CSE 1201
Data Structure

Chapter 2: Arrays

Introduction

- Arrays
  - Structures of related data items
  - Static entity (same size throughout program)
- A few types
  - Pointer-based arrays (C-like)
  - Arrays as objects (C++)

Introduction

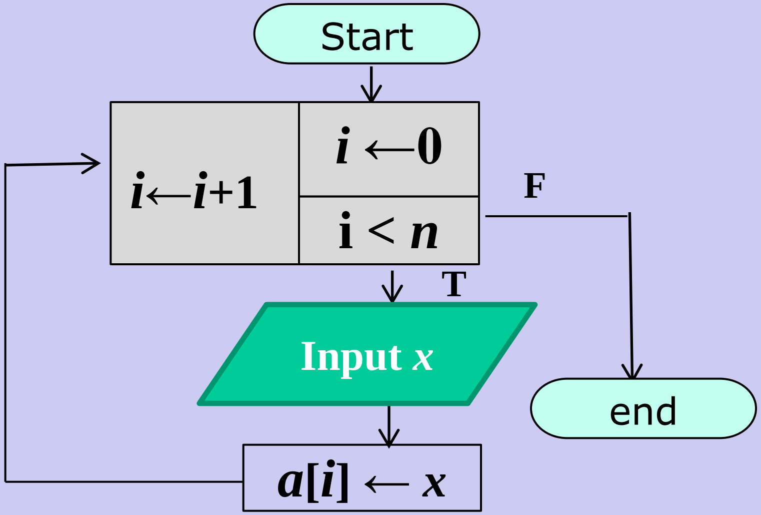
- Array
  - Consecutive group of memory locations
  - Same name and type (**`int`**, **`char`**, etc.)
- To refer to an element
  - Specify array name and position number (index)
  - Format: arrayname[position number]
  - First element at position 1 (0 for C)
- N-element array c

  `c[ 0 ]`, **`c[ 1 ]`** … **`c[ n - 1 ]`**

  - Nth element as position N-1

Array Creation

Topic 1: Write an Algorithm to insert *n* elements in an array

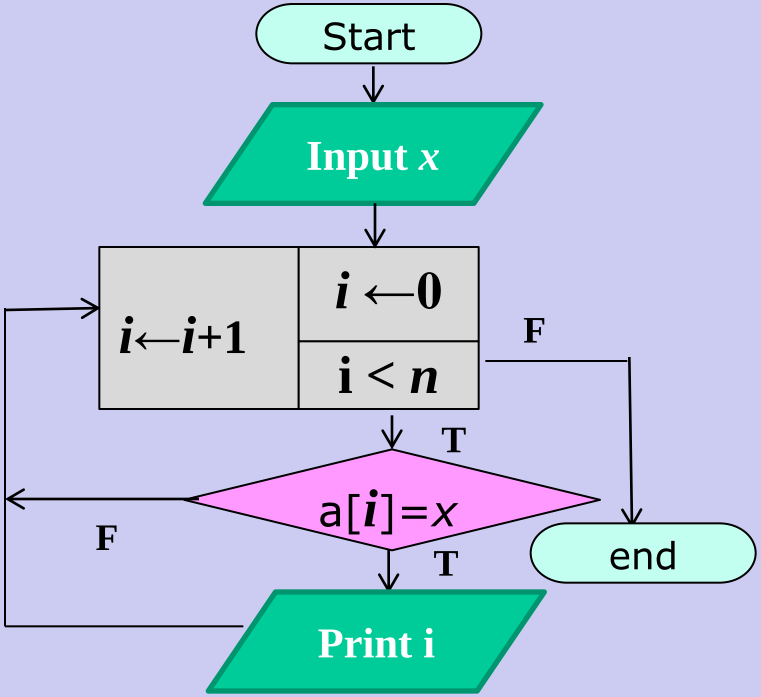


n: total elements
x: input variable
Array Elements: a[0]…..a[*n*-1]

