Revision

→ Data Structure
- Linear
- Non-Linear

Recursion & Backtracking

→ Algorithm
- D&C
- Greedy
- DP

Module 1 → complexity analysis

- Apriori analysis
- Apostiary analysis

Time complexity

↳ Loop

↳ No Loop → O(1)

↳ Complexity classes (Increasing)

Space complexity → Extra space (Data Structure)

Logarithmic Properties | Example

① $\log_a m^n = n * \log_a m$ ⟶ $\log_2 2^4 = 4$

$4 \log_2 2 = 4$

② $\log_a a = 1$

③ $\log_n m = \frac{1}{\log_m n}$ ⟶ $\log_2 10 = \frac{1}{\log_{10} 2}$

④ $\log_a (mn) = \log_a m + \log_a n$

⑤ $\log_a (m/n) = \log_a m - \log_a n$

$\log_2 (21/8) = \log_2 21 - \log_2 8$

⑥ $\log_b m = \frac{\log_a m}{\log_a b}$

Example

$\log_{10} 23 = \frac{\log_2 23}{\log_2 10}$

Asymptotic Notation

Omega

Best case

(Ω)

$f(n) = \Omega(g(n))$

$f(n) >= c \cdot g(n)$

LHS

Big O

Worst case

(O)

$f(n) = O(g(n))$

$f(n) <= c \cdot g(n)$

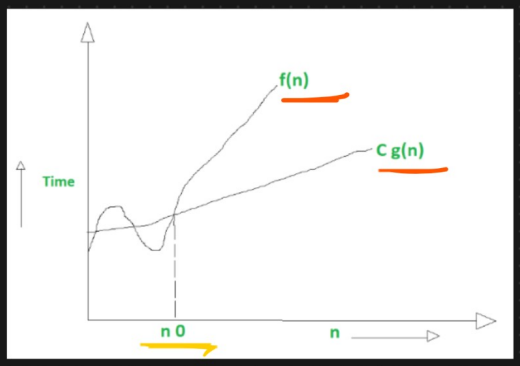
Theta

Average case

(Θ)

↳ Both omega & big O satisfied

Omega



Example 1

$f(n) = n$

$g(n) = 5n$

$n_0 >= 1$

$c > 0$

True ← $f(n) = \Omega(g(n))$

$f(n) >= c \cdot g(n)$

$n >= c \cdot 5n$ ← $c = 1/5$

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

false

$f(n) = \Omega(g(n))$

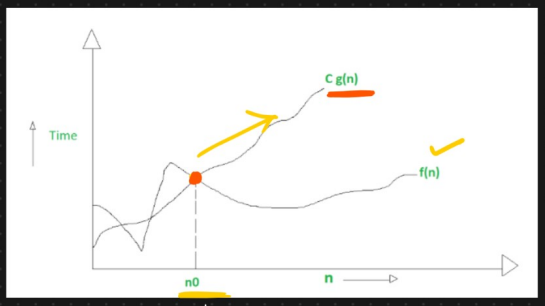
$n >= c \cdot n^2$

inverse proportion to n

$c = 1/n$

Big O → Worst case

Mathematical Idea



Time ↑

10 ↓ n →

$f(n) = O(g(n))$

$f(n) <= c \cdot g(n)$

$n \geq n_0$
$c > 0$
$n_0 \geq 1$

f(n) = n-1 / n-2 / n+4 / 5n / 7n

→ O(n)

$f(n) = 5n$

$g(n) = n$

$f(n) = O(g(n))$

$f(n) \leq c \cdot g(n)$

$5n \leq c \cdot n$ — True

$c \geq 5$ — any constant

$f(n) = n$

$g(n) = 5n$

True

$f(n) = O(g(n))$ RHS

LHS

$f(n) <= c \cdot g(n)$

$n <= c \cdot 5n$

True

$n <= 1/5 \cdot 5n$

$c = 1/5$

Example 3

$f(n) = n^2$

$g(n) = n$

False

$f(n) = O(g(n))$

$f(n) <= c \cdot g(n)$

$n^2 <= c \cdot n$

$c = n$

Constant?

