Spring 2019 (Course Number 009952)

Data Structures and Lab (Lecture 02: Array and Its Operations)

Prof. A. P. Shrestha, Ph.D.

Dept. of Computer Science and Engineering, Sejong University

Today

- Array
- Insertion and Deletion
- Sorting in array

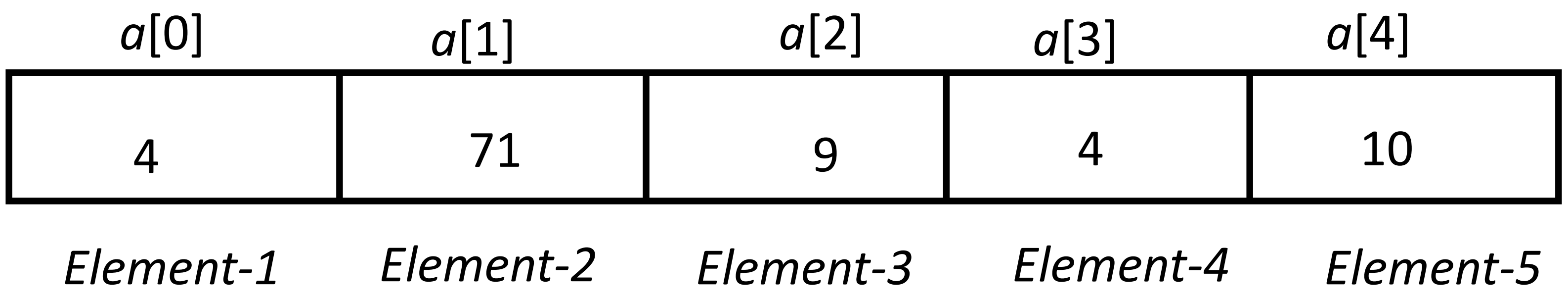
Next class

- Linked lists

2.1.1 Array-Introduction

- An array is collection of items stored at *continuous memory locations*.
- The idea is to store multiple items of *same type* together.
- Continuous memory locations make it easier to calculate the position of each element by simply adding *an offset* to *a base value*

```
int a[5] = {4,71,9,4,10};
```



2. 1.2 Arrays-Practice Question

```
#include<iostream>

using namespace std;

void main()
{
        int a[5] = {4,71,9,4,10};
        cout << sizeof(int)<<endl;
        cout << a << endl;
        cout << a + 1 << endl;
        cout << &a + 1 << endl;
        cout << *a + 3 << endl;
        cout << *(a + 3) << endl;
        cout << *(a + 3)+1 << endl;
        system("Pause");
}
```

Output:

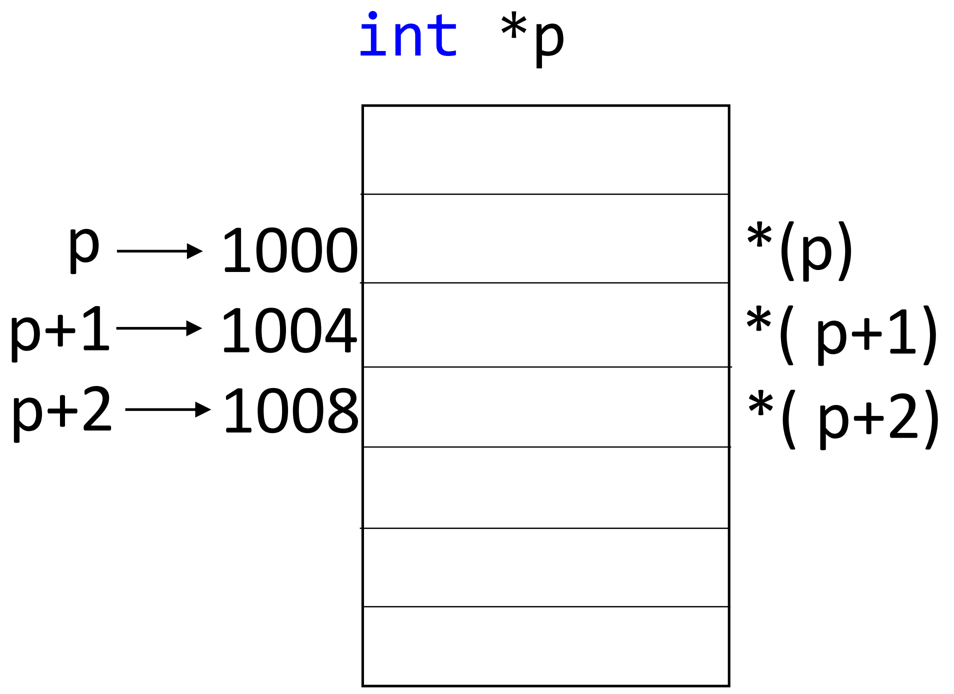
```
4
008FF890
??
??
??
??
??
```

Note:

C	C++
printf()	cout <<
scanf()	cin >>
\n	\n or endl

2.2.1 Dynamic Arrays

- What if the size of the array is *unknown before the compile time*?
- We can use
- C : `calloc, malloc` / `free` and C++ : `new` and `delete`



The array elements can now be access as
*(p+0) first array element
*(p+1) second array element
*(p+2) third array element
:
:
*(p+i) ith array element

2.2.2 Dynamic Array-Practice Program

```
#include<iostream>

using namespace std;

void main()
{
  int *arr;
  int i, n;
  cout << "Enter total number of elements:";
  cin >> n;
  arr = new int(n);//dynamic memory allocation

  for (i = 0; i<n; i++)
  {
      cout<<"Input element "<<i+1;
      cin >> arr[i];
  }

  cout << "Entered elements are: ";
  for (i = 0; i<n; i++)
       cout << arr[i] << " ";
  cout << endl;
  delete (arr);
  system("Pause");
}
```

Output:

```
Enter total number of elements: 2¶
Input element 1 11 ¶
Input element 2  22 ¶
Entered elements are: 11 22
```

2.3.1 Arrays- Advantages

- Allow easy random access of elements

  -This makes accessing elements by position faster.

- Good for locality of reference

  - Make a quite big difference in terms of performance.

Locality of reference: same values, or related storage locations, are frequently accessed, depending on the memory access pattern

2.3.2 Disadvantages of Arrays

- The size of the arrays is fixed

    - Upper limit on the number of elements should be known in advance

    - Allocated memory is constrained by upper limit irrespective of the usage.

- Allocating "large enough” arrays has two disadvantages

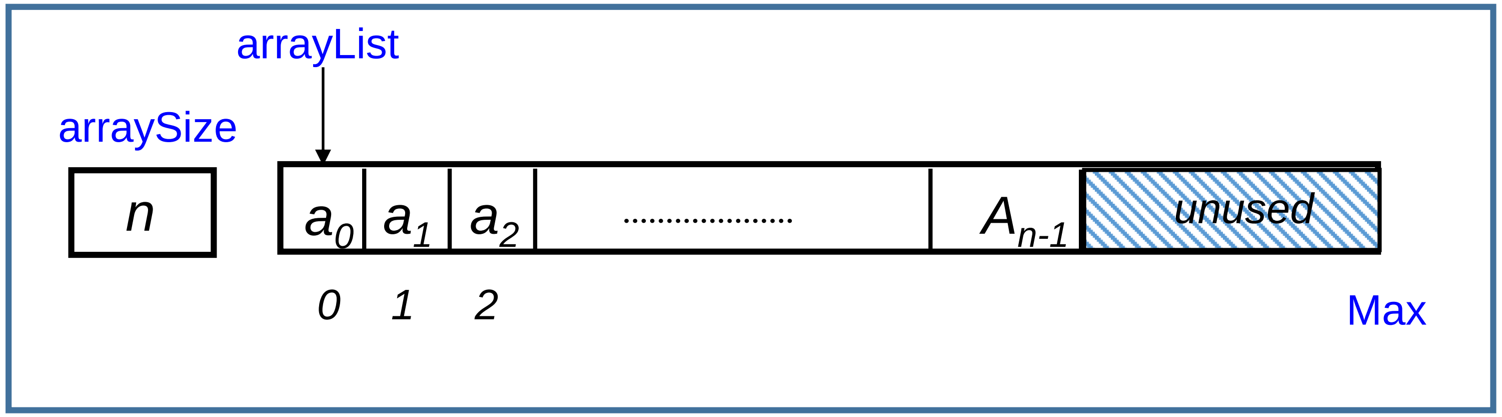
    - Large chunk of the space in the array really is wasted

    - If we need to process more than the declared size, the code breaks.

