General Guideline

Module Objective

An array is linear list in coded form. It is a structure used to store data of similar type in static manner. In this module we will learn various types of array, their indexing formulas . We will also learn various operation of 1 D & 2 D Array.

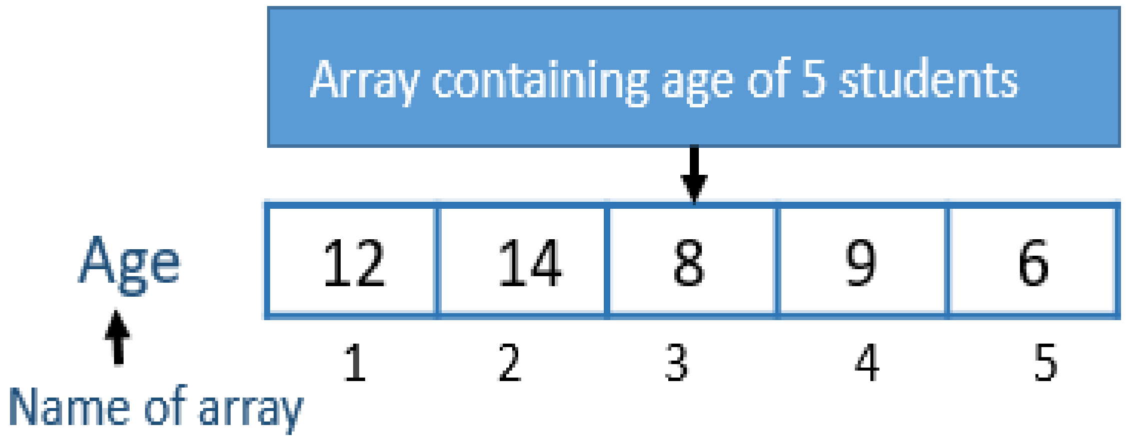
References

- “Fundamentals of **data structure** in C” Horowitz, Sahani & Freed, **Computer Science**.
- “Fundamental of **Data Structure**” (Schaums Series)
- Robert Kruse, Data Structures and Program Design , Prentice Hall, 1984

ABES
Estd. 2000

Introduction to Array

An Array is a Data Structure which can be defined as a finite ordered set of Homogenous elements



Question:

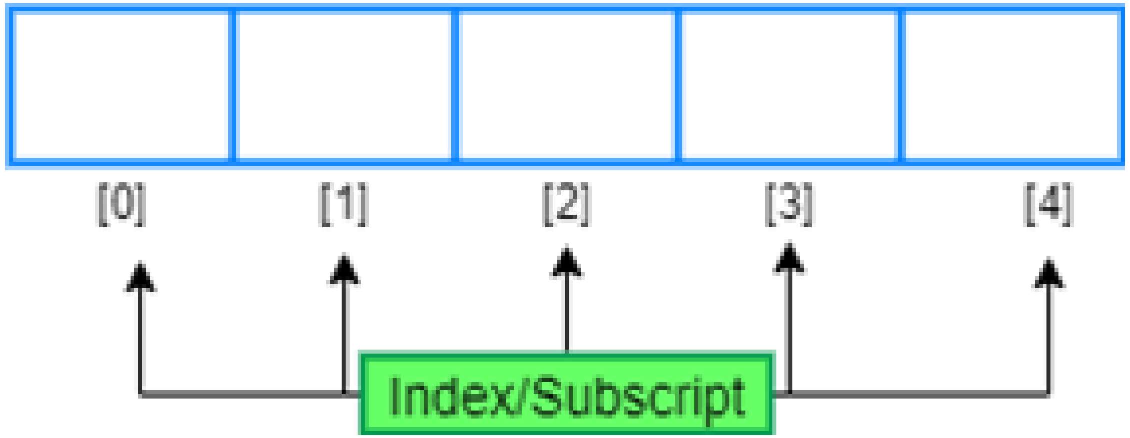
What is an array?	
A. An array is a series of elements of the same type in contiguous memory locations	B.An array is a series of element
D. None of the mentioned	C. An array is a series of elements of the same type placed in non-contiguous memory locations

Properties of Array

Finite means fixed element , Arrays have a fixed size where the size of the array is defined when the array is declared. In the below given figure, the size of the array is fixed i.e., the size 5 is fixed and we cannot add one more element in the array

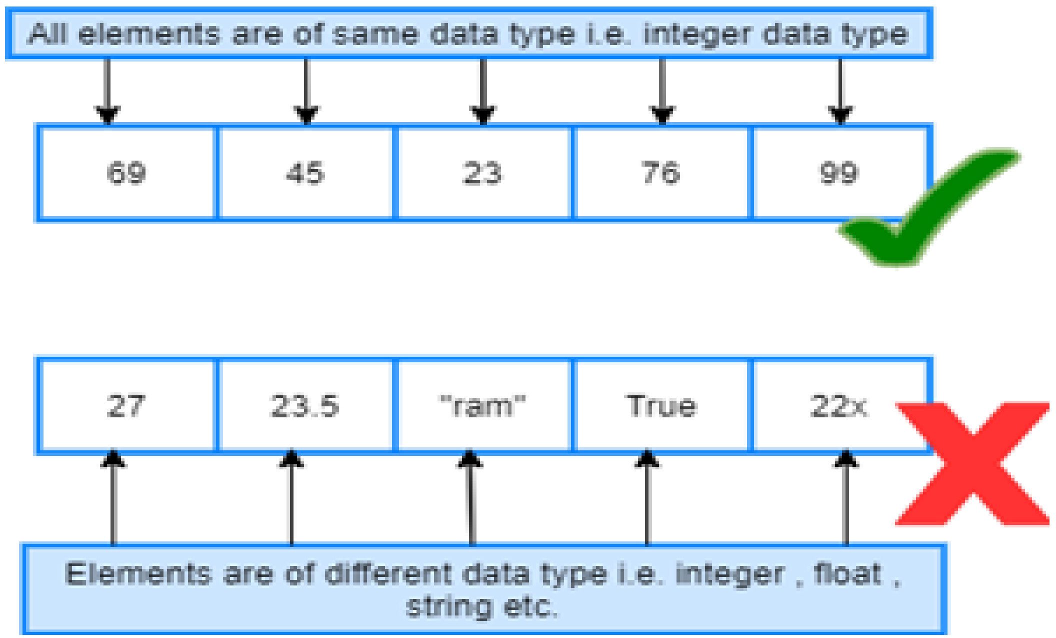
Properties of Array

Ordered Set means every number will be in Sequence and will be denoted by numbers called Index (“indices” in plural) or ”subscript.



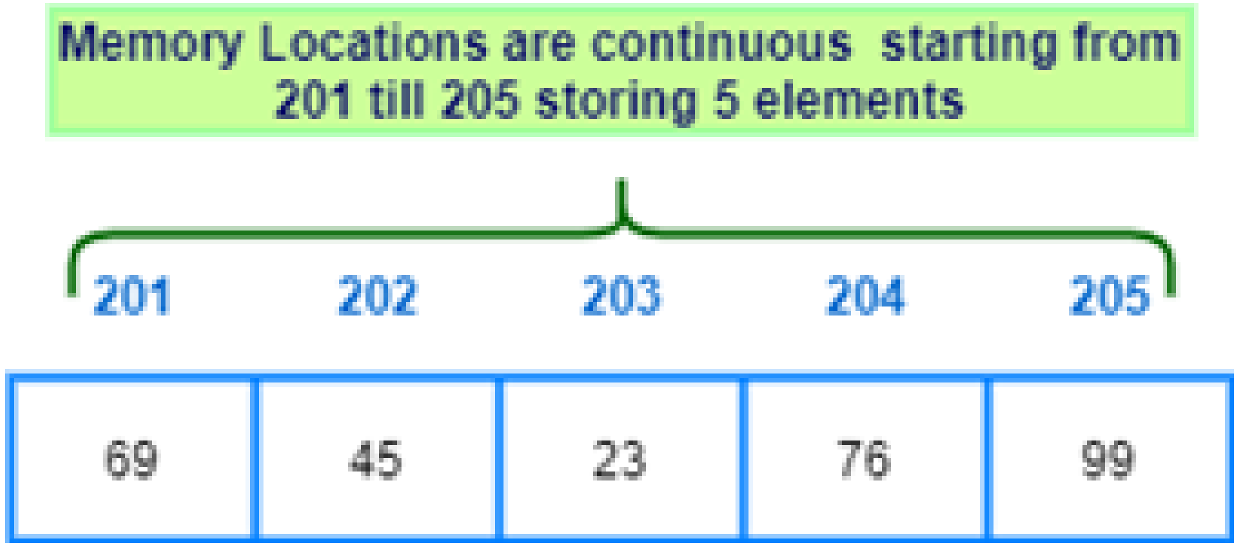
Properties of Array

Homogenous elements means Data type of all the elements will be same



Properties of Array

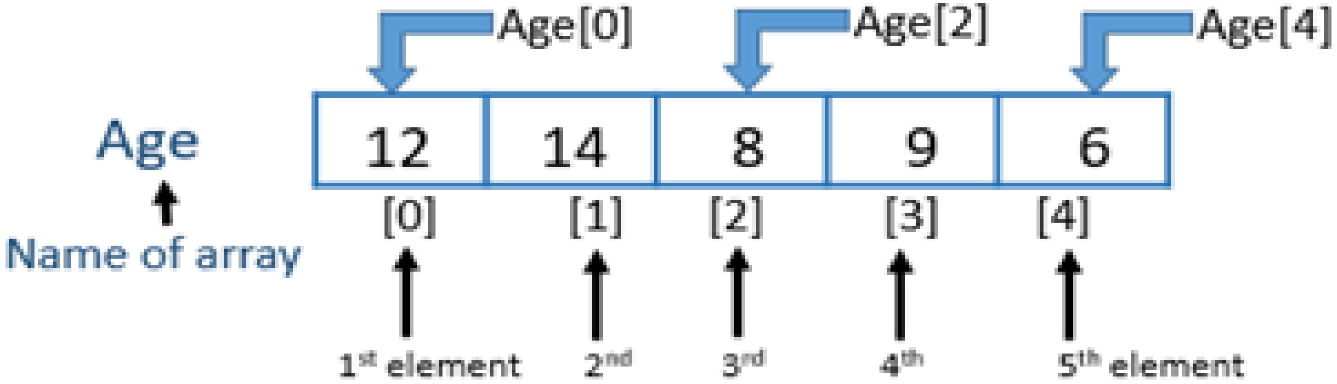
Contiguous Memory allocation



Properties of Array

Random Access,

The general syntax to access an element is: <*name_of_array>[<index>]*



Continued...

There are 3 types of indexing provided by different languages to access the array.

<u>0 (zero-based indexing):</u> The first element of the array is indexed by a subscript 0. The index of nth element is “n-1” in this case. (C, C++, etc.).

<u>1 (one-based indexing):</u> The first element of the array is indexed by the subscript 1. (Basic, MATLAB, R, Mathematica

<u>n (n-based indexing):</u> The base index of an array can be freely chosen. Usually, programming languages allowing n-based indexing also allow negative index values. (Fortran, Pascal, ALGOL, etc.)

Types Of Array

The array is represented in various ways based on the number of dimensions

1. **One-Dimensional array**
2. **Two-Dimensional array**
3. **Multi-Dimensional array**

One-Dimensional Array

A one-dimensional array (or single dimension array) is an array with one subscript only.

Declaration of one-dimensional array

Syntax:

<Data Type> <Arrayname> [<Array_Size>]

Example:

Declaration of one-dimensional array "A" having "int" data type and size 10 elements in C.

int A[10]

One-Dimensional Array

Accessing the element in one-dimensional array

Accessing its elements involves a single subscript.

Syntax: *Arrayname[index]*

Example:

To access 2nd element in the array A, we write *A[1]*

To access 9th element in the array A, we write *A[8]*

(Here, we that assume the first index is 0)

Memory Representation of One-Dimensional Array

The memory representation of one-dimensional array is shown in below diagram.

Index	0	1	2	3	4	5	6	7	8	9
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
Address	100	104	108	112	116	120	124	128	132	136

In the above diagram A[0], A[1], A[2],. . . , A[9] are the array elements. The address mentioned for these elements represent the physical location of data elements in the main memory. It is assumed that each element requires 4 bytes for storage in this scenario.

Question:

A one dimensional array is always considered as____?	
A. Complex	B. Sequential
D. Both C and B	C. Linear

Question:

```
int main()
{
static int ary[ ] = {1, 3, 5};
printf("%d %d", ary[-1], ary[5]);
return 0;
}
```

What is the output of C Program with arrays?	
A. 0 0	B. -1 -1
D. None of the above	C. Compile error

Question:

What is the index number of the last element of an array with 9 elements?	
A. 9	B. 8
D. Programmer-defined	C. 0

Question:

Which of the following gives the memory address of the first element in array?	
A. array[0];	B. array[1];
D. array;	C. array(2);

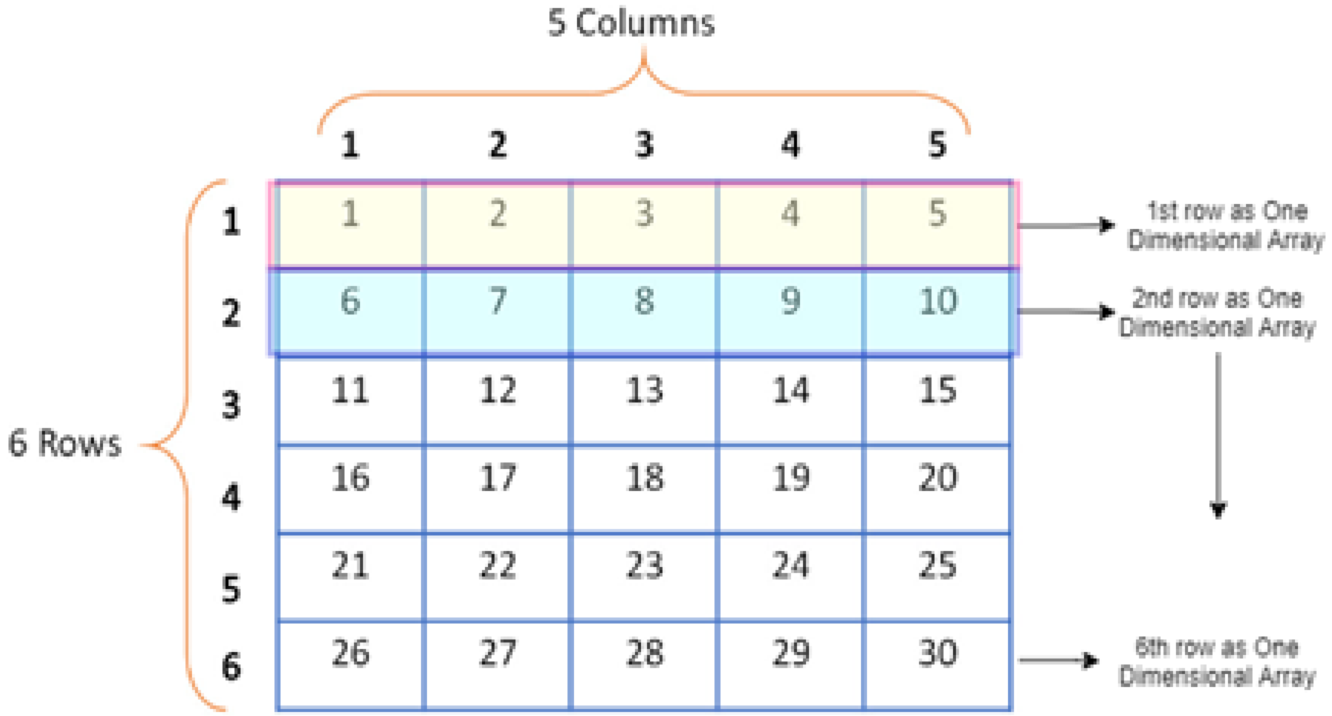
Question:

The information about an array used in program will be stored in	
A. Symbol Table	B. Activation Record
D. Both A and B	C. Dope Vector

Two-Dimensional Array

A two-dimensional array (2-D array) has two subscripts. The elements are stored in the form of rows and columns. It can also be termed as an array of one-dimensional arrays.

Matrix is an example of two-dimensional array



Continued...

Declaration the two-dimensional array:

Syntax: *<Data Type> <Arrayname> [m][n]*

Where,

m = Number of rows

n = Number of columns

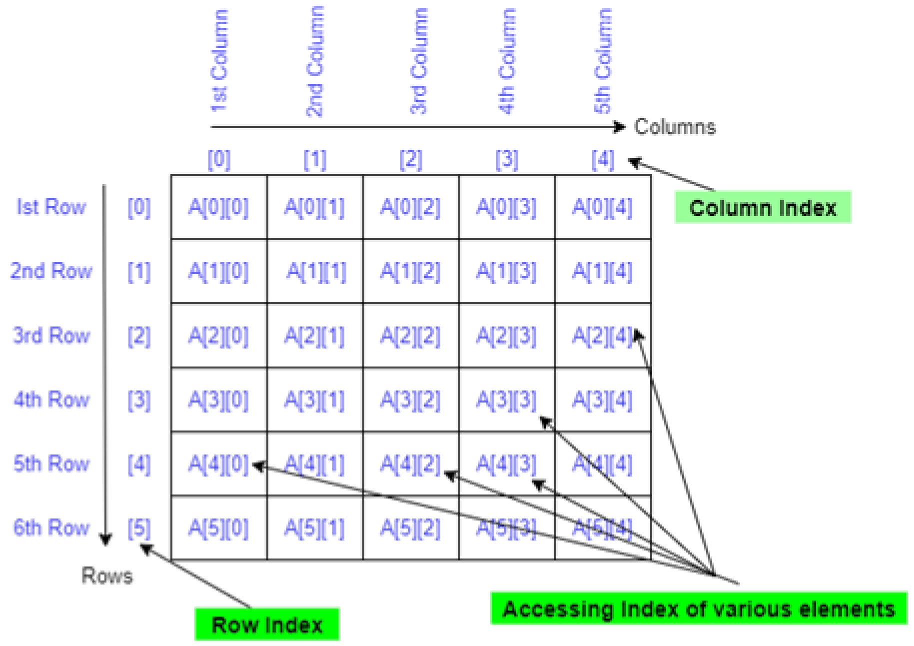
Example: Declaration of two-dimensional array "A" having "int" datatype and row size 6 and column size 5.

int A[6][5]

Continued...

Accessing the element in two-dimensional array

Accessing its elements involves two subscripts; one for row and second for column.



syntax:

Arrayname[row_index][column_index]

Example:

To access 2nd element of 1st row in the array A, we write *A[0][1]*

Memory Representation of Two-Dimensional Array

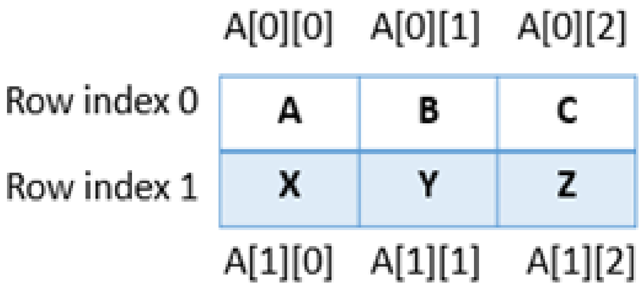
There are two-ways by which the 2D array elements can be represented in Memory.

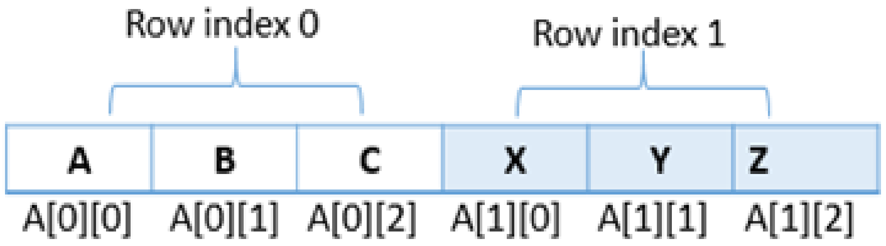
a) **Row Major**

b) **Column Major**

Row Major Representation

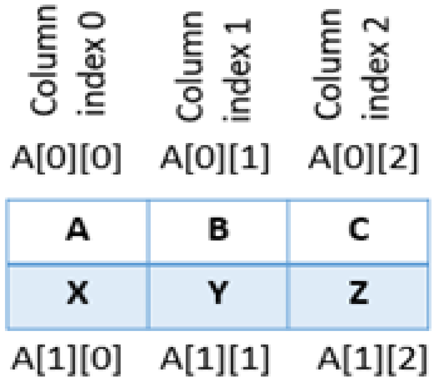
In row-major order, storage of the elements in the memory is row-wise i.e. storage of elements of first row is followed by the storage of elements of second row and so on so forth.

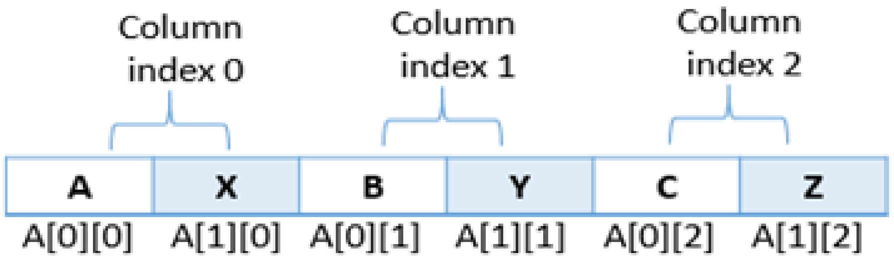




Column-major Representation

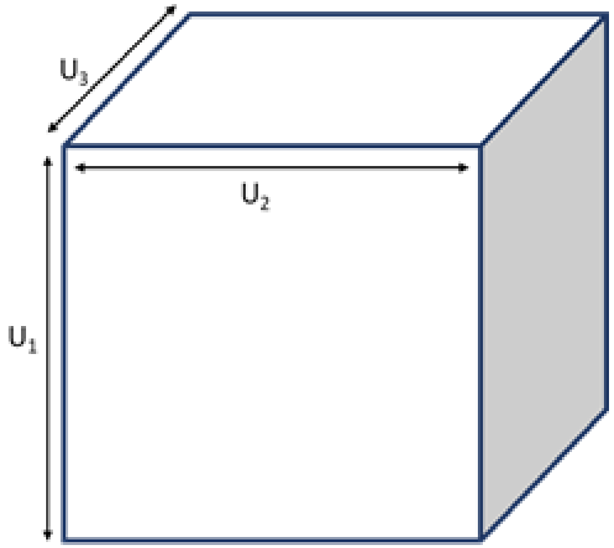
In column-major order, elements are stored column wise i.e., storage of elements of the first column followed by storage of second column elements and so on so forth.





