SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-(IOT Including Cyber security &BCT)

COURSE NAME : 19ITT201 - DATA STRUCTURES

II YEAR / III SEMESTER

Unit 1- LINEAR STRUCTURES AND TREES

Topic 2 : Linked List Based Implementation

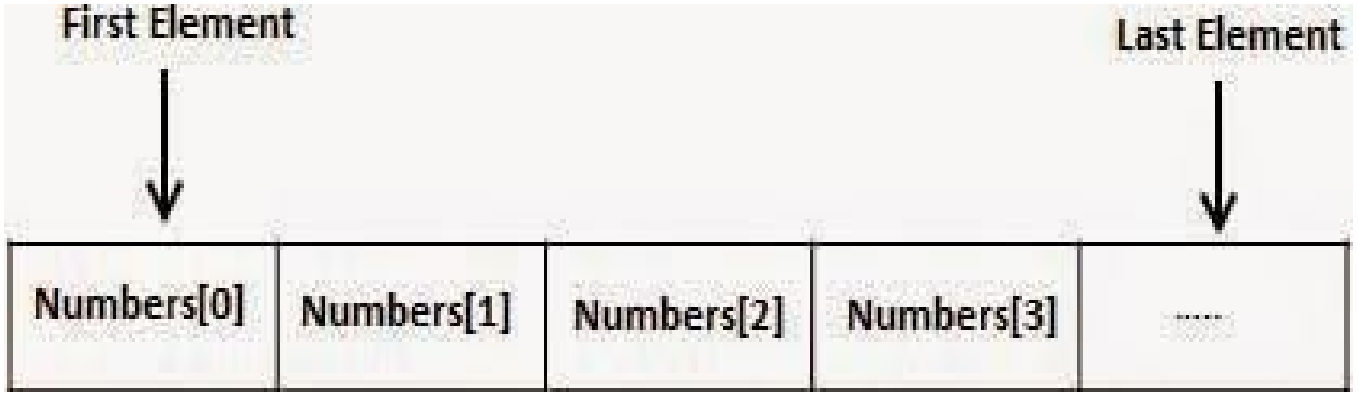
Problem

- Insertion and deletion are expensive
- Even if the array is dynamically allocated, an estimate of the maximum size of the list is required. Usually this requires a high over-estimate, which wastes considerable space. This could be a serious limitation, if there are many lists of unknown size.
- Simple arrays are generally not used to implement lists. Because the running time for insertion and deletion is so slow and the list size must be known in advance

Array Based Implementation

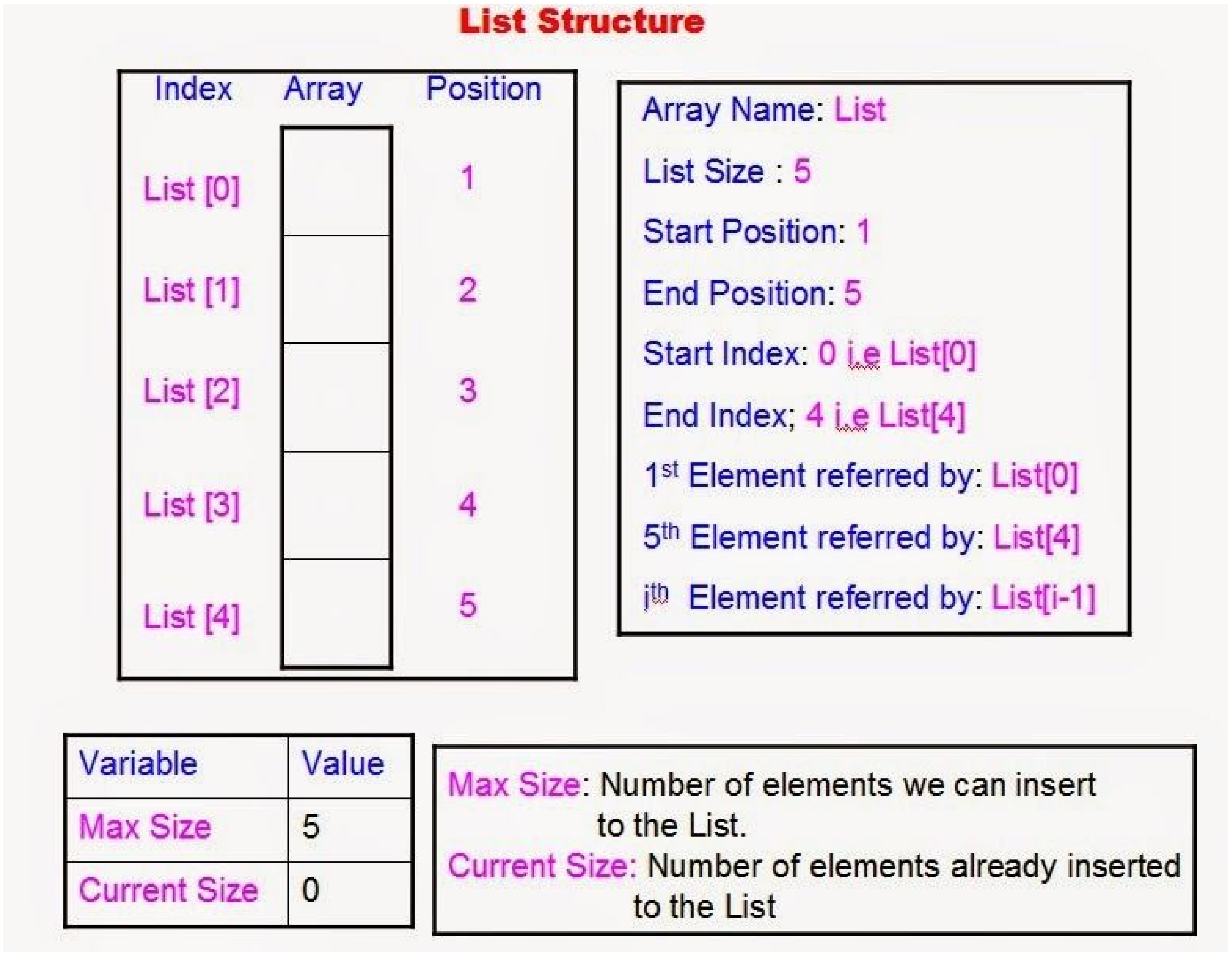
- **What is Array?**
- An Array is a data structure which can store a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.
- A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element

Array Based Implementation –Cont..



Array Based Implementation of LIST

Array Based Implementation –Cont..



List Structure

Index	Array	Position
List [0]		1
List [1]		2
List [2]		3
List [3]		4
List [4]		5

Array Name: List

List Size : 5

Start Position: 1

End Position: 5

Start Index: 0 i.e List[0]

End Index; 4 i.e List[4]

1st Element referred by: List[0]

5th Element referred by: List[4]

ith Element referred by: List[i-1]

Variable	Value
Max Size	5
Current Size	0

Max Size: Number of elements we can insert to the List.

Current Size: Number of elements already inserted to the List

Array Based Implementation –Cont..

Operations

1. Is Empty(LIST)

2.Is Full(LIST)

3.Insert Element to End of the LIST.

4.Delete Element from End of the LIST.

5.Insert Element to front of the LIST.

6.Delete Element from front of the LIST.