$n^2 \neq O(n)$

$f(n) \mid g(n)$

Reason

Example 4

$$\begin{cases} f(n) = n^3 \\ g(n) = n \end{cases}$$

$n^3 = O(n)$ ← 4

↓

Not true

$f(n) \propto c \cdot g(n)$

$n^3 \propto c \cdot n$

↓

$c = n^2$

↳ variable

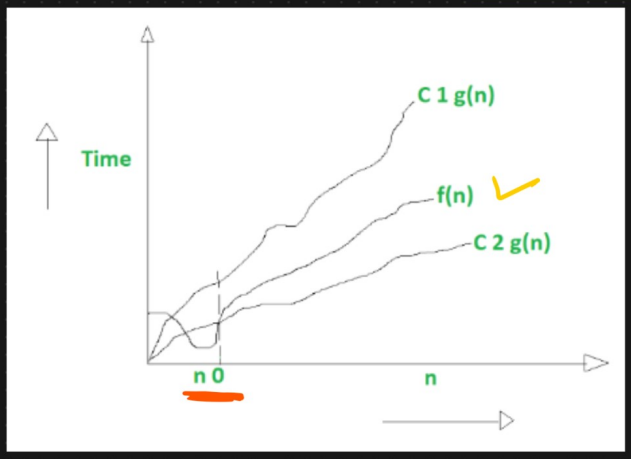
$f(n) = n^2$

$g(n) = n^2$

True

$f(n) = O(g(n))$

Theta Notation



Big O

$f(n) <= c_1 \cdot g(n)$ — (1)

$f(n) >= c_2 \cdot g(n)$ — (2)

Omega

$f(n) = n$, $g(n) = 5n$

$f(n) = \Theta(g(n))$ — True

Big O

$f(n) <= c_1 \cdot g(n)$

$n <= c_1 \cdot 5n$

$c_1 = 1/5$

constant

Omega

$f(n) >= c_2 g(n)$

$n >= c_2 \cdot 5n$

$c_2 = 1/5$

Big O — Resource allocation

$f(n) = O(g(n))$

$f(n) <= c \cdot g(n)$

↓

constant

Omega — Conclusion

$f(n) = \Omega(g(n))$

$f(n) >= c \cdot g(n)$

Theta

$f(n) = O(g(n))$

$f(n) = \Omega(g(n))$

↓

Web application

↓

Response time

n_0 — Constant ($n \longrightarrow$ very high)

$n_0 < n$

$n \geq n_0$

Array → Linear Data Structure

Data Structure

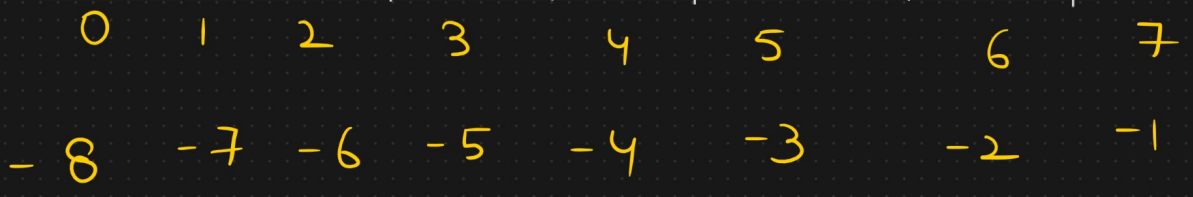
Python

List

Numpy (2D Array)

arr

1D array



n = 8

Random Access

accessing any element in an array → O(1)