Three-Dimensional Array

When an array is represented in the form of 3 different dimensions, it is called 3-D Array. It can also be called as an array of 2-dimensional arrays.



3-Dimensional array which has 3 dimensions named U_1, U_2, and U_3.

Continued...

Declaration of three-dimensional array:

Syntax: *<Data Type> <Arrayname> [m][n][o]*

Where,

m = 1^{st} Dimension

n = 2^{nd} Dimension

o = 3rd Dimension

Example: Declaration of three-dimensional array "A" having "int" datatype with first dimension size 6, second dimension size 5, third dimension size 4.

int A[6][5][4]

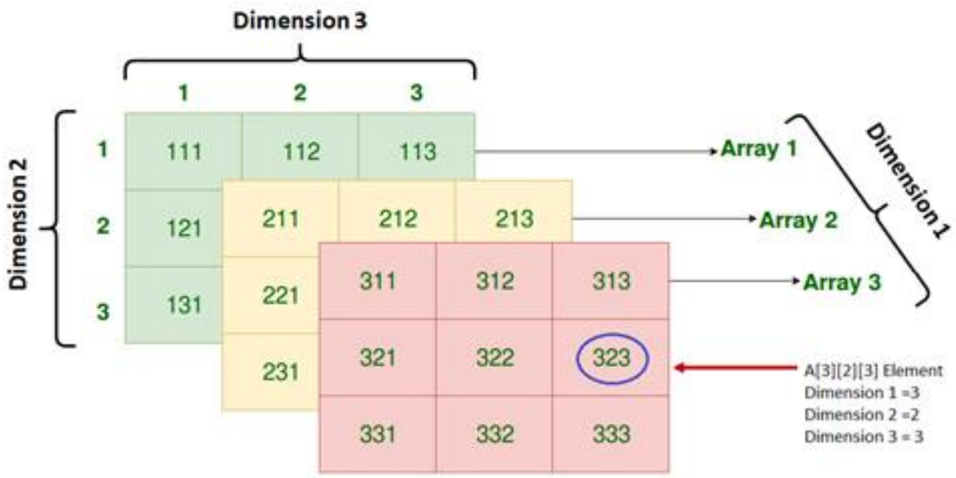
If the array is declared as A[3][4][5], it will have a total of 3 x 4 x 5 = 60 elements.

Continued...

Accessing the element in three-dimensional array

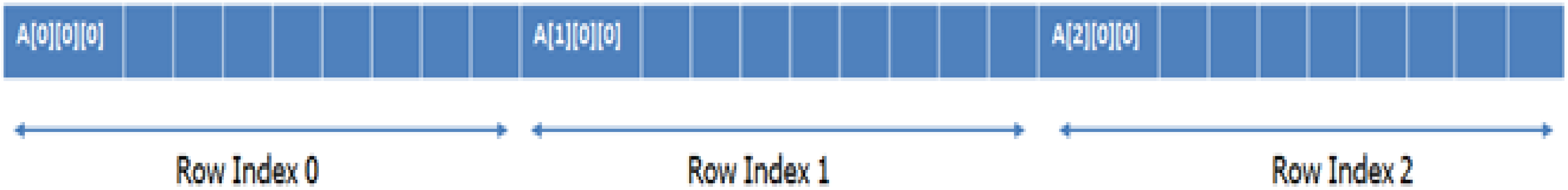
Accessing its elements involves three subscripts, one for each dimension.

Syntax: *Arrayname[index1][index2][index3]*



Memory Representation of Three-Dimensional Array

Memory Representation in Row Major Order



Here, the first dimension is considered as row.

(Here, we assume the first index for row and column is 0)

In the similar way Column major order arrangement can be done.

Question:

```
int main()
{
int ary[2][2][3] = {
{{1,2,3},{4,5,6}},
{{7,8,9},{10,11,12}}
};
int *p;
p = &ary;
printf("%d %d",*p, *p+11);
return 0;
}
```

What is the output of C Program with arrays?	
A. 1 11	B. 1 12
D. Compile Error	C. 2 13

Question:

Given A [1:15], bytes per cell = 3, base address = 1000 find the address of A [9].	
A. 1022	B. 1021
D. 1024	C. 1023

Question:

If the address of A[1][1] and A[2][1] are 1000 and 1010 respectively and each element occupies 2 bytes then the array has been stored in _________ order.	
A. row major	B. column major
D. None of these	C. matix major

Question:

Given an array, arr[1………10][1………15] with base value 100 and the size of each element is 1 Byte in memory. Find the address of arr[8][6] with the help of row-major order?	
A. 12	B. 210
D. 200	C. 120

Question:

Consider the following declaration of a ‘two-dimensional array in C: char a[100][100]; Assuming that the main memory is byte-addressable and that the array is stored starting from memory address 0, the address of a[40][50] is:	
A. 4040	B. 4050
D. 5080	C. 5040

Question:

Which of the following expressions accesses the (i,j)th entry of an (m x n) matrix stored in column major form?	
A. n x (i -1) + j	B. m x(j -1) + i
D. m x (n-j) + j	C. n x(m-i) + j

Question:

The smallest element of an array's index is called its	
A. lower bound	B. upper bound
D. extraction	C. range

Question:

Given an array [1..8, 1..5, 1..7] of integers. Calculate address of element A[5,3,6], by using rows and columns methods, if BA=900?	
A. 1120	B. 1122
D. 1123	C. 1121

Question:

Given an array [1..8, 1..5, 1..7] of integers. Calculate address of element A[5,3,6], by using rows and columns methods, if BA=900?	
A. 1120	B. 1122
D. 1123	C. 1121

Question:

Given an array arr[1:8, -5:5, -10:5] with base value 400 and size of each element is 4 Bytes in memory find the address of element arr[3][3][3] with the help of column-major order?	
A. 4676	B. 4670
D. 4696	C. 4690

Question:

If more than one subscript is used, an array is known as a___________.	
A. One- dimensional array	B. Single dimensional array
D. None of the above	C. Multi- dimensional array

Question:

The memory address of fifth element of an array can be calculated by the formula	
A. LOC(Array[5]=Base(Array)+w(5-lower bound), where w is the number of words per memory cell for the array	B. LOC(Array[5])=Base(Array[5])+(5-lower bound), where w is the number of words per memory cell for the array
D. None of above	C. LOC(Array[5])=Base(Array[4])+(5-Upper bound), where w is the number of words per memory cell for the array

Index Formula Derivation for Array

One Dimensional Array

Index Formula Derivation in One Dimensional Array

Suppose there is a one-dimensional array A[L : U]

Number of elements /lengths of the array can be found by the formula N = U – L + 1

Where U = Upper Bound of the array (Last index)

L = Lower Bound of the array (First index)

A[0:9] will contain 9-0+1 =10 elements

Index Formula Derivation in One Dimensional Array

- **To find the address of i^{th} index element, we will take two assumptions**
  - **Assumption 1:** 1 Byte storage for each element.
  - **Assumption 2:** First element is at index 1.

Index Formula Derivation in One Dimensional Array

- Assume base address of an array = α.
- So

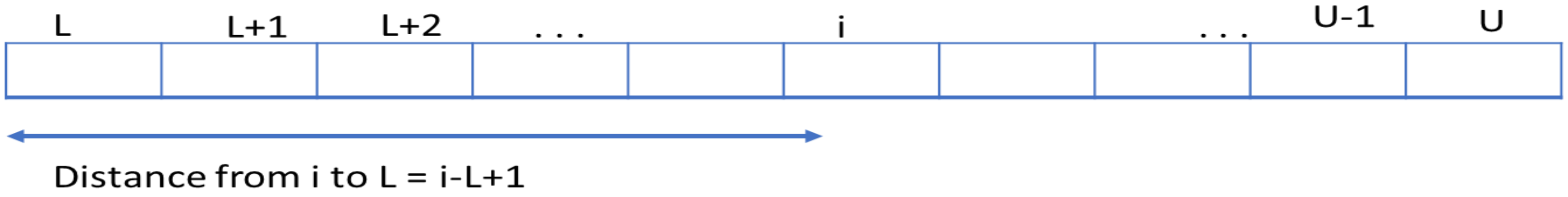
- address of A[1]= α //address of 1st element of array A as given.
- address of A[2]= α +1 //address of 2nd element of array A is address of base address + no. of bytes i.e.1 (α +1)
- address of A[3]= α +2 //address of 3rd element of array A is address of 2nd element + no. of bytes i.e.1 (α +1+1)
- address of A[4]= α +3 //address of 4th element of array A is address of 3rd element + no. of bytes i.e.1 ((α +1+1+1)
- address of **A[i]= α + (i-1) ………………. (equation 1)**

Step 2: Removal of first assumption i.e., 1 Byte storage for each element.

- An element can take ‘n’ number of bytes depending upon the datatype
  - **A[i]= α +(i-1)………………. (equation 1)**
  - **A[i]= α +(i-1) *n ………………. (equation 2)**

Step 3: Removal of 2nd Assumption, i.e. First element is at index 1

- If index starts from L then, for some i^{th} index element distance from first index will be i-L+1.



Index Formula Derivation in One Dimensional Array

- Address of **A[i]= α +(i-1) *n** //As given in equation 2.
- Replacing i with i-L+1
- Address of **A[i] = α + (i-L+1-1) *n**
- So, Address of **A[i] = α +(i-L) *n**

Address of A[i] = Base Address +n*(i – Lower Bound)

Index Formula Derivation in One Dimensional Array

Question 1: Given A [-1:10], bytes per cell = 4, base address = 2000 find the address of A [7].

Solution:

Here, i = 7

n= 4

Lower Bound = -1

Upper Bound = 10

Address of A [i] = Base Address +n*(i-Lower Bound)

Address of A [7] = 2000 + 4*(7-(-1)) = 2032

Question 2:Given A [1:15], bytes per cell = 3, base address = 1000 find the address of A [9].

Solution:

Here, i = 9

n= 3

Lower Bound = 1

Upper Bound = 15

Address of A [i] = Base Address +n*(i-Lower Bound)

Address of A [9] = 1000 + 3*(9 - 1) = 1024

Index Formula Derivation in One Dimensional Array

Question 3: Why does indexing in most of the languages start with 0?

Solution:

In address calculation we can skip the offset value. E.g. A[10] = {1,2,3,4,5,6,7,8,9,10}. If we start the indexing with 1, calculation of address of A[5] = 1000+(5-1)*2

If we start the indexing with 0, calculation of address of A[5] = 1000+(5-0) *2. This can directly be written as 1000+5*2. In this case, we do not need to perform the additional arithmetic operation i.e. subtraction.

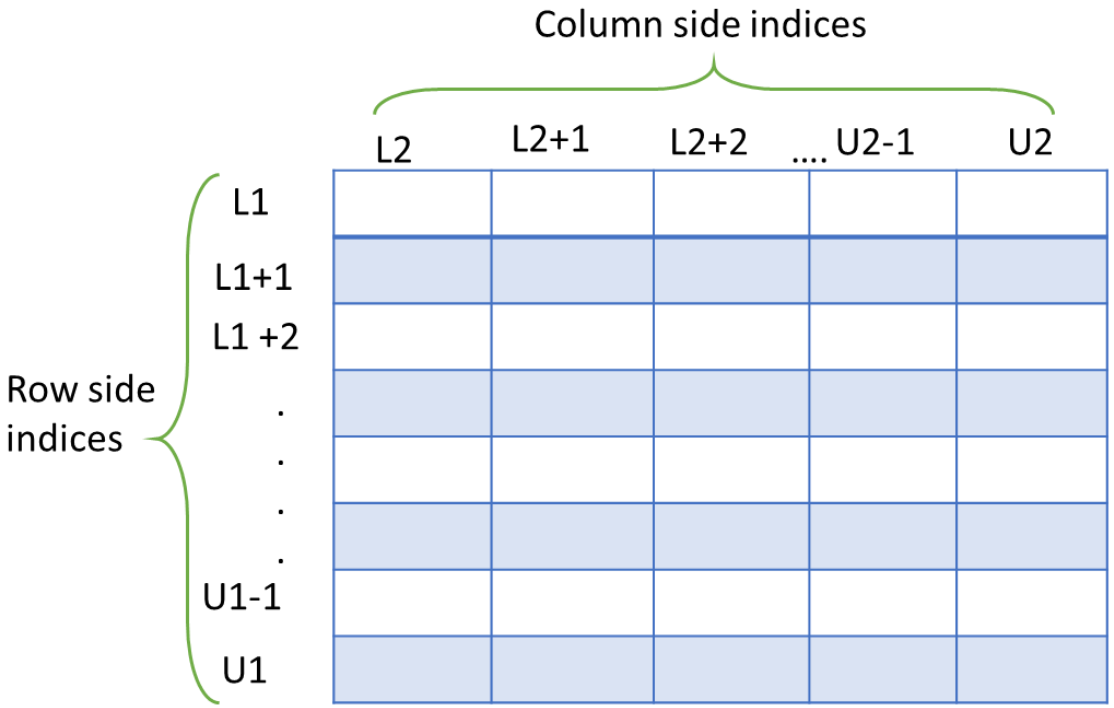
(5-1) in the above computations is known as extra subtraction or offset value.

Index Formula Derivation for Array

Two-Dimensional Array

Row Major Order Arrangement

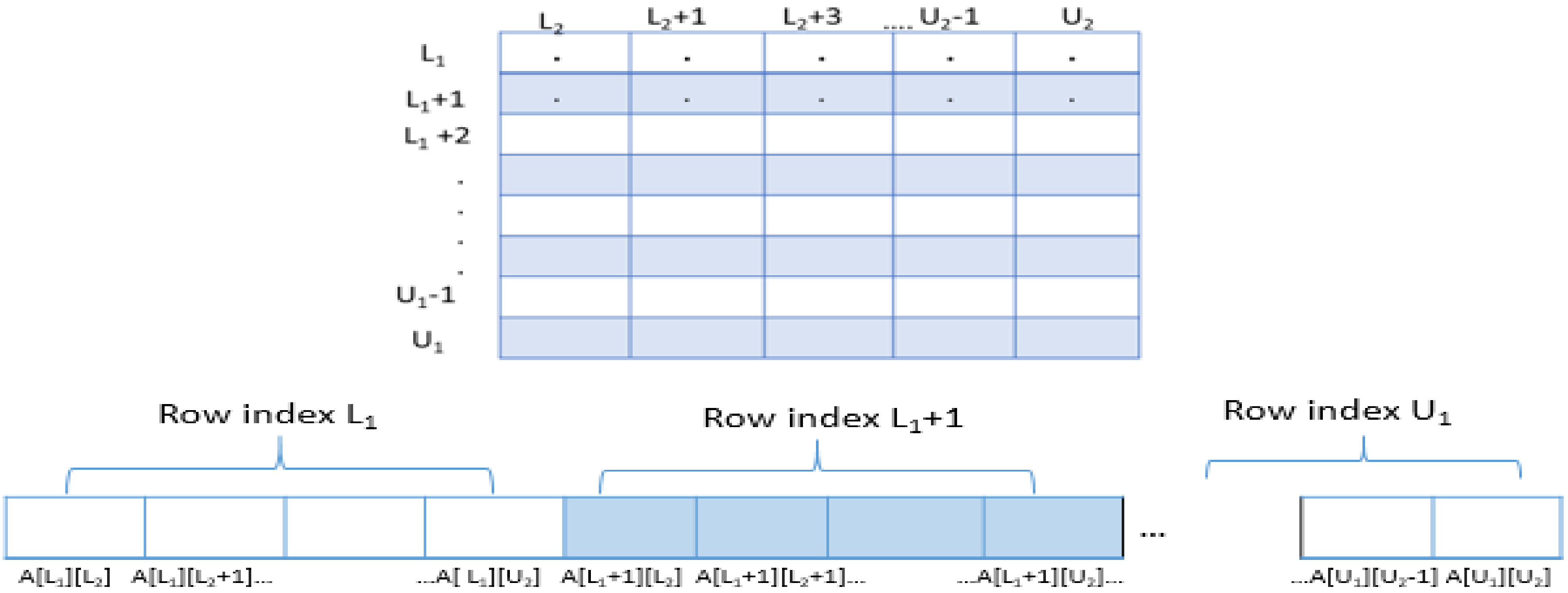
- Suppose we have a 2–D array $A[L_1{:}U_1, L_2{:}U_2]$ given as shown:



- Row side indices are L_1, L_1+1, L_1+2, …, $U_1–1$, U_1.
- Column side indices are L_2, L_2+1, L_2+2, …, $U_2–1$, U_2.

Row Major Order Arrangement

- In memory it will look like 1-D array as it will be stored row-wise.



Row Major Order Arrangement

- For simplicity, we will assume that the first index is 1 and each element requires 1 byte for storage. The array becomes $A[1:U_1, 1:U_2]$.

If the base address of array is α then,

Address of A[1,1] = α

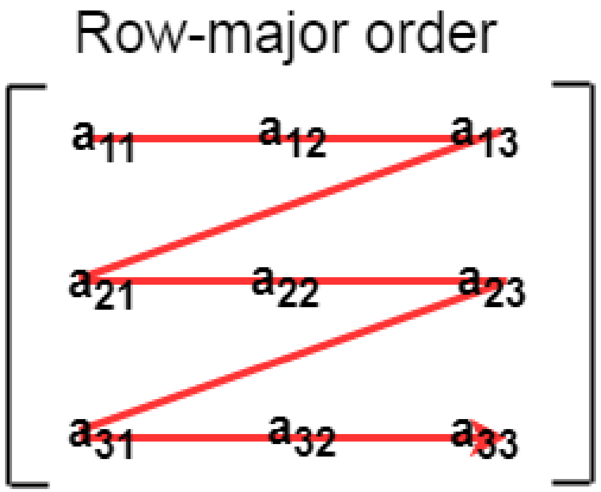
Address of A[1,2] = $\alpha + 1$

Address of A[1,3] = $\alpha + 2$

Address of A[1,4] = $\alpha + 3$

…

Address of $A[1, U_2] = \alpha + (U_2 - 1)$



Row Major Order Arrangement

Address of A[2,1] = $\alpha + (U_2 - 1) + 1$

Address of A[2,1] = $\alpha + U_2$

Address of A[3,1] = $\alpha + U_2 + U_2$

Address of A[3,1] = $\alpha + 2\,U_2$

…

Address of A[i, 1] = $\alpha + (i - 1)*U_2$

Address of A[i,2] = $\alpha + (i - 1)*U_2 + 1$

Address of A[i,3] = $\alpha + (i - 1)*U_2 + 2$

…

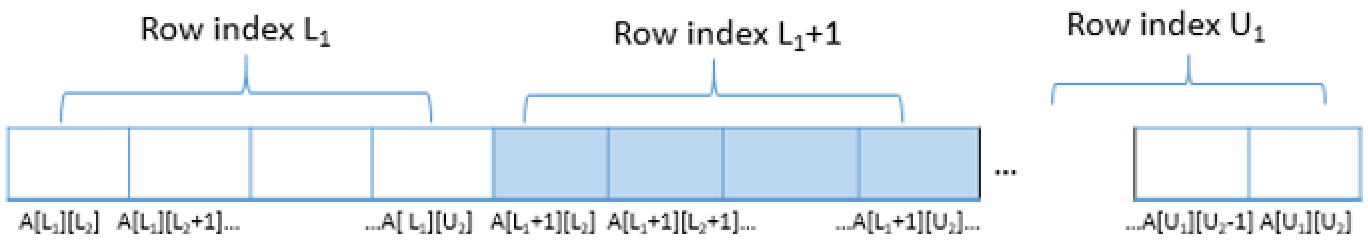
Similarly, A[i, j] = $\alpha + (i - 1)*U_2 + (j - 1)$

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Row Major Order Arrangement

Now, let us remove the assumption that every element takes 1 byte of storage with n bytes for storage. So, the formula will change to

$$\textbf{Address of } A[i, j] = \alpha + [(i - 1)*U_2 + (j - 1)]*n$$



Row Major Order Arrangement

Now, remove the assumption that first index is 1 in row and column with L1 and L2, respectively. Replacing U_2 as $U_2 - L_2 + 1$ (length formula), i with $i - L_1 + 1$ and j with $j - L_2 + 1$

Address of A [i, j] = $\alpha + [(i - L_1 + 1 - 1) * (U_2 - L_2 + 1) + (j - L_2 + 1 - 1)] * n$

Address of A [i, j] = $\alpha + [(i - L_1) * (U_2 - L_2 + 1) + (j - L_2)] * n$

$$\boxed{\text{Address of } A[i, j] = \text{Base address} + [(i - L_1) * (U_2 - L_2 + 1) + (j - L_2)] * n}$$

Can you answer these questions?

Suppose a 2D array A is declared as A[–2:2 , 2:6], words per cell = 4, base address = 200. Consider Row major order arrangement.