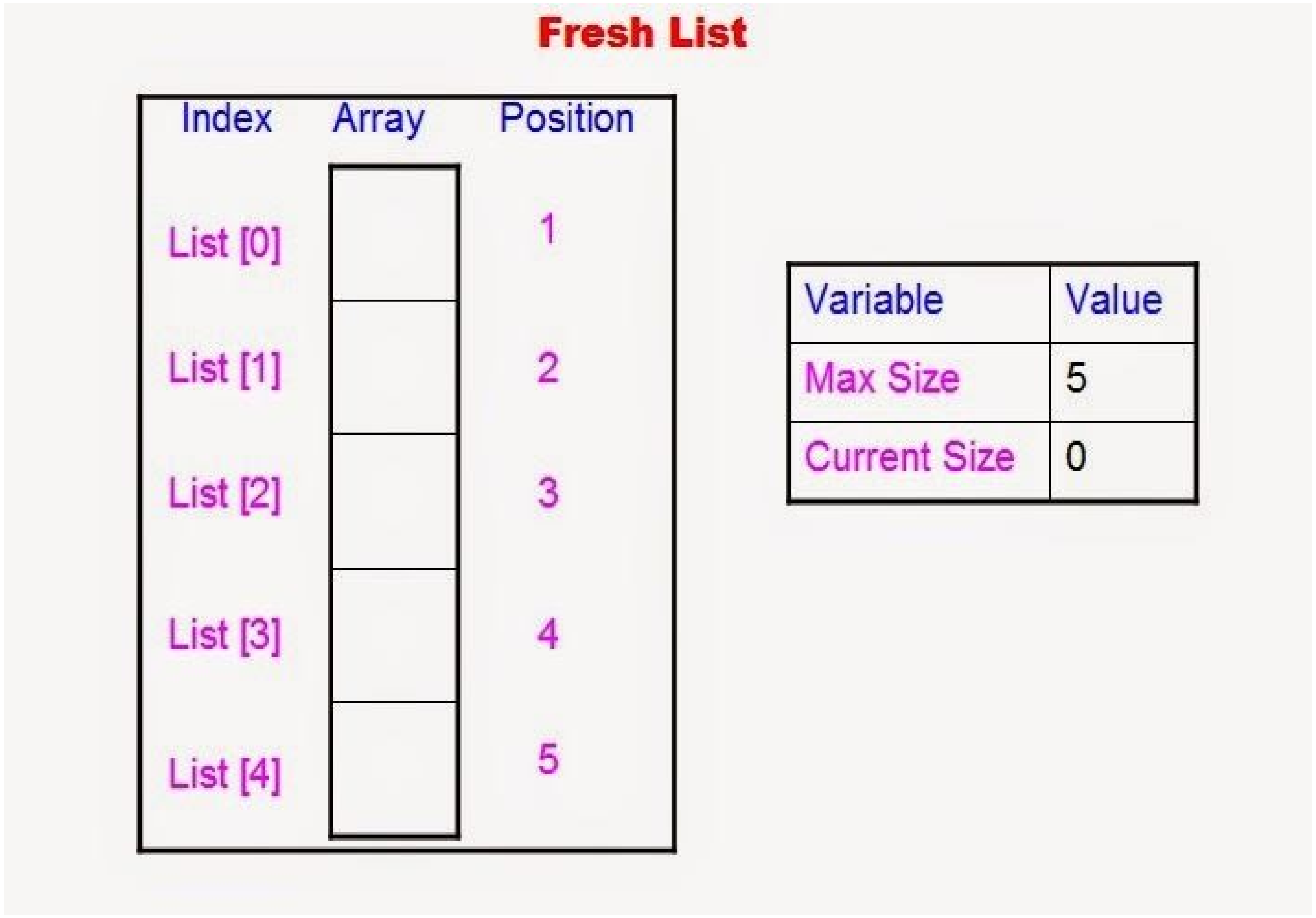
7.Insert Element to nth Position of the LIST.

8.Delete Element from nth Position of the LIST.

9.Search Element in the LIST.

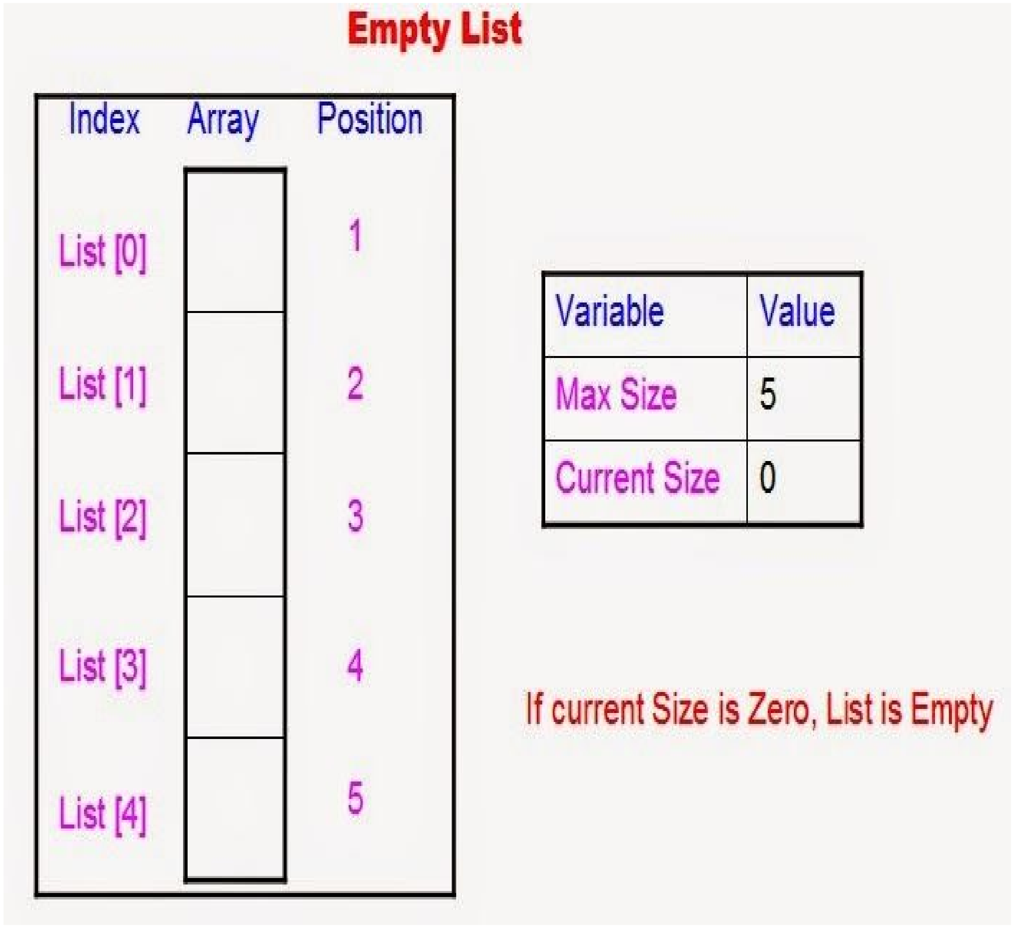
10.Print the Elements in the LIST.

Array Based Implementation –Cont..