- Inserting (and Deleting) a new element in an array is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.

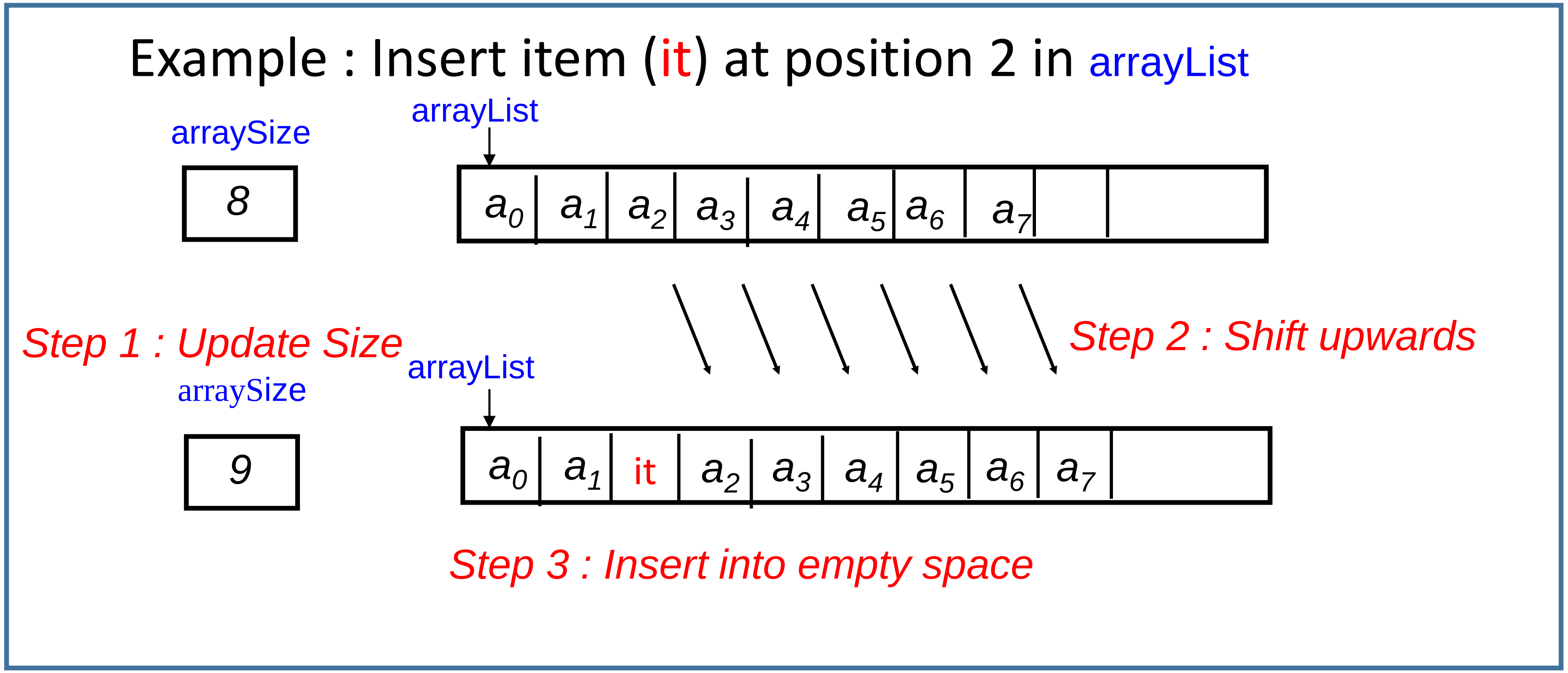
2.4.1 Array based List Implementation

- Maximum size is anticipated a priori.
- Variables:
  - Maximum size (`MAX`)
  - Current size (`arraySize`)
  - Array of elements (`arrayList`)