- A) Find out length of each dimension and the number of elements in array.
- B) Find the location of A[1,2]

Solution

By formula:

Here Lower Bound of row(L_1) = –2

Here Upper Bound of row(U_1) = 2

Here Lower Bound of column(L_2) = 2

Here Upper Bound of column(U_2) = 6

n = 4

Length of row = $U_1 – L_1 + 1$

= 2 – (–2) + 1 = 5

Length of column = $U_2 – L_2 + 1$

= 6 – 2 + 1 =5

No. of elements = 5*5 = 25

A[i, j] = Base address + [(i – L_1)*($U_2 – L_2$ + 1) + (j – L_2)] *n

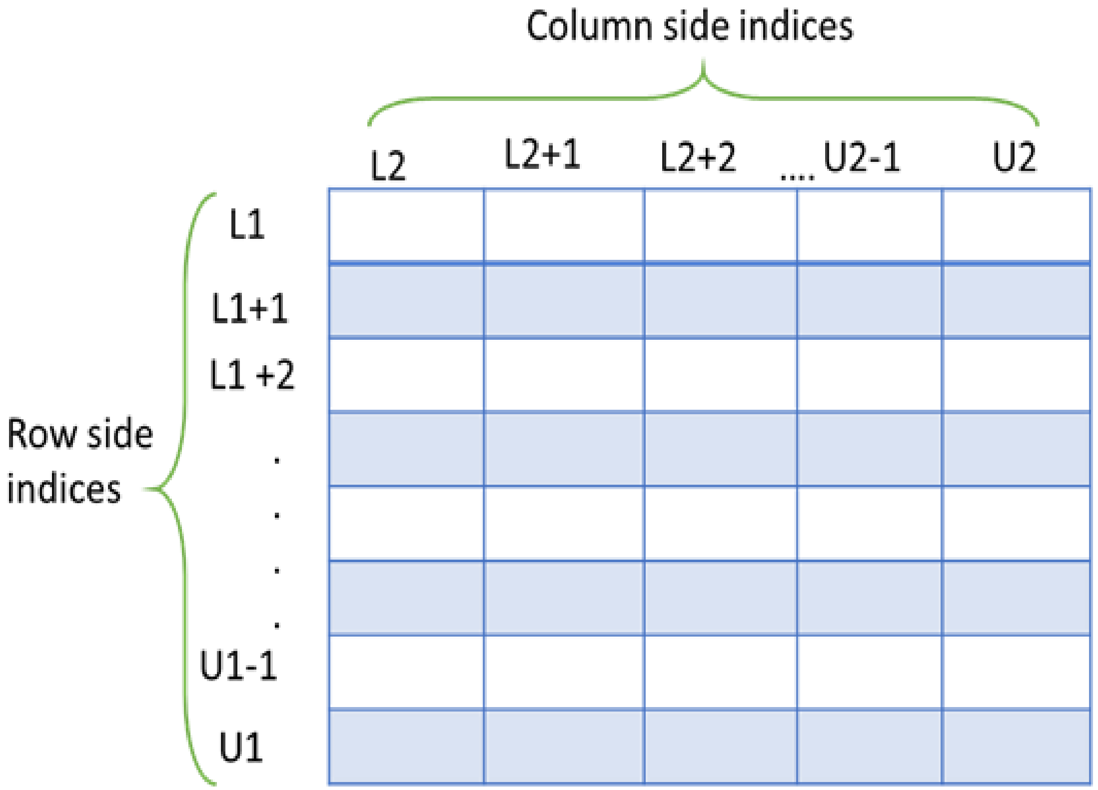
A[1, 2]= 200 + [(1 – (–2) * (6 – 2 + 1) + (2 – 2)] * 4

= 200 + 15* 4

= 260

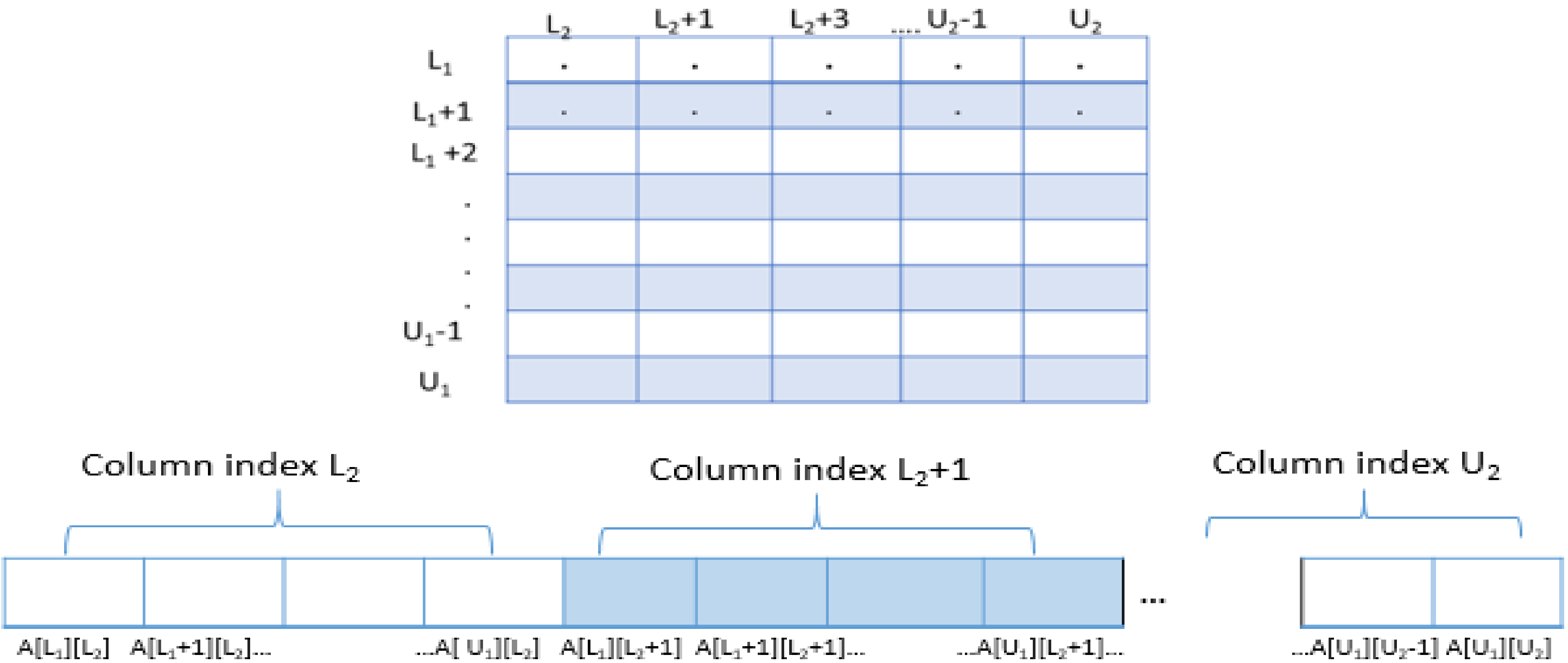
Column Major Order Arrangement

- Suppose we have a 2–D array $A[L_1:U_1, L_2:U_2]$ given as shown:
- Row side indices are L_1, L_1+1, L_1+2, …, $U_1–1$, U_1.
- Column side indices are L_2, L_2+1, L_2+2, …, $U_2–1$, U_2.



Column Major Order Arrangement

In memory it will look like 1-D array as it will be stored Column-wise.



Column Major Order Arrangement

For simplicity, we will assume that the first index is 1 and each element requires 1 byte for storage. The Array becomes $A[1:U_1, 1:U_2]$. Another assumption: every element requiring 1 byte for storage. So that, address of first element say:

If the base address of array is α then,

Address of $A[1,1] = \alpha$

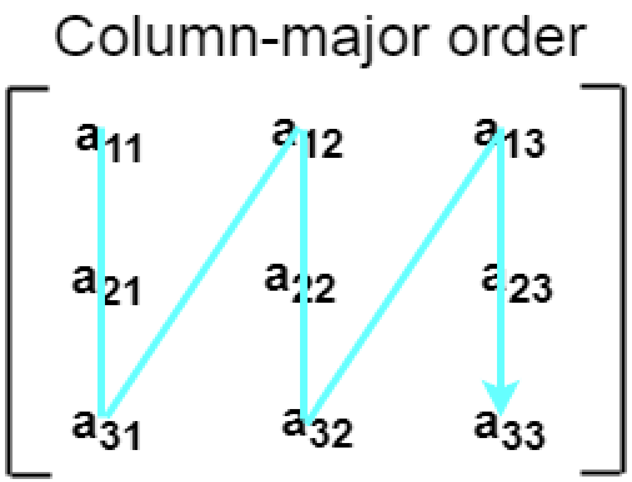
Address of $A[2,1] = \alpha + 1$

Address of $A[3,1] = \alpha + 2$

Address of $A[4,1] = \alpha + 3$

…

Address of $A[U_1,1] = \alpha + (U_1 – 1)$



Column Major Order Arrangement

Address of A[1,2] = α + (U_1 – 1) +1

Address of A[1,2] = α + U_1

Address of A[1,3] = α + U_1 + U_1

Address of A[1,3] = α + 2*U_1

…

Address of A[1,j] = α + (j – 1)*U_1

Address of A[2,j] = α + (j – 1)*U_1 +1

Address of A[3,j] = α +(j – 1)*U_1 +2

…

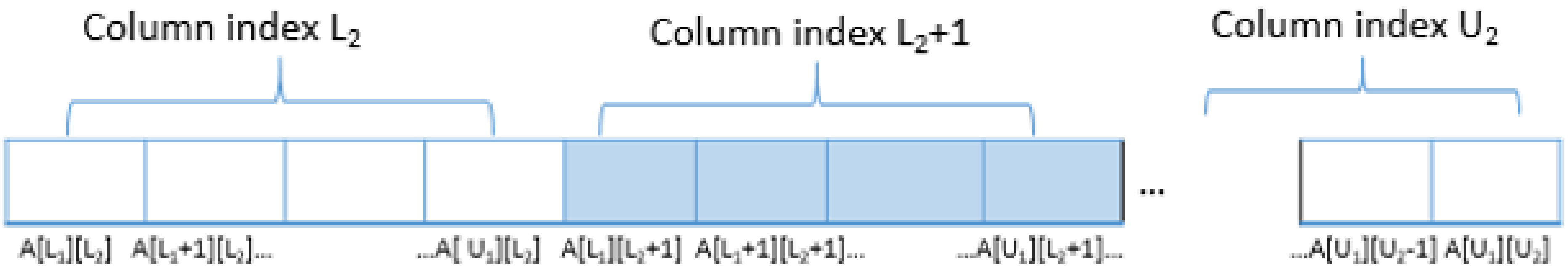
Similarly, A[i, j] = α +(j – 1)*U_1 + (i – 1)

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Column Major Order Arrangement

Now, let us remove the assumption that every element takes 1 byte of storage with n bytes for storage. So, the formula will change to

Address of A[i, j] = α +[(j – 1)*U1 + (i – 1)]*n



Column Major Order Arrangement

Now, remove the assumption that first index is 1 in row and column with L_1 and L_2, respectively. Replacing U_1 as $U_1 – L_1 +1$ (length formula), i with $i – L_1 + 1$ and j with $j – L_2 + 1$

Address of A [i, j] = $\alpha + [(j – L_2 + 1 – 1)*(U_1 – L_1 + 1) + (i – L_1 + 1 – 1)]*n$

Address of **A [i, j] = $\boldsymbol{\alpha + [(j – L_2)*(U_1 – L_1 + 1) + (i – L_1)] *n}$**

Address of A[i, j] = Base address + $[(j – L_2)*(U_1 – L_1 + 1) + (i – L_1)] *n$

Can you answer these questions?

Suppose a 2D array A is declared as A[–2:2, 2:6], words per cell = 4, base address = 1024. Consider Column Major order arrangement.

- Find the length of each dimension and number of elements in array.
- Find the location of A[2,5]

Solution

Here Lower Bound of row(L_1) = –2

Here Upper Bound of row(U_1) = 2

Here Lower Bound of column(L_2) = 2

Here Upper Bound of column(U_2) = 6

n = 4

Length of row = $U_1 - L_1 + 1$

= 2 – (–2) + 1 = 5

Length of column = $U_2 - L_2 + 1$

= 6 – 2 + 1 =5

No. of elements = 5*5 = 25

By formula:

Address of A[i, j] = Base address + [(j – L_2)*($U_1 - L_1$ + 1) + (i – L_1)] *n

Address of A[2,5] = 1024 + [(5 – 2) * (2 – (–2) + 1) + (2 – (–2))] * 4

= 1024 + [15 + 4]*4

= 1024 + 76

= 1100

Three Dimensional Array

- An array represented in the form of 3 different dimensions, is called 3-D Array

Declaration of three-dimensional array:

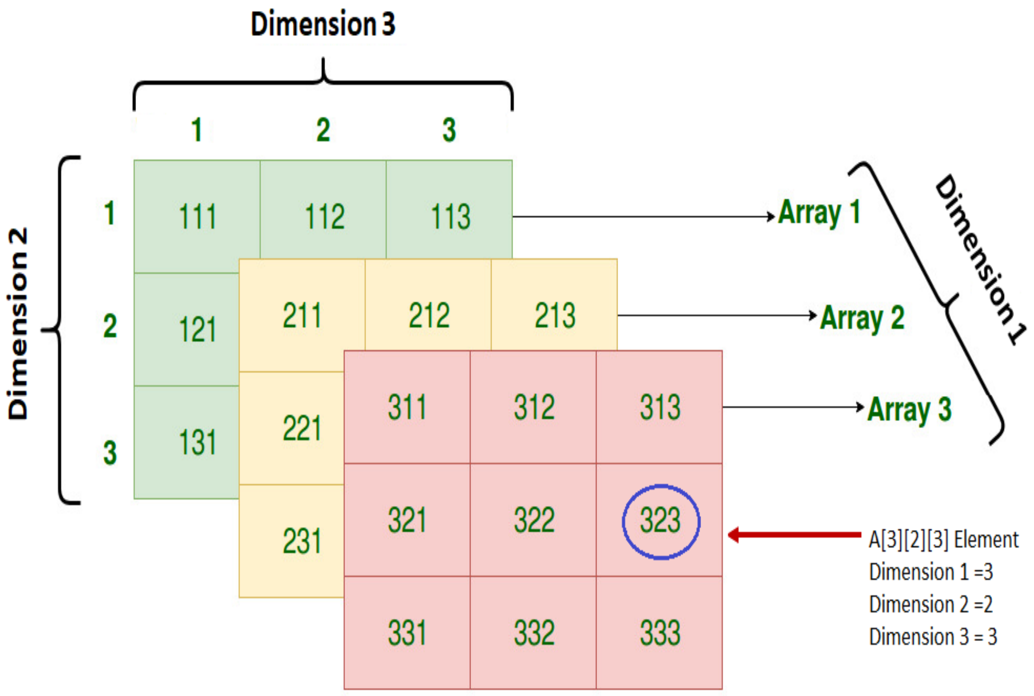
Syntax: *<Data Type> <Arrayname> [m][n][o]*

Where,

m → 1^{st} Dimension

n → 2^{nd} Dimension

o → 3rd Dimension

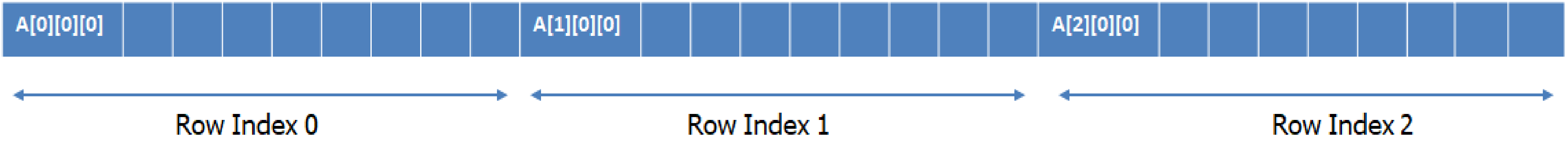


- A[3][4][5], it will have a total of 3 x 4 x 5 = 60 elements.

Three Dimensional Array

Memory Representation

Row Major Representation



- The first dimension is considered as row.

Column Major Representation

Three Dimensional Array

Row Major Representation

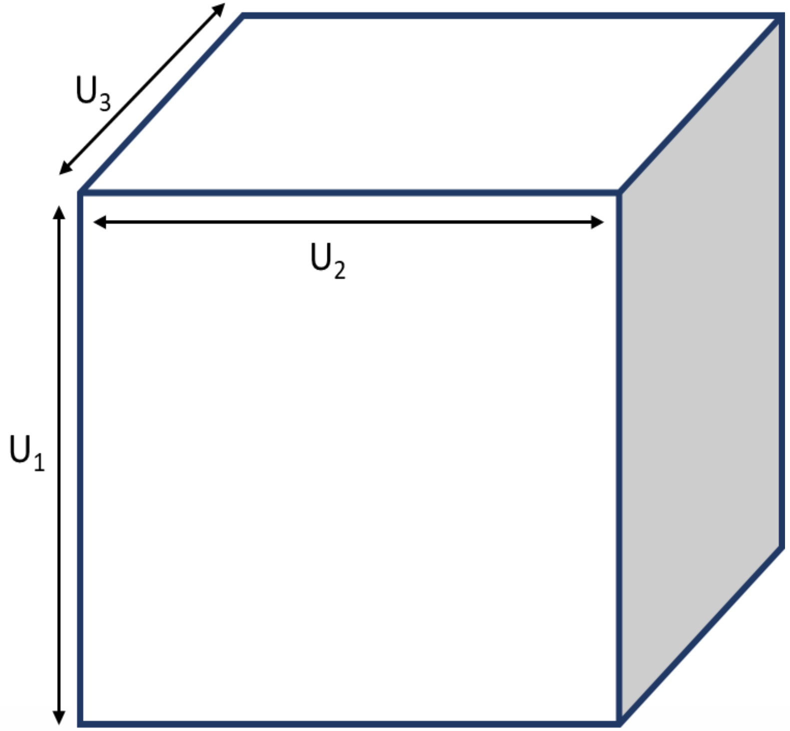
- Imagine a cuboid of size U_1 x U_2 x U_3.
- The 3-D Array can be represented as $A[L_1{:}U_1, L_2{:}U_2, L_3{:}U_3]$

where,

L_1 = lower bound of first dimension

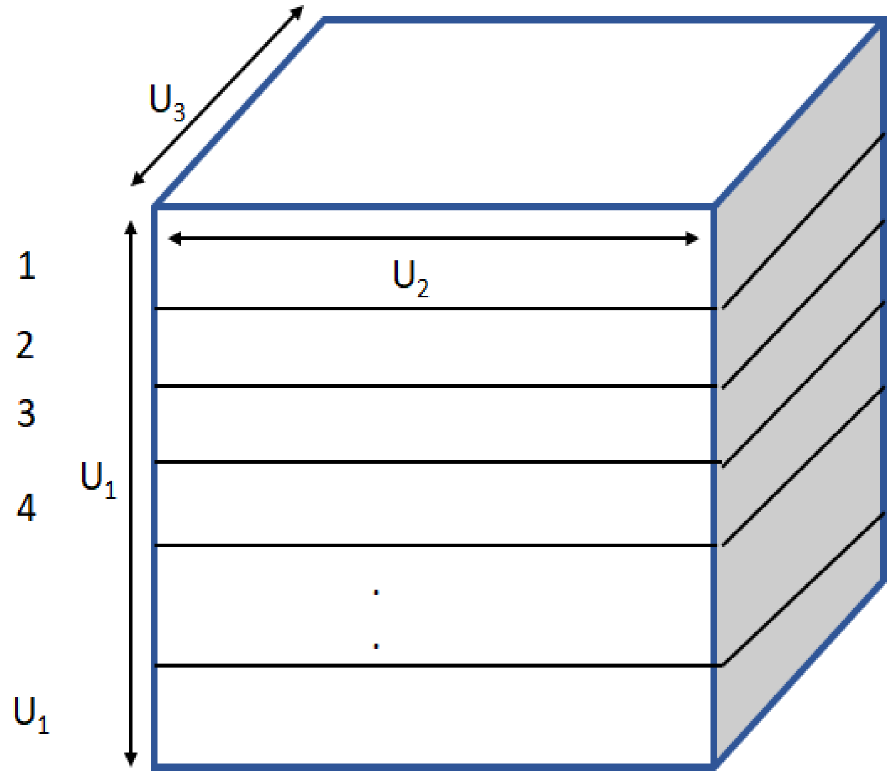
L_2 = lower bound of second dimension

L_3 = lower bound of third dimension

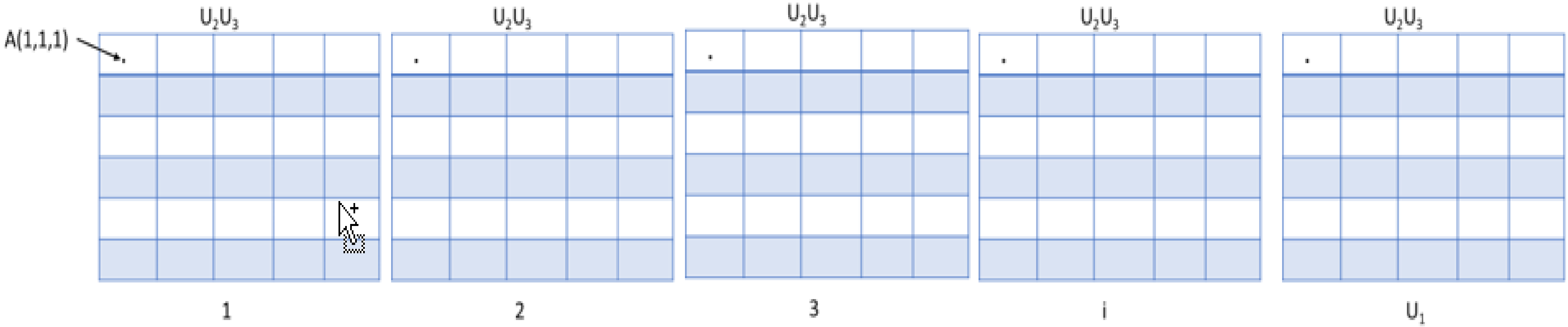


Index Formula Derivation in Three Dimensional Array

The diagram can be viewed as U_1 2-D arrays of size $U_2 x U_3$.