Variable	Value
Max Size	5
Current Size	0

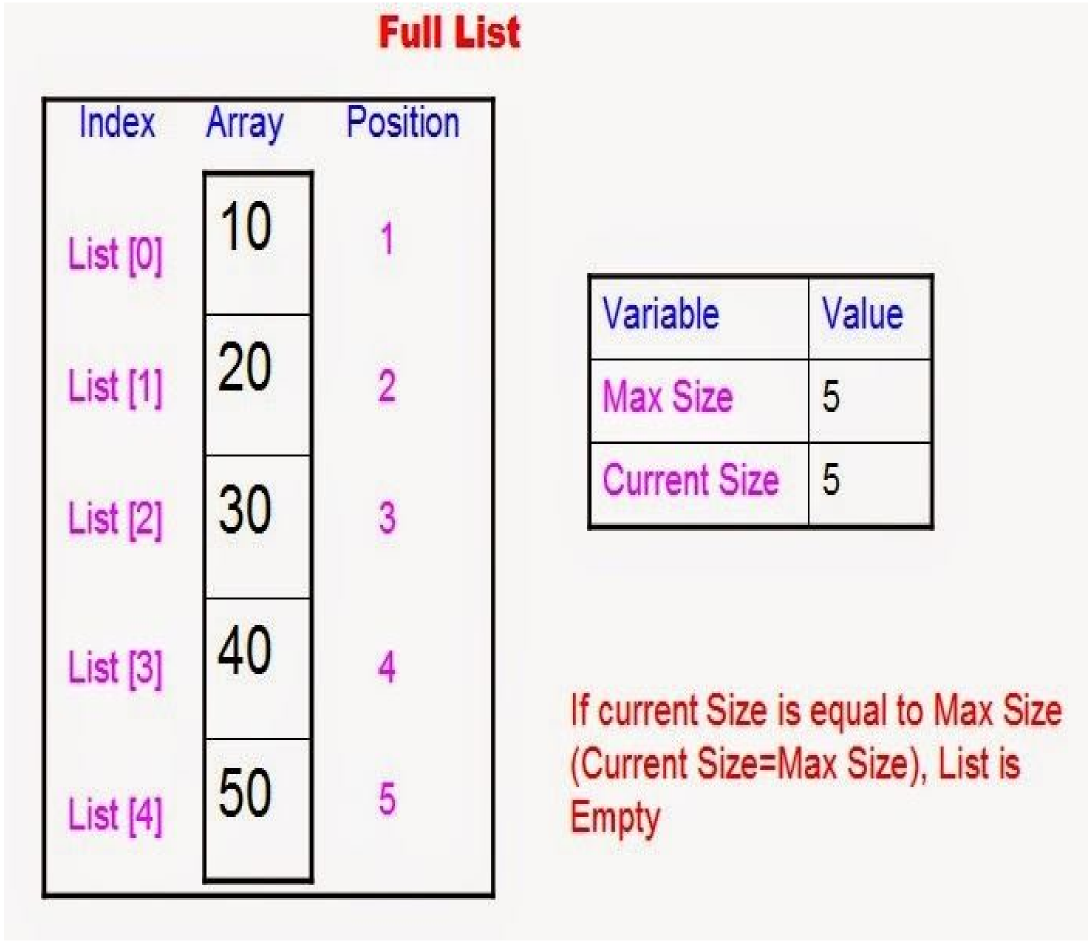
Array Based Implementation –Cont..



Variable	Value
Max Size	5
Current Size	0

- **Is Empty(LIST)**
- If (**Current Size==0**) **"LIST is Empty"**
- else "LIST is not Empty"

Array Based Implementation –Cont..


Full List

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]	50	5

Variable	Value
Max Size	5
Current Size	5

If current Size is equal to Max Size (Current Size=Max Size), List is Empty

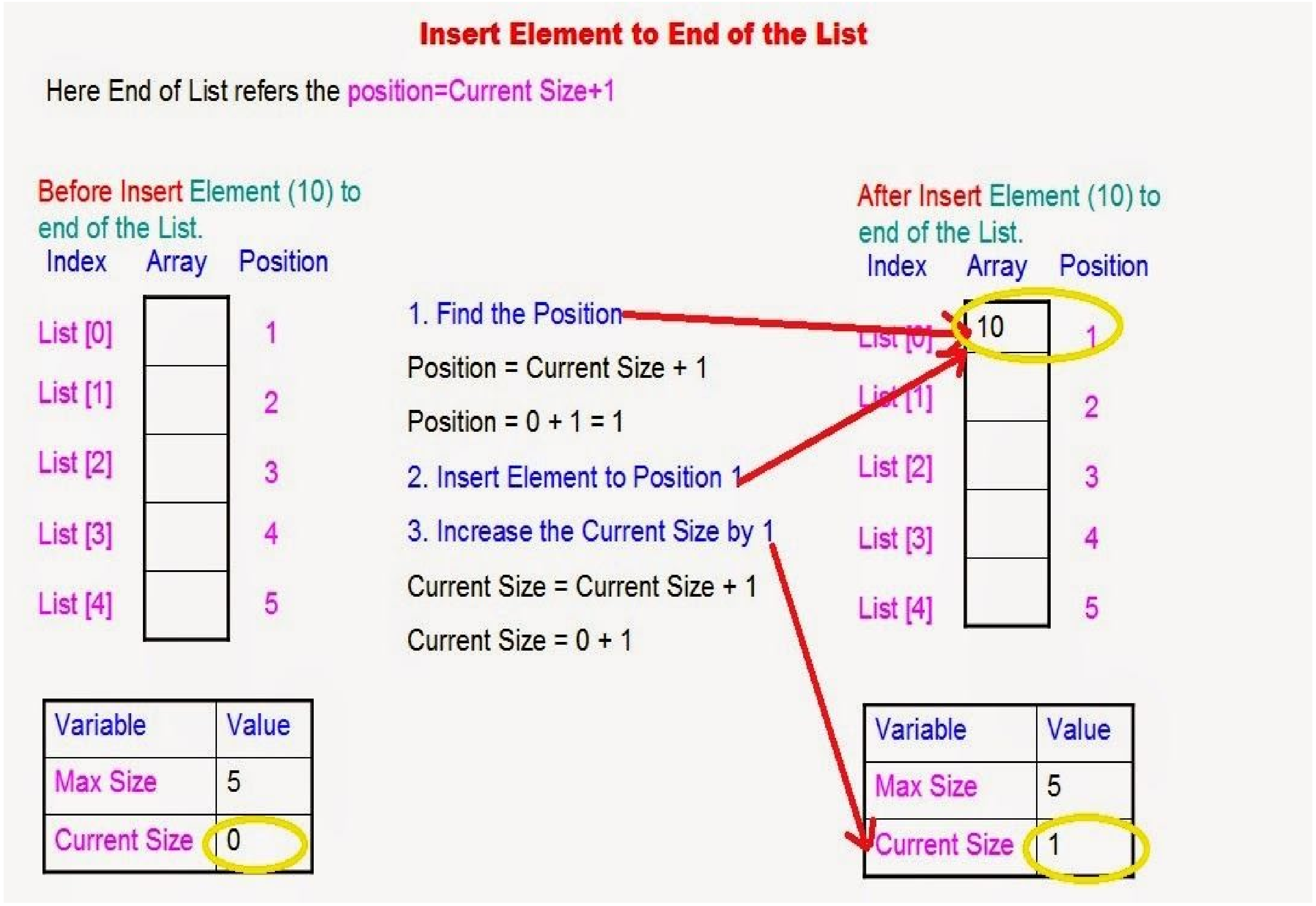
- **Is Full(LIST)**
- If (**Current Size=Max Size**) "**LIST is FULL**"

else "LIST is not FULL"

Array Based Implementation –Cont..

- **Insert Element to End of the LIST.**
- Check that weather the List is full or not
  - –If List is full return error message **”List is full. Can’t Insert”.**
  - –If List is not full.
    - Get the position to insert the new element by **Position=Current Size+1**
    - Insert the element to the Position
    - Increase the Current Size by 1 i.e. **Current Size=Current Size+1**

Array Based Implementation –Cont..



Insert Element to End of the List

Here End of List refers the position=Current Size+1

Before Insert Element (10) to end of the List.
Index
Array
Position
List [0]
List [1]
List [2]
List [3]
List [4]
1
2
3
4
5
After Insert Element (10) to end of the List.
10

1. Find the Position

Position = Current Size + 1

Position = 0 + 1 = 1

2. Insert Element to Position 1

3. Increase the Current Size by 1

Current Size = Current Size + 1

Current Size = 0 + 1

Before:

