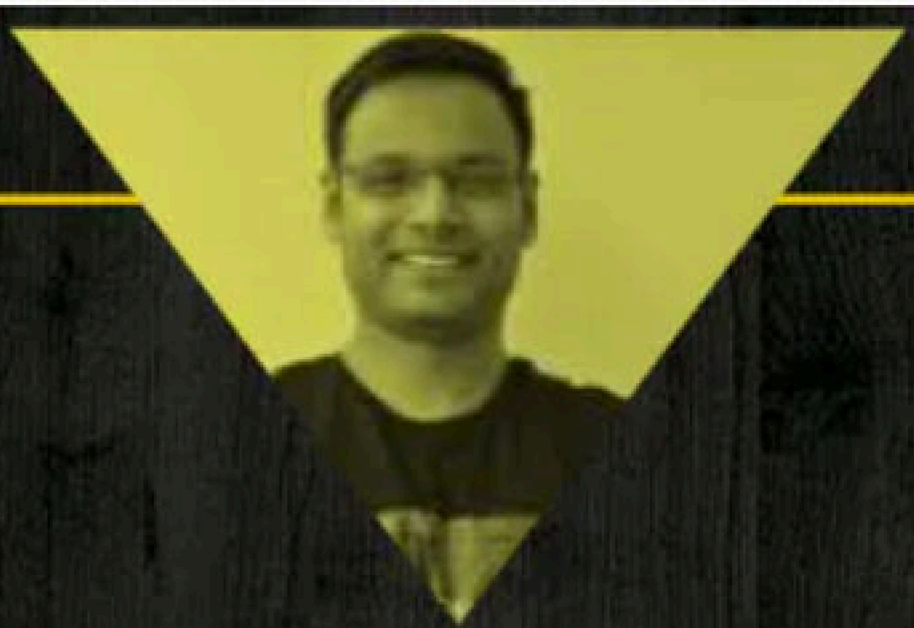
Array in Data Structure

Course on C-Programming & Data Structures: GATE - 2024 & 2025

Vishvadeep Gothi • Lesson 21 • Dec 20, 2022

Data Structure: Asymptotic Notations & Array

By: Vishvadeep Gothi



Hello!

I am Vishvadeep Gothi

I am here because I love to teach

Vishvadeep Gothi: Profile

- **GATE Ranks:**
  - 682 (2009) – 3rd year
  - 19 (2010) – 4th year
  - 119, 440 etc.
- **Education:**
  - ME from IISc Bangalore
  - Mtech from BITS-pilani in Data Science
- **Work:**
  - 15 Year Teaching Experience
  - 12+ in GATE/IES (GateForum, Gate Academy, ACE)
  - Worked in Cisco, Audience Communication
- **Professions:**
  - Freelance S/W developer
  - Educator
  - CrossFit Trainer

Analysis of Algorithm

- Space Complexity
- Run-Time Complexity

Analysis of Algorithm

ex:-

Input => int A[n] = { 6, 19, 20, 5, ... };

complicity
⇓
upper bound => n

```
for(i = 0; i < n; i++)      — 1
{
  if(A[i] == 15)            — 1
  {
    return i;               — 1
  }
}
return -1;
```

→ 3

O(n)

Asymptotic Notation

↳ used for bounding complexities

Big O :- O ⇒ It provides tightest upper-bound $O(n)$

$O(\log n)$

Omega :- Ω ⇒ It provides tighest lower-bound $\Omega(n)$

$\Omega(n \log n)$

Theta :- Θ ⇒ It provides exact bound

$\Theta(n) \Rightarrow \Omega(n)$ and $O(n)$

Asymptotic Notation

Constant complexity $\Rightarrow$ $O(1)$ or $\Theta(1)$

Types of cases (Types of inputs)

① Best case :— Type of input for which, algo takes min. time

② worst case :— — " ——— " ——— " —— max time

③ Avg case :— The input which is not best or worst

Question 1

Consider an algorithm which takes n number of inputs and performs an operation on it, which requires n-1 operations. The best possible run time complexity for the algorithm can be represented as:

(A) $O(n)$

$\theta(n)$

(B) $\Theta(n)$

(C) $O(n \log_2 n)$

(D) A & B both

Question 2

Consider an algorithm which takes n number of inputs and performs an operation on it. The operation is performed by algorithm in such a way that it is not dependent on number of inputs. Which of the following can be the run time complexity for the algorithm?