2.4.2 Array based List Implementation-Insertion

- Insertion has to shift upwards to create space



Example : Insert item (it) at position 2 in arrayList
arraySize
arrayList
8
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
Step 1 : Update Size
Step 2 : Shift upwards
arraySize
arrayList
9
a_0 a_1 it a_2 a_3 a_4 a_5 a_6 a_7
Step 3 : Insert into empty space

2.4.3 Array based List Implementation-Code

```
int arrayList[MAX], arraySize;
```

```
void initializeList()
{
        cout << "Enter size of array :";
        cin >> arraySize;
        cout << "Enter elements of array :";
        for (int i = 0; i<arraySize; i++)
                cin >> arrayList[i];
}
```

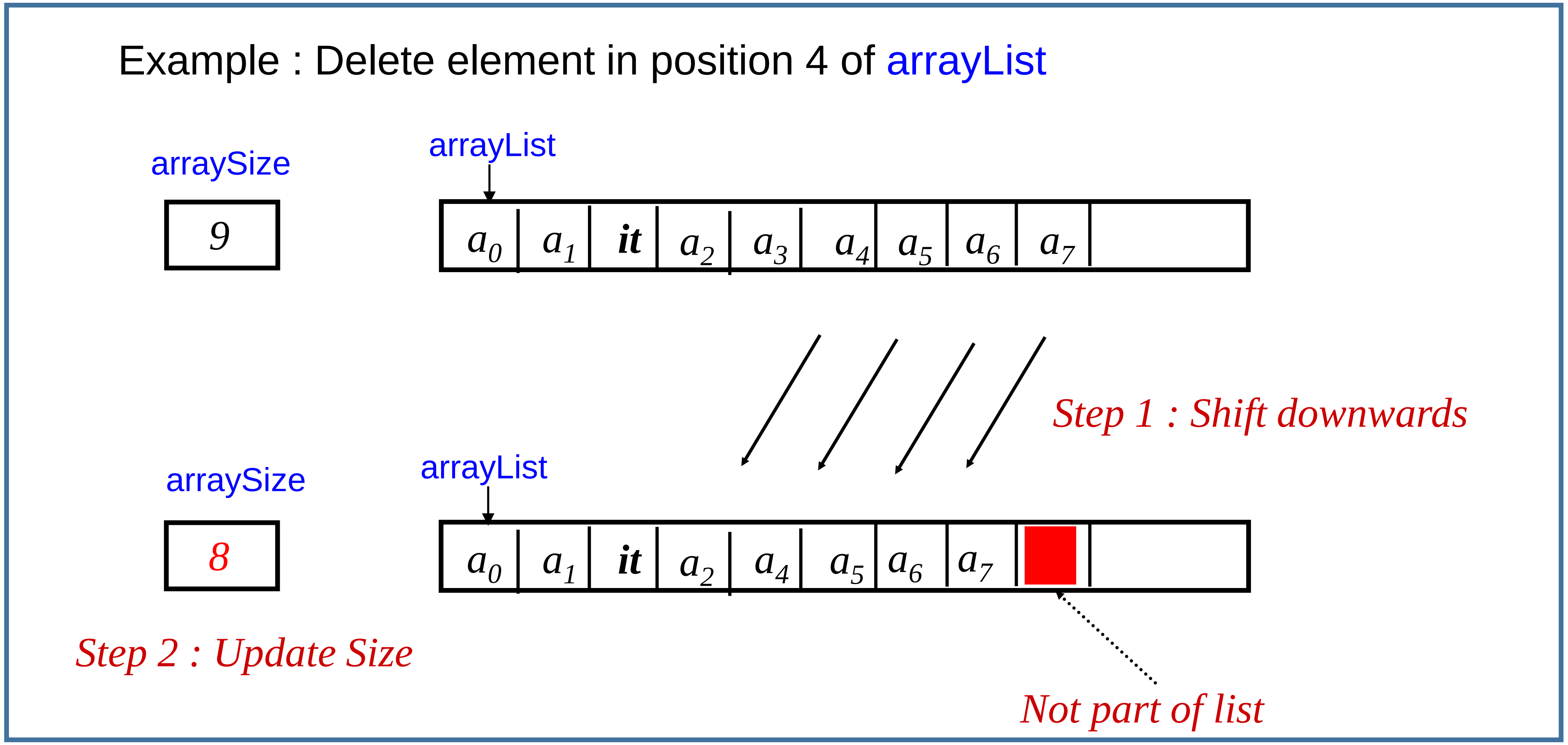
```
void insertion()
{
        int pos, item, j;
        cout << "Enter position;
        cin >> pos;
        cout << "Enter item";
        cin >> item;
        j = arraySize;
        arraySize = arraySize + 1;

        while (j >= pos)
        {
            arrayList[j + 1] = arrayList[j];
            j = j - 1;
        }

        arrayList[pos] = item;
}
```