Variable	Value
Max Size	5
Current Size	0

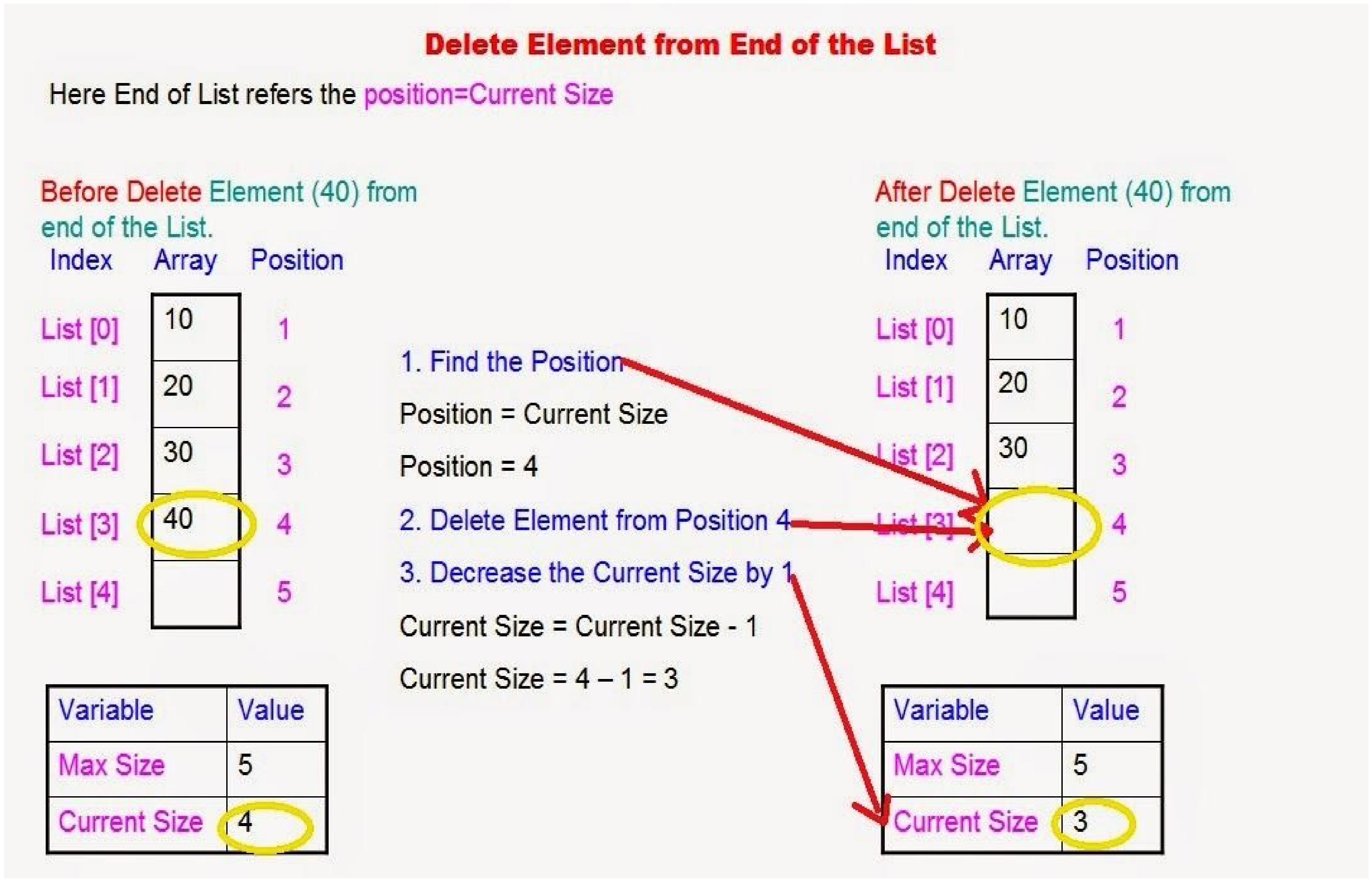
After:

Variable	Value
Max Size	5
Current Size	1

Array Based Implementation –Cont..

- **Delete Element from End of the LIST.**
- Check that weather the List is empty or not
  - –If List is empty return error message ”**List is Empty. Can't Delete”**.
  - –If List is not Empty.
    - Get the position of the element to delete by **Position=Current Size**
    - Delete the element from the **Position**
    - Decrease the Current Size by 1 i.e. **Current Size=Current Size-1**

Array Based Implementation –Cont..



Delete Element from End of the List

Here End of List refers the position=Current Size

1. Find the Position

Position = Current Size

Position = 4

2. Delete Element from Position 4

3. Decrease the Current Size by 1

Current Size = Current Size - 1

Current Size = 4 – 1 = 3

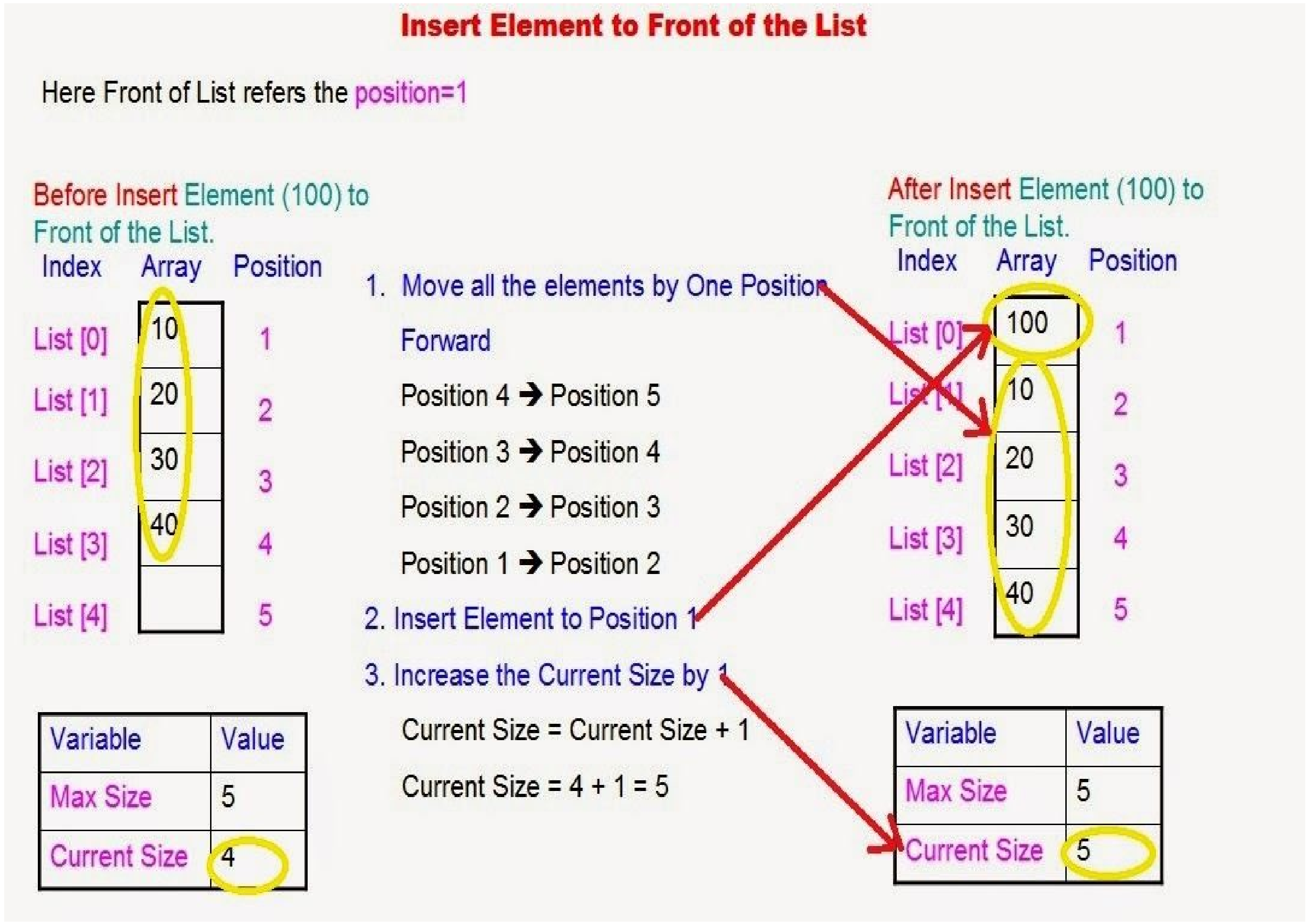
Variable	Value
Max Size	5
Current Size	4

Variable	Value
Max Size	5
Current Size	3

Array Based Implementation –Cont..

- **Insert Element to front of the LIST.**
- Check that weather the List is full or not
  - –If List is full return error message ”**List is full. Can't Insert**”.
  - –If List is not full.
    - Free the 1st Position of the list by moving all the Element to one position forward i.e**New Position=Current Position + 1**.
    - Insert the element to the **1st Position**
    - Increase the Current Size by 1 i.e. **Current Size=Current Size+1**

Array Based Implementation –Cont..



