TIME COMPLEXITY

Time Complexity of the algorithm is an indicator of how the execution time depends on the size of data structure.

Beginner Level (do refer to this playlist for concepts)

Problem 1: Constant Time Complexity

```
int getFirstElement(int arr[], int size)
    return arr[0];
}
```

here the code runs only once as output is the first element itself

⇒ Time Complexity = $T(n) = O(1)$ = constant time complexity

Problem 2: Linear Time Complexity

```
int sumElements(int arr[], int size) {
    int total = 0;
    for (int i = 0; i < size; i++) {
        total += arr[i];
    }
    return total;
}
```

this however only executes with $T(n) = O(1)$ since it's only incrementing

this loop will run for (n+1) times because the 'i' increments everytime the loop condition is satisfied and loop is executed. to check if 'i' still belongs to the array (i.e. within the condition), the code will iterate the loop (n+1) times.

⇒ Time Complexity = $T(n) = O(n)$ = Linear time complexity.

$\uparrow = (n+1) + 1 \Rightarrow n+2 \Rightarrow O(n)$

Problem 3: Linear Time Complexity with Conditionals

```
int findFirstEven(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        if (arr[i] % 2 == 0) {
            return arr[i];
        }
    }
    return -1;
}
```

executed (n+1) time (similar logic as previous)

executed $\left(\frac{n}{2}\right)$ times because the probability of 'if' loop condition getting satisfied is 50% (i.e. $1/2$) and total no. of times the loop executed = n.

simple print/return statement $\therefore T(n) = O(1)$

⇒ Time complexity = $T(n) = O(n)$ = Linear time complexity

$\uparrow = (n+1) + n/2 \Rightarrow \frac{2n+2+n}{2} = \frac{3n+2}{2} = O(n)$

Key Takeaway

1. RULES OF CALCULATING TIME COMPLEXTIY:

- Drop constant multipliers (i.e. coefficients)

for eg: in Problem 3: total time = $\frac{3n+2}{2}$ ← here constant multipliers/coefficients are $\frac{3}{2}$ for 'n'.

⇒ according to the Rule, the multipliers/coefficients must be dropped. i.e. their existence is negligible because we only care about the degree/order of time complexity and not about the magnitude.

- Drop lower order terms

considering the same example; the lower order terms here refer to be constants since n^0 = constant and that's the smallest order here

⇒ constants in $\frac{3n+2}{2} = \frac{2}{2}$ ⇒ which must be neglected/dropped according to the rule.

- Running time of program = Running time of all fragments

- Running time of program gives magnitude of time through which the order of time complexity can be calculated by applying the above rules.