Index Formula Derivation in Three Dimensional Array



The above diagram can be viewed as U_1 2-D arrays of size $U_2 x U_3$

If the base address of array is α then,

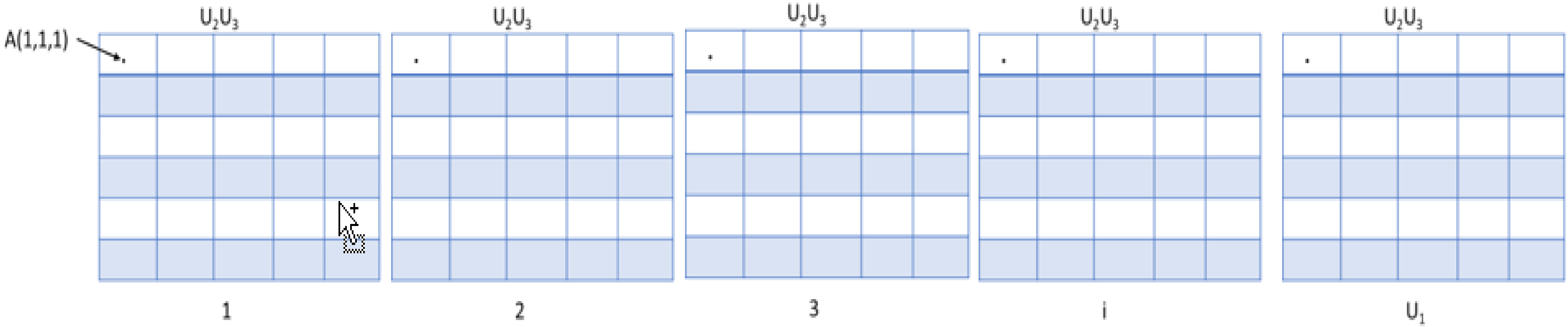
Address of A[1,1,1] = α

Address of A[2,1,1] = $\alpha + U_2 * U_3$ (There are $U_2 x U_3$ elements in the first slice)

Address of A[3,1,1] = $\alpha + U_2 * U_3 + U_2 * U_3$

$= \alpha + 2 * U_2 * U_3$

Index Formula Derivation in Three Dimensional Array



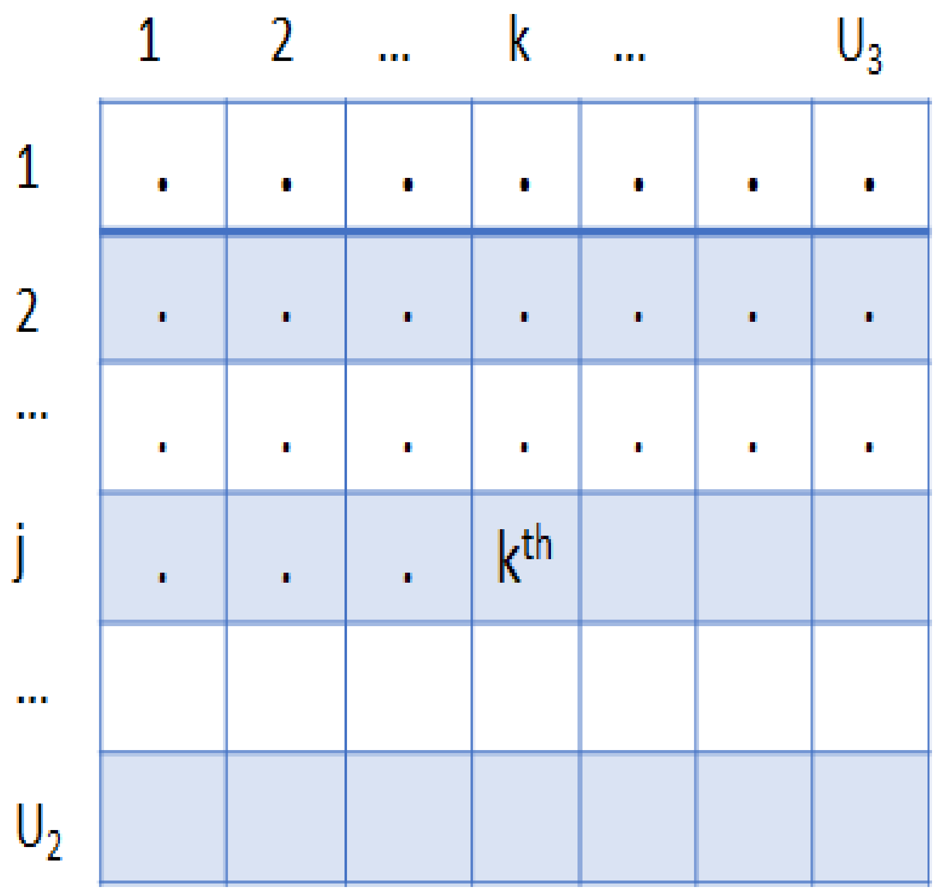
Address of A[4,1,1] = $\alpha + 3*U_2*U_3$

Address of A[i, 1, 1] = $\alpha + (i - 1)*U_2*U_3$

Index Formula Derivation in Three Dimensional Array

let us expand the i^{th} array. This will be a 2-D array of size $U_2 x U_3$

Address of A[i,2,1] = $\alpha + (i - 1)*U_2*U_3 + U_3$ (There are U3 elements in the first row of this 2-D array)



Address of A[i,3,1] = $\alpha + (i - 1)*U_2*U_3 + U_3 + U_3$

Address of A[i,3,1] = $\alpha + (i - 1)*U_2*U_3 + 2*U_3$

Index Formula Derivation in Three Dimensional Array

Address of 1st element of j^{th} row

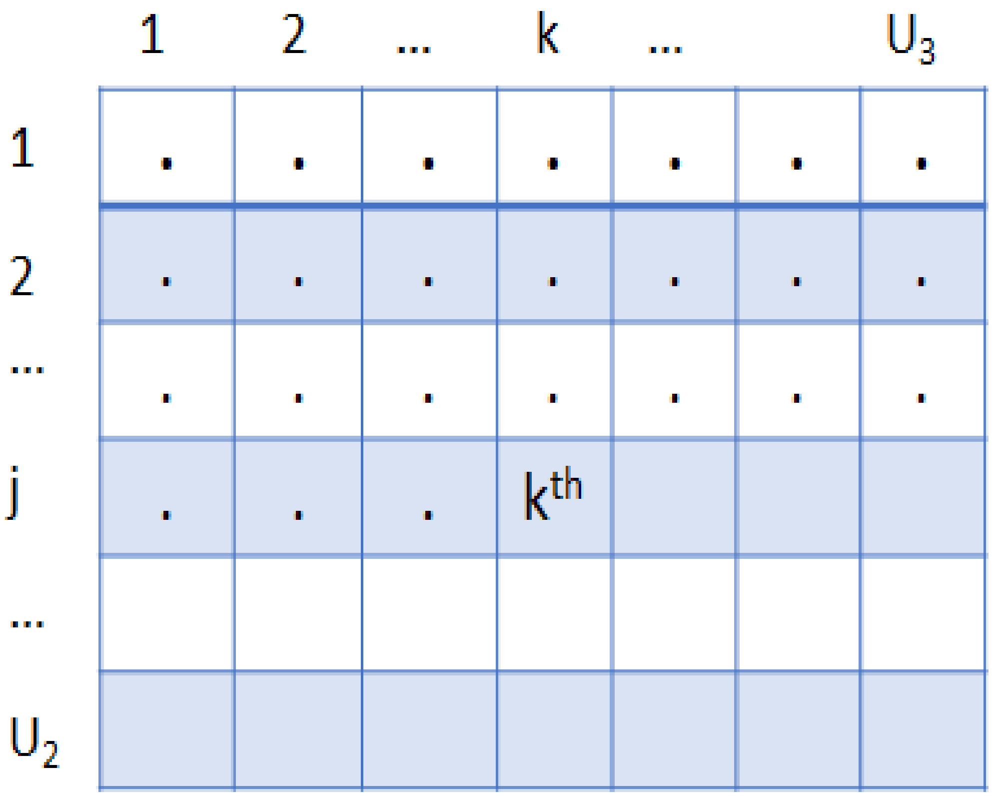
Address of $A[i,j,1] = \alpha + (i - 1)*U_2*U_3 + (j - 1)*U_3$

Address of $A[i,j,2]= \alpha + (i - 1)*U_2*U_3 + (j - 1)*U_3 +1$

Address of $A[i,j,3] = \alpha + (i - 1)*U_2*U_3 + (j - 1)*U_3 +2$

Similarly, for k^{th} element of j^{th} row

Address of $A[i, j, k] = \alpha + (i - 1)*U_2*U_3 + (j - 1)*U_3 + (k - 1)$



Index Formula Derivation in Three Dimensional Array

- Remove the assumption that every element takes 1 byte of storage

$$A[i, j, k] = \alpha + [(i - 1)*U_2*U_3 + (j - 1)*U_3 + (k - 1)]*n$$

- Remove the assumption that first index is 1 in each dimension with L_1, L_2 and L_3 respectively.

	1	2	...	k	...		U_3
1	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.
...	.	.	.	.	.	.	.
j	.	.	.	k^{th}			
...							
U_2							

$$A[i, j, k] = \alpha + [(i - L_1)*(U_2 - L_2 + 1)*(U_3 - L_3 + 1) + (j - L_2)*(U_3 - L_3 + 1) + (k - L_3)]*n$$

Column Major Representation

- The 3-D Array can be represented as $A[L_1:U_1, L_2:U_2, L_3:U_3]$

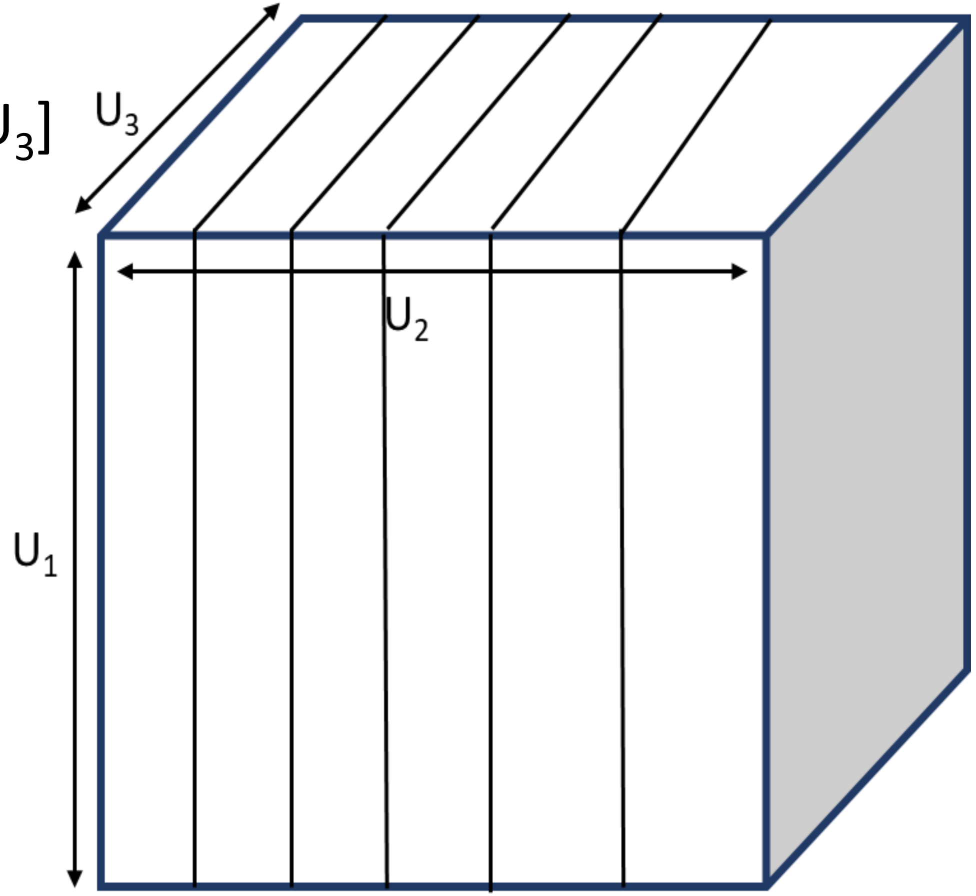
where,

L_1 = lower bound of first dimension

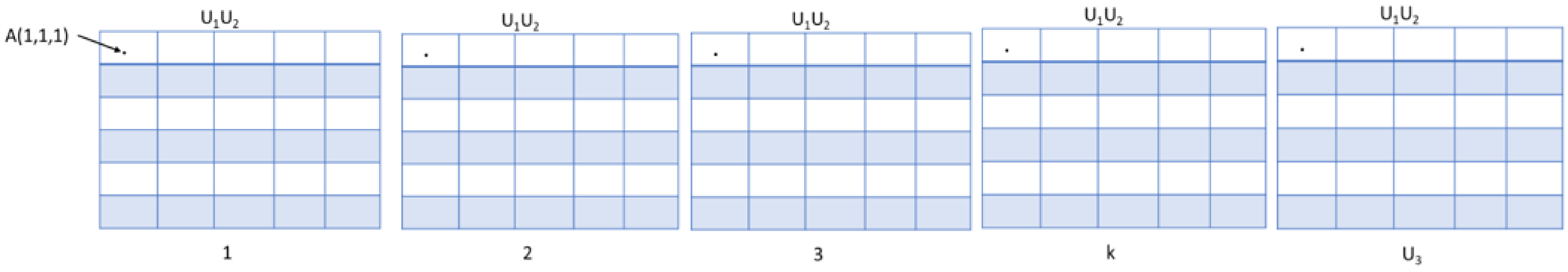
L_2 = lower bound of second dimension

L_3 = lower bound of third dimension

- Cut the cuboid horizontally across the first dimension. The slices obtained are of size $U_1 x U_2$. So there will be U_3 such slices.



Index Formula Derivation in Three Dimensional Array



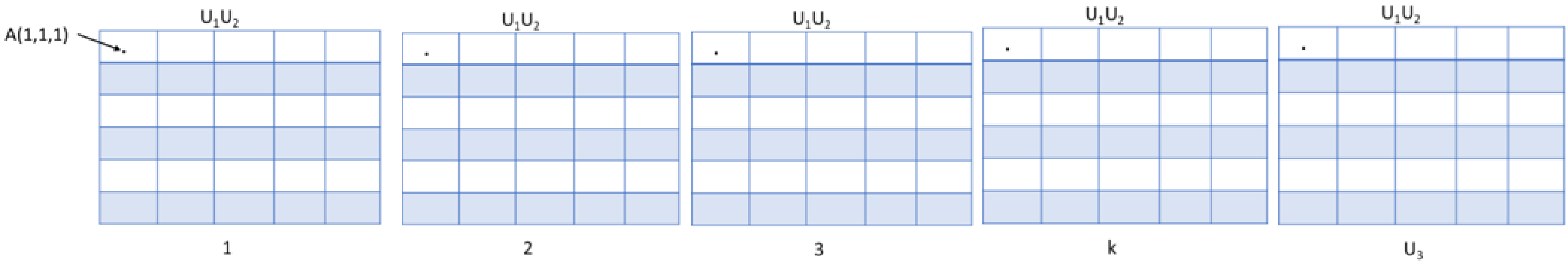
The above diagram can be viewed as U_3 2-D arrays of size $U_1 x U_2$.

If the base address of array is α then,

Address of A[1,1,1] = α

Address of A[1,1,2] = $\alpha + U_1 * U_2$ (There are $U_1 x U_2$ elements in the first slice)

Index Formula Derivation in Three Dimensional Array

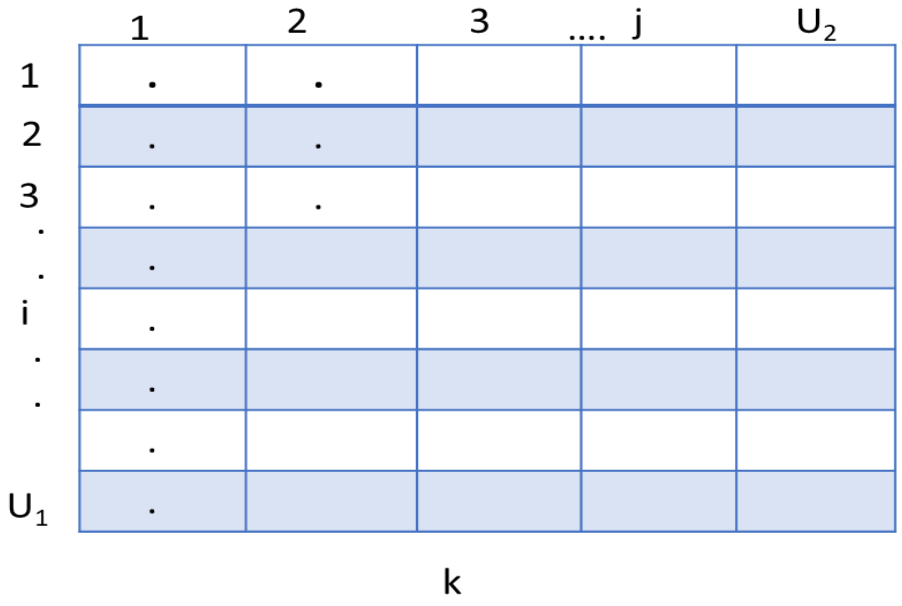


Address of A[1,1,3] = $\alpha + U_1*U_2 + U_1*U_2$

$= \alpha + 2*U_1*U_2$

Address of A[1,1,4] = $\alpha + 3*U_1*U_2$

Address of A[1, 1, k]= $\alpha + (k-1)*U_1*U_2$



Index Formula Derivation in Three Dimensional Array

Address of A[1,2,k] = $\alpha + (k-1)*U_1*U_2 + U_1$ (There are U1 elements in the first column of this 2-D array)

Address of A[1,3,k] = $\alpha + (k-1)*U_1*U_2 + U_1 + U_1$

Address of A[1,3,k] = $\alpha + (k-1)*U_1*U_2 + 2*U_1$

Address of 1st element of jth column

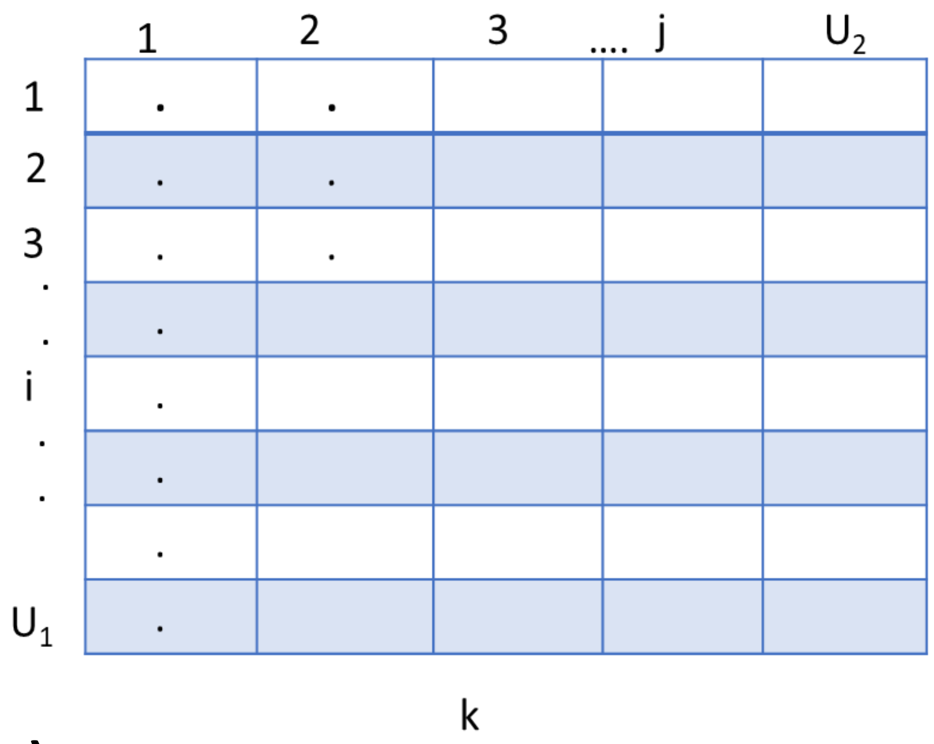
Address of A[1,j,k] = $\alpha + (k-1)*U_1*U_2 + (j-1)*U_1$

Address of A[2,j,k]= $\alpha + (k-1)*U_1*U_2 + (j-1)*U_1 + 1$

Address of A[3,j,k] = $\alpha + (k-1)*U_1*U_2 + (j-1)*U_1 +2$

Similarly, for kth element of jth column

Address of A[i,j,k] = $\alpha + (k-1)*U_1*U_2 + (j-1)*U_1 + (i-1)$



Index Formula Derivation in Three Dimensional Array

Remove the assumption that every element takes 1 byte of storage with n bytes for storage. So, the formula will be

Address of A[i,j,k] = $\alpha + [(k - 1)*U_1*U_2 + (j - 1)*U_1 + (i - 1)]*n$

Index Formula Derivation in Three Dimensional Array

Remove the assumption that first index is 1 in each dimension with L_1, L_2 and L_3 respectively, i will be replaced by $i–L_1+1$, j by $j–L_2+1$ and k by $k–L_3+1$

$$A[i, j, k] = \alpha + [(k - L_3)*(U_1 - L_1 + 1)*(U_2 - L_2 + 1) + (j - L_2)*(U_1 - L_1 + 1) + (i - L_1)]*n$$

$$\boxed{A[i, j, k] = \text{Base Address} + [(k-L_3)*(U_1-L_1+1)*(U_2-L_2+1) + (j-L_2)*(U_1-L_1+1) + (i-L_1)]*n}$$

Index Formula Derivation in Three Dimensional Array

Question: Given a 3D array A[2:8, –4:1, 6:10] with Base(A)= 200. Number of words per cell =4. Calculate address of A[5,–1,8] if elements are stored in

- Row major order fashion
- Column major order.

Index Formula Derivation in Three Dimensional Array

Solution: Given:

Lower Bound of dimension 1(L_1) = 2

Upper Bound of dimension 1 (U_1) = 8

Lower Bound of dimension 2(L_2) = -4

Upper Bound of dimension 2(U_2) = 1

Lower Bound of dimension 3(L_3) = 6

Upper Bound of dimension 3(U_3) = 10

Base Address = 200

n = 4

Index Formula Derivation in Three Dimensional Array

Solution: By formula (Row major order):

Address of A[i, j, k] = Base Address + $[(i–L_1)(U_2 –L_2+1)(U_3–L_3+1) + (j–L_2)(U_3–L_3+1) + (k–L_3)]*n$

Address of A[5,-1,8] = 200 + [(5 – 2) * (1 – (-4) + 1)* (10 – 6+1) + (-1 –(-4)*(10-6 +1) + (8 -6)] * 4

= 200 + [3 * 6* 5 + 3*5 + 2]*4

= 200 + 107*4 = 628

By formula (Column major order):

Address of A[i, j, k] = Base Address + $[(k–L_3)(U_1–L_1+1)(U_2–L_2+1) + (j–L_2) (U_1–L_1+1) + (i–L_1)]*n$

Address of A[5,-1,8] = 200 + [(8 – 6) * (8 – 2 + 1)* (1 – (–4)+1) + (-1 –(-4)*(8 -2 +1) + (5 -2)] * 4

= 200 + [2 * 7 *6 + 3*7 + 3]*4

= 200 + 432 = 632

Index Formula Derivation for Array

Multi-Dimensional Array

N-Dimensional Array

A 3-D array A[5][4][6] can be considered as the 5 two dimensional arrays of 4x6. For writing the Index formula for a N-dimensional array, the observation of 2-D and 3-D array derivations are used. Here it is assumed that an element requires B bytes for storage.