Insert Element to Front of the List

Here Front of List refers the position=1

Before Insert Element (100) to Front of the List.

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]		5

Variable	Value
Max Size	5
Current Size	4

1. Move all the elements by One Position Forward

   Position 4 → Position 5

   Position 3 → Position 4

   Position 2 → Position 3

   Position 1 → Position 2

2. Insert Element to Position 1
3. Increase the Current Size by 1

   Current Size = Current Size + 1

   Current Size = 4 + 1 = 5

After Insert Element (100) to Front of the List.

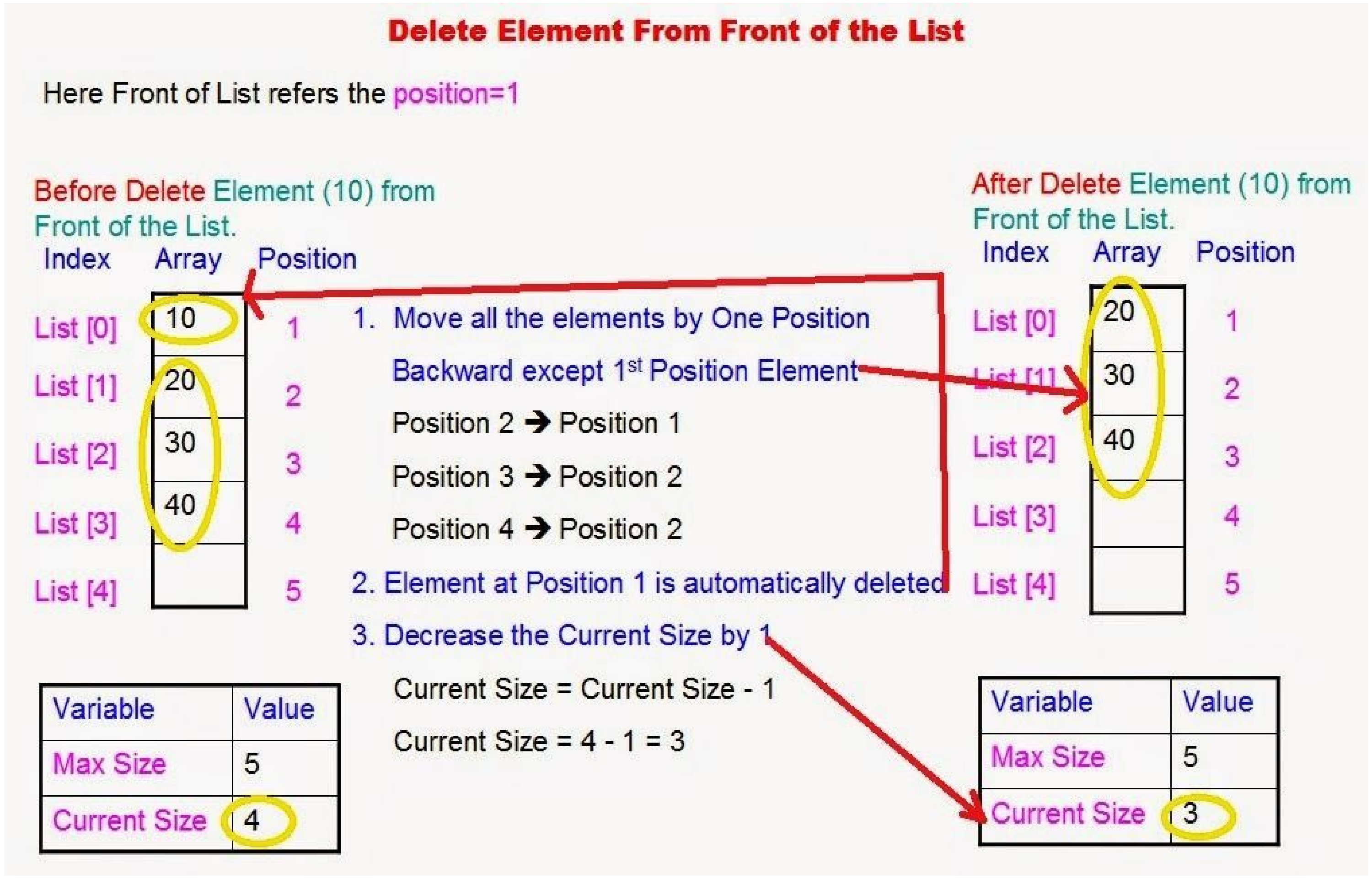
Index	Array	Position
List [0]	100	1
List [1]	10	2
List [2]	20	3
List [3]	30	4
List [4]	40	5

Variable	Value
Max Size	5
Current Size	5

Array Based Implementation –Cont..

- **Delete Element from front of the LIST.**
- Check that weather the List is empty or not
  - –If List is empty return error message "**List is Empty. Can't Delete**".
  - –If List is not Empty.
    - Move all the elements except one in the 1st position to one position backward i.e**New Position= Current Position -1**
    - After the 1st step, element in the **1st position will be automatically deleted.**
    - Decrease the Current Size by 1 i.e. **Current Size=Current Size-1**

Array Based Implementation –Cont..



Delete Element From Front of the List

Here Front of List refers the position=1

Before Delete Element (10) from Front of the List.

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]		5

Variable	Value
Max Size	5
Current Size	4

1. Move all the elements by One Position Backward except 1st Position Element

   Position 2 ➔ Position 1

   Position 3 ➔ Position 2

   Position 4 ➔ Position 2

2. Element at Position 1 is automatically deleted

3. Decrease the Current Size by 1

   Current Size = Current Size - 1

   Current Size = 4 - 1 = 3

After Delete Element (10) from Front of the List.

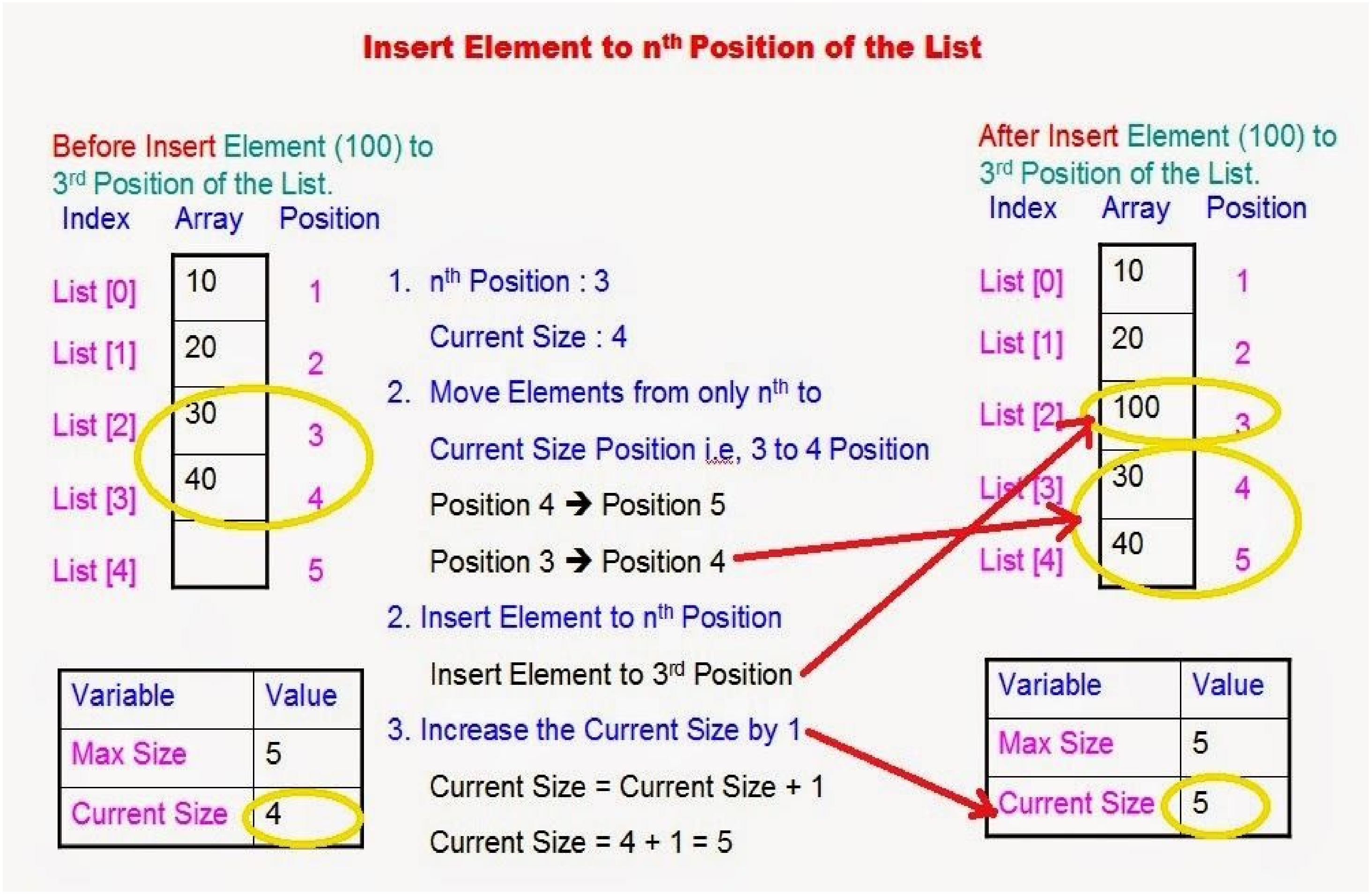
Index	Array	Position
List [0]	20	1
List [1]	30	2
List [2]	40	3
List [3]		4
List [4]		5