(A) $O(1)$

(B) $\Theta(1)$

(C) $O(n)$

(D) All

Array

- Collection of homogeneous elements
- Characteristics:
  1. All elements stored on consecutive memory locations
  2. All elements can be accessed using a set of indexes

Array

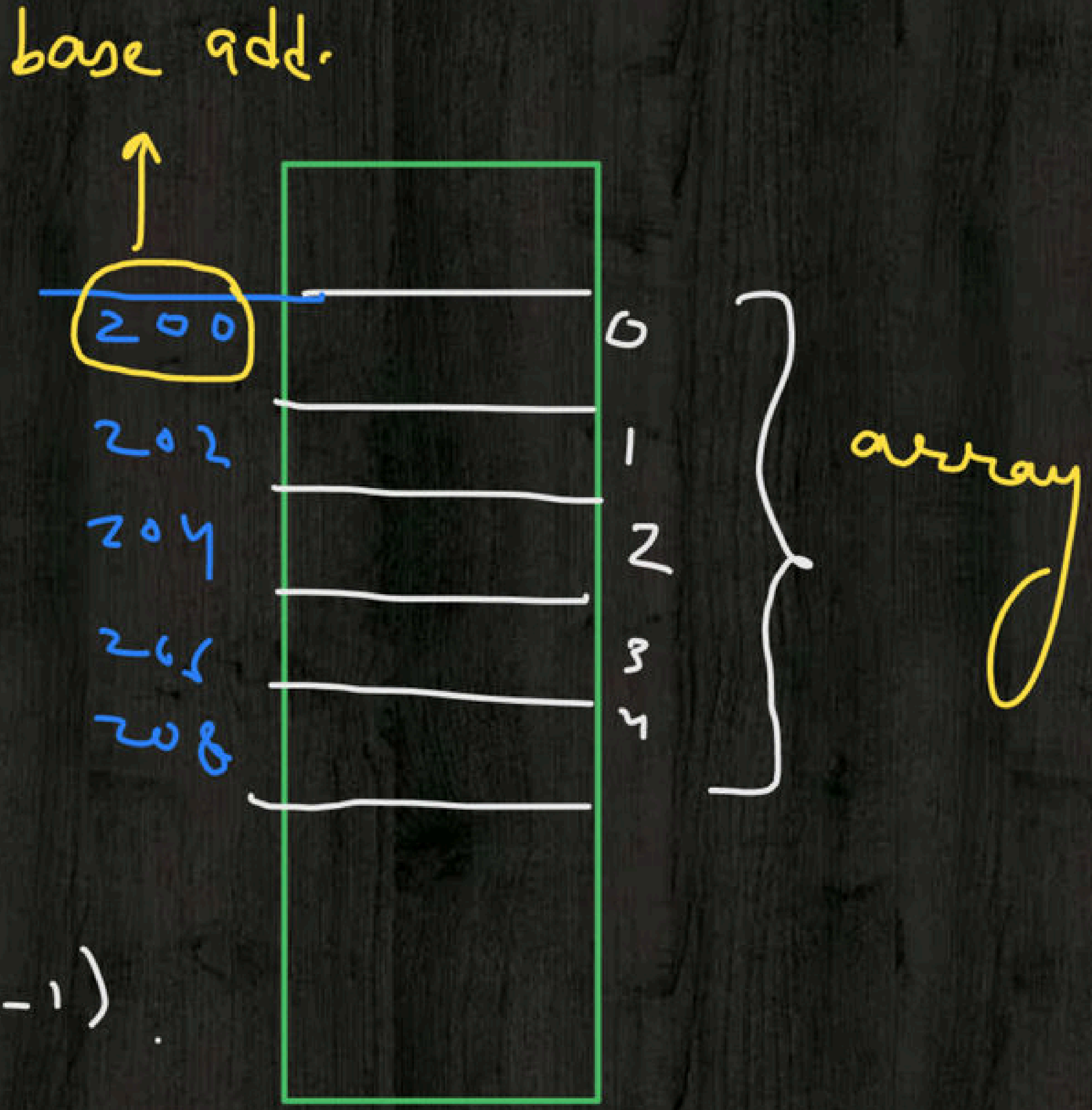
In C-programming :-

datatype name[size];

int A[5];

Lower bound (LB) => starting index = 0

Upper bound (UB) => last index = 4 (size-1).