***a*[]**	**10**	**32**	**45**					
	0	1	2	3	…..	….	*n*-2	*n*-1

Array Searching

Topic 2: Write an Algorithm to search a element(s) in an array

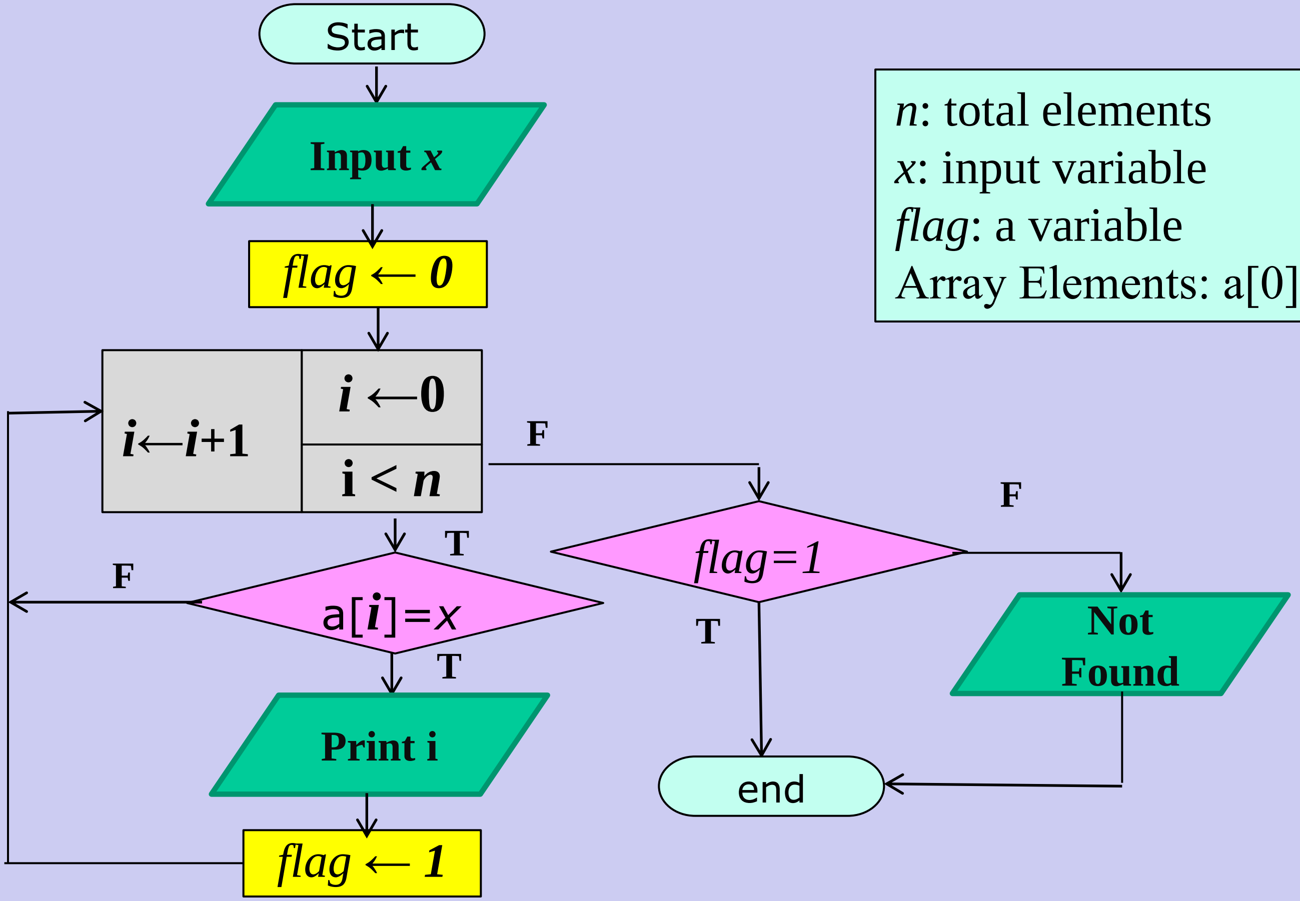


n: total elements
x: input variable
Array Elements: a[0]…..a[*n*-1]

a[]	**10**	**32**	**45**					
	0	1	2	3	…..	….	*n*-2	*n*-1

Array Searching

Topic 3: Modifications of previous algorithm



n: total elements
x: input variable
flag: a variable
Array Elements: a[0]…..a[*n*-1]

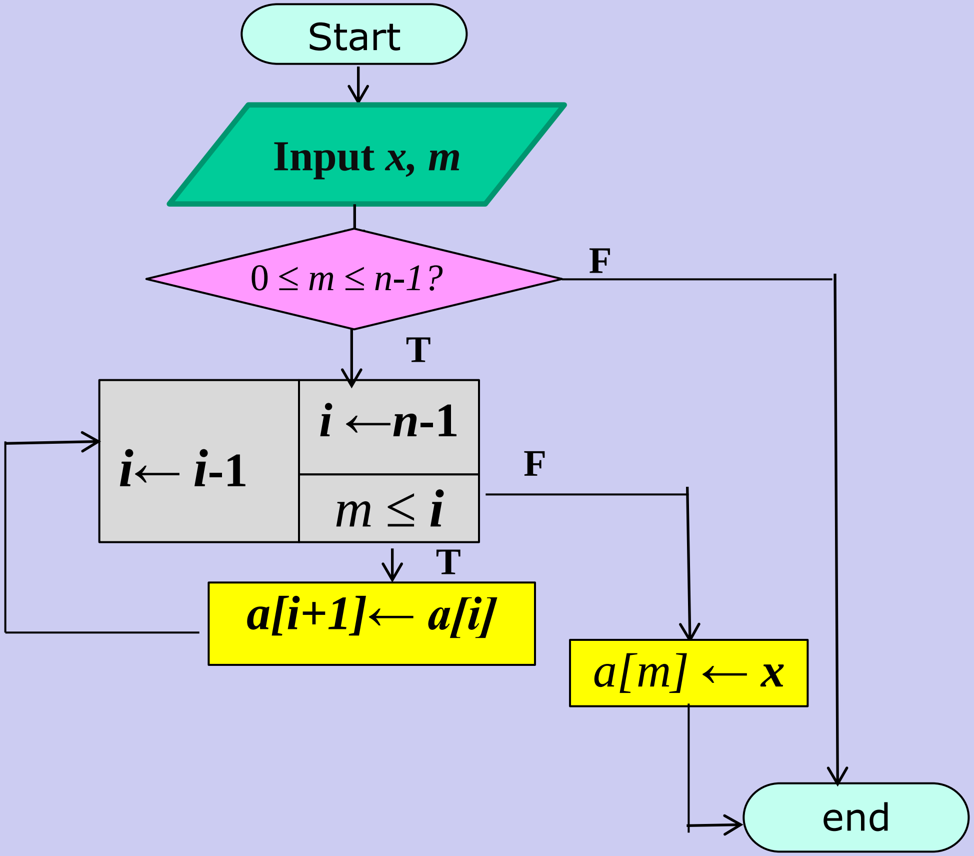
Complexity

O(1) for best-case

O(n) for worst-case

Array Insertion

Topic 4: Insert a new element at index *m*.



