ops

T.C. $O(\frac{n}{2}) \approx O(n)$

$O(Kn) \approx O(n)$
(K is constant)

```
for( int i = 1 ; i <= n-7 ; i++){
|   cout << "PW";
}
```

n-7 iterations

$\Rightarrow$ T.C. $O(n-7) \approx O(n)$

$O(n \pm K) \approx O(n)$

1) $O(5n^3 + 3) = O(5n^3) = O(n^3)$

2) $O(6n^2 - 8) = O(n^2)$

3) $O(6n^2 + n) = O(n^2)$

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

$$O(k_1 n^m \pm k_2 n^{m-1} \pm k_3 n^{m-2}) \approx O(n^m)$$

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

→ 'n' times → O(n)

'm' times → O(m)

T.C. = O(n+m)

Q// Calculate the T.C. of this given code

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

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

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

SKILLS

Q11 Calculate the T.C. of this given code

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

$i=1 \rightarrow j=1 \text{ to } i \rightarrow 1 \text{ to } 1 : 1$

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

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

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

$\vdots$

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

No. of iterations $\Rightarrow 1+2+3\cdots+n$

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

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

C.W. : Calculate the T.C. of this given code

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

What if this time we traverse them in a nested manner?

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

→ 'm' times

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

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

⋮

$i = n-1 \rightarrow j = 0, 1, 2 .. m-1 : m$

No. of iterations : $n * m$

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

*Ques : Calculate the time complexity for the below code snippet.

$k \neq 1$

$k = 2, 3, 4, 5, 6 \ldots$

```
int c = 0;
for(int i = 1; i ≤ n; i*=k) {
   c++;
}
```

Ex: $n = 100$, $k = 2$ → no. of ops → 7

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

Khatam

Ques : Calculate the time complexity for the below code snippet.

no. of iterations → no. of times 'i' is changing

```
int c = 0;
for(int i = 1; i ≤ n; i*=k) {
   c++;
}
```

T.C. = O(n)

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

→ this loop will end when $k^x > n$

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

$= O(\log n)$

$= k^x = n$

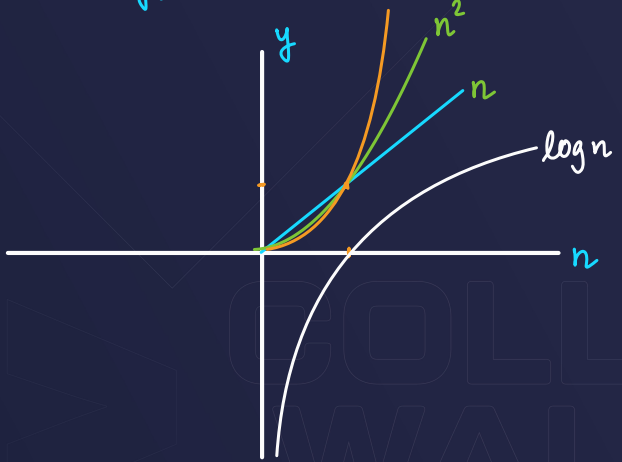
$= \log_k n = x$

Logarithms : $a^b = c$

$\Rightarrow \log_a c = b$

$n > 1$

$n^3 > n^2 > n > \log n$



Ques : Calculate the time complexity for the below code snippet.

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

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

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

$i=2 \rightarrow j=3,4,5 \ldots m-1 : m-3$

$\vdots$

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

$\therefore m-n-1$

total no: $(m-1)+(m-2)+(m-3) \ldots + [m-(n+1)]$ → $(n+1)$ terms

$$S = \frac{N}{2}[a_1 + a_2] = \frac{n}{2}[m-1+m-n-1]$$

$$S = O\left[\frac{n}{2}[2m-n-2]\right] \Rightarrow O(n*m-n^2) \approx O(m*n)$$

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

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

$m-3$

$\vdots$

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

total iterations : $(m-1) + (m-2) + (m-3) \ldots (m-n)$

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

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

$-\left[(m-n-1) + \ldots 2 + 1\right]$

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

Ques : Calculate the space complexity for the below code snippet.

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

No. of ops : n

T.C. : O(n)

S.C. : O(n)

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

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

No extra space

T.C. → O(n)

S.C. → O(1)

Ques : Calculate the space complexity for the below nested loop code snippet.

```
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);

m*n elemen

n times

m times

T.C. : O(n*m)

S.C. : O(n*m)

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

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

T.C. = O(m*n)

S.C. = O(n+m)

Space Complexity of creating a 2d matrix

```
int arr[m][n];
```

rows cols

cells : $m * n$

S.C. = $O(m * n)$

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)

≈ O(n)

Ques : Calculate the time and space complexity for the below nested loop code snippet.

Space used : $n * \frac{n}{2} = \frac{n^2}{2}$

S.C. → $O(\frac{n^2}{2}) \approx O(n^2)$

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

$\log_2 n$

$\sim \frac{n}{2}$ operations

T.C. → $\frac{n}{2} * \log_2 n$

$O(n)$

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

$i = 1, 2^1, 2^2, 2^3 \cdots 2^x \rightarrow x$ ops

$2^x = n$

$\log_2 n = x$

A.P., G.P., Basic Math → Log

↓

Easy ♡

Doubts

↓

Form

↓

Telegram

Thank you!

↓

Maza aa gaya