Variable	Value
Max Size	5
Current Size	3

Array Based Implementation –Cont..

- **Insert Element to nth Position of the LIST.**
- Check that weather the List is full or not
  - If List is full return error message "**List is full. Can't Insert**".
  - If List is not full.
    - If List is Empty, Insert element at Position 1.
    - If (nth Position > Current Size)
      - Return message "**nth Position Not available in List**"
    - else
      - Free the nth Position of the list by moving all Elements to one position forward**except n-1,n-2,... 1** Position i.e **move only from n to current size position Elements**. i.e **New Position=Current Position + 1.**
      - Insert the element to the **nth Position**
      - Increase the Current Size by 1 i.e. **Current Size=Current Size+1**

Array Based Implementation –Cont..



Insert Element to nth Position of the List

Before Insert Element (100) to 3rd Position of the List.

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]		5

Variable	Value
Max Size	5
Current Size	4

1. nth Position : 3

   Current Size : 4

2. Move Elements from only nth to Current Size Position i.e, 3 to 4 Position

   Position 4 ➔ Position 5

   Position 3 ➔ Position 4

2. Insert Element to nth Position

   Insert Element to 3rd Position

3. Increase the Current Size by 1

   Current Size = Current Size + 1

   Current Size = 4 + 1 = 5

After Insert Element (100) to 3rd Position of the List.

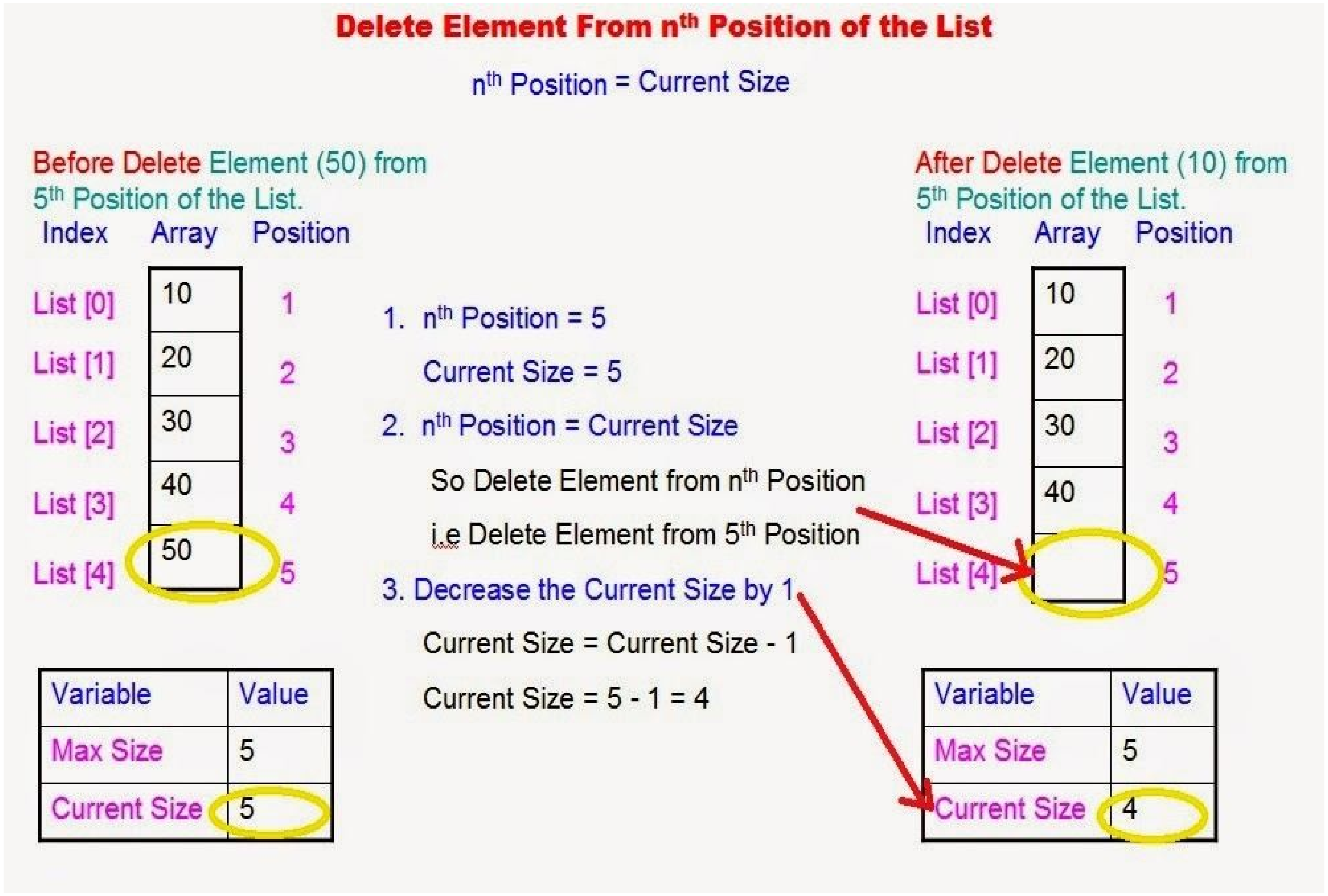
Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	100	3
List [3]	30	4
List [4]	40	5

Variable	Value
Max Size	5
Current Size	5

Array Based Implementation –Cont..

- **Delete Element from nth Position of the LIST.**
- Check that weather the List is Empty or not
  - If List is Empty return error message **”List is Empty**.”
  - If List is not Empty.
    - If (nth Position > Current Size)
      - Return message “**nth Position Not available in List**”
    - If (nth Position == Current Size)
      - Delete the element from **nth Position**
      - Decrease the Current Size by 1 i.e. **Current Size=Current Size-1**
    - If (nth Position < Current Size)
      - Move all the Elements to one position **backward** except n,n-1,n-2,... 1 Position i.e **move only from n+1 to current size position Elements**. i.e New Position=Current Position - 1.
      - After the previous step, **nth element will be deleted automatically.**
      - Decrease the Current Size by 1 i.e. **Current Size=Current Size-1**

Array Based Implementation –Cont..



Delete Element From nth Position of the List

nth Position = Current Size

Before Delete Element (50) from 5th Position of the List.