Row Major Order

$$A[k_1, k_2, \ldots, k_N] = \text{Base address} + [(k_1-L_1)(U_2-L_2+1)(U_3-L_3+1)\ldots(U_N-L_N+1) + (k_2-L_2)(U_3-L_3+1)\ldots(U_N-L_N+1) + \ldots + (k_{N-1}-L_{N-1})(U_N-L_N+1) + (k_N-L_N)]*B$$

Column Major Order

$$A[k_1, k_2, \ldots, k_N] = \text{Base address} + [(k_N-L_N)(U_1-L_1+1)(U_2-L_2+1)\ldots(U_{N-1}-L_{N-1}+1) + (k_{N-1}-L_{N-1})(U_1-L_1+1)(U_2-L_2+1)\ldots(U_{N-2}-L_{N-2}+1) + \ldots + (k_2-L_2)(U_1-L_1+1) + (k_1-L_1)]*B$$

4.4 Primitive operations on Array

There are various primitive operations that get operate on linear data structure like array are:

a) Traversing

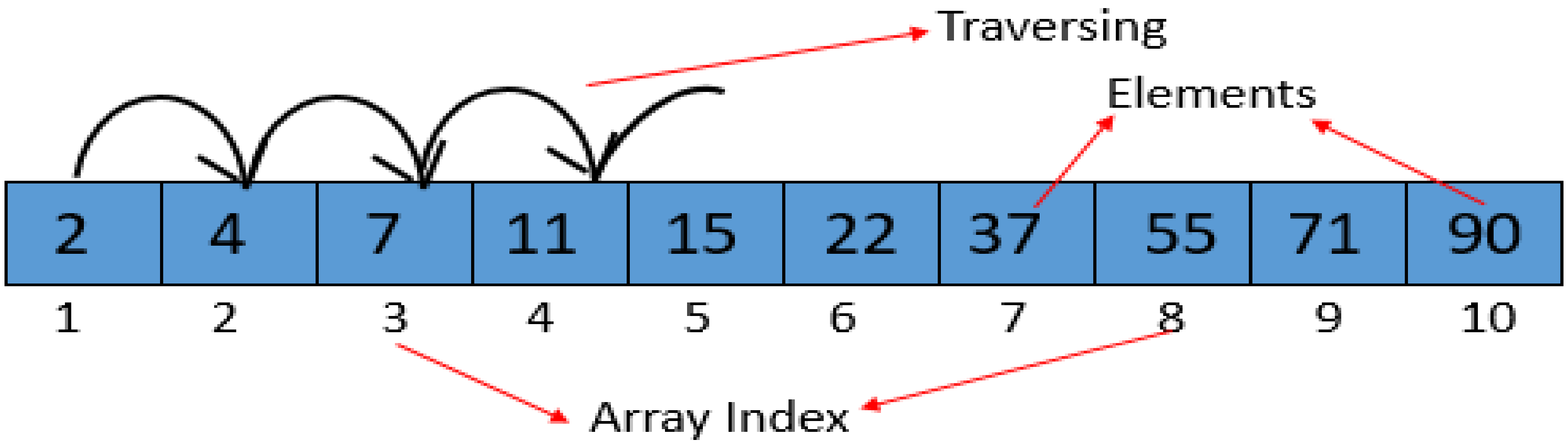
b) Insertion

c) Deletion

Note: For writing the algorithm, we are assuming that the lower bound of array is 1.

4.4.1 Traversal of an Array

- Explore the array elements one by one in sequential order..
- **Traversing elements exactly once.**
- Also called the visiting of an array.



Algorithm for Traversing an Array

In the given Algorithm, A[ ] is the array and N is the size of the array.

ALGORITHM Traverse (A[ ], N)

Input: Array A[ ] of size N

Output: Array elements in sequence

BEGIN:

```
        FOR i = 1 TO N DO
                WRITE(A[i])
```

Exploring the array elements from first index to last index.

END;

Complexity of Traversing an array

- **Time Complexity**: ϴ(N)

  Reason: The above algorithm requires execution of for loop N times. Hence, the number of statements to be executed is N.

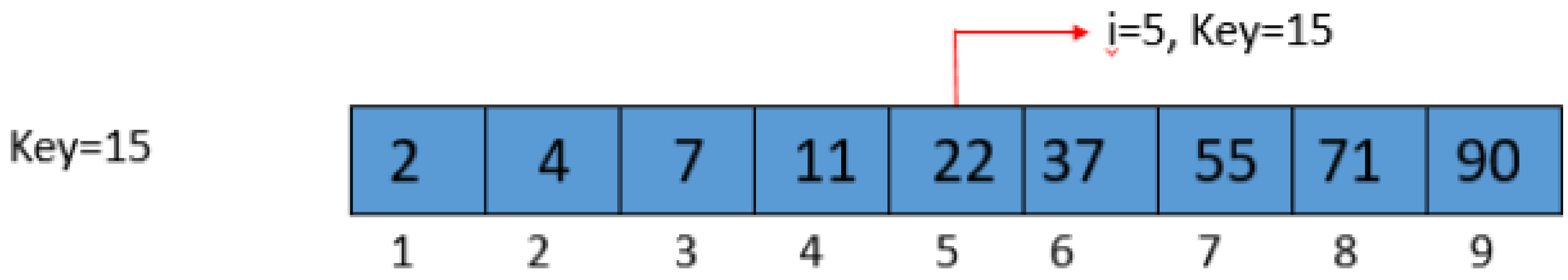
- **Space Complexity**: ϴ(1)

  Reason : The only extra variable taken here is i. Hence, the space complexity is constant.

4.4.2 Insertion in Array

- To insert a data element in an array.
- A new element can be added at the beginning, end, or at any given index based on the requirement.

Example: Insert an element (Key=15) at specific index (i=5) in the given array of size 9.



Continued.....

i=5, Key=15

Key=15

2	4	7	11	22	37	55	71	90
1	2	3	4	5	6	7	8	9

V shift IV shift III shift II shift I shift

FOR j = N TO i STEP–1 DO
A[j+1] = A[j]

2	4	7	11	22 →	37 →	55 →	71 →	90 →	
1	2	3	4	5	6	7	8	9	10

2	4	7	11	22	22	37	55	71	90
1	2	3	4	5	6	7	8	9	10

Insert Key=15

A[i] = Key
N = N+1

2	4	7	11	15	22	37	55	71	90
1	2	3	4	5	6	7	8	9	10

Algorithm for Inserting an Element in an Array

ALGORITHM Insertion (A[ ], N, i, Key)

Input: Array A[ ] of size N, position of insertion i, data element for insertion Key

Output: Updated array after insertion

BEGIN:

```
    FOR j = N TO i STEP–1 DO
        A[j+1] = A[j]
    A[i] =Key
    N = N+1
```

Shifting of elements from N^{th} index to i^{th} index. An element is shifted at one higher index to the current index.

Placing Key at i^{th} index.

Incrementing the array size by 1.

END;

Complexity of Inserting an element in an Array

- **Time Complexity:**

Worst Case: O(N)

Reason: When the element is to be inserted at the beginning, N number of shifting will be required and two statements to assign the value and increase the value of N. Hence, N+2 statements will be executed. Average case complexity is same as worst case complexity.

Best Case : Ω(1)

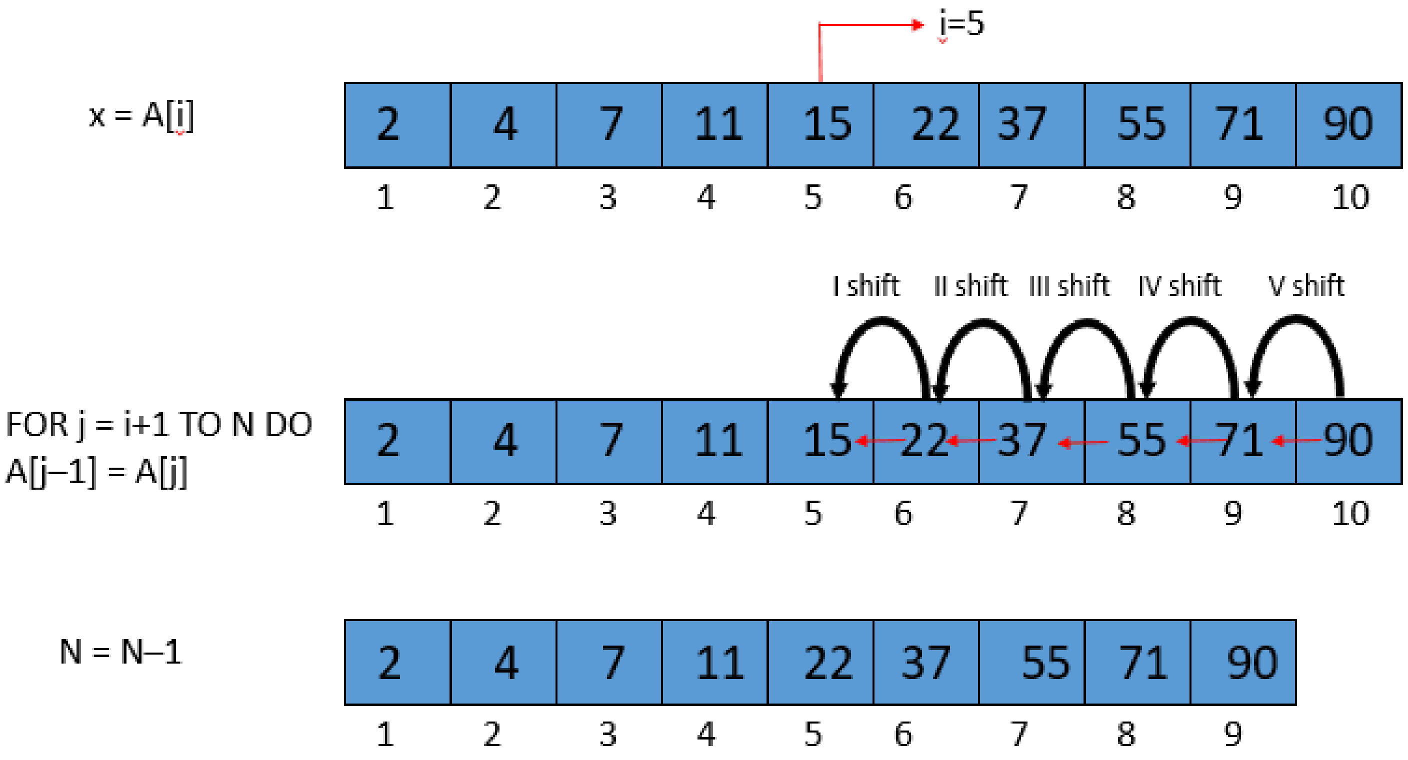
Reason: When the element is to be inserted at the end, no shifting is required. Therefore, only two statements will be executed.

- **Space Complexity :ϴ(1).**

Reason: The only extra variable taken here is j, hence the space complexity is ϴ(1).

4.2.3 Deletion in Array

- To delete an element from the given index in the array and re-organizes the array elements with shifting.



Algorithm for Deleting an Element in an Array

Input: Array A[ ] of size N, position of deletion i

Output: Updated array after deletion

BEGIN:

```
        x = A[i]                    }  Saving the element to be deleted

        FOR j = i+1 TO N DO         }  Shifting of elements from (i+1)th index to Nth index. An
              A[j–1] =A[j]          }  element is shifted at one lower index to the current index.

        N = N–1                     }  decrementing the array size by 1.

        RETURN x                    }  Returning the deleted element
END;
```

***Note:** When we delete an element from any data structure, deleted element should be returned to the calling function.*

Application Problems Related to Array

Insertion in sorted 1-D Array

To insert an element “Key” in a sorted array (increasing order), the following steps need to be performed:

1. A search operation for the appropriate position of insertion.
2. This position needs to be made vacant by shifting the elements to their right.
3. Insert the element at this position.

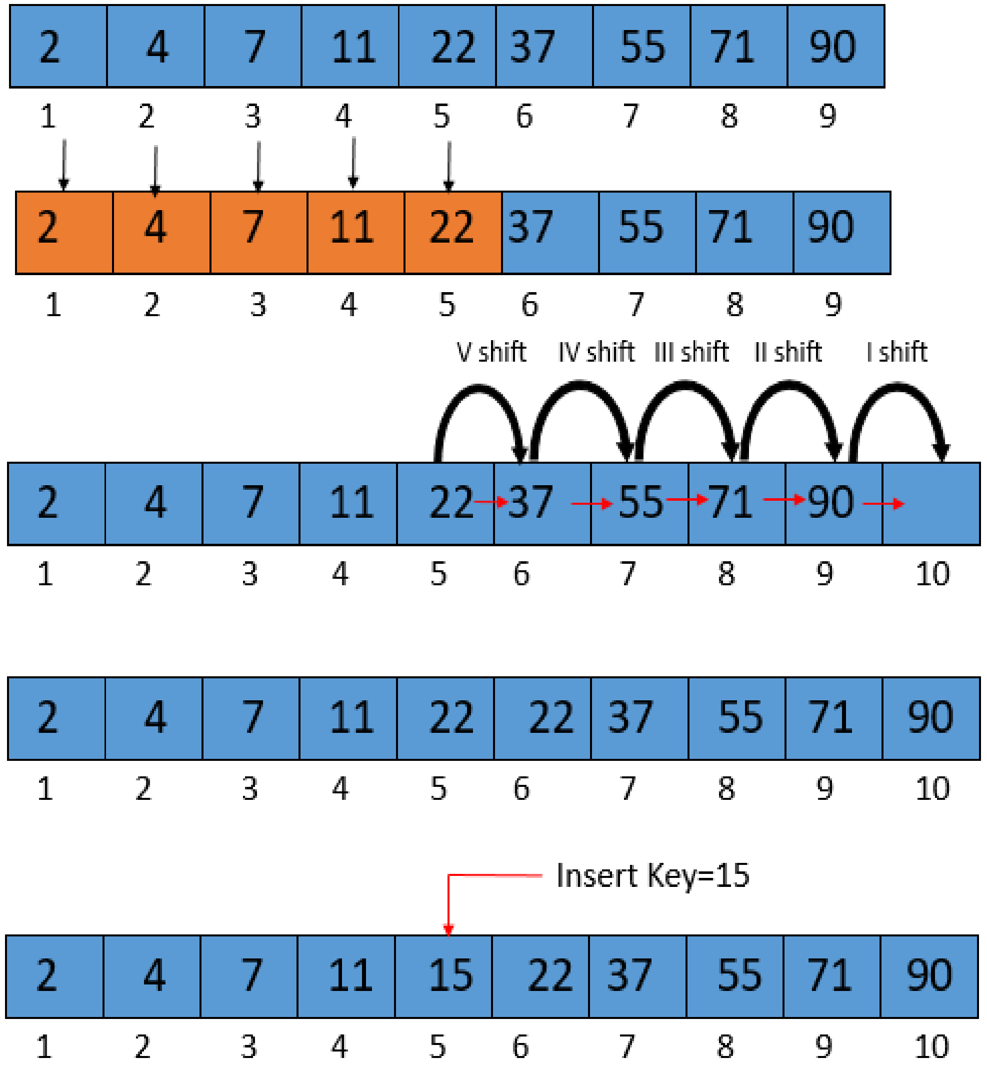
Insertion in sorted 1-D Array

Key= 15

i=1
WHILE A[i]<Key DO
i = i+1

FOR j = N TO i STEP–1 DO
A[j+1] = A[j]

A[i] = key
N = N+1



Insertion in sorted 1-D Array(Contd.)

ALGORITHM InsertionSortedArray(A[ ], N, key)

Input: Array A[ ] of size N, data element for insertion Key

Output: Updated array after insertion

```
BEGIN:
        i =1
        WHILE A[i]<key DO
                i = i+1
        FOR j =N TO i STEP–1 DO
                A[j+1] = A[j]
        A[i] = key
        N=N+1
END;
```

Insertion in sorted 1-D Array(Contd.)

Time Complexity: Ɵ(N)

If the desired place for the insertion is ‘I’ (found with the search) then ‘n–I’ shifting will be required.

Search operations require I statement executions and shifting requires n-I statement executions.

Apart from searching and shifting, three other statements will get executed. Total statements execution is N+3 i.e. **Ɵ(N).**

Space Complexity: Ɵ (1)

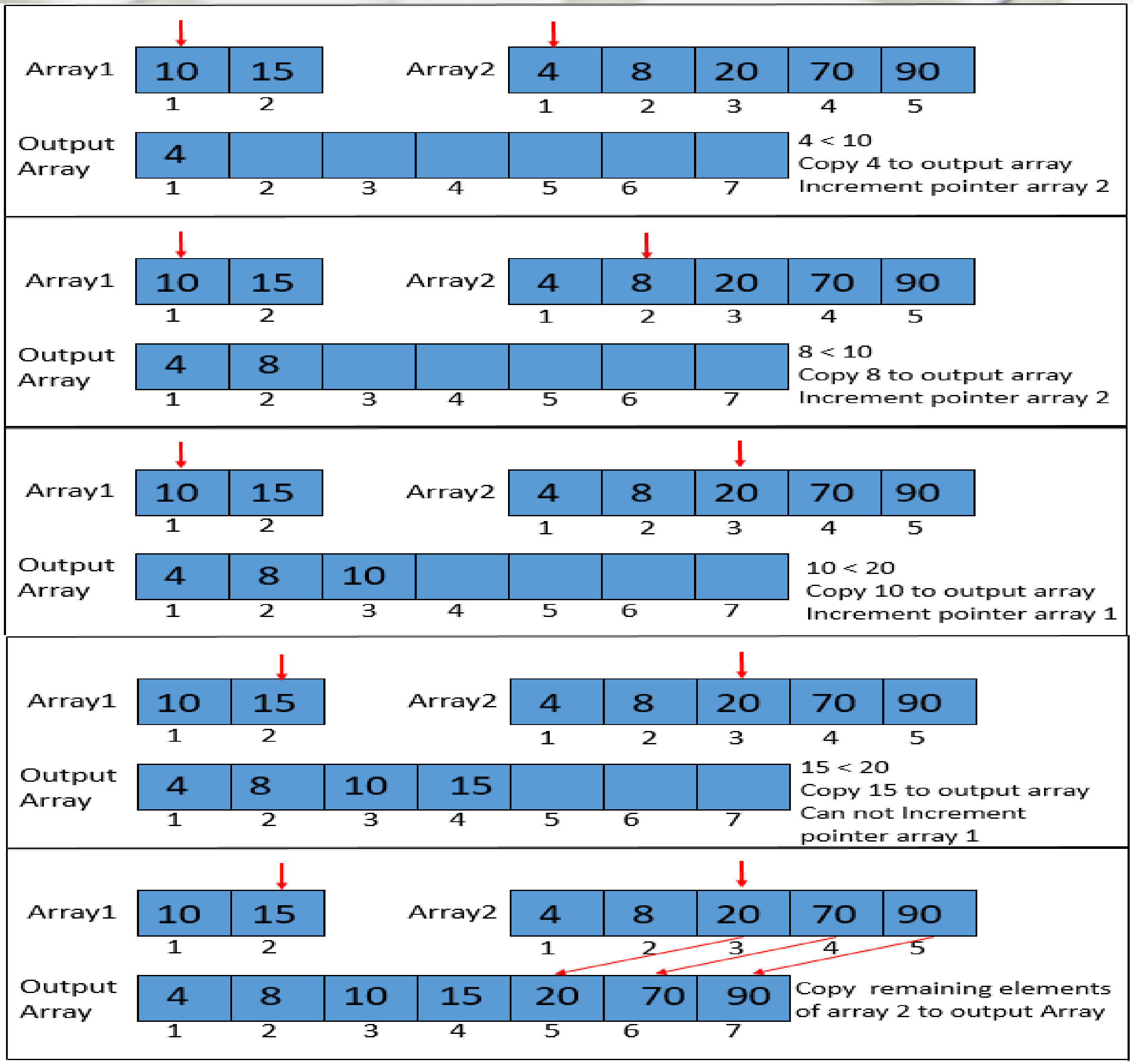
The only variables taken here are i and j; hence the space complexity is constant, i.e. Ɵ(1).

Merging of two sorted arrays

To merge two sorted arrays, the following steps need to be performed:

- Given two sorted Arrays array1 and array2.
- The task is to combine these two sorted arrays to form a single sorted array.
- The steps given below are used to perform the merging.
- It is assumed that m represents the size of array1 and n represents the size of array2.

Example:



Array1
10 15
1 2
Array2
4 8 20 70 90
1 2 3 4 5
Output Array
4
1 2 3 4 5 6 7
4 < 10
Copy 4 to output array
Increment pointer array 2
Array1
10 15
1 2
Array2
4 8 20 70 90
1 2 3 4 5
Output Array
4 8
1 2 3 4 5 6 7
8 < 10
Copy 8 to output array
Increment pointer array 2
Array1
10 15
1 2
Array2
4 8 20 70 90
1 2 3 4 5
Output Array
4 8 10
1 2 3 4 5 6 7
10 < 20
Copy 10 to output array
Increment pointer array 1
Array1
10 15
1 2
Array2
4 8 20 70 90
1 2 3 4 5
Output Array
4 8 10 15
1 2 3 4 5 6 7
15 < 20
Copy 15 to output array
Can not Increment
pointer array 1
Array1
10 15
1 2
Array2
4 8 20 70 90
1 2 3 4 5
Output Array
4 8 10 15 20 70 90
1 2 3 4 5 6 7
Copy remaining elements
of array 2 to output Array

Merging of two sorted arrays(Contd.)

ALGORITHM: MergeArr(A[ ], m, B[ ], n)

Input: Array A[ ] of size m, Array B[ ] of size n

Output: Array after merging of elements in A[ ] and B[ ]

```
BEGIN:
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        C[k]=A[i]
                        i=i+1
                        k=k+1
                ELSE
                        C[k]=B[j]
                        j=j+1
                        k=k+1
        WHILE i<=m DO
                C[k]=A[i]
                i=i+1
                k=k+1
        WHILE j<=n DO
                C[k]=B[j]
                j=j+1
                k=k+1
        RETURN C
END;
```

Merging of two sorted arrays(Contd.)

Time Complexity:

- The process of merging requires the comparison of each element of array1 with that of array2.
- An element is added to the output array after the comparison.
- Since m+n elements will be added in the output array, total m+n comparisons are required.
- Hence Time Complexity is **ϴ (m+n)**.

