Data Structure and Algorithms-I

Arrays: Memory Mapping, Linear and Binary Search, Linear Time Sorting (Counting Sort)

Lecturer Saifur Rahman, Dept. of CSE, United International University

Arrays

- An array is an indexed sequence of components
  - The components of an array are all of the same type
- Typically, the array occupies sequential storage locations
- Array is a static data structure, that is, the length of the array is determined when the array is created, and cannot be changed
- Each component of the array has a fixed, unique index
  - Indices range from a lower bound to an upper bound
- Any component of the array can be inspected or updated by using its index
  - This is an efficient operation: $O(1)$ = constant time

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
7	6	11	17	3	15	5	19	30	14

Representation of Arrays in Memory

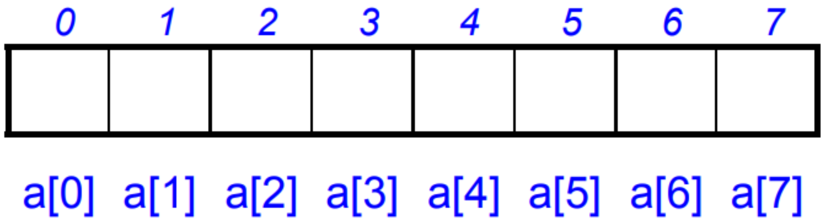
- **Linear (1 D) Arrays:**

  A 1-dimensional array a is declared as:

  int a[8];

  The elements of the array a may be shown as

  a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]



Representation of Arrays in Memory

- **2 D Arrays:**

A 2-dimensional array a is declared as:

```
int a[3][4];
```

The elements of the array a may be shown as a table

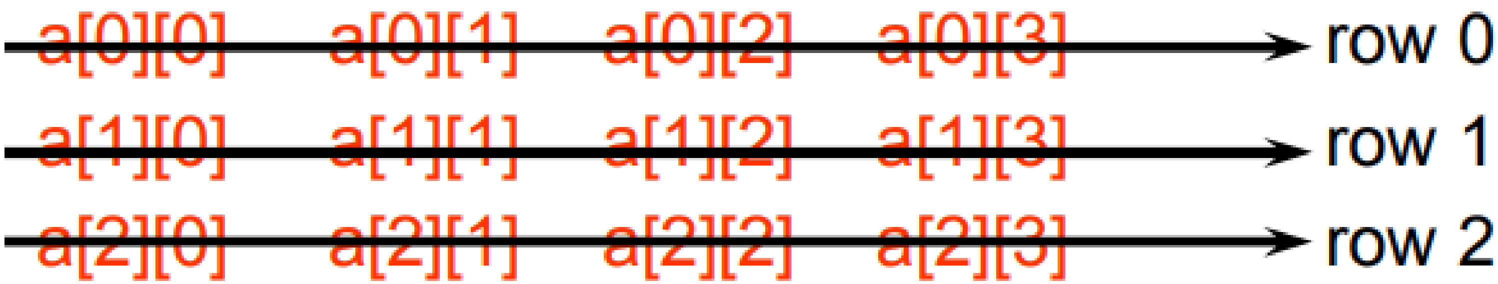
a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

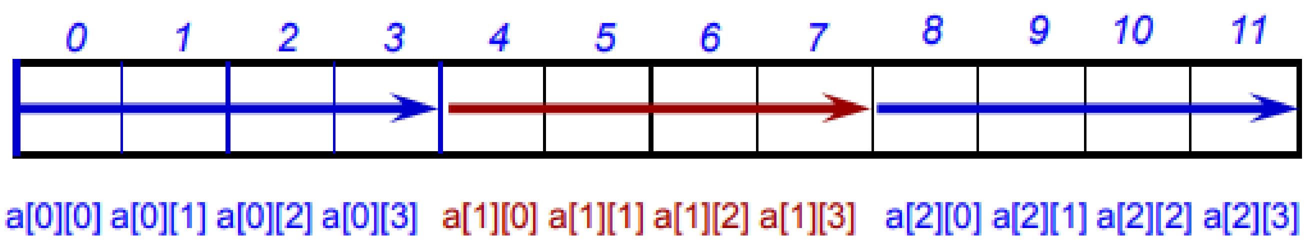
In which order are the elements stored?

- Row major order (C, C++, Java support it)
- Column major order (Fortran supports it)