2.4.4 Array based List Implementation-Deletion

- Deletion has to shift downwards to close gap of deleted item



Example : Delete element in position 4 of arrayList
arraySize
9
arrayList
a_0 a_1 it a_2 a_3 a_4 a_5 a_6 a_7
Step 1 : Shift downwards
arraySize
8
arrayList
a_0 a_1 it a_2 a_4 a_5 a_6 a_7
Step 2 : Update Size
Not part of list

2.4.5 Array based List Implementation-Code

```
void deletion()
{
        int pos, j;
        cout << "Enter position of element to be deleted";
        cin >> pos;
        j = pos;
        while (j < arraySize)
        {
            arrayList[j - 1] = arrayList[j];
            j = j + 1;
        }

        arraySize = arraySize - 1;
}
```

2.5.1 Sorting Array

Problem: “Order the elements of a list”.

- For example, sorting the list

  7, 2, 1, 4, 5, 9

  produces the list

  1, 2, 4, , 5, 7, 9.

- Similarly, sorting the list

  d, h, c, a, f

  produces

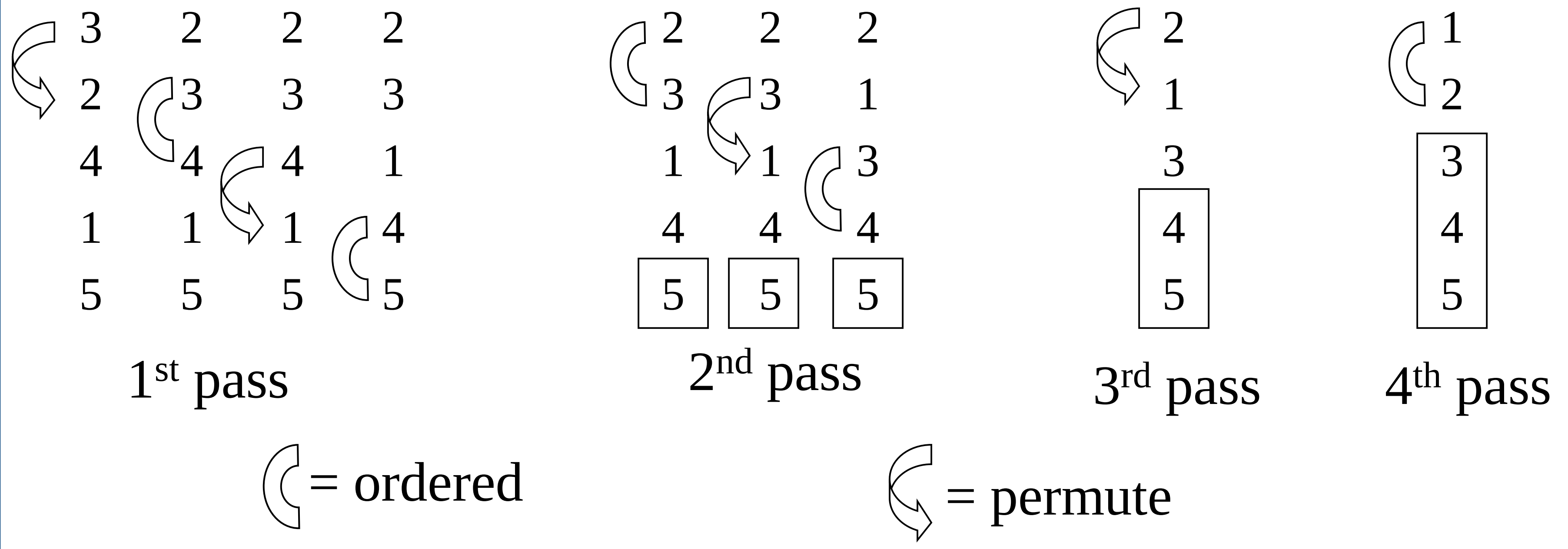
  a, c, d, f, h.