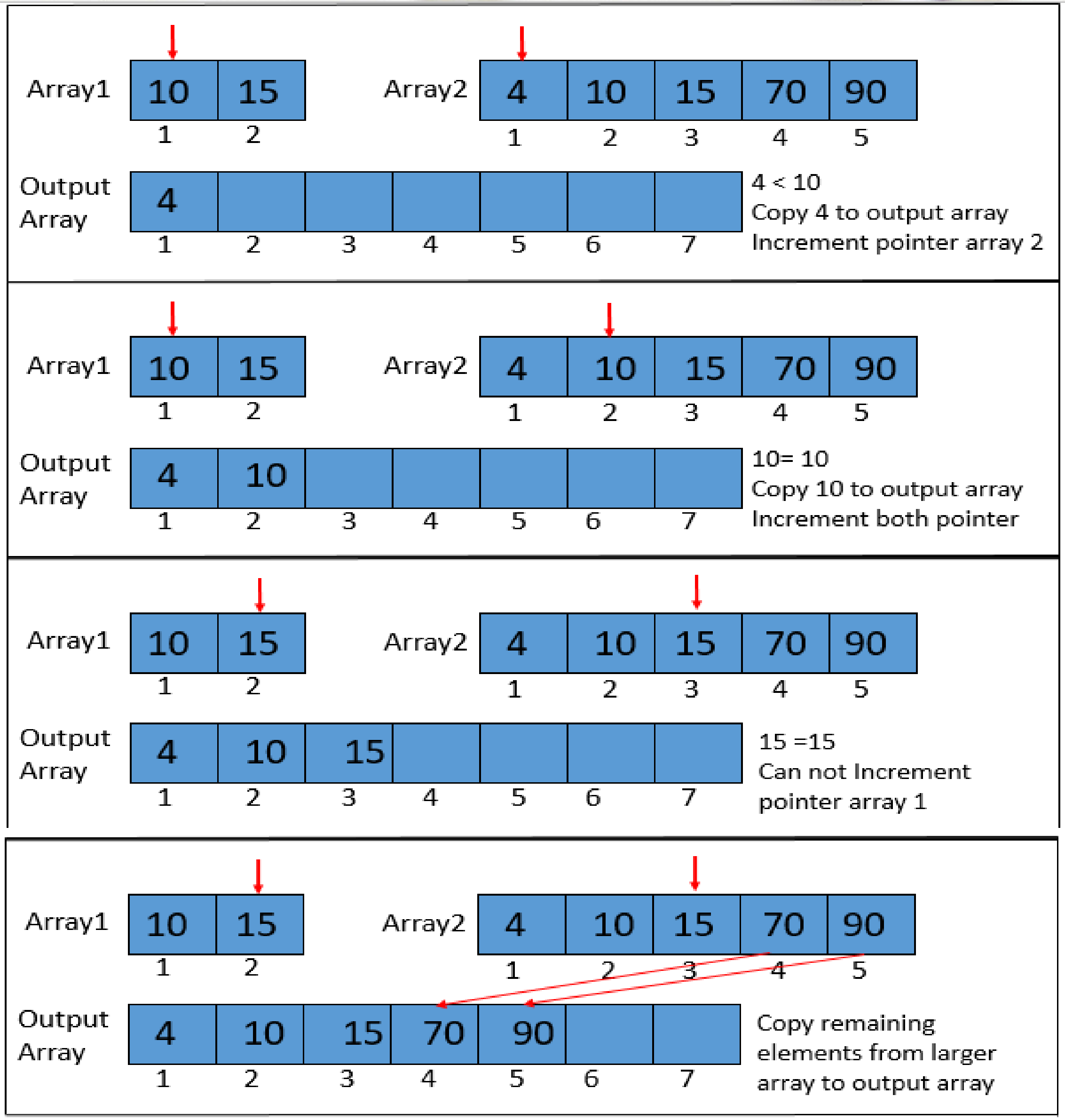
Space Complexity:

- An array of size m+n is required for storage of the output.
- Alongside, space is required for variables i, j and k.
- the total space required is m+n+3 which can be represented as **ϴ (m+n)**.

Set Union operation

- It is assumed here that the set elements are arranged in an array in ascending sequence.
- The task is to combine these two sorted arrays to form a single sorted array (common elements should be added only once in the output array).
- The steps given below are used to perform the union operation. It is assumed that m represents the size of Set 1 (array1) and n, the size of Set 2 (array2) .

Example:



Array1
10 15
1 2
Array2
4 10 15 70 90
1 2 3 4 5
Output Array
4
1 2 3 4 5 6 7
4 < 10
Copy 4 to output array
Increment pointer array 2
Array1
10 15
1 2
Array2
4 10 15 70 90
1 2 3 4 5
Output Array
4 10
1 2 3 4 5 6 7
10= 10
Copy 10 to output array
Increment both pointer
Array1
10 15
1 2
Array2
4 10 15 70 90
1 2 3 4 5
Output Array
4 10 15
1 2 3 4 5 6 7
15 =15
Can not Increment
pointer array 1
Array1
10 15
1 2
Array2
4 10 15 70 90
1 2 3 4 5
Output Array
4 10 15 70 90
1 2 3 4 5 6 7
Copy remaining
elements from larger
array to output array

Set Union operation(Contd.)

ALGORITHM: SetUnion(A[ ], m, B[ ], n)

Input: Array A[ ] of size m, Array B[ ] of size n

Output: Array after union of elements in A[ ] and B[ ]

BEGIN:

```
        C[m+n]
        Output array of size m+n
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        C[k]=A[i]
                        i=i+1
                        k=k+1
                ELSE
                        IF A[i]==B[j] THEN
                                C[k]=B[j]
                                i=i+1
                                j=j+1
                                k=k+1
                        ELSE
                                C[k]=B[j]
                                j=j+1
                                k=k+1
        WHILE i<=m DO
                C[k]=A[i]
                i=i+1
                k=k+1
        WHILE j<=n DO
                C[k]=B[j]
                j=j+1
                k=k+1
        RETURN C
```

END;

Set Union operation(Contd.)

Time Complexity: This process of merging requires comparison of each element of Array1 with that of Array2. An element is added to the output array after the comparison. Since maximum m+n elements will be added in the output array, total m+n comparisons are required. Hence Time Complexity is **ϴ (m+n)**.

Space Complexity: An array of size m+n is required for storage of the output. Alongside, space is required for variables i, j and k. Thus the total space required is m+n+3 which can be represented as **ϴ (m+n)**.

Problems for practice

1. **Finding the number which is not repeated in Array of integers**

ALGORITHM: NonRepetitions(A[ ], N, k)

Input: Array A[ ] of size N, the largest element k

Output: The elements which are not repeated

BEGIN:

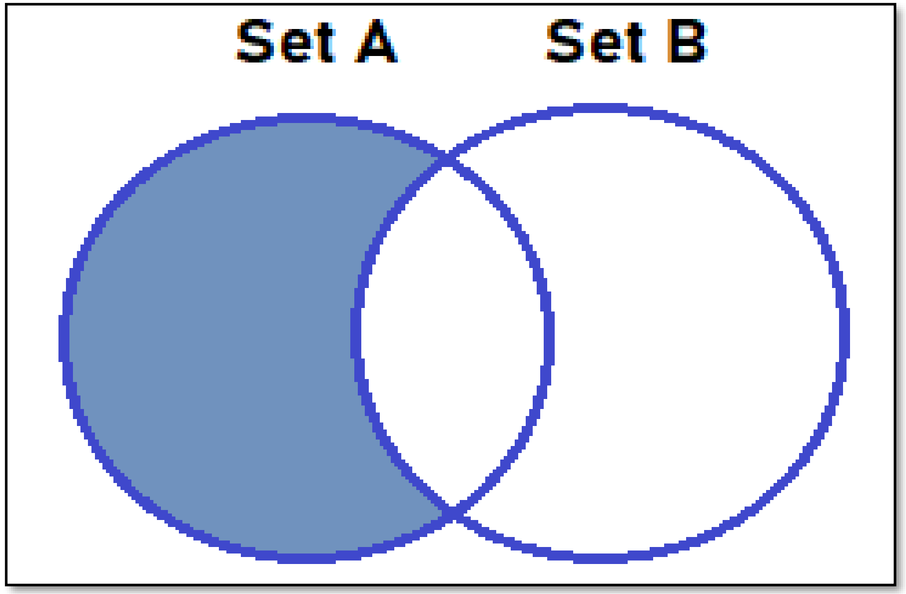
```
        C[k] ={0}                      } DAT of size k+1 with all elements initialized with 0

        FOR i = 1 TO N DO              } Frequency count of all elements in array
              C[A[i]]=C[A[i]]+1

        FOR i =1 TO k DO               } Finding the elements which have frequency 1 i.e.,
              IF C[i]==1 THEN            not repeated
                    WRITE(C[i])
```

END;

Continued...

2. Finding the elements of one set that does not belong to the other set

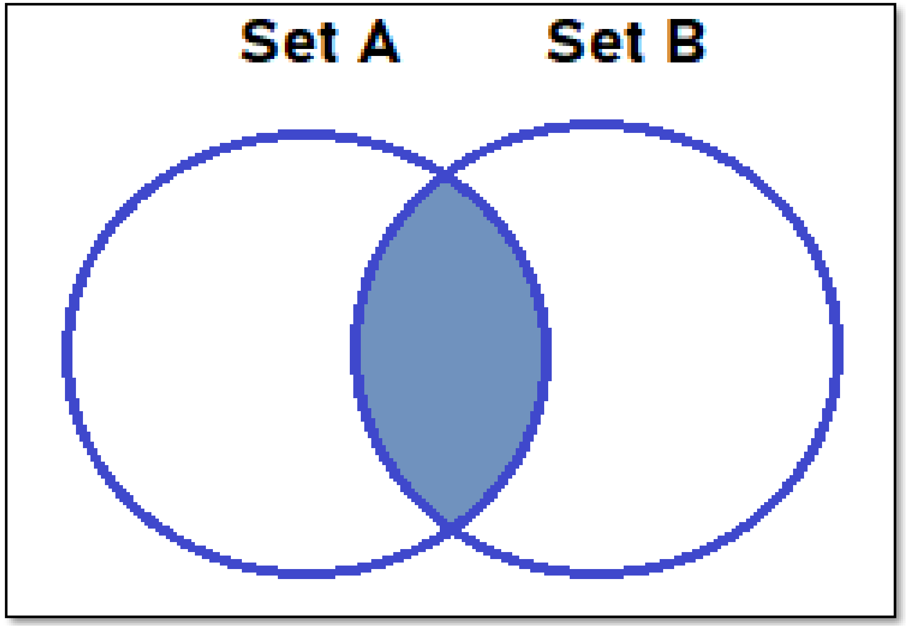
It is assumed here that the set elements are arranged in ascending sequence. The algorithm traverses both the array simultaneously and finds the common elements. These elements are not included in the output array. Rest of the elements from set A are added to the output array. This is more like finding the set Difference of A from B.



Continued...

3. Set Intersection Operation

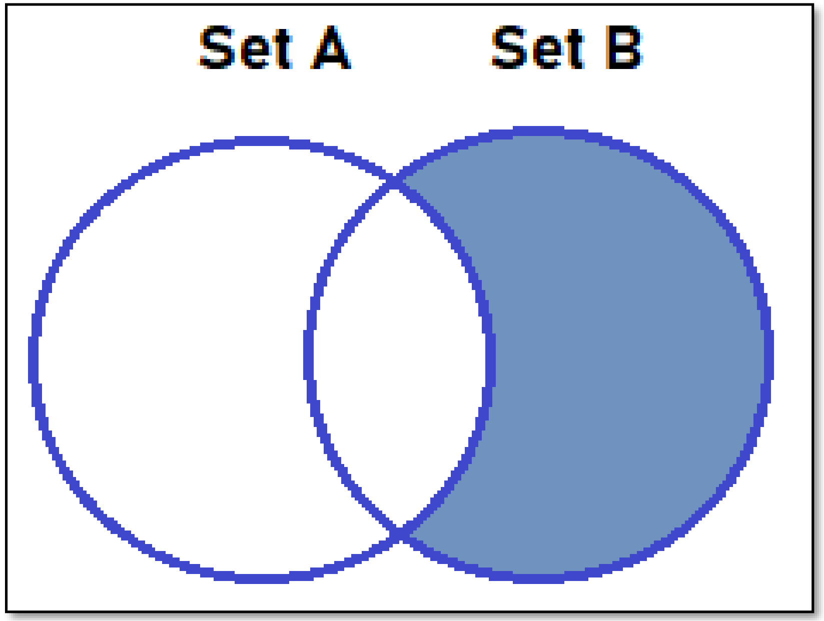
It is assumed here that the set elements are arranged in an array in ascending sequence. The task is to find the common elements and add them to the final array. The steps given below are used to perform the intersection operation. It is assumed that m represents the size of Set 1 (array1) and n, the size of Set 2 (array2).



Continued...

4. **Set Difference Operation**

The difference (B–A) will output the elements from **array B** that are not in the **array A.** It is assumed here that the set elements are arranged in ascending sequence. The algorithm traverses both the array simultaneously and finds the common elements. These elements are not included in the output array. The rest of the elements from set B are added in the output array.



Question:

int val[2][4]={1,2,3,4,5,6,7,8};

4 will be value of

What would be the output of the above code ? Choose the correct option.	
A. val [0][4]	B. val [1][4]
C. val [1][1]	D. None of these

Question:

```
# include <stdio.h>
int main()
{
int arr[5],i=0;
while(i<5)
        arrr[i]=++i;
for(i=0;i<5;i++)
        printf("%d",arr[i]);
printf("\n");
return 0;
}
```

Tell the output	
A. 12345	B. 01234
C. garbage value 1234	D. 23456

Question:

```
# include <stdio.h>
int main()
{
        int a[3] = {1, 2, 3};
        int *p = a;
        printf("%p\t%p", p, a);
}
```

What will be the output of this code	
A. Compile time error	B. Same answer is printed
B.Different answer is printed	D. None

Question:

```
# include <stdio.h>
int main()
{
        int a[3] = {1, 2, 3};
        int *p = a;
        printf("%p\t%p", p, a);
}
```

What will be the output of this code	
A. Compile time error	B. Same answer is printed
B. Different answer is printed	D. None

Question:

```
int main() {
        int i; int arr[5] = {1};
        for (i = 0; i < 5; i++)
                printf("%d ", arr[i]);
        return 0;
}
```

What will be the output of this code	
A. followed by four garbage values:	B1 1 1 1 1
C. 10000	D. None

Question:

```
#include<stdio.h>
int main()
{
 int a[5] = {5, 1, 15, 20, 25};
 int i, j, m;
 i = ++a[1];
 j = a[1]++;
m = a[i++];
printf("%d, %d, %d", i, j, m);
return 0;}
```

What will be the output of this code	
A. 2 5 15	B 1 2 5
C. 3 2 15	D. None

Question:

```
#include<stdio.h>
 int main()
{
    int a[3] = {20,30,40};
    int b[3];
    b=a;
    printf("%d", b[0]);
}
```

What will be the output of this code	
A. 20 30 40	B. Compiler error
C. 0	D. None

Question:

A program P reads in 500 integers in the range [0..100] experimenting the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies?

What will be the output of this code	
A. An array of 50 numbers	B. An array of 100 numbers
C. An array of 500 numbers	D. A dynamically allocated array of 550 numbers

Question:

```
#include<stdio.h>
int main()
{
int arr[1]={10};
printf("%d\n", 0[arr]);
 return 0; }
```

What will be the output of this code	
A. 10	B.1
C. 6	D. 8

Question:

```
#include<stdio.h>
int main()
{
   int arr[1]={10};
   printf("%d\n", 0[arr]);
   return 0;
}
```

What will be the output of this code	
A. 10	B.1
C. 6	D. 8

Question:

- Which of the following concepts make extensive use of arrays?

What will be the output of this code	
A. Binary Trees	B. Scheduling
C. Caching	D. Spatial locality

Question:

- Let A[1...n] be an array of n distinct numbers. If i < j and A[i] > A[j], then the pair (i, j) is called an inversion of A. What is the expected number of inversions in any permutation on n elements ?

What will be the output of this code	
A. θ(n)	B.θ(n2)
C. O(1)	C. O(n)

Question:

Which of the following operations is not O(1) for an array of sorted data. You may assume that array elements are distinct.

What will be the output of this code	
A. Find the ith largest element	B. Delete an element
C. Find the ith smallest element	C. All of the above

Application of 2D Array

Applications of Two Dimensional Array

a) **Graph:** Representation of Graph data structure using adjacency matrix.

b) **Computer Graphics:** Representation of pixel information. Various operations like translation, rotation, scaling etc. can be performed with direct computations on pixel matrix.

c) **Geology:** Matrices are used for the representation of dynamics of Earth e.g., seismic surveys. Matrices can also be used to draw graphs, statistics, perform scientific calculations.

d) **Economics:** Use of Decision matrix can help to select the best course plan for business processes. Decision matrix provides a way to compare different solutions to a given problem and select the best one.

Applications of Two Dimensional Array

e) **In Auto CAD Civil Construction:** here structure of buildings can be represented as matrices by storing their coordinates and angles.

f) **Manage Databases:** Managing Tables

g) **Robotics and Automation:** Here, matrices are used to store the direction coordinates and angles related to the robot's movement.

h) **Data Security by providing encryption and Decryption:** In cryptography, encryption and decryption is done by performing the basic matrix operations like matrix multiplication, transpose etc.

Matrix Traversal

P[ ][ ] is the array and M x N is the size of the array.

ALGORITHM Traverse (A[ ], M, N)

Input: Array P[ ][ ] of size M x N

Output: Array elements in sequence

BEGIN:

```
        FOR i = 1 TO M DO
                FOR j = 1 TO N
                        WRITE(P[i][j])
END;
```

Exploring the array elements in row-major order. For this, first fix the row number i.e. i and traverse through all the columns. Then, repeat this for the next value of i until all rows are covered.

Column Index

Row Index

	1	2	3
1	2	7	3
2	11	3	4
3	2	1	6

Matrix P[ ][ ]

Matrix Traversal

Time Complexity: ϴ(N)

The above algorithm requires execution of loop M x N times. Hence, the number of statements to be executed are **ϴ(**Mx N**).**

Space Complexity: ϴ(1)

The only extra variables taken here is I and j. Hence, the space complexity is constant.

Matrix Addition

P and Q are the two matrices of the same order (same number of rows and columns).

Column Index

Row Index	1	2	3
1	5	7	3
2	11	3	4
3	2	1	6

Matrix P[ ][ ]

+

Column Index

Row Index	1	2	3
1	7	1	1
2	2	3	4
3	1	1	2

Matrix Q[ ][ ]

=

S[1][1]= P[1][1] + Q[1][1] = 5 + 7 = 12

Column Index

Row Index	1	2	3
1	12	8	4
2	13	6	8
3	3	2	8

Matrix S[ ][ ]

Matrix Addition

```
ALGORITHM: MatrixAddition(P[ ][ ], Q[ ] [ ], R1, C1, R2, C2)
Input: Array P[ ][ ] of size R1 x C1 and Array Q[ ][ ] of size R2 x C2
Output: Array of size R1 x C1 or R2 x C2
BEGIN:
        S[R1] [C1]
        IF R1 = = R2 AND C1 == C2 THEN
                FOR i = 1 TO R1 DO
                        FOR j = 1 TO C1
                                S[i][j] = P[i][j] + Q[i][j]

                RETURN S
        ELSE
                WRITE("ADDITION is not possible")

END;
```

Checking the order(same number of rows and columns). If the order is the same addition can be performed.

Performing the addition and saving the result in output matrix S.

Returning the output array

If the order is not same.

Matrix Addition

Time Complexity: Since addition is performed element by element and there are R_1xC_1 elements, the time complexity will be **$\Theta(R_1*C_1)$**, where R1 is the number of rows and C1 is the number of columns.

Space complexity: An additional matrix of size R_1xC_1 is used and two variables i and j. Hence, the space complexity is **$\Theta(R_1*C_1)$.**

Matrix Subtraction

P and Q are the two matrices of the same order (same number of rows and columns).

Column Index

Row Index	1	2	3
1	9	7	3
2	11	3	4
3	2	1	6

Matrix P[ ][ ]

−

Column Index

Row Index	1	2	3
1	7	1	1
2	2	3	4
3	1	1	2

Matrix Q[ ][ ]

=

S[1][1]= P[1][1] – Q[1][1] = 9 – 7 = 2

Column Index

Row Index	1	2	3
1	2	6	2
2	9	0	0
3	1	0	4

Matrix S[ ][ ]

Matrix Subtraction

ALGORITHM: MatrixSubtraction(P[ ][ ], Q[ ] [ ], R1, C1, R2, C2)

Input: Array P[ ][ ] of size R1 x C1 and Array Q[ ][ ] of size R2 x C2

Output: Array of size R1 x C1

BEGIN:

```
        S[R1] [C1]
        IF R1 == R2 AND C1 == C2 THEN
                FOR i = 1 TO R1 DO
                        FOR j = 1 TO C1
                                S[i][j] = P[i][j] – Q[i][j]
                RETURN S
        ELSE
                WRITE("SUBTRACTION is not possible")
```

Checking the order(same number of rows and columns). If the order is the same subtraction can be performed.

Performing the subtraction and saving the result in output matrix S.

Returning the output array

If the order not same.

END;

Matrix Subtraction

Time Complexity: Subtraction is performed element by element and there are R_1xC_1 elements, the time complexity will be **$\Theta(R_1*C_1)$**, where R1 is the number of rows and C1 is the number of columns.

Space complexity: An additional matrix of size R_1xC_1 is used and two variables i and j. Hence, the space complexity is **$\Theta(R_1*C_1)$.**

Applications of Two Dimensional Array

Matrix Multiplication

Let P and Q are the two matrices to be multiplied(P.Q), number of columns in P must be equal to the number of rows in Q.

Let P be a R_1xC_1 matrix and Q be a R_2xC_2 matrix. Then the product of the matrices P and Q will be of the order of mxp.

Column Index

Row Index	1	2	3
1	2	7	3
2	11	3	4
3	2	1	6

Matrix P[ ][ ]

×

Column Index

Row Index	1	2	3
1	3	1	1
2	2	3	4
3	1	1	2

Matrix Q[ ][ ]

=

S[1][1]= P[1][1] x Q[1][1] + P[1][2] x Q[2][1] + P[1][3] x Q[3][1]
= 2x3 + 7x2 + 3x1
= 6 + 14 + 3
= 23

Column Index

Row Index	1	2	3
1	23	26	36
2	43	24	31
3	14	11	18

Matrix S[ ][ ]

Matrix Multiplication

```
ALGORITHM: MatrixMultiply(P[ ][ ], Q[ ] [ ], R1, C1, R2, C2)
BEGIN:
        M[R1] [C2]
        IF C1 == R2 THEN
                FOR i = 1 to R1 DO
                        FOR j = 1 to C2 DO
                                M[i][j] = 0
                                FOR k = 1 to C1 DO
                                        M[i][j] = M[i][j]+ P[i][k] + Q[k][j]
                RETURN M
        ELSE
                WRITE("Matrix multiplication is not possible")
END;
```

Checking the order whether multiplication can be performed.

Performing the multiplication and saving the result in output matrix M.

Returning the output array

If C1 is not equal to R2, matrices are not multipliable.