n: total elements
m: index $0 \le m \le n\text{-}1$
x: input variable for new data
Array Elements: *a*[0]…..*a[n-1*]

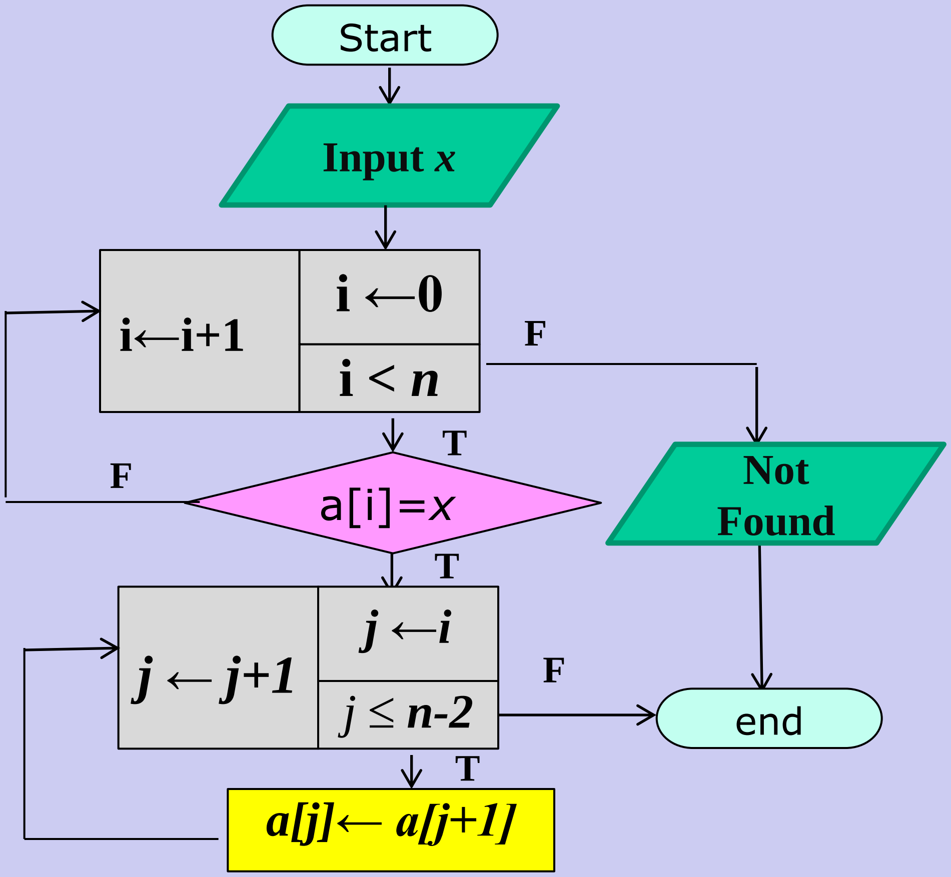
Shifting Required

all elements from index ***m*** to ***(n-1)*** are needed to be shifted to index ***(m+1)*** to ***n*** respectively. Total ***(n-m)*** elements to be shifted.

a[]	**10**	**32**	**45**				
	0	1	….	*m*	…	*n-1*	*n*

Array Insertion

Topic 4: Delete a specific element.



n: total elements
m: index $0 \le m \le n$
x: input variable to be deleted
Array Elements: *a[0]*.....*a[n-1]*

Shifting Required

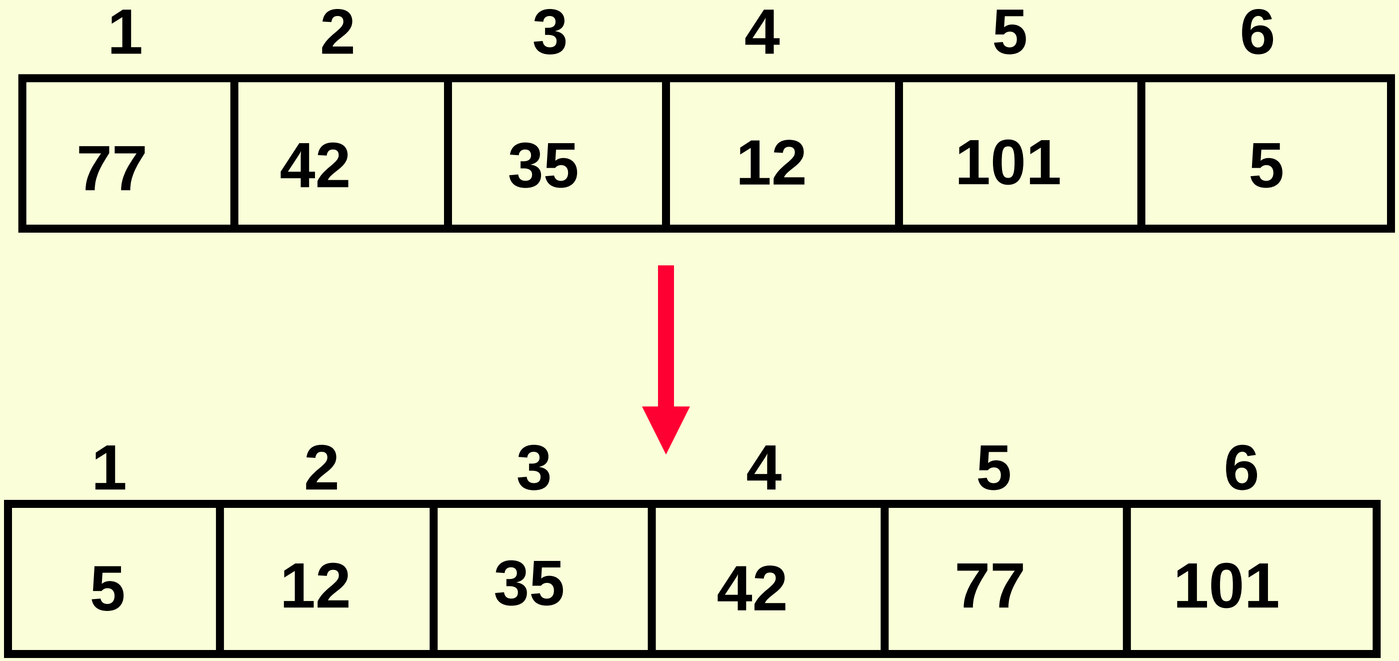
all elements from index (***m*+1)** to ***(n-1)*** are needed to be shifted from index ***m*** to ***(n-2)*** respectively. Total ***(n-m-1)*** elements to be shifted.