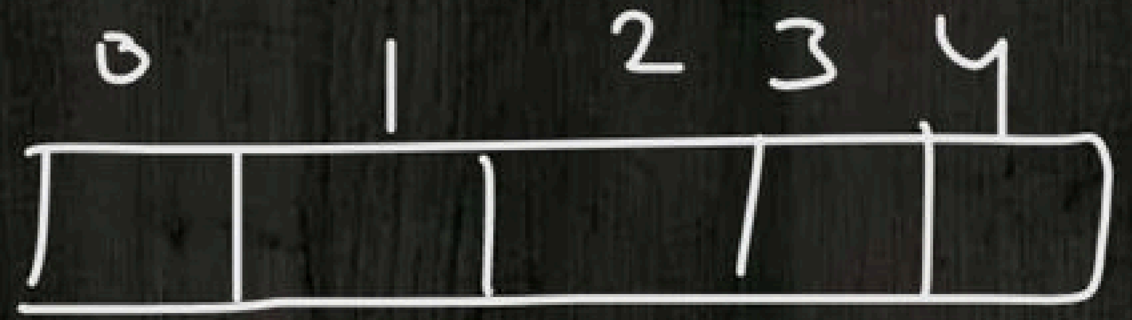
Array

Data - structure :-

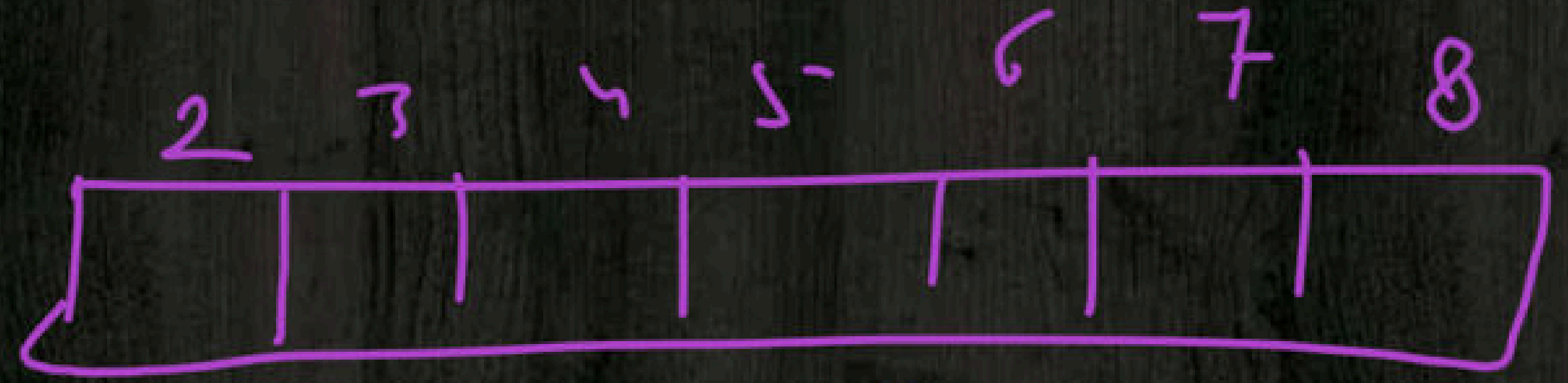
name [LB : UB]

Size = UB - LB + 1

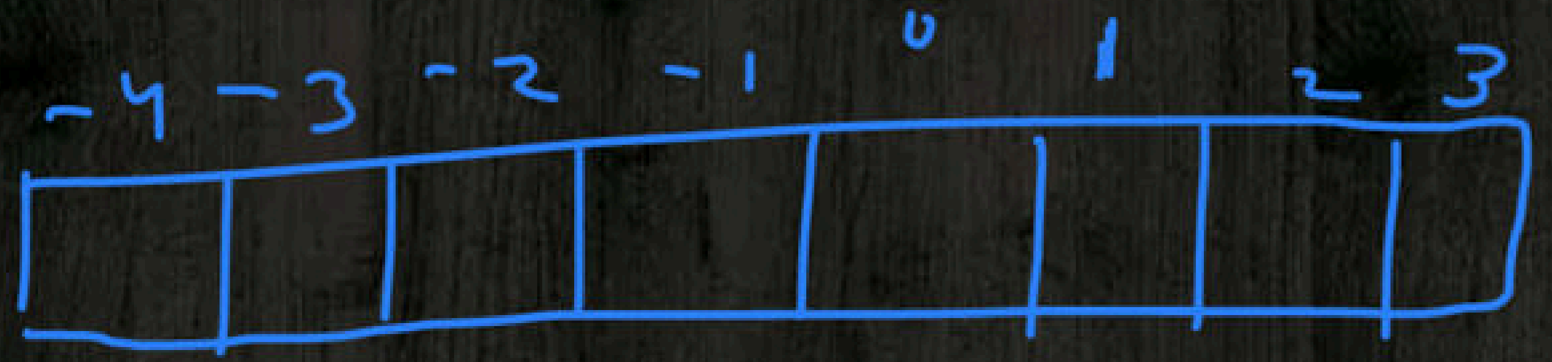
A [0 : 4]



B [2 : 8]



C [-4 : 3]