- Three sorting problems are covered:
  1. Bubble Sort
  2. Insertion Sort
  3. Merge Sort

Hungarian Folk Dance: https://www.youtube.com/watch?v=lyZQPjUT5B4

2.5.2 Bubble Sort

Example: Sort the list 3, 2, 4, 1, 5 into increasing order using the Bubble sort



- At the first pass, the largest element will be put in the correct position
- At the end of second pass, the 2nd largest element will be put in the correct position
- In each subsequent, pass, an additional element will be put in the correct position.

2.5.3 Bubble Sort-Pseudocode

```
Procedure Bubblesort ($a_1$, …, $a_n$)
for i = 1 to n-1 {count number of passes}
      for j = 1 to n-i
           if $a_j > a_{j+1}$ then interchange $a_j$ and $a_{j+1}$
 {$a_1$, …, $a_n$ is the increasing order}
```

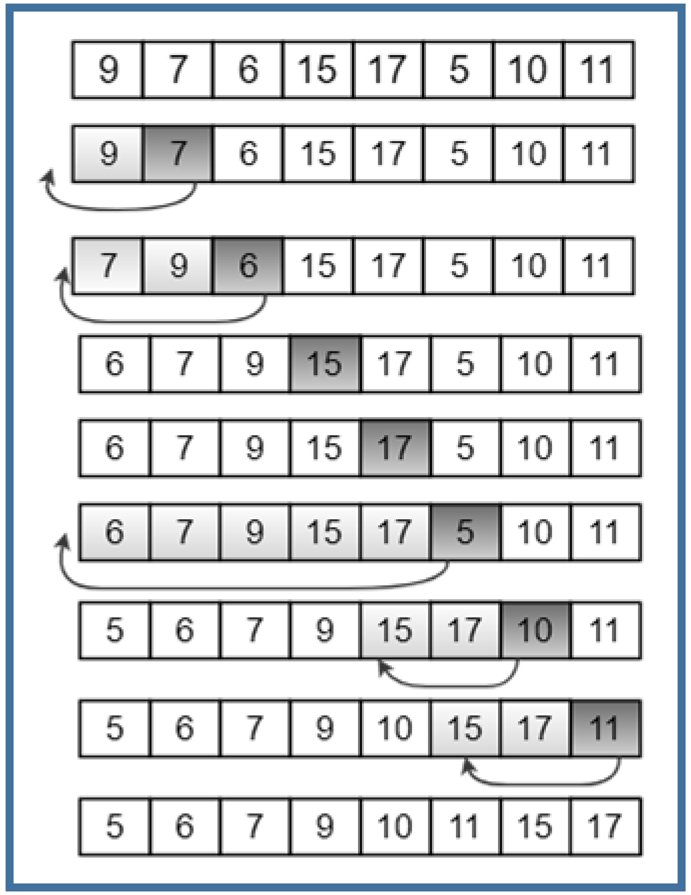
Romanian Folk Dance: https://www.youtube.com/watch?v=ROalU379l3U

2.5.4 Insertion Sort

- In each subsequent pass, the nth element is put into its correct position among the first n elements

Example: Use the insertion sort to put the following elements of the list in ascending order.