a[]	**10**	**32**	**45**				
	0	1	….	*m*	…	*n-1*	*n*

Bubble Sort

Sorting

- Sorting takes an unordered collection and makes it an ordered one.



1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
5	12	35	42	77	101

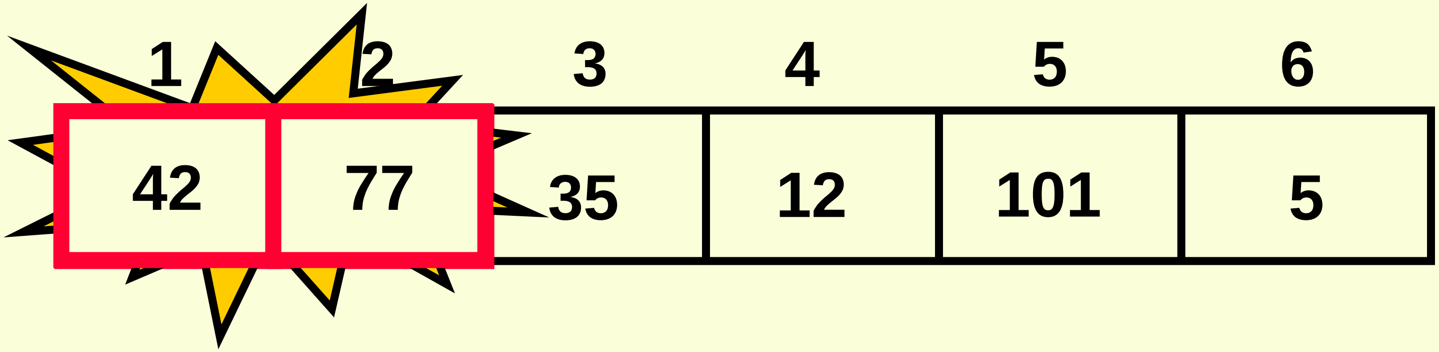
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - – Move from the front to the end
  - – "Bubble" the largest value to the end using pair-wise comparisons and swapping

1	2	3	4	5	6
77	42	35	12	101	5

"Bubbling Up" the Largest Element

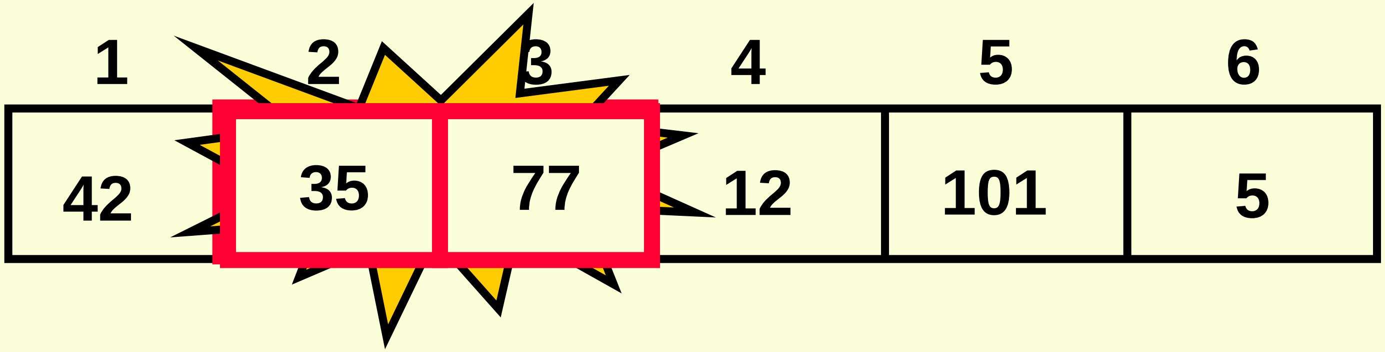
- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



1	2	3	4	5	6
42	77	35	12	101	5

"Bubbling Up" the Largest Element

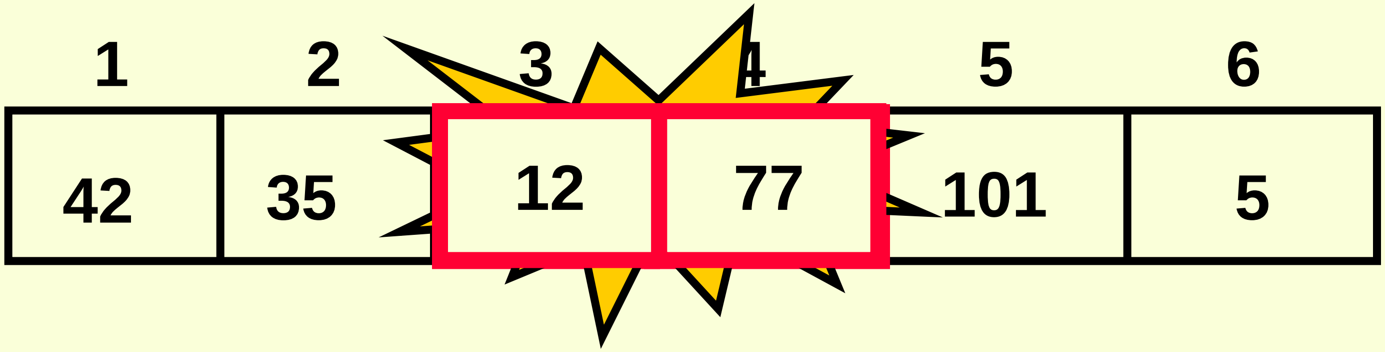
- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