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]	50	5

Variable	Value
Max Size	5
Current Size	5

1. nth Position = 5

   Current Size = 5
2. nth Position = Current Size

   So Delete Element from nth Position

   i.e Delete Element from 5th Position
3. Decrease the Current Size by 1

   Current Size = Current Size - 1

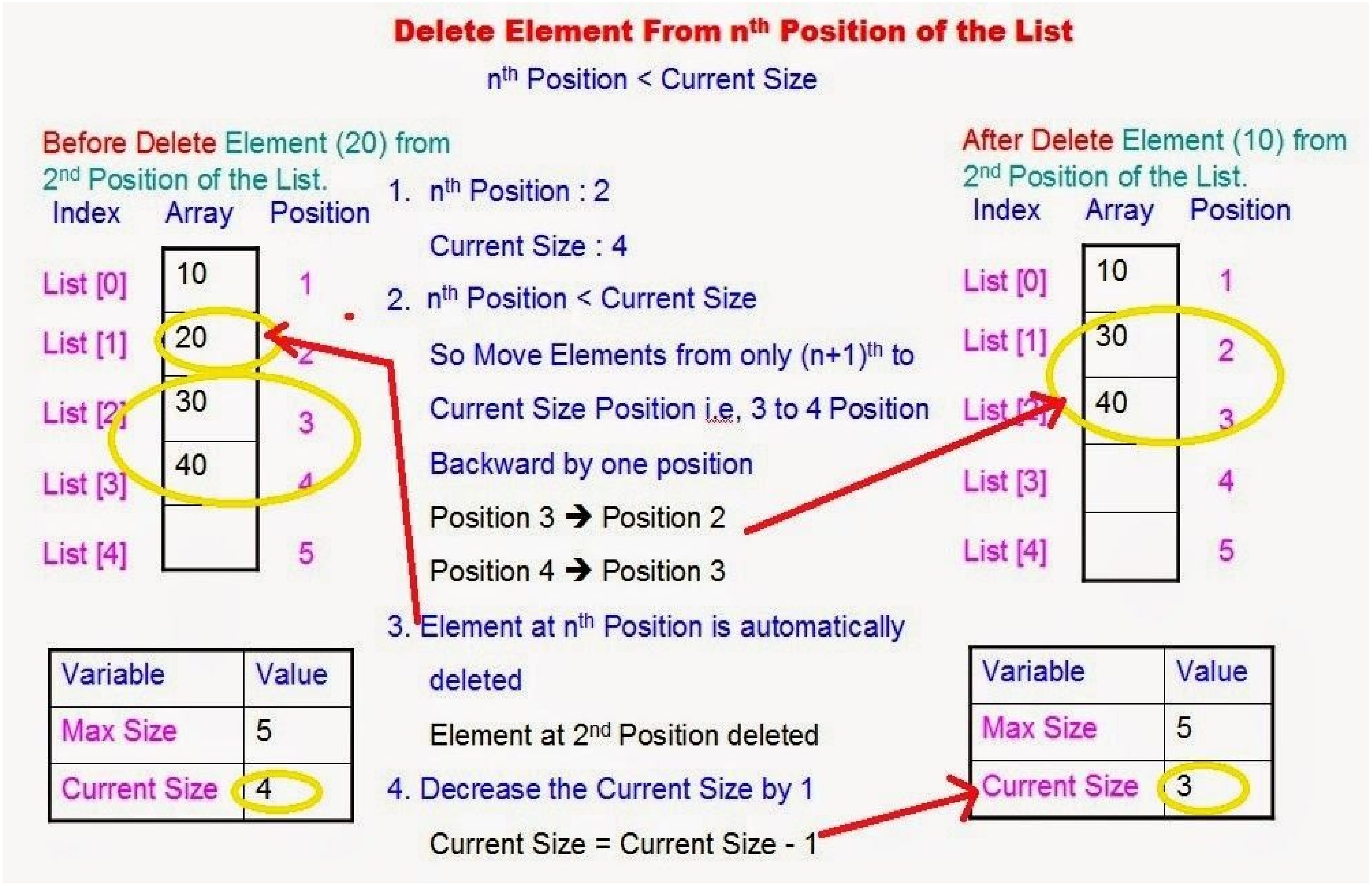
   Current Size = 5 - 1 = 4

After Delete Element (10) from 5th Position of the List.

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]		5

Variable	Value
Max Size	5
Current Size	4

Array Based Implementation –Cont..



Delete Element From nth Position of the List

nth Position < Current Size

Before Delete Element (20) from 2nd Position of the List.

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]		5

Variable	Value
Max Size	5
Current Size	4

1. nth Position : 2

   Current Size : 4

2. nth Position < Current Size

   So Move Elements from only (n+1)th to Current Size Position i.e, 3 to 4 Position Backward by one position

   Position 3 ➔ Position 2

   Position 4 ➔ Position 3

3. Element at nth Position is automatically deleted

   Element at 2nd Position deleted

4. Decrease the Current Size by 1

   Current Size = Current Size - 1

After Delete Element (10) from 2nd Position of the List.

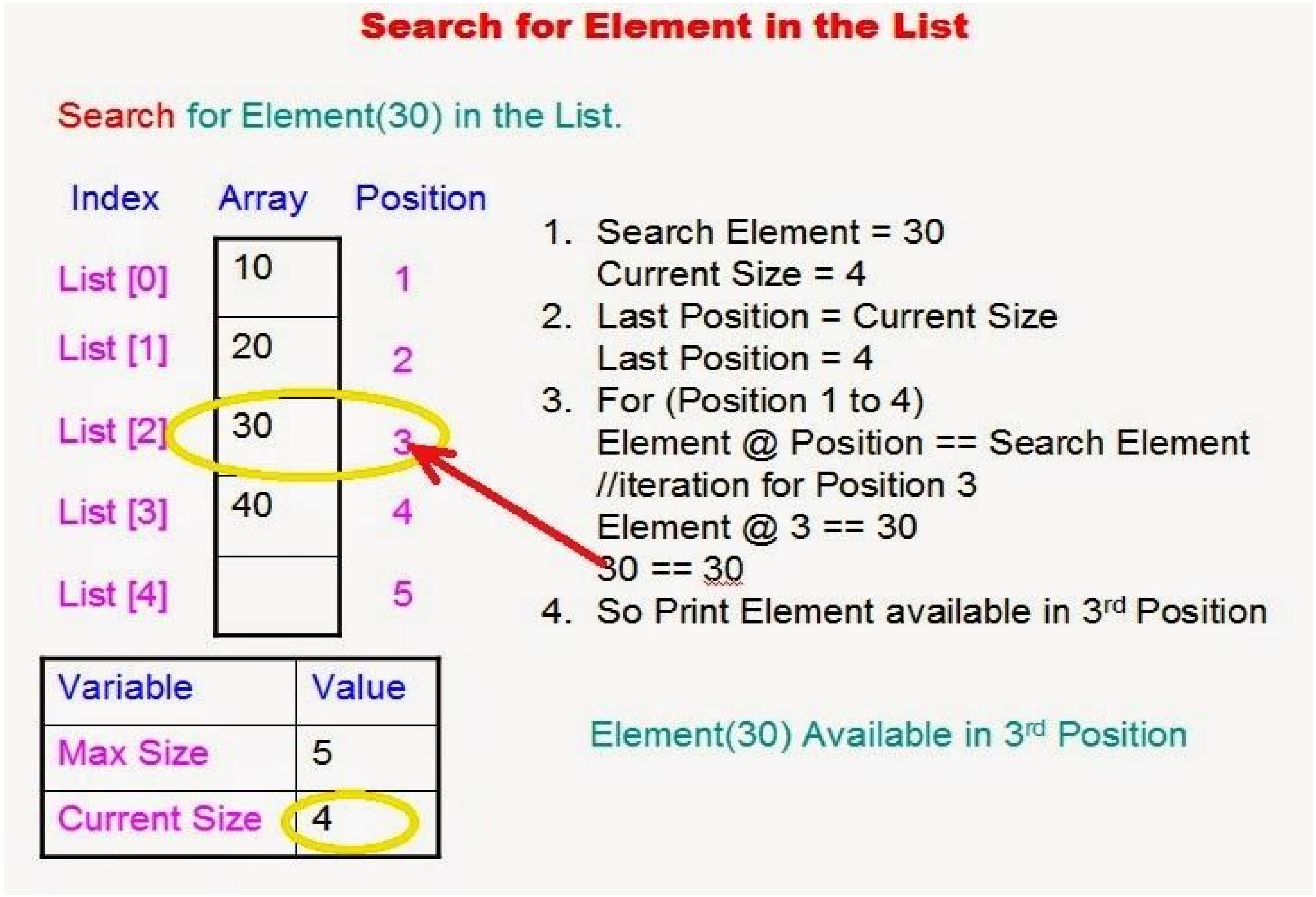
Index	Array	Position
List [0]	10	1
List [1]	30	2
List [2]	40	3
List [3]		4
List [4]		5

Variable	Value
Max Size	5
Current Size	3

Array Based Implementation –Cont..