9,7,6,15,17,5,10,11



2.5.5 Insertion Sort

```
procedure insertion sort($a_1$, $a_2$, . . . , $a_n$: with $n \geq 2$)
for j := 2 to n
        i = 1
        while $a_j > a_i$
                i = i + 1
        m = $a_j$
        for k = 0 to j - i - 1
                $a_{j-k} = a_{j-k-1}$
        $a_i$ = m
{$a_1$, . . . , $a_n$ is in increasing order}
```

2.5.6 Recursive Merge Sort

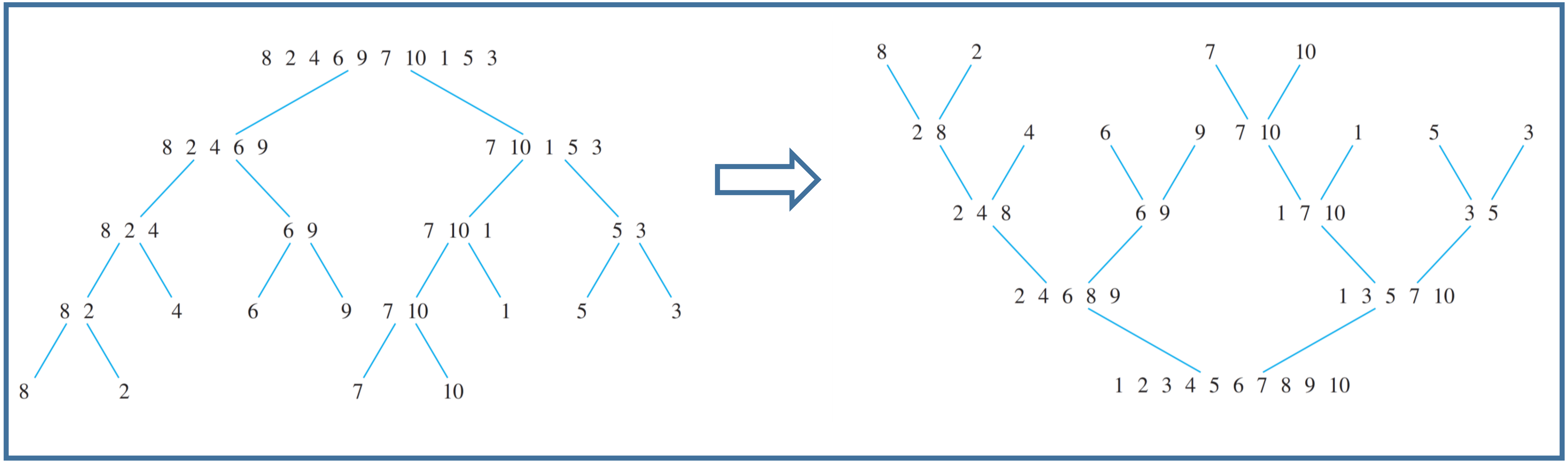
- **Merge Sort** works by iteratively splitting a list (with an even number of elements) into two sublists of equal length until each sublist has one element.

- At each step a pair of sublists is successively merged into a list with the elements in increasing order. The process ends when all the sublists have been merged.

2.5.7 Recursive Merge Sort

German Folk Dance:

https://www.youtube.com/watch?v=XaqR3G_NVoo



1. Divide the list into the smallest unit (1 element),
2. Compare each element with the adjacent list to sort
3. Merge the two adjacent lists.
4. Finally all the elements are sorted and merged.

2.5.8 Recursive Merge Sort

Example: Construct a recursive merge sort algorithm.

Solution: Begin with the list of *n* elements *L*.

```
procedure mergesort(L = $a_1, . . . , a_n$)
if $n > 1$ then
$m = n/2$
$L1 = a_1, a_2, . . . , a_m$
$L2 = a_{m+1}, a_{m+2}, . . . , a_n$
$L = merge(mergesort(L_1), mergesort(L2))$
```

2.5.9 Recursive Merge Sort-Pseudocode

- Subroutine *merge*, which merges two sorted lists.

```
procedure merge(L1, L2:sorted lists)
L = empty list
while $L_1$ and $L_2$ are both nonempty
      remove smaller of first elements of $L_1$ and $L_2$ from its list;
      put at the right end of L
if this removal makes one list empty
      then
      remove all elements from the other list and append them to L
Return L
```

2.5.10 Recursive Merge Sort- List Merging Example

- **Example**: Merge the two lists 2, 3, 5, 6and 1, 4.

