CSE 1201
Data Structure

Week 2

Lecture 3

Chapter 2:
Arrays, Pointers and Records

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++)

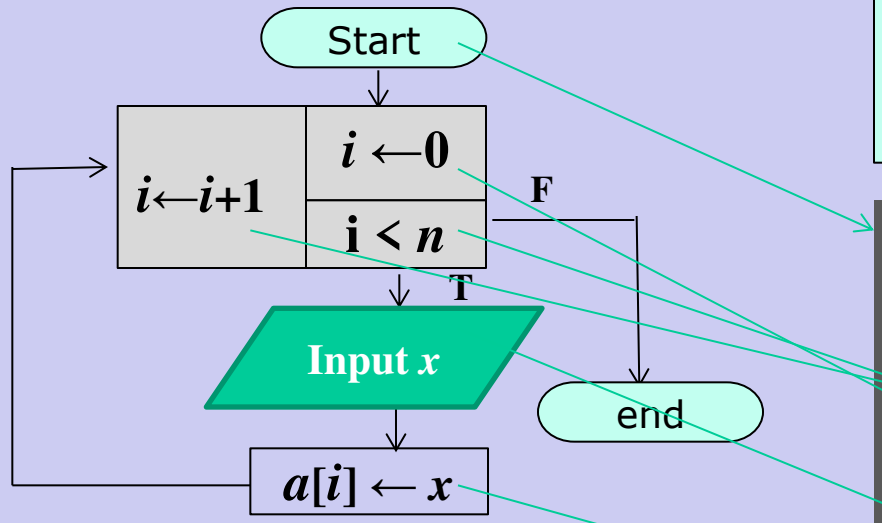
Introduction

- 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

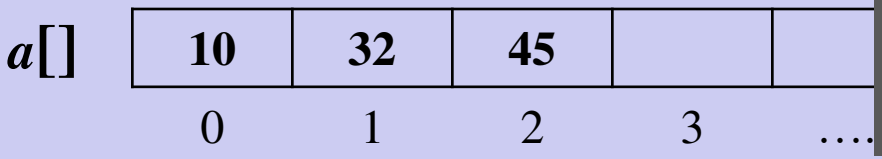
Array Creation

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



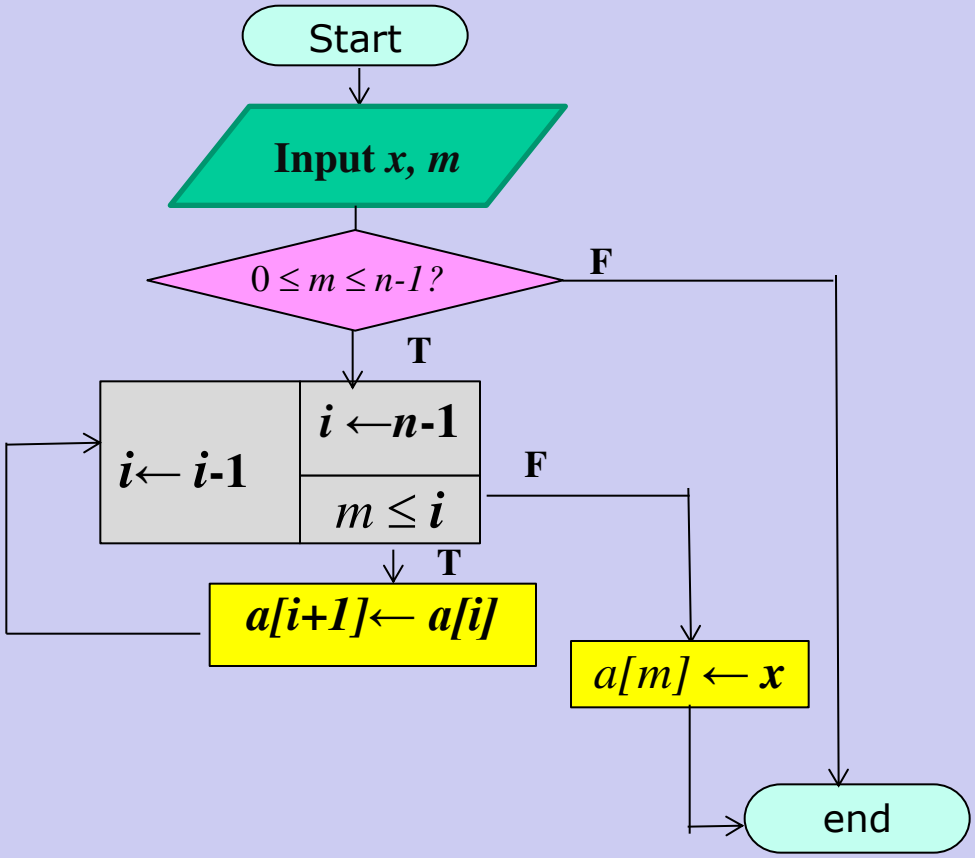
n: total elements
x: input variable
Array Elements: a[0].....a[*n*-1]

```
int main(){
  int i, a[100], n;
  n=10;
  for(i=0;i<n;i++){
    scanf("%d",&a[i]);
    a[i]=x;
}
}
```