6th element

arr[5] = 27

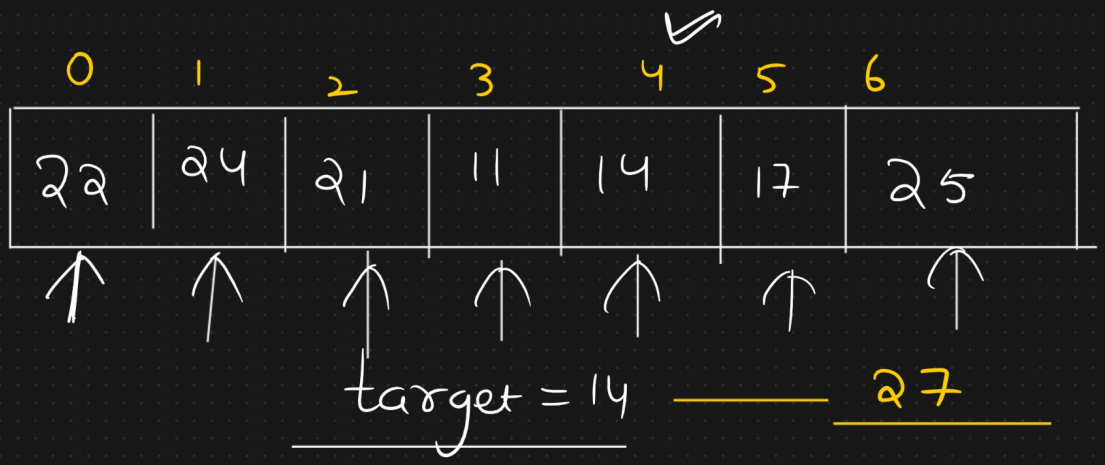
arr[-1] = 37

Coding

1D Array

① Sum of all elements in an array

② Searching Algorithms



Output = 4

```
def LinearSearch (arr, n):
    for i in range(n):
        if arr(i) == target:
            return i
    return -1
```

Linear Search

Time complexity

O(n)

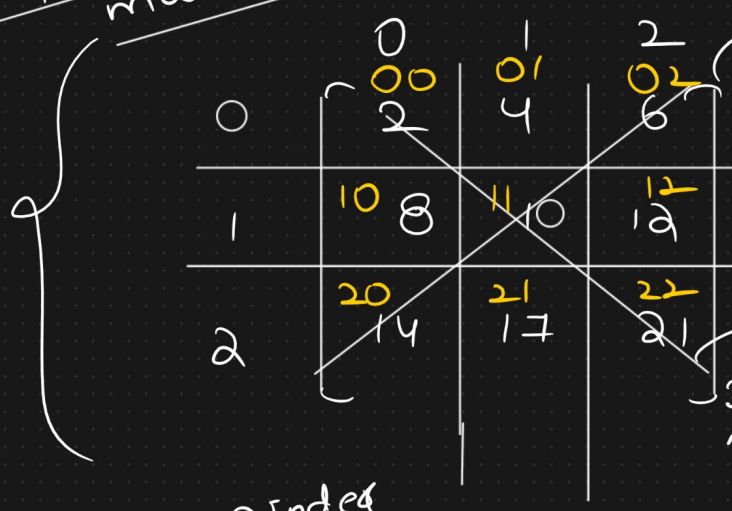
-1 indicates that the target value is not available

in the array

Space complexity → O(1)

2D Arrays → Rows / columns

Square matrix



Secondary diagonal → 6+10+14

arr[rowIndex][colIndex]