1	2	3	4	5	6
42	35	77	12	101	5

"Bubbling Up" the Largest Element

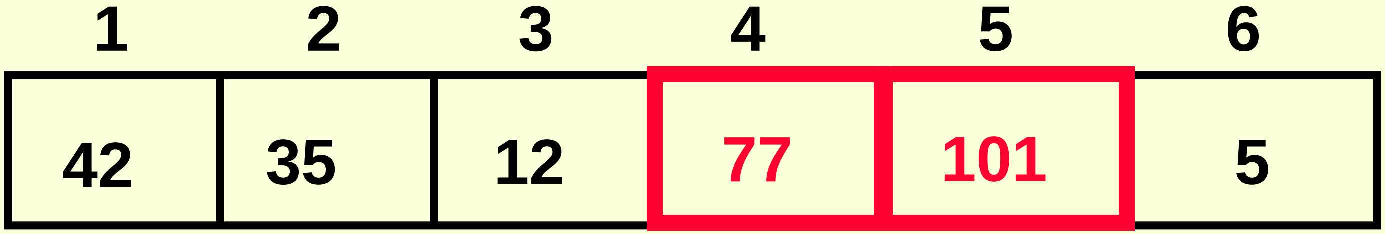
- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



1	2	3	4	5	6
42	35	12	77	101	5

"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

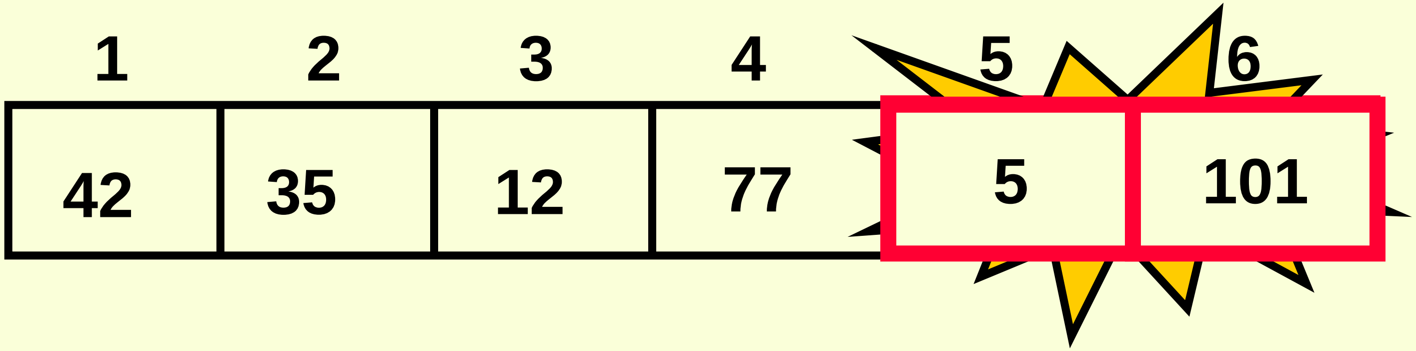


1	2	3	4	5	6
42	35	12	77	101	5

No need to swap

"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



1	2	3	4	5	6
42	35	12	77	5	101

"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

Items of Interest

- Notice that only the largest value is correctly placed
- All other values are still out of order
- So we need to repeat this process

1	2	3	4	5	6
42	35	12	77	5	101

Largest value correctly placed

Repeat “Bubble Up” How Many Times?

- If we have N elements…
- And if each time we bubble an element, we place it in its correct location…
- Then we repeat the “bubble up” process N – 1 times.
- This guarantees we’ll correctly place all N elements.