Array Insertion

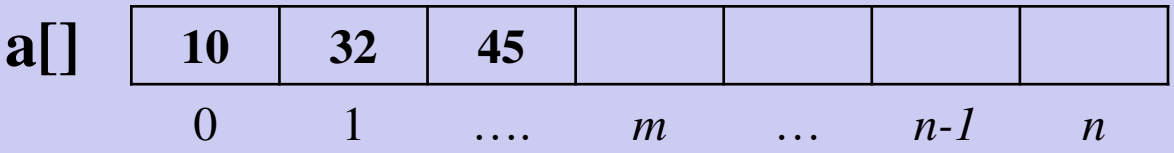
Topic : Insert a new element at index *m*.



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

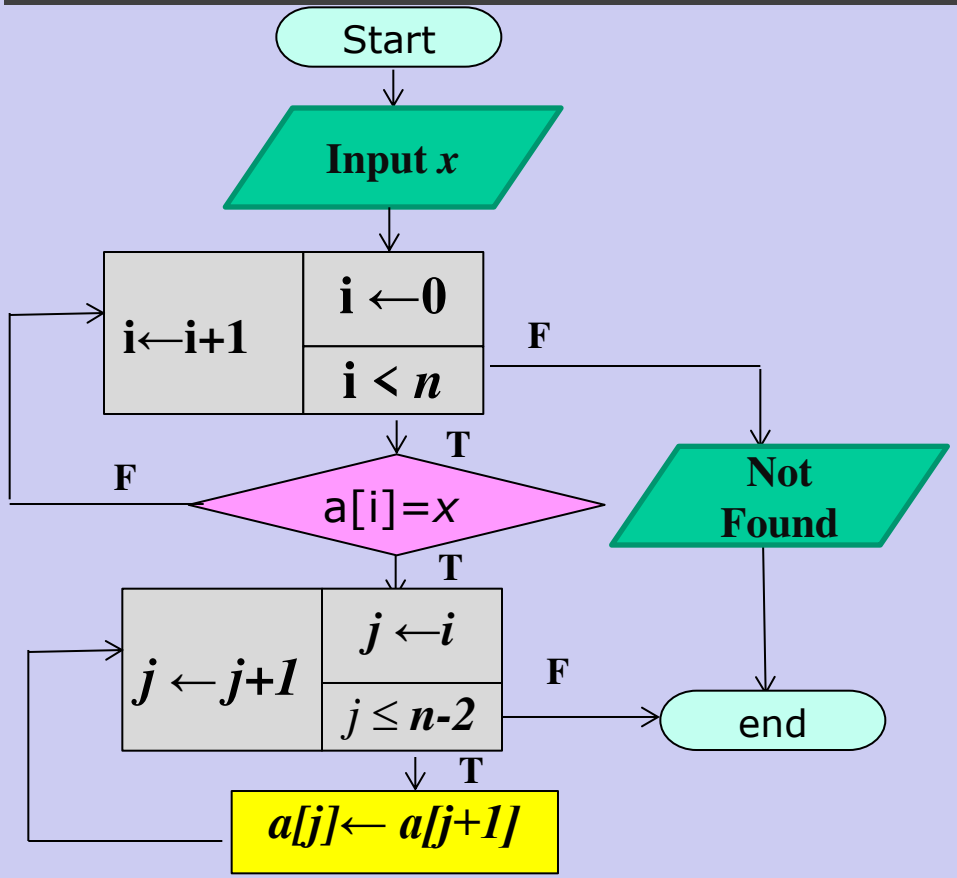
Shifting Required

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.