Representation of Arrays in Memory

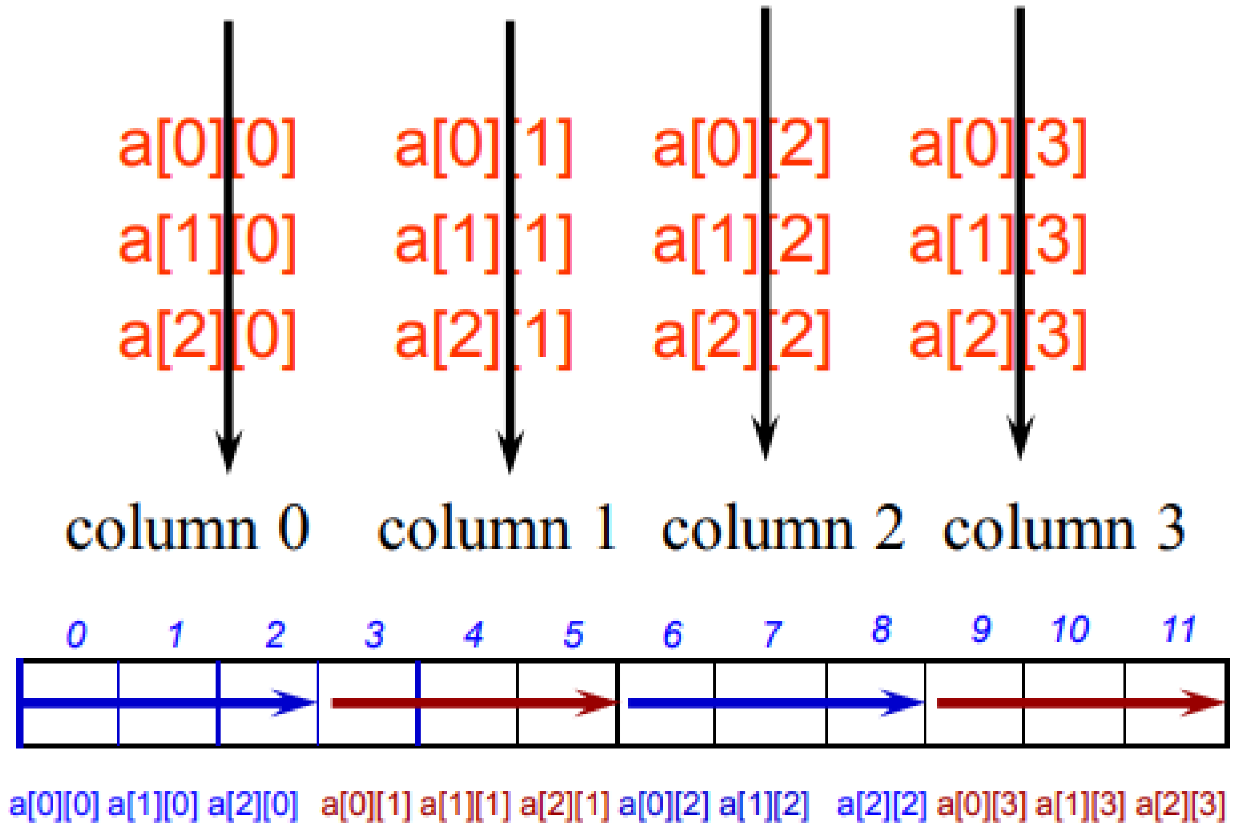
Row Major Order: the array is stored as a sequence of 1-D arrays consisting of rows





Representation of Arrays in Memory

Column Major Order: The array is stored as a sequence of arrays consisting of columns instead of rows



Representation of Arrays in Memory: Parameters

- **Base Address (*b*):** The memory address of the first byte of the first array component.
- **Component Length (*L*):** The memory required to store one component of an array.
- **Upper and Lower Bounds (l_i, u_i):** Each index type has a smallest value and a largest value.
- **Dimension**

Representation of Arrays in Memory

- **Array Mapping Function (AMF)**
  - AMF converts index value to component address

- **Linear (1D) Arrays:**

  a : array $[l_1 \,..\, u_1]$ of element_type

  $$\text{Then addr}(a[i]) = b + (i - l_1) \times L$$
  $$= c_0 + c_1 \times i$$

  Therefore, the time for calculating the address of an element is same for any value of i.

Representation of Arrays in Memory

- **Array Mapping Function (AMF): 2D Arrays**

  Row Major Order:

  a : array $[l_1 \,..\, u_1, l_2 \,..\, u_2]$ of element_type

  Then $\text{addr}(a[i,j]) = b + (i - l_1)\times(u_2 - l_2 + 1)\times L + (j - l_2)\times L$

  $$= c_0 + c_1 \times i + c_2 \times j$$

  Therefore, the time for calculating the address of an element is same for any value of (i, j).

Representation of Arrays in Memory

- **Array Mapping Function (AMF): 2D Arrays**

  Column Major Order:

  a : array $[l_1 \,..\, u_1, l_2 \,..\, u_2]$ of element_type

$$\text{Then addr}(a[i,j]) = b + (j - l_2)\times(u_1 - l_1 + 1)\times L + (i - l_1)\times L$$
$$= c_0 + c_1\times i + c_2\times j$$

Therefore, the time for calculating the address of an element is same for any value of (i, j).

Representation of Arrays in Memory

- **Array Mapping Function (AMF): 3D Arrays :**

  a : array $[l_1 \,..\, u_1, l_2 \,..\, u_2\,, l_3 \,..\, u_3]$ of element_type

Then $\text{addr}(a[i, j, k]) = b + (i - l_1)\times(u_2 - l_2 + 1)\times(u_3 - l_3 + 1)\times L + (j - l_2)\times(u_3 - l_3 + 1)\times L + (k - l_3)\times L$

$$= c_0 + c_1\times i + c_2\times j + c_3\times k$$

Therefore, the time for calculating the address of an element is same for any value of $(i, j\,, k)$.

Summary on Arrays

- **Advantages:**
  - Array is a random access data structure.
  - Accessing an element by its index is very fast (constant time)
- **Disadvantages:**
  - Array is a static data structure, that is, the array size is fixed and can never be changed.
  - Insertion into arrays and deletion from arrays are very slow.

- An array is a suitable structure when
  - a lot of searching and retrieval are required.
  - a small number of insertions and deletions are required.