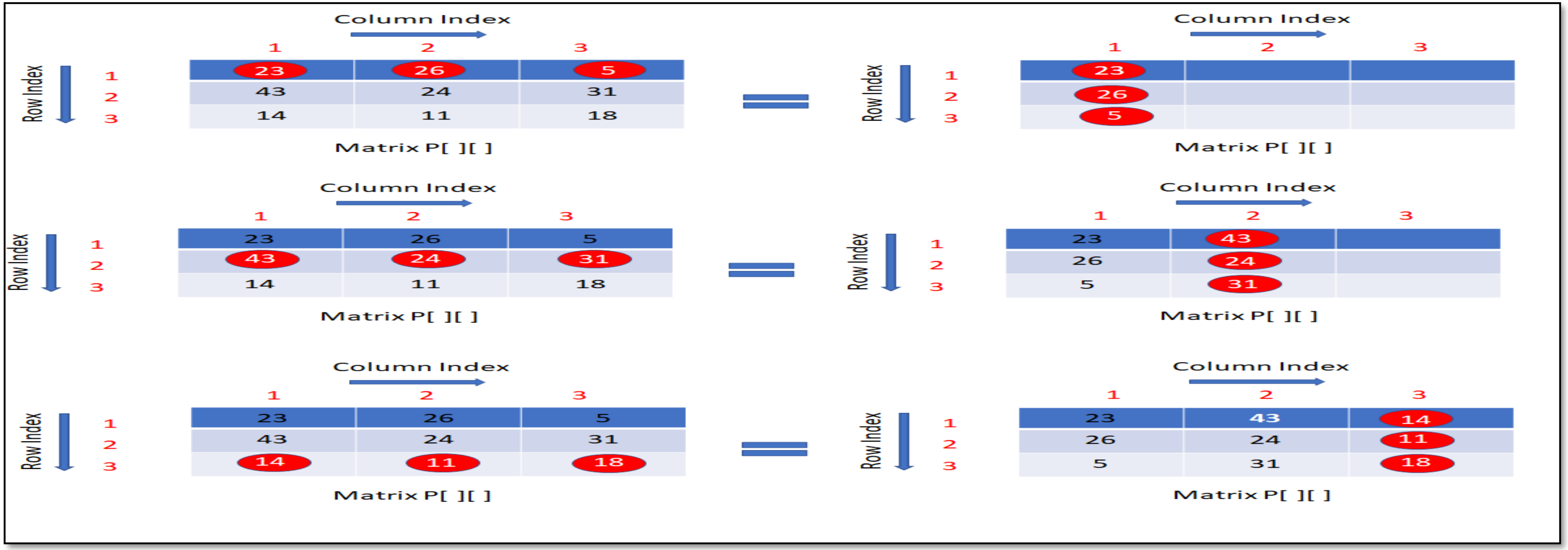
Matrix Multiplication

Time Complexity: The order of the first matrix is R_1xC_1, the order of the second matrix is R_2xC_2. Total multiplications performed to obtain the output matrix will be of the order of R_1xC_2 will be $R_1.C_1.C_2$. Hence, the time complexity is **$\Theta(R_1.C_1.C_2)$**.

Space complexity: An additional matrix of size R_1xC_2 is used and three variables i, j and k. Hence, the space complexity is **$\Theta(R_1C_2)$.**

Transpose of a matrix

Interchange the rows elements with the corresponding columns elements. If a matrix P is of order rxc then transposed matrix will be of order cxr.



Transpose of a matrix

```
ALGORITHM: MatrixTranspose(P[ ][ ], R, C)
BEGIN:
        T[C][R]
        FOR i = 1 to R DO
                FOR j = 1 to C DO
                        T[i][j] = P[j][i]
        RETURN T
END;
```

Performing the transpose and saving the result in output matrix P.

Returning the output array

Transpose of a matrix

Time complexity: Let P be an RxC matrix. Transpose will require RxC times placement of data from original matrix to transposed matrix. Thus, complexity of transpose operation will be **Ѳ(C.R)**.

Space complexity: An additional matrix of size CxR is used and two variables i, j. Hence, the space complexity is **Ѳ(C.R).**

Determinant of a Matrix

Determinant of a matrix is calculated by the element of a square matrix. The determinant of a matrix is denoted by the |A|, det(A), det[A].

a	b
c	d

Determinant of a 2X2 matrix is |A| =a*d – b*c

a	b	c
d	e	f
g	h	i

Determinant of a 3x3 matrix is |A| = a(e*i – f*h) – b(d*i – f*g) + c(d*h – e*g)

Determinant of a Matrix

Column Index →

Row Index ↓	1	2	3
1	1	1	1
2	2	3	2
3	3	4	5

Matrix A[ ][ ]

Finding Determinant

$$1\begin{vmatrix} 3 & 2 \\ 4 & 5 \end{vmatrix} - 1\begin{vmatrix} 2 & 2 \\ 3 & 5 \end{vmatrix} + 1\begin{vmatrix} 2 & 3 \\ 3 & 4 \end{vmatrix}$$

$$|A| = 1*(3*5 - 4*2) - 1*(\ 2*5 - 3*2) + 1*(2*4 - 3*3) = 7 - 4 - 1 = 3$$

Determinant of a Matrix

ALGORITHM: MatrixDeterminant(P[ ][ ], R1, C1)

BEGIN:

Det = Det+ P[1,1]*(P[2,2]*P[3,3] – P[2,3]*P[3,2]) – P[1,2]*(P[2,1]*P[3,3] – P[3,1] * P[2,3]) + P[1,3]*(P[2,1]*P[3,2] – P[2,2]*P[3,1])

END;

Time Complexity: As N^2 multiplications are required for finding determinants, Time complexity will be **ϴ** (N^2).

Space Complexity: There are no additional space required; hence the space complexity will be **ϴ(1)**

Question:

Two matrices M_1 and M_2 are to be stored in arrays A and B respectively. Each array can be stored either in row-major or column-major order in contiguous memory locations. The time complexity of an algorithm to compute $M_1 \times\times M_2$ will be	
A. Best if A is in row-major and B is in column-major order	B. Best if A is in row-major and B is in column-major order
C. Best if both are in column-major order	D. Independent of the storage scheme

Question:

Two matrices M_1 and M_2 are to be stored in arrays A and B respectively. Each array can be stored either in row-major or column-major order in contiguous memory locations. The time complexity of an algorithm to compute $M_1 \times\times M_2$ will be	
A. Best if A is in row-major and B is in column-major order	B. Best if A is in row-major and B is in column-major order
C. Best if both are in column-major order	D. Independent of the storage scheme

Question:

Which of the following property does not hold for matrix multiplication?	
A. Associative	B. Distributive
C. Commutative	D. Additive Inverse

Question:

Which of the following property does not hold for matrix multiplication?	
A. Associative	B. Distributive
C. Commutative	D. Additive Inverse

Question:

If row-major order is used, how is the following matrix stored in memory? a b c d e f g h i	
A. abcdefghi	B. adgbehefi
C. Ihgfedcba	D. ifchebgda

Question:

If row-major order is used, how is the following matrix stored in memory? a b c d e f g h i	
A. abcdefghi	B. adgbehefi
C. Ihgfedcba	D. ifchebgda

Question:

if column-major order is used, how is the following matrix stored in memory? a b c d e f g h i	
A. abcdefghi	B. adgbehefi
C. Ihgfedcba	D. ifchebgda

Question:

if column-major order is used, how is the following matrix stored in memory? a b c d e f g h i	
A. abcdefghi	B. adgbehefi
C. Ihgfedcba	D. ifchebgda

Question:

Suppose you are given an array s[1..n] and a procedure reverse (s, i, j) which reverse the order of elements in s between positions i and j (both inclusive). What does the following sequence do, where 1≤k<n:

reverse (s, 1, k);
reverse (s, k+1, k);
reverse (s, 1, n);

A. Rotates s left by k positions	B. Leaves s unchanged
C. Reverse all elements of s	D. None of the above

Question:

Suppose you are given an array s[1..n] and a procedure reverse (s, i, j) which reverse the order of elements in s between positions i and j (both inclusive). What does the following sequence do, where 1≤k<n: reverse (s, 1, k); reverse (s, k+1, k); reverse (s, 1, n);	
A. Rotates s left by k positions	B. Leaves s unchanged
C. Reverse all elements of s	D. None of the above

Question

Let A be a two dimensional array declared as follows: A : array [1... 10] [1... 15] of integer; Assuming that each integer takes one memory locations the array is stored in row-major order and the first element of the array is stored at location 100, what is the address of the element A[i] [j]?	
A. 15i + j + 84	B. 15j + i + 84
C. 10i + j + 89	D. 10j + i + 89

Question

Let A be a two dimensional array declared as follows: A : array [1... 10] [1... 15] of integer; Assuming that each integer takes one memory locations the array is stored in row-major order and the first element of the array is stored at location 100, what is the address of the element A[i] [j]?	
A. 15i + j + 84	B. 15j + i + 84
C. 10i + j + 89	D. 10j + i + 89

Question

A matrix has p number of rows and q number of columns. That matrix is called Sparse Matrix when	
A. Total number of Zero elements > (p*q)/2	B. Total number of Zero elements = p+q
C. Total number of Zero elements = p/q	D. Total number of Zero elements = p-q

Question

A matrix has p number of rows and q number of columns. That matrix is called Sparse Matrix when	
A. Total number of Zero elements > (p*q)/2	B. Total number of Zero elements = p+q
C. Total number of Zero elements = p/q	D. Total number of Zero elements = p-q

Question:

From the following data structures which cannot be used to represent a Sparse Matrix?	
A. Linked List	B. Arrays
C. Heap	D. Dictionary of Keys

Question:

From the following data structures which cannot be used to represent a Sparse Matrix?	
A. Linked List	B. Arrays
C. Heap	D. Dictionary of Keys

Question:

What is true for Sparsity and Density of a matrix from the given relations?	
A. Sparsity = 1 - Density	B. Sparsity = 1 + Density
C. Sparsity = Density/total number of elements	D. Sparsity = Density*total number of elements

Question:

What is true for Sparsity and Density of a matrix from the given relations?	
A. Sparsity = 1 - Density	B. Sparsity = 1 + Density
C. Sparsity = Density/total number of elements	D. Sparsity = Density*total number of elements

Question:

```
int main ()
{
int c[3][4]={2,3,1,6,4,1,6,2,2,7,1,10};
printf("%u, %u\n", c+1, &c+1);
return 0;
}
```

What will be output of the following program where c=100 and int=1 bytes.	
A. 104, 112	B. 108, 124
C. 112,124	D. No output

Question:

```
int main ()
{
int c[3][4]={2,3,1,6,4,1,6,2,2,7,1,10};
printf("%u, %u\n", c+1, &c+1);
return 0;
}
```

What will be output of the following program where c=100 and int=1 bytes.	
A. 104, 112	B. 108, 124
C. 112,124	D. No output

Question:

```
int main ()
{
int a[5]={1,2,3,4,5};
int b[5];
ba=;
printf("%d\n",b[1]);
}
```

What will be output of the following program	
A. 1	B. Program crashes
C. Compile time error	D. No output

Question:

```
int main ()
{
int a[5]={1,2,3,4,5};
int b[5];
ba=;
printf("%d\n",b[1]);
}
```

What will be output of the following program	
A. 1	B. Program crashes
C. Compile time error	D. No output

Question:

In a compact single dimensional array representation for lower triangular matrices (i.e. all the elements above the diagonal are zero) of size n*n, non-zero elements (i.e. elements of the lower triangle) of each row are stored one after another starting from the first row, the index of the (i,j)th element of the lower triangular matrix in this new representation is	
A. i+j	B. i+j-1
C. (j-1)+ i(i-1)/2	D. i+j(j-1)/2

Question:

In a compact single dimensional array representation for lower triangular matrices (i.e. all the elements above the diagonal are zero) of size n*n, non-zero elements (i.e. elements of the lower triangle) of each row are stored one after another starting from the first row, the index of the (i,j)th element of the lower triangular matrix in this new representation is	
A. i+j	B. i+j-1
C. (j-1)+ i(i-1)/2	D. i+j(j-1)/2

Question:

An n*n array V is defines as: v[I,j]=i-j for all I,j, 1<=i<=n, 1<=j<=n, the sum of the elements of the array v is	
A. 0	B. n-1
C. $N^{2} - 3n+2$	D. $N^{2}(n+1)/2$

Question:

An n*n array V is defines as: v[I,j]=i-j for all I,j, 1<=i<=n, 1<=j<=n, the sum of the elements of the array v is	
A. 0	B. n-1
C. $N^{2} - 3n+2$	D. $N^2(n+1)/2$

Question:

A program P reads in 500 integers in the range [0, 100] representing the cores of 500 students. It then print the frequency of each score above 50. What would be the best way for P to store the frequencies?	
A. An array of 50 numbers	B. An array of 100 numbers
C. An array of 500 numbers	D. A dynamically allocated array of 550 numbers

Question:

A program P reads in 500 integers in the range [0, 100] representing the cores of 500 students. It then print the frequency of each score above 50. What would be the best way for P to store the frequencies?	
A. An array of 50 numbers	B. An array of 100 numbers
C. An array of 500 numbers	D. A dynamically allocated array of 550 numbers

Question:

```
int MyX(int *E, unsigned int size){
 int Y = 0;
 int Z;
 int i,j,k;
 for(i = 0; i < size; i++)
      Y = Y + E[i];
 for(i = 0; i < size; i++)
   for(j = i; j < size; j++){
     Z = 0;
     for(k = i; k <= j; k++)
        Z = Z + E[k];
     if(Z > Y)
        Y = X;
   }
   return Y;
}
```

A Consider the following C functions in which size is the number of elements in the array E: What is the value returned by the function MyX ?	
A. Maximum possible sum of elements in any sub-array of array E.	B. Maximum element in any sub-array of array E.
C. Sum of the maximum elements in all possible sub-arrays of array E.	D. The sum of all elements in the array E.

Question:

```
int MyX(int *E, unsigned int size){
 int Y = 0;
 int Z;
 int i,j,k;
 for(i = 0; i < size; i++)
      Y = Y + E[i];
 for(i = 0; i < size; i++)
   for(j = i; j < size; j++){
    Z = 0;
    for(k = i; k <= j; k++)
       Z = Z + E[k];
    if(Z > Y)
       Y = X;
   }
   return Y;
}
```

A Consider the following C functions in which size is the number of elements in the array E: What is the value returned by the function MyX ?	
A. Maximum possible sum of elements in any sub-array of array E.	B. Maximum element in any sub-array of array E.
C. Sum of the maximum elements in all possible sub-arrays of array E.	D. The sum of all elements in the array E.

Question:

```
Int main()
{
unsigned int x[4][3]= {{1,2,3}, {4,5,6}, {7,8,9}, {10,11,12}};
printf("%u, %u, %u", x+3,*(x+3),*(x+2)+3);
}
```

What is the output of the following code? Assume the address of x is 20000 and an integer requires 4 bytes of memory	
A. 2036, 2036, 2036.	B. 2012, 4, 2204
C. 2036, 10, 10	D. 2012, 4, 6

Question:

```
Int main()
{
unsigned int x[4][3]= {{1,2,3}, {4,5,6}, {7,8,9}, {10,11,12}};
printf("%u, %u, %u", x+3,*(x+3),*(x+2)+3);
}
```

What is the output of the following code? Assume the address of x is 20000 and an integer requires 4 bytes of memory	
A. 2036, 2036, 2036.	B. 2012, 4, 2204
C. 2036, 10, 10	D. 2012, 4, 6

Question:

```
C=100;
For i=1to n do
      For j=1 to n do
      {
        Temp=A[i][j] + c;
       A[i][j]= A[j][i];
       A[j][i] = temp-c;
        }
For i=1to n do
      For j=1 to n do
          Print A[i][j];
```

Let A be a square matrix of size n*n. Consider the following pseudocode and find the expected output:	
A. Matrix A itself	B. Transpose of the matrix a
C. Adding 100 to the upper diagonal and subtracting 100 from lower diagonal elements of A	D. none

Question:

```
C=100;
For i=1to n do
      For j=1 to n do
      {
        Temp=A[i][j] + c;
       A[i][j]= A[j][i];
       A[j][i] = temp-c;
        }
For i=1to n do
      For j=1 to n do
          Print A[i][j];
```

Let A be a square matrix of size n*n. Consider the following pseudocode and find the expected output:	
A. Matrix A itself	B. Transpose of the matrix a
C. Adding 100 to the upper diagonal and subtracting 100 from lower diagonal elements of A	D. none

Question:

```
int main()
{
int a[3] = {20,30,40};
int b[3];
b=a;
printf("%d", b[0]);
}
```

What is the output of the above code?	
A. 20	B. 30
C. 0	D. Compile time error

Question:

```
int main()
{
int a[3] = {20,30,40};
int b[3];
b=a;
printf("%d", b[0]);
}
```

What is the output of the above code?	
A. 20	B. 30
C. 0	D. Compile time error

Question:

```
int main()
 {
 int a[];
a[4]= {1,4,6,8};
 int b[4] = {5,9,7,4};
printf("%d,%d", a[0], b[0]);
}
```

What is the output of the above code?	
A. 1,5	B. 2,6
C. 0.0	D. Compile time error

Question:

```
int main()
 {
 int a[];
a[4]= {1,4,6,8};
 int b[4] = {5,9,7,4};
printf("%d,%d", a[0], b[0]);
}
```

What is the output of the above code?	
A. 1,5	B. 2,6
C. 0.0	D. Compile time error

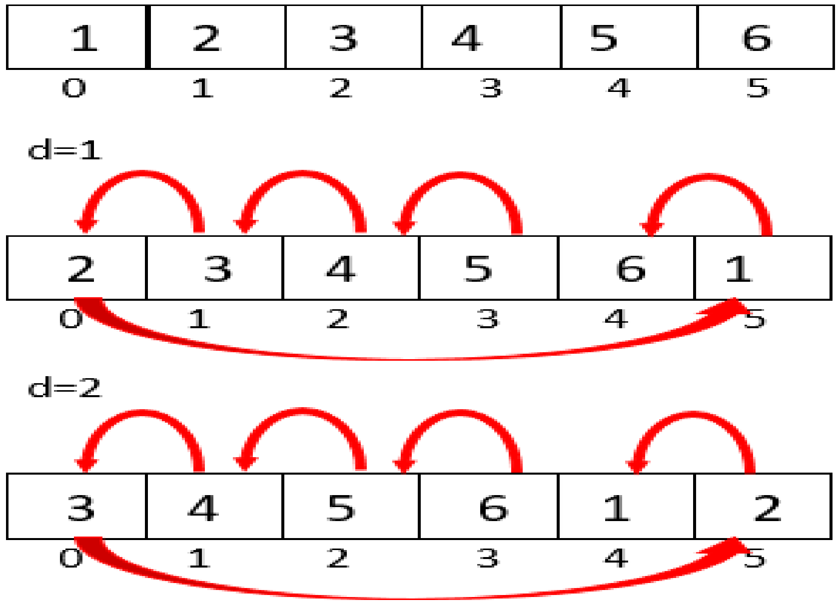
Types of Problem in Array

Problem1– Given an Array and a number d, how will you rotate an Array by d positions.

e.g. Arr[ ]={1,2,3,4,5,6,7,8} , d=2

Output = {3,4,5,6,7,8,1,2}

Method 1:



Type 1: Rotation Type Problem

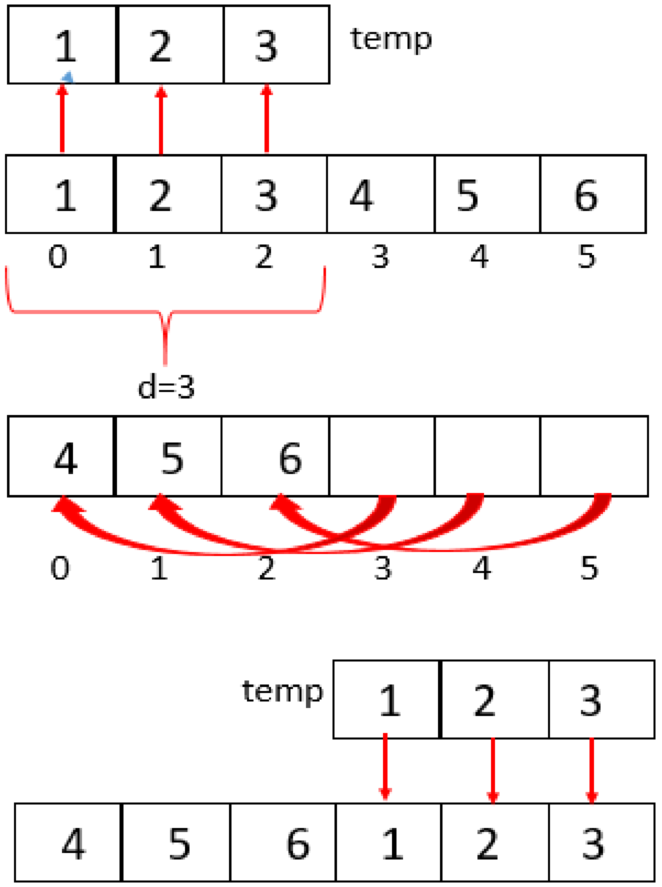
In this method, outer loop executes d+1 times i.e., number of elements rotated and the inner loop executes n times and shifts elements one position left every time. Hence **Time complexity** of this process will be **O(n*d)**.

```
ALGORITHM Rotate(Arr[ ], d, n)
BEGIN:
        FOR i=1 TO d DO
        Temp=Arr[0]
        FOR j=1 TO n–1 DO
                Arr[j–1]=Arr[j]
        Arr[n–1] = Temp
END;
```

Type 1: Rotation Type Problem

Method2:

- In this method, first take a Temporary Array of size d and copy first d elements from the original Array to temporary Array (loop execution d+1 times).
- In the second loop, shift elements by d position into the left (loop execution n–d+1 times).
- Finally, in the last loop, copy the elements from temporary Array to original Array in last d positions (loop execution time d+1 times).



Total time =d+1+n–d+1+d+1 =n+d+2= O(n+d).

Time complexity O(n+d).