- **Search Element in the LIST.**
- Check that weather the list is empty or not.
  - –If List is empty, return error message "**List is Empty**".
  - –If List is not Empty
    - Find the Position where the last element available in the List by
      Last Position = Current Size
    - For(**Position 1 to Last Position**)
      - –If(**Element @ Position== Search Element**)//If Element matches the search element
      - –return the Position by message "**Search Element available in Position**"
    - Else return message "**Search Element not available in the List**"

Array Based Implementation –Cont..



Search for Element in the List

Search for Element(30) in the List.

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]		5

Variable	Value
Max Size	5
Current Size	4

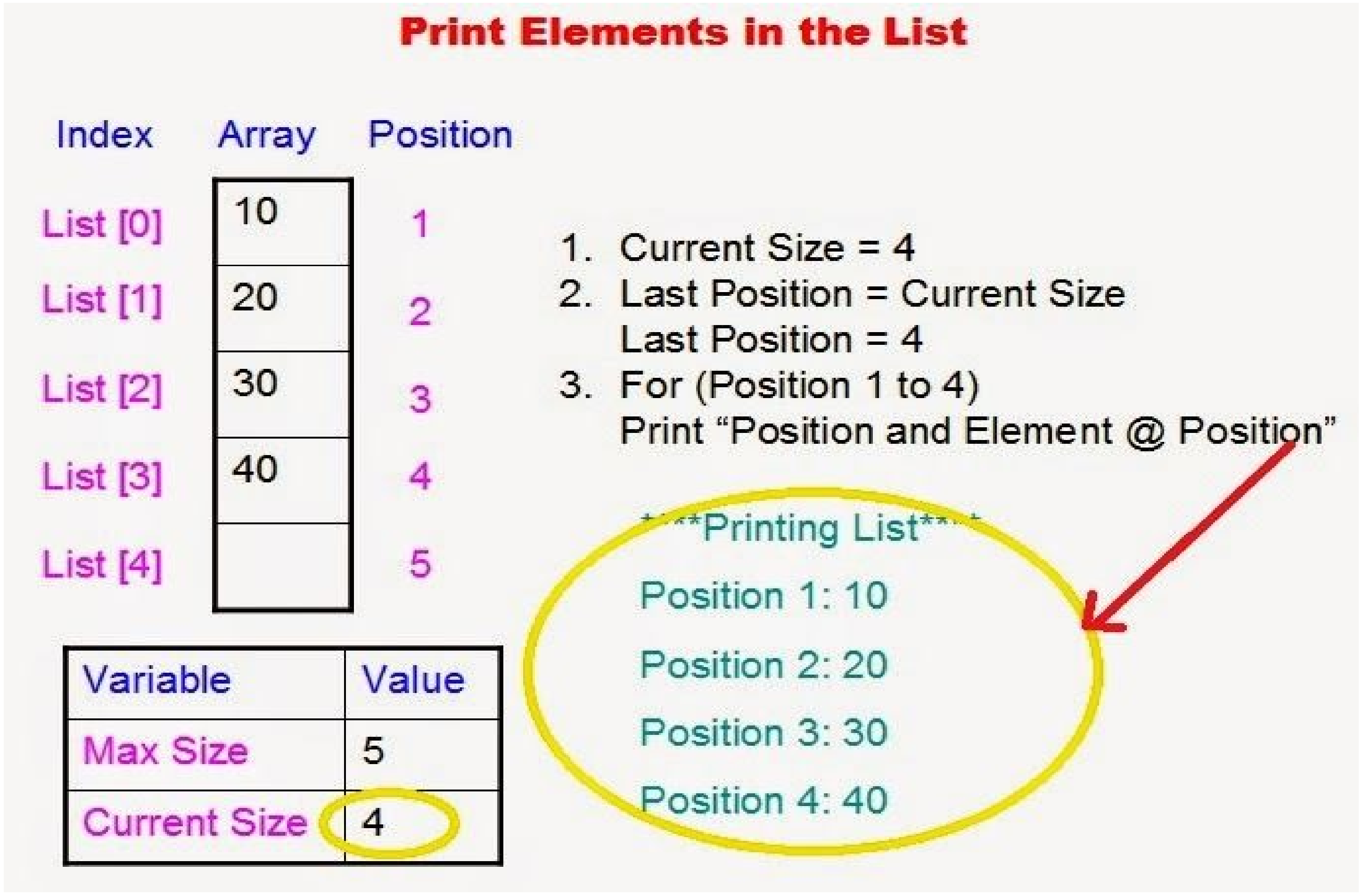
1. Search Element = 30
   Current Size = 4
2. Last Position = Current Size
   Last Position = 4
3. For (Position 1 to 4)
   Element @ Position == Search Element
   //iteration for Position 3
   Element @ 3 == 30
   30 == 30
4. So Print Element available in 3rd Position

Element(30) Available in 3rd Position

Array Based Implementation –Cont..

- **Print the Elements in the LIST.**
- Check that weather the list is empty or not.
  - –If List is empty, return error message "**List is Empty**".
  - –If List is not Empty
    - Find the Position where the last element available in the List by **Last Position = Current Size**
    - For(**Position 1 to Last Position**)
    - Print the Position and Element available at the position of List.

Array Based Implementation –Cont..



Print Elements in the List

Index	Array	Position
List [0]	10	1
List [1]	20	2
List [2]	30	3
List [3]	40	4
List [4]		5

Variable	Value
Max Size	5
Current Size	4

1. Current Size = 4
2. Last Position = Current Size
   Last Position = 4
3. For (Position 1 to 4)
   Print "Position and Element @ Position"

```
****Printing List****
Position 1: 10
Position 2: 20
Position 3: 30
Position 4: 40
```

Activity

Advantages

✓No need to declare Large number of variables Individually

✓Variables are not scattered in Memory, they are stored in contiguous memory.

✓Ease the handling of large number of variables of same data type.

Disadvantages

✓Rigid Structure

✓Can be hard to add/remove elements.

✓Cannot be dynamically re-sized in most Languages.

✓Memory Loss

Assessment 1

1. List out the advantages of array based linked list

   a)______________________________________

   b)______________________________________

   c)______________________________________

   d)______________________________________

2. Identify the disadvantages of array based linked list

   a)______________________________________

   b)______________________________________

   c)______________________________________

   d)______________________________________

REFERENCES

1. M. A. Weiss, “Data Structures and Algorithm Analysis in C”, Pearson Education, 8th Edition, 2007. [Unit I, II, III, IV,V]

2. A. V. Aho, J. E. Hopcroft and J. D. Ullman, “Data Structures and Algorithms”, Pearson Education, 2nd Edition, 2007 [Unit IV].

3. A.M.Tenenbaum, Y. Langsam and M. J. Augenstein, “Data Structures using C”,PearsonEducation, 1st Edition, 2003.(UNIT I,II,V)

4.https://www.youtube.com/watch?v=ubyK2PUfDXI&t=129s

THANK YOU