Array Deletion

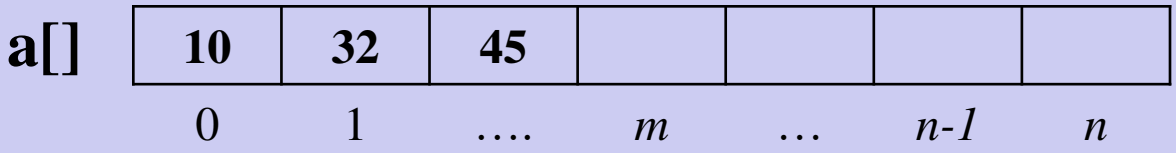
Topic : Delete an specific element x



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

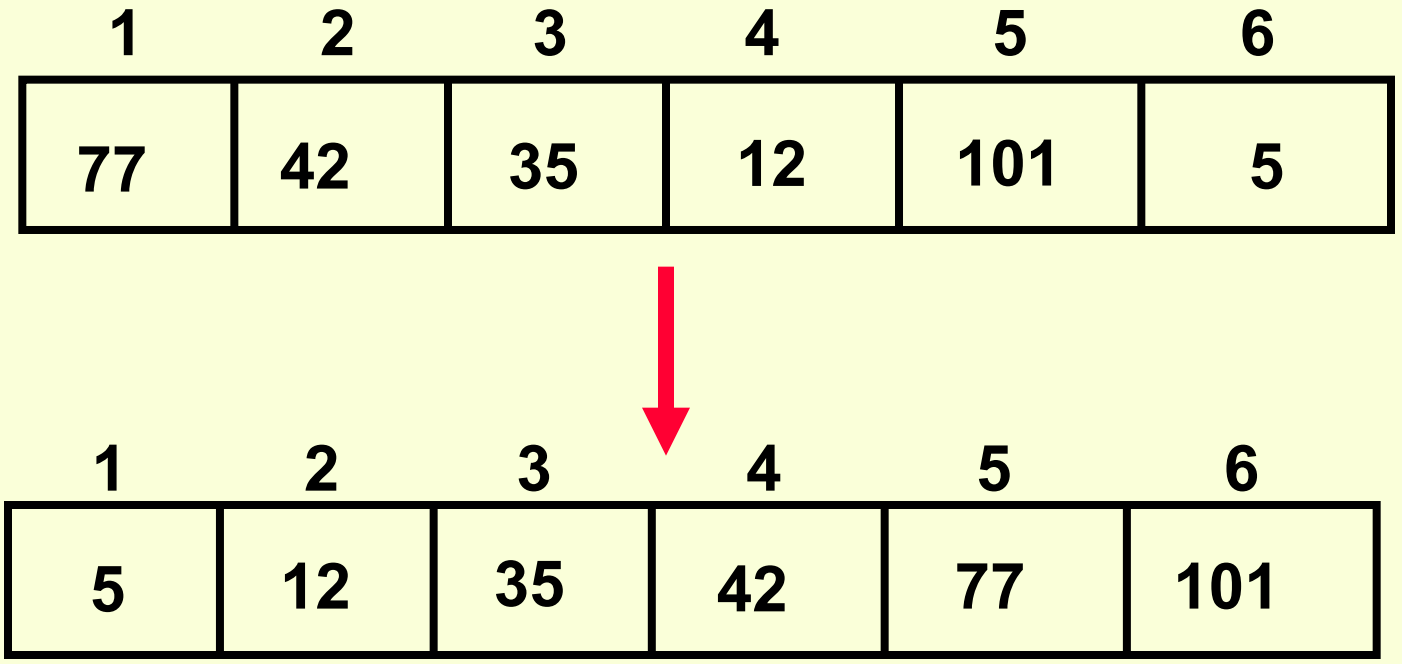
all elements from index ***(m+1)*** to ***(n-1)*** are needed to be shifted from index ***m*** to ***(n-2)*** respectively. Total ***(n-m-1)*** elements to be shifted.