“Bubbling” All the Elements

N - 1

1	2	3	4	5	6
42	35	12	77	5	101

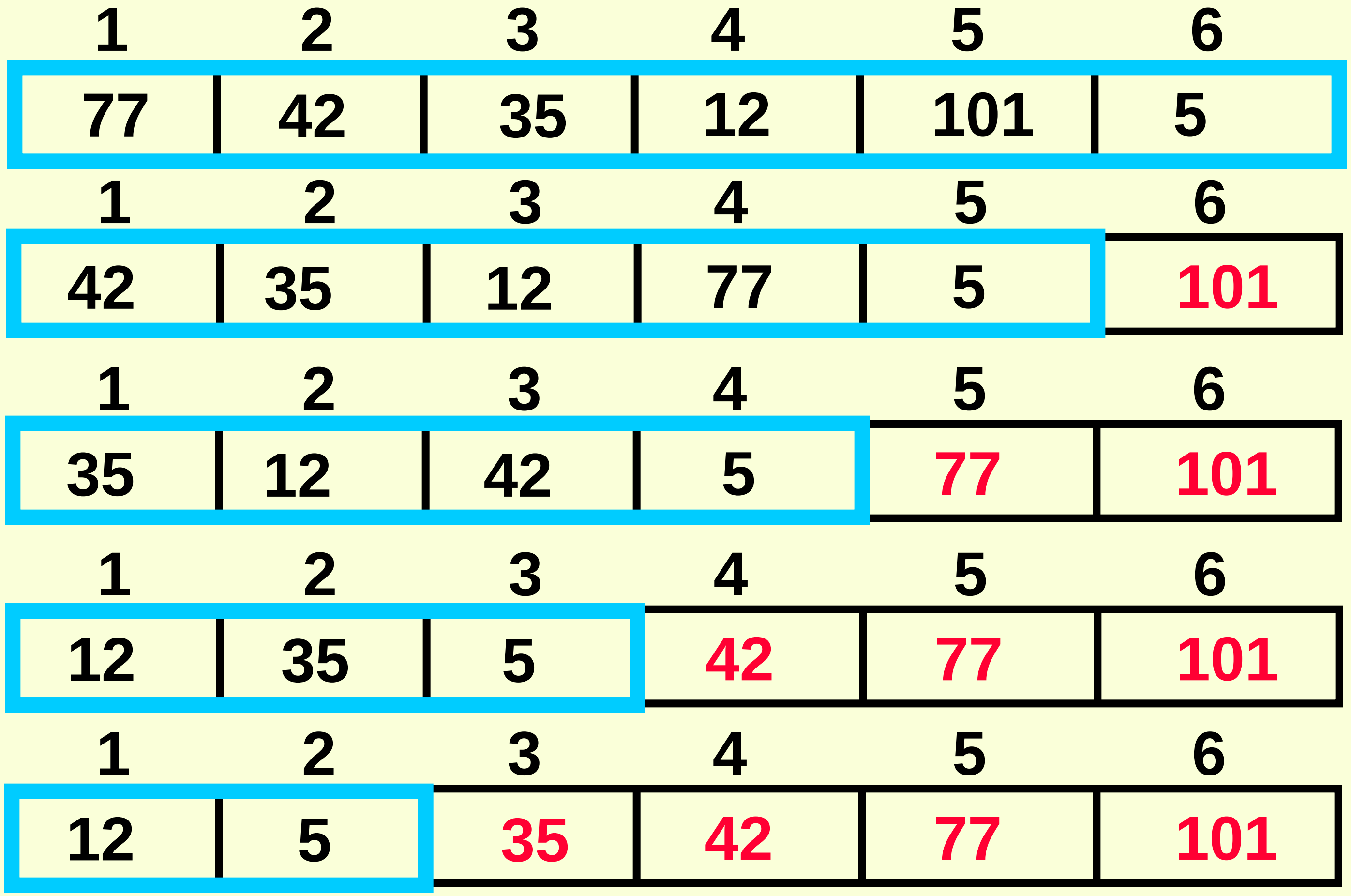
1	2	3	4	5	6
35	12	42	5	77	101

1	2	3	4	5	6
12	35	5	42	77	101

1	2	3	4	5	6
12	5	35	42	77	101

1	2	3	4	5	6
5	12	35	42	77	101

Reducing the Number of Comparisons



1	2	3	4	5	6
77	42	35	12	101	5

1	2	3	4	5	6
42	35	12	77	5	101

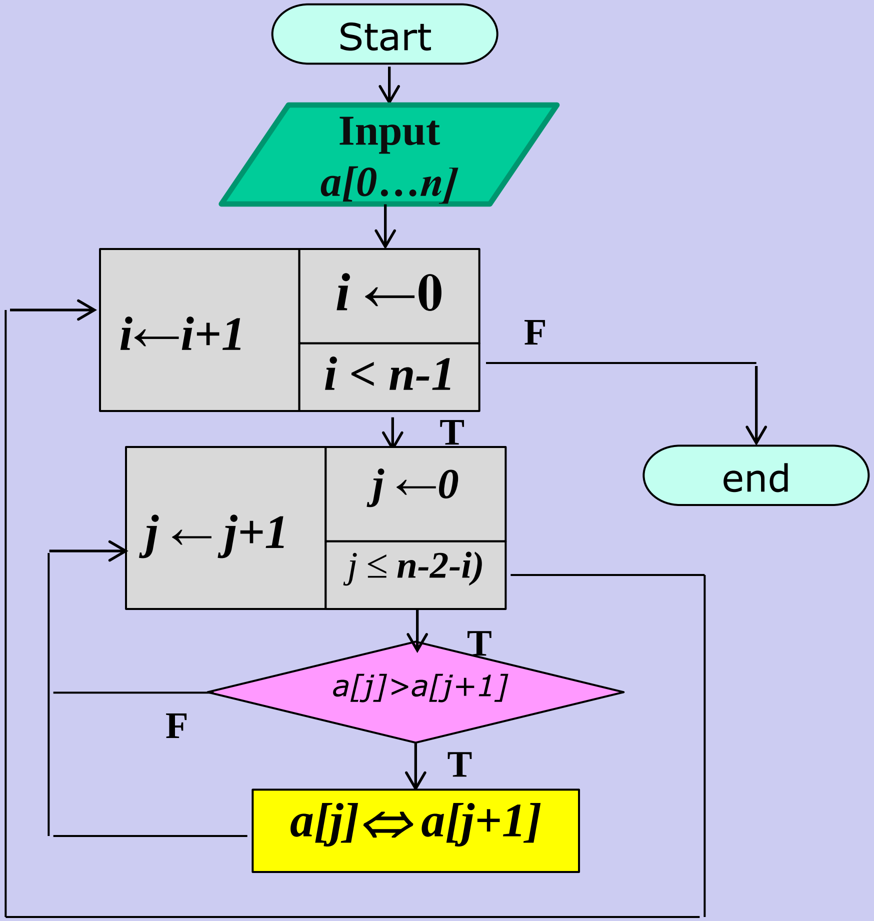
1	2	3	4	5	6
35	12	42	5	77	101

1	2	3	4	5	6
12	35	5	42	77	101

1	2	3	4	5	6
12	5	35	42	77	101

Bubble Sort

Topic 5: Wrte an algorithm to sort an array using Bubble Sort.



