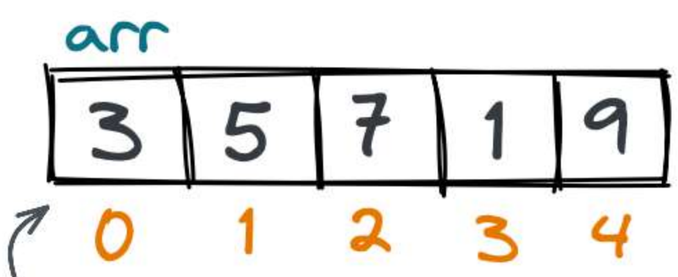
Time Complexity Simply Explained

→ We can find the time taken by an algorithm by using the frequency time method.



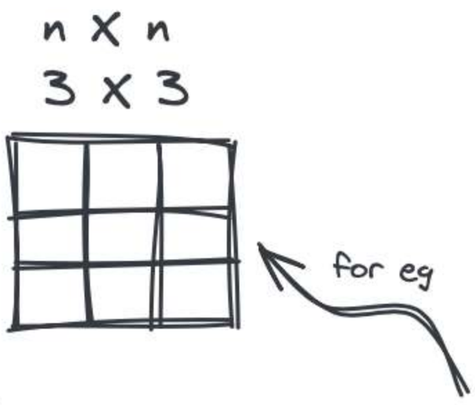
Checks how many times the iteration happens.
This case the loop will end at index 5, which is (n + 1).

```
def sum_array(arr):
    sum = 0                  ——→ 1
    for x in arr:            ——→ n + 1
        sum += x             ——→ n
    return sum               ——→ 1
```

$f(n) = 2n + 3$

The degree of this polynomial is 1.
Therefore, the time complexity here is order of n.

$O(n)$



Sum of two square matrices

```
def matrix_sum(A, B):
    n = len(A)
    C = [[0 for i in range(n)] for j in range(n)]
    for i in range(n):                          ——→ n + 1
        for j in range(n):                      ——→ n x (n + 1)
            C[i][j] = A[i][j] + B[i][j]         ——→ n x n
    return C                                    ——→ 1
```

first loop
second loop

$f(n) = 2n^2 + 2n + 2$

The degree of this polynomial is 2.
Therefore, the time complexity here is order of n^2

$O(n^2)$

Matrix Multiplication

```
def matrix_multiply(A, B):

    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0])
    C = [[0 for _ in range(cols_B)] for _ in range(rows_A)]

    for i in range(rows_A):                          ⟶ n + 1
        for j in range(cols_B):                      ⟶ n (n + 1)
            for k in range(cols_A):                  ⟶ n x n x (n + 1)
                C[i][j] += A[i][k] * B[k][j]         ⟶ n x n x n

    return C
```

$$f(n) = 2n^3 + 2n^2 + 2n + 1$$

The degree of this polynomial is 3.
Therefore, the time complexity here is:

$$O(n^3)$$

Don Joe Jacob

Data Scientist

linkedin.com/donjoejacob twitter.com/taurean_joe