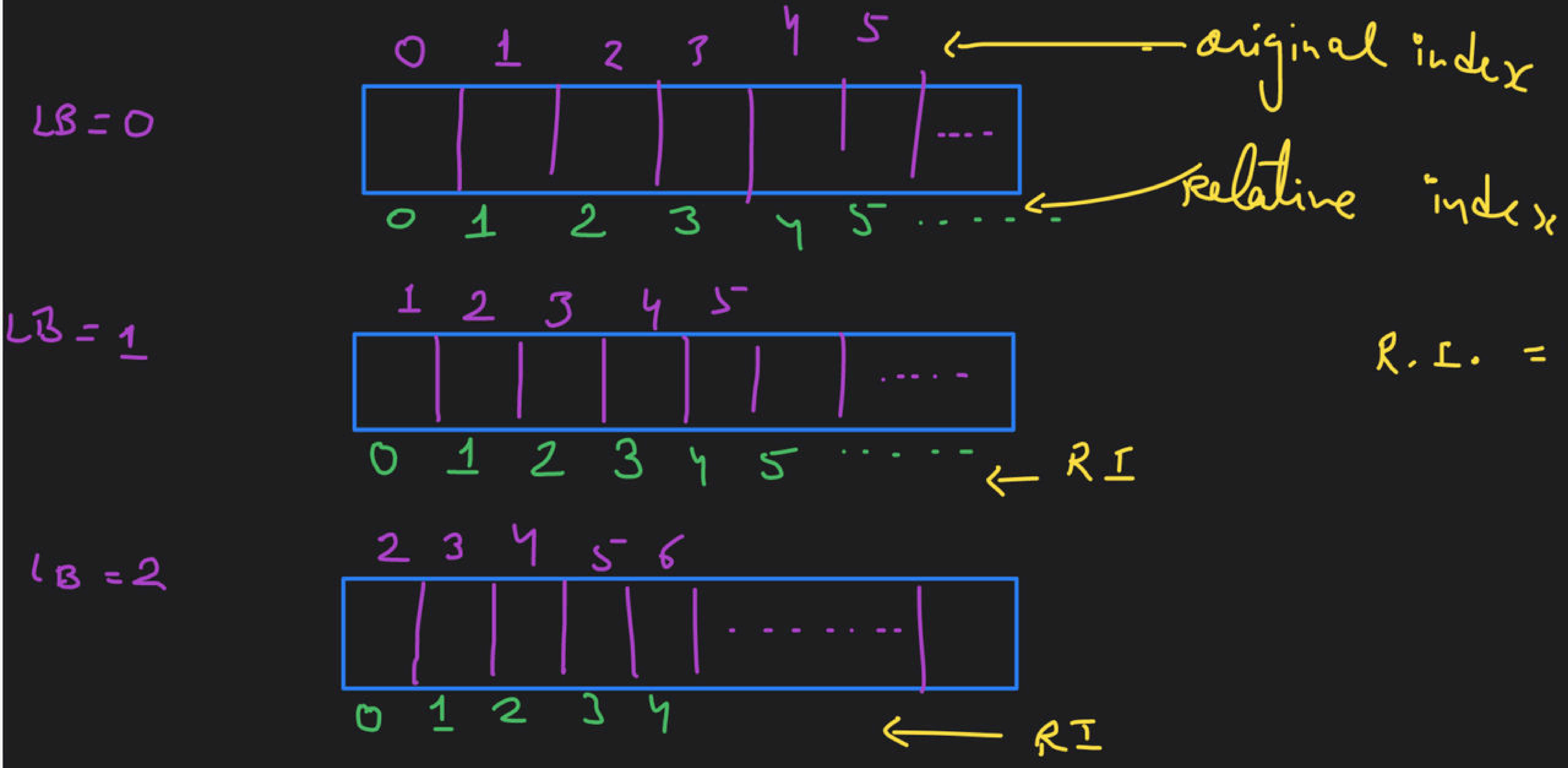
Location of an Array Element

C-prog. :-

Base add. ⇒ base

Locatn (A[i]) = Base + size of element in memory * i

Relative index of any element x (R. I.) $\Rightarrow$ No. of elements stored in memory before x.



R. I. = Index − LB

Location of an element = Base + size of an element in memory (ω) * Relative index

$$\boxed{\text{location}(A[i]) = \text{Base} + \omega * [i - LB]}$$

Ques) Consider an array A[-4 : 200], which is stored in memory from location 2500. Each element takes 4 locations in memory. The location of array element A[17] is?

Ans)

$$= 2500 + 4 * (17 - (-4))$$

$$= 2584$$

Ans

Ques) Array A[-6 : 13]

each element occupies ⇒ 8 locations in memory (1 location = 1 Byte)

(1) total no. of elements in array = ?

$13 - (-6) + 1 = 20$ elements

(2) Size of memory required to store complete array = ____ Bytes ?

$\Rightarrow 20 * 8\,B = 160\,B$

Why Indexing from Zero?

To save $(i - LB)$ calculation time, everytime an array element accessed

performance improvement

Traversing in Array


```
for(i=0; i<n; i++)
{
  process A[i]
}
```

```
for k = LB to UB, step by 1
    process A[k]
```

Run time complexity $\Rightarrow \Theta(n)$

Happy Learning