Primary diagonal → 3x3 / nxn

↳ 2+10+21 = 33

rowIndex, colIndex

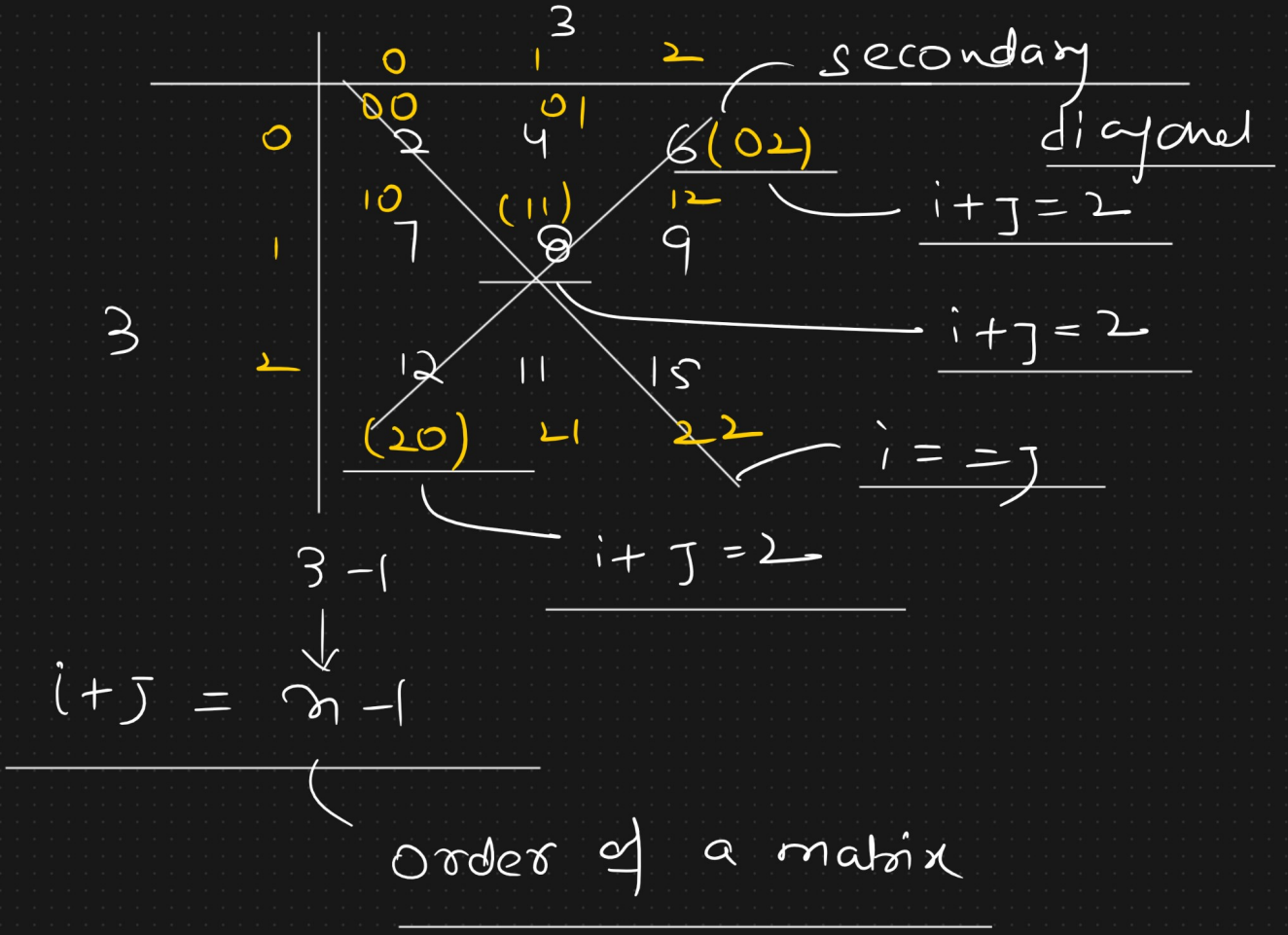
3-1=2

i+j = n-1

↳ Secondary diagonal elements

i == j

primary diagonal elements


3
0
1
2
secondary
diagonal
00
01
0
2
4
6(02)
10
(11)
12
1
7
8
9
i+j=2
3
i+j=2
2
12
11
15
(20)
21
22
i==j
i+j=2
3-1
i+j = n-1
order of a matrix