n: total elements
Array Elements: a[0]…..a[*n*-1]

4.4 Examples Using Arrays

- Initializing arrays
  - For loop
    - Set each element
  - Initializer list
    - Specify each element when array declared

    ```
    int n[ 5 ] = { 1, 2, 3, 4, 5 };
    ```

    - If not enough initializers, rightmost elements 0
    - If too many syntax error
  - To set every element to same value

    ```
    int n[ 5 ] = { 0 };
    ```

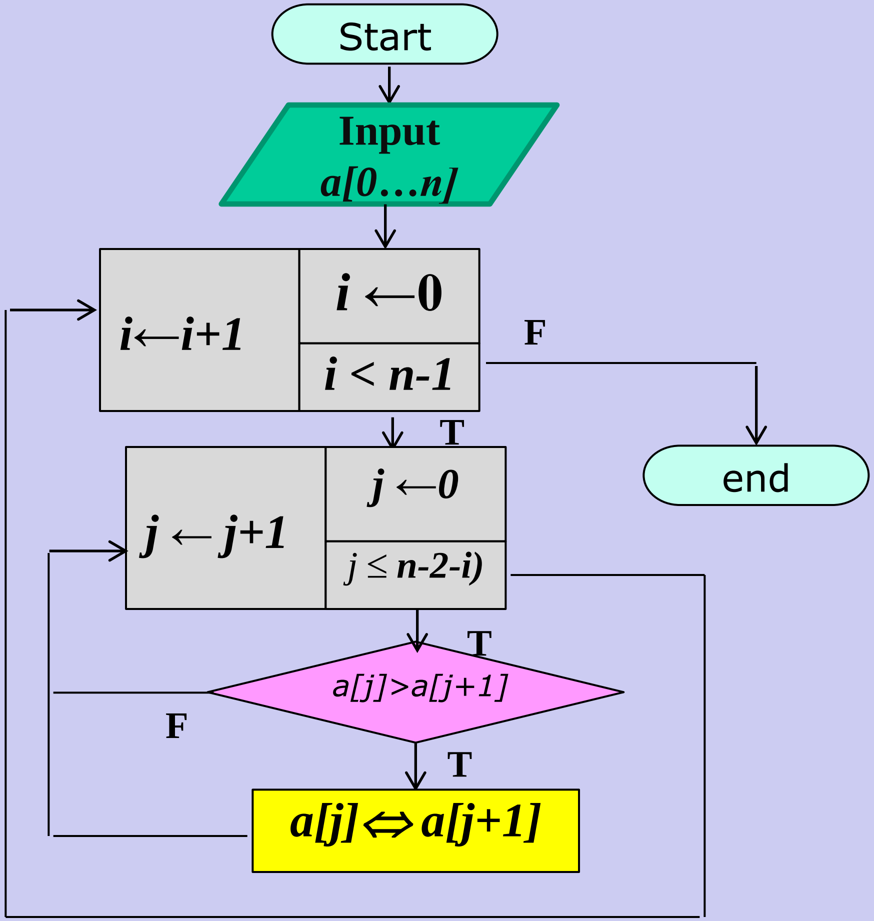
  - If array size omitted, initializers determine size

    ```
    int n[] = { 1, 2, 3, 4, 5 };
    ```

    - 5 initializers, therefore 5 element array

Bubble Sort

Topic 5: Wrte an algorithm to sort an array using Bubble Sort.



n: total elements
Array Elements: a[0]…..a[*n*-1]