Type 1: Rotation Type Problem

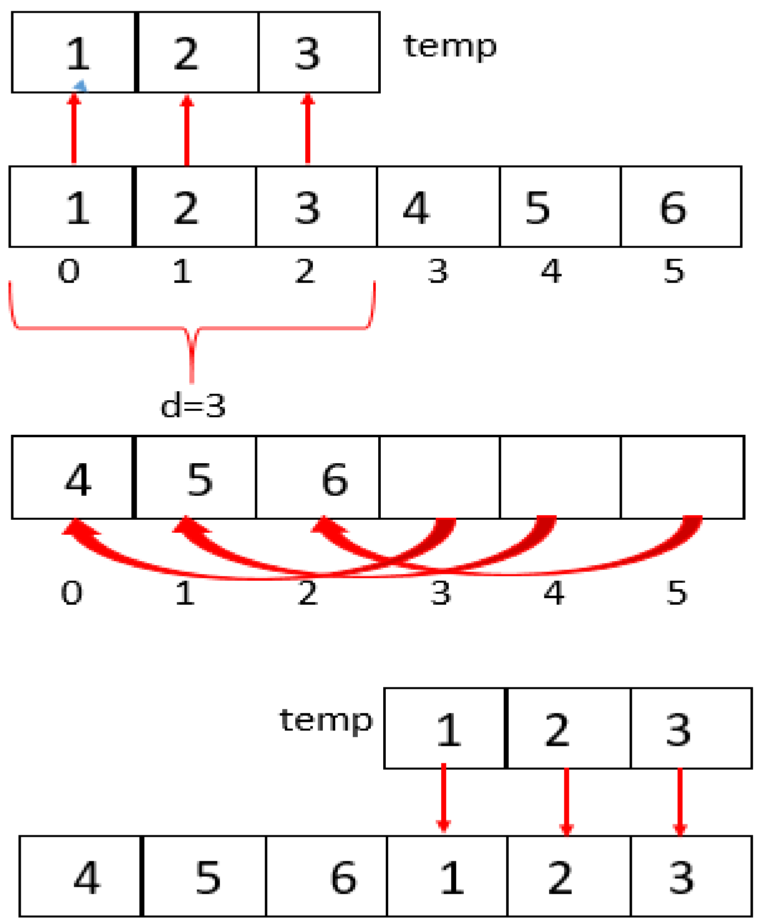
Method2:

ALGORITHM Rotate(Arr[ ], d, n)

BEGIN:

```
FOR i=0 TO d–1 DO
        Temp[i]=Arr[i]
FOR i=0 TO n–d–1 DO
        Arr[i]=Arr[i+d]
FOR i=n–d TO n–1 DO
        Arr[i]=temp[i–n+d]
```

END;



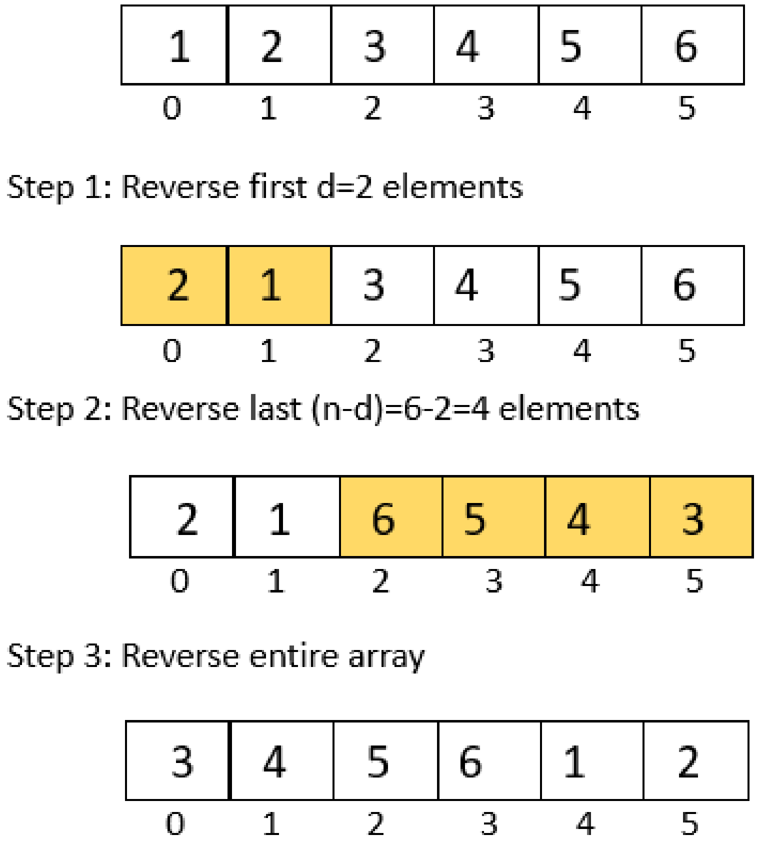
Type 1: Rotation Type Problem

Method 3 – Reversal Algorithm

To perform the rotation, the process can be broken in three parts.

- Reverse the first d elements.
- Reverse the last n–d elements.
- Reverse the entire Array.

The resulting Array would be rotated by d positions.



Can you answer these questions?

Problem 2:

Write an algorithm to rotate an array of n elements by d positions that rotate Arr[ ] of size n by d elements using block swap algorithm.

Hint: Block swap means swapping the Array elements by making a group of elements (Block).

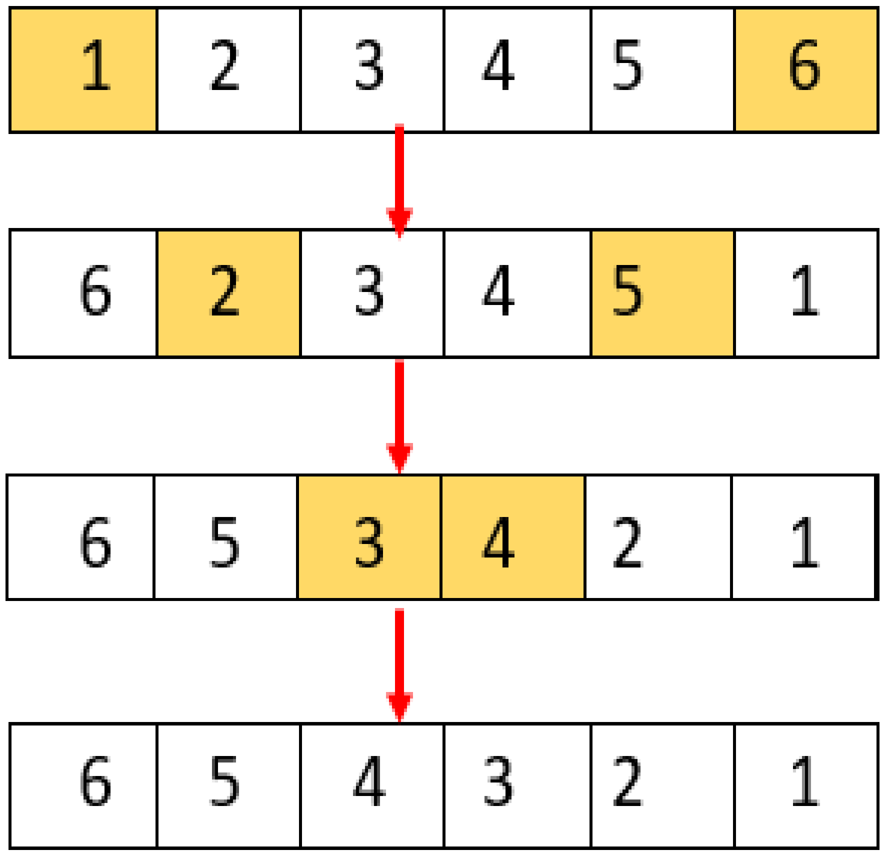
Problem 3:

Write an algorithm to search an element into sorted and rotated array.

Type 2: Arrangement and de–arrangement problem (Reversal of an Array)

Method1:

ALGORITHM Reverse(Arr[ ], n)
BEGIN:
  Low=0
  High=n–1
  WHILE Low < High DO
    Swap(Arr[Low], Arr[High])
    Low++
    High—
END;

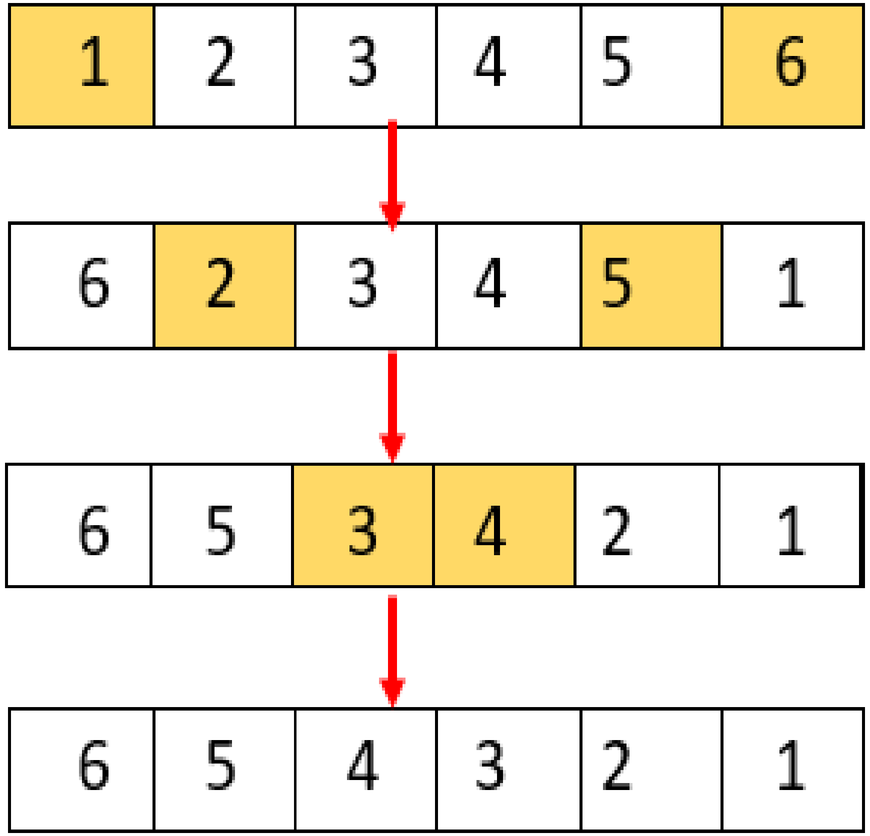


Time complexity:

The Algorithm performs swaps n/2 times.

The loop executes for n/2 times and a total of 5 statements run in each loop execution.

Total statement execution required for the operation is 5*n/2 + 2 i.e. **O(n).**



Type 2: Arrangement and de–arrangement problem (Reversal of an Array)

Method 2:

The same operation of pair wise swap can be performed by making use of 1 variable instead of 2.

The Algorithm performs swaps n/2 times.

The loop executes for n/2 times and a total of 5 statements run in each loop execution.

Total statement execution required for the operation is 5*n/2 + 2 i.e. **O(n).**

```
ALGORITHM reverse(Arr[ ],  n)
BEGIN:
        FOR i=0 TO n/2 – 1 DO
            swap(Arr[i], Arr[n–i–1])
END;
```

Type 2: Arrangement and de–arrangement problem (Reversal of an Array)

Method 3:

ALGORITHM Reverse(Arr[ ], Low, High)
BEGIN:
    IF Low>=High THEN
        RETURN
    Swap(Arr[Low], Arr[High]);
    Reverse(Arr, Low+1, High–1);
END;

Time Complexity

The Algorithm performs swaps n/2 times conditionally. Hence the Time complexity of the operations would be O(n).

Can you answer these questions?

Problem 1:

Write an algorithm to ReArrange positive and negative numbers in O(n) time and O(1) extra space.

Problem 2:

Write an algorithm to Shuffle a given Array using Fisher-Yates shuffle Algorithm.

Type 3: Order Statistics Problem

Given an array and a number k where k is smaller than the size of the Array, we need to find the k^{th} smallest element in the given Array. Therefore, it is given that all Array elements are distinct.

Examples

Input: Arr[ ] = {7, 10, 4, 3, 20, 15}

k = 3

Output: 7

Input: Arr[ ] = {7, 10, 4, 3, 20, 15}

k = 4

Output: 10

Type 3: Order Statistics Problem

Method1: BRUTE FORCE

- **Step 1:** Sort the given Array by using any sorting algorithm. It is suggested to use the sorting algorithms that take minimum time. We can either pick the Quick sort, Heap sort or Merge sort that takes O(nlogn) time.
- **Step 2:** Return the element at (k–1)st index.

```
ALGORITHM kthSmallest(Arr[ ], n, k)
BEGIN:
        Sort(Arr, n)
        RETURN Arr[k–1]
END;
```

Type 3: Order Statistics Problem

Time Complexity:

Since sorting takes O(nlogn) time and 1 return statement is used in the given algorithm, total time can be represented as **O(nlogn)**

Can you answer these questions?

Write an algorithm for finding **Mean** and **Median** of an unsorted Array.

Type 4: Range query problem

To find the GCDs of Array elements in the given index range. Given an Array A[ ] of size n. We should be able to efficiently find the GCD from index qstart (query start) to qend (query end) where 0 <= qstart <= qend <= n–1.

Example :

Input : a[ ] = {2, 3, 60, 90, 50}

Index Ranges : {1, 3}

Output: GCD of given range is 3

Input : a[ ] = {2, 3, 60, 90, 50}

Index Ranges : {2, 4}

Output: GCD of given range is 10

Input : a[ ] = {2, 3, 60, 90, 50}

Index Ranges : {0, 2}

Output: GCD of given range is 1

Type 4: Range query problem

```
ALGORITHM RangeGcd(qstart, qend)
BEGIN:
    FOR i = qstart TO qend DO
        x=GCD(A[qstart], A[qend])
    RETURN x
END;
```

```
ALGORITHM GCD(a, b)
BEGIN:
    IF a<b THEN
        Swap(a,b)
    IF b==0 THEN
        RETURN a
    RETURN GCD(b, a%b)
END;
```

Type 5: Optimization problem

Subset sum Problem

Given a set of non–negative integers and a value *sum*, determine if a subset of the given set with Total equals the given *sum*.

Example:

Input: set[ ] = {3, 34, 4, 12, 5, 2}, sum = 9

Output: True

There is a subset (4, 5) with sum 9.

Input: set[ ] = {3, 34, 4, 12, 5, 2}, sum = 30

Output: False

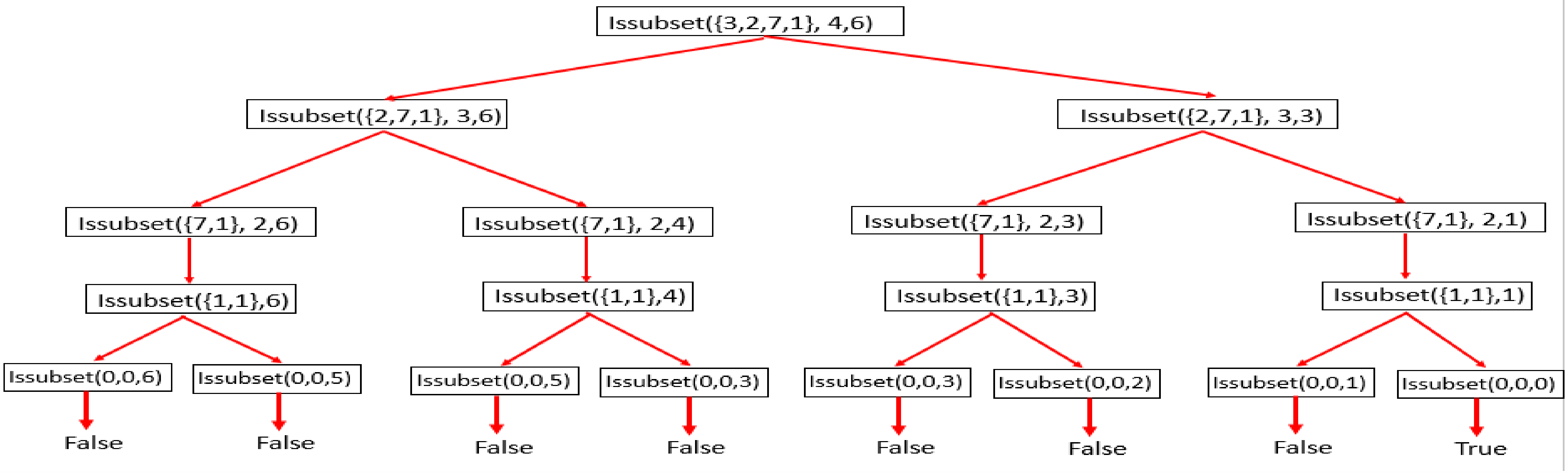
There is no subset that add up to 30.

Type 5: Optimization problem: Recursive Solution

```
ALGORITHM IsSubset(Arr[ ], n, x)
BEGIN:
        IF x==0 THEN
                RETURN TRUE
        IF n==0THEN
                RETURN FALSE
        IF Arr[0]>x THEN
                RETURN IsSubset(Arr+1, n–1, x)
ELSE
        RETURN IsSubset(Arr+1, n–1, x) || IsSubset(Arr+1, n–1, x–Arr[0])
END;
```

Type 5: Optimization problem: Recursive Solution

let us suppose that Arr[4]={3,2,7,1},x=6,n=4 x is sum and n is number of elements. In recursive calls, if the current item is greater than the sum, then simply ignore that item. If the current item is not greater than the sum, either include the item or exclude that item. If the item is included, then sum = sum – item; otherwise, there will be no change in the sum.



Can you answer these questions?

Write an algorithm to find the permutation of given Array elements.

Type 6: Sorting problem

Alternative Sorting

Given an Array of integers, print the Array in such a way that the first element is first maximum and second element is first minimum, third element is second maximum and fourth element is second minimum and so on so forth.

Examples:

Input : Arr[ ] = {7, 1, 2, 3, 4, 5, 6}

Output: 7 1 6 2 5 3 4

Input : Arr[ ] = {1, 6, 9, 4, 3, 7, 8, 2}

Output: 9 1 8 2 7 3 6 4

Type 6: Sorting problem

Step 1:

First, sort the Array by using any sorting algorithms which take a minimum of O(nlogn) time.

Step 2:

Set Beg=0 and End=n–1. Print elements alternatively A[Beg] followed by A[End]. Increase Beg by 1 and decrease End by 1. Again, print A[Beg] and A[End]. The process repeats until Beg and End meet each other or cross.

Type 6: Sorting problem

```
ALGORITHM AlternateSort(Arr[ ], n)
BEGIN:
        WHILE Beg < End DO
            WRITE(Arr[Beg])
            WRITE(Arr[End])
            Beg=Beg+1
            End=End–1
        IF n%2 !=0 THEN
            WRITE(Arr[Beg])
END;
```

Can you answer these questions?

Problem 1: Sort an Array in wave form.

Given an unsorted Array of integers, sort the Array into a wave like Array. An Array 'Arr[n]' is sorted in wave form if Arr[0] >= Arr[1] <= Arr[2] >= Arr[3] <= Arr[4] >= …

Problem 2: Merge two sorted Arrays with O(1) extra space

We are given two sorted Arrays. We need to merge these two Arrays such that the initial numbers (after complete sorting) are in the first Array and the remaining numbers are in the second Array. Extra space allowed in O(1).

Type 7: Searching Problem

Leaders in an Array

Write an Algorithm to print all the LEADERS in the Array. An element is a leader if it is greater than all the elements to its right side and the rightmost element is always a leader. For example, int the Array {16, 17, 4, 3, 5, 2}, leaders are 17, 5 and 2.

```
ALGORITHM Leader(Arr[ ], n)
BEGIN:
FOR i=0 TO n–1 DO
        FOR j=i+1 TO n–1 DO
                IF Arr[i]<=Arr[j] THEN
                        BREAK
        IF j==n THEN
                WRITE(Arr[i])
END;
```

Type 7: Searching Problem

Method

Step 1: Outer loop runs from 0 to n – 1 and one by one select all elements from left to right.

Step 2: The inner loop compares the selected element to all the elements to its right side.

Step 3: If the selected element is greater than all the elements to its right side, then the selected element is the leader.

Can you answer these questions?

Problem 1: Majority Element

Write a function that takes an Array and prints the majority element (if it exists); otherwise, print "No Majority Element." A *majority element* in an Array A[ ] of size n is an element that appears more than n/2 times (and hence there is at most one such element).

Problem 2: Peak Element

Given an Array of integers. Find a peak element in it. An Array element is a peak if it is not smaller than its neighbors. For corner elements, we need to consider only one neighbor.

4.8 Generic Array

- The method of Generic Programming is implemented to increase the efficiency of the code.
- Generic Programming enables the programmer to write a general algorithm which will work with all data types.
- It eliminates the need to create different algorithms if the data type is an integer, string or a character.
- In C programming ,Generic Array Is not Allowed.

Continued...

- In C language

Array can be declared as– int a[10];

From the above declaration, we can calculate how much memory is allocated to an Array. However, if we declare an Array of void type, it is impossible to calculate how much memory will be allocated to an Array.

void a[10]; This is an invalid declaration

Continued...

- Array of void pointers used as generic Array

void*a[4] //valid declaration

Now let us convert the Array elements in such a way that each element of an Array points to different types of memory

a[0] = malloc(sizeof(bool));

a[1] = malloc(sizeof(int));

a[2] = malloc(sizeof(float));

a[3] = malloc(sizeof(char));

Continued...

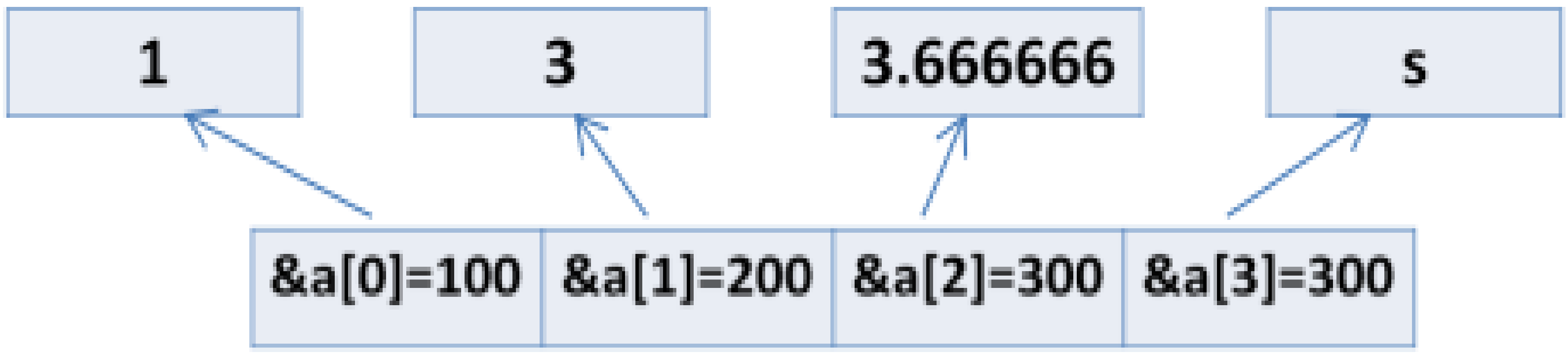
- Now let us store different data type values into this Array

```
 *((bool*)&(a[0]))=1
 *((int*)&(a[1]))=3
*((float*)&(a[2]))=3.666666
 *((char*)&(a[0]))='s'
```

In the generic Array, it is very important to know which type of data element Array element points to.

Continued...

- In Java Array of object data type is generic because the object is parent class of all.



Summary

- Types of Array
- Index Formulas of Array
- Primitive operations of Array.
- Application of 1 D Array
- Application of 2 D array
- Generic Array

Thank You