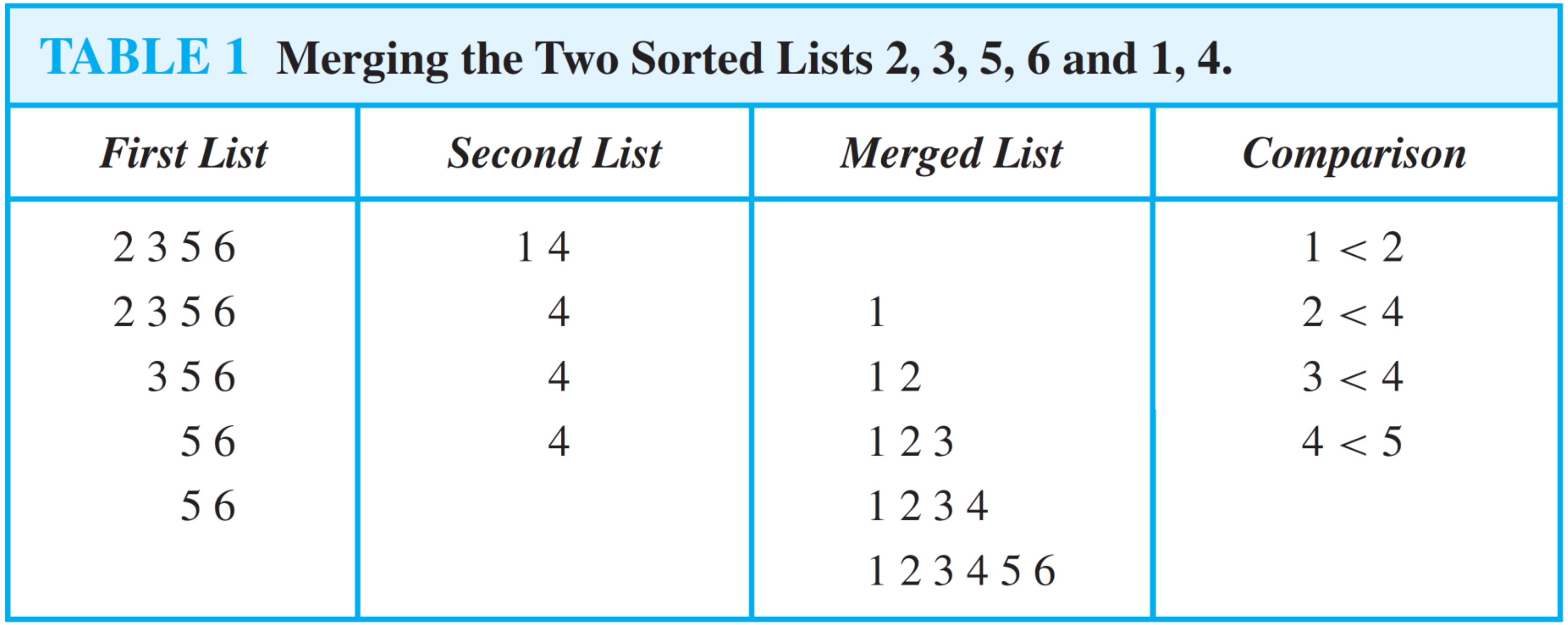


TABLE 1 **Merging the Two Sorted Lists 2, 3, 5, 6 and 1, 4.**

First List	*Second List*	*Merged List*	*Comparison*
2 3 5 6	1 4		$1 < 2$
2 3 5 6	4	1	$2 < 4$
3 5 6	4	1 2	$3 < 4$
5 6	4	1 2 3	$4 < 5$
5 6		1 2 3 4	
		1 2 3 4 5 6	

Note: Slides related to sorting algorithms are based on following book
Discrete Mathematics and Its Applications – Kenneth H. Rosen

Q & A ?