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int a[6]={10,3,41,12,77,21};
   int n=6;
   int i,j,t;
   for(i=0;i<n-1;i++)
    for(j=0;j<=n-2-i;j++)
     if(a[j]>a[j+1])
   {
       t=a[j];a[j]=a[j+1];a[j+1]=t;
   }
   for(i=0;i<n;i++)
    printf("%d ",a[i]);
    return 0;
}
```

Corresponding C program

Two-dimensional Arrays

Matrix

➔In computer programming, a **matrix** can be defined with a 2-dimensional array. Any array with 'm' columns and 'n' rows represents a mXn matrix.

Sparse Matrix

➔There may be a situation in which a matrix contains more number of ZERO values than NON-ZERO values. Such matrix is known as sparse matrix

Example

consider a matrix of size 100 X 100 containing only 10 non-zero elements. In this matrix, only 10 spaces are filled with non-zero values and remaining spaces of matrix are filled with zero. That means, totally we allocate 100 X 100 X 2 = 20000 bytes of space to store this integer matrix. And to access these 10 non-zero elements we have to make scanning for 10000 times.

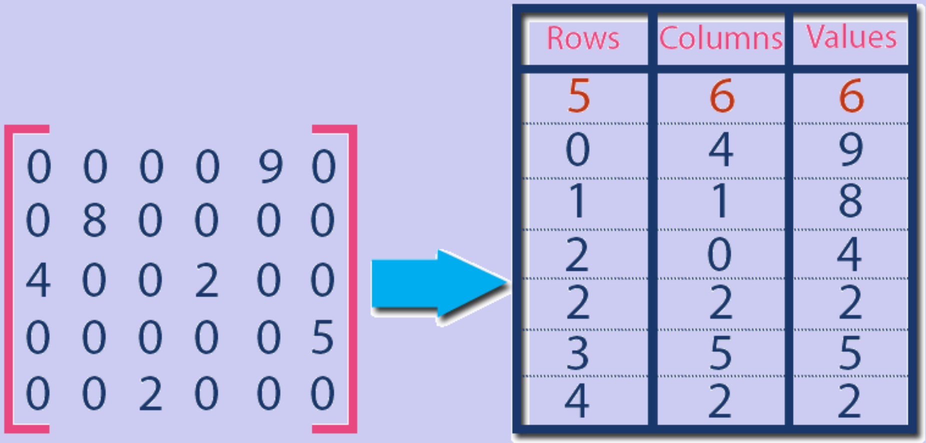
Sparse Matrix

Representation

- ➔Triplet representation
- ➔Linked representation

Triplet Representation

In this representation, we consider only non-zero values along with their row and column index values. In this representation, the 0th row stores total rows, total columns and total non-zero values in the matrix. consider a matrix of size 5 X 6 containing 6 number of non-zero values. This matrix can be represented as shown in the image.

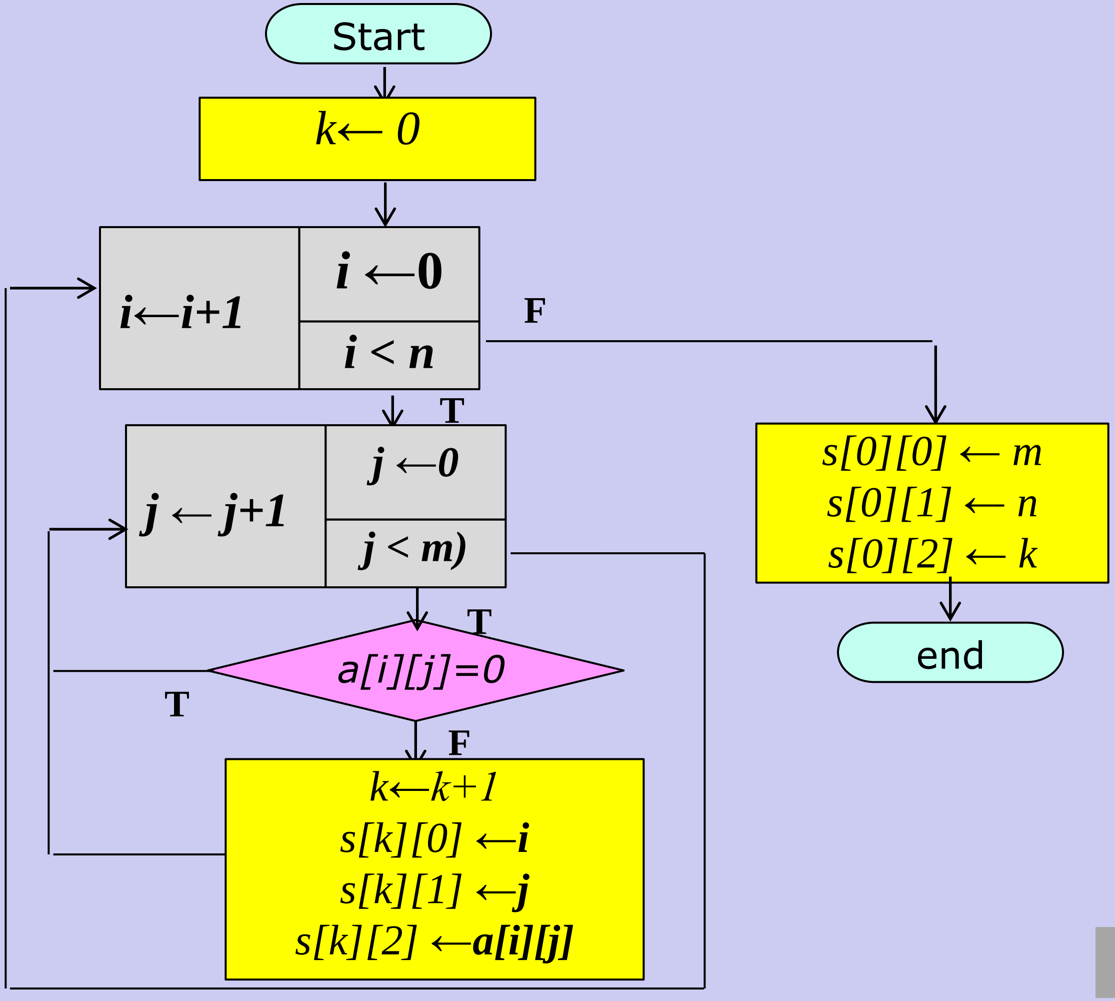


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

Rows	Columns	Values
5	6	6
0	4	9
1	1	8
2	0	4
2	2	2
3	5	5
4	2	2

Sparse Matrix

Topic 5: Algorithm for Triplex Representation



mxn: total elements

Array: *a[0…m-1,0…n-1]*

s[0…N, 0..2]

i: stores row of non-zero value

j: store col of non-zero value

k: counter for non-zero values

Corresponding C program

Assignments

Prob 1: Write an algorithm to insert an element after a specific element.

Prob 2: Write an algorithm to delete all the multiple matching elements.

Prob 3: Write an algorithm to split an array using a particular condition.

Prob 4: Write an algorithm to merge two sorted arrays into one sorted array.

Prob 5: Write an algorithm to multiply to matrices.

Prob 6: Write C programs for Creating Triplex form of Sparse Matrix.

End of Chapter 2