Array

1D Array → [2, 4, 6, 8, 12, 15, 20, 27, 37] → arr

0 1 2 3 4 5 6 7 8 — index

⌐1008

↳1004

Address of first element 1000 ↳ Base address

int → 4 Bytes

arr[6] ↳ 20

Using base address we can access all the elements of the array arr[6] → 1000 + (6 − 0) * 4

1000 + 24 = 1024

(contiguous storage of an array elements) (Index)

(By default → 0)

Memory address

Lower bound is the index of base element which by default is 0 ⟹ arr(i) = Base address + (i − Lower Bound of an index) * size of an element

Random access Primary feature of an array

2D Array → {Rows, columns}

↳ Matrix computation

{ i → Row index
J → column index

$$\begin{array}{c|ccc} & 0 & 1 & 2 \rightarrow \text{columns} \\ \hline \text{rows} \rightarrow 0 & 2 & 4 & 6 \\ 1 & 8 & 10 & 12 \\ 2 & 14 & 17 & 20 \end{array}$$

20 → arr[2] [2]

arr[0](1)

Row major

[2, 4, 6], [8, 10, 12], (14, 17, 20)

↳ rows

⌐ columns

Column major

[2, 8, 14], (4, 10, 17], [6, 12, 20]

1000

↓

0	1	2	3	4	5	6	7	8
2	4	6	8	10	12	14	17	20
2	8	14	4	10	17	6	12	20

→ Row major form

→ CMF (Col major form)

1) Row major form → Row-wise

2) column major form → column-wise

In terms of memory allocation even 2D arrays are stored in 1D using either row major approach or column major approach.

At visual level there can be multiple dimensions but at memory level we are storing element always in 1D where multiple dimensions are converted into 1D either using row major or column major approach. By default it is following row major approach.