Bubble Sort

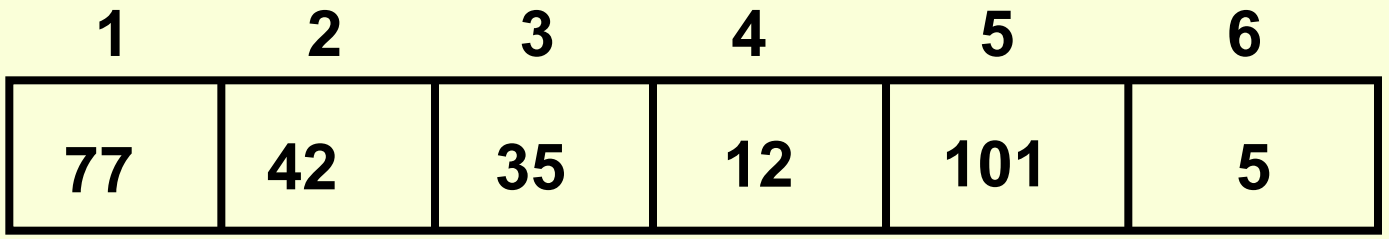
Sorting

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



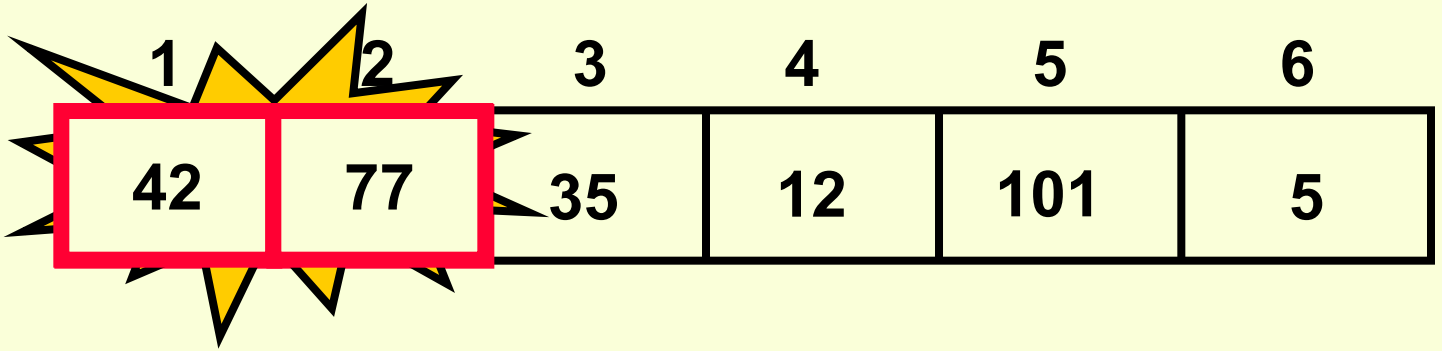
"Bubbling Up" the Largest Element

- 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



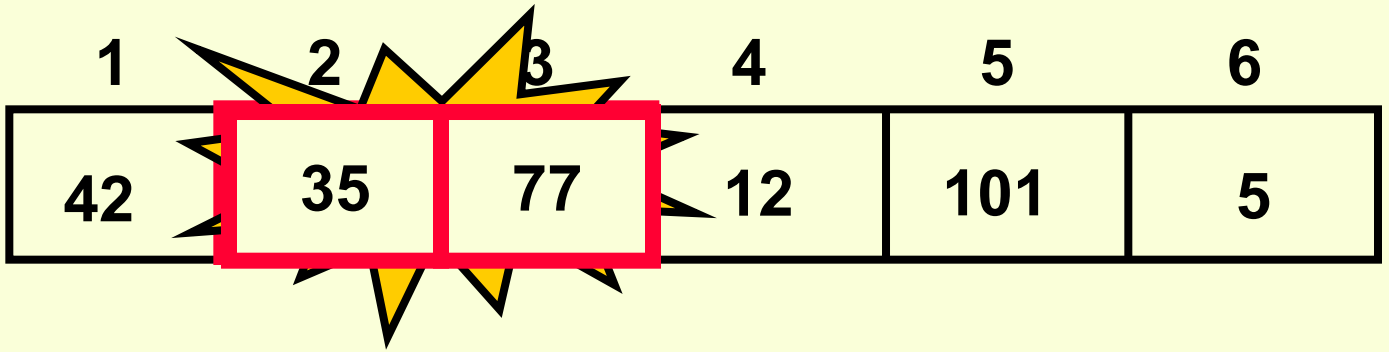
"Bubbling Up" the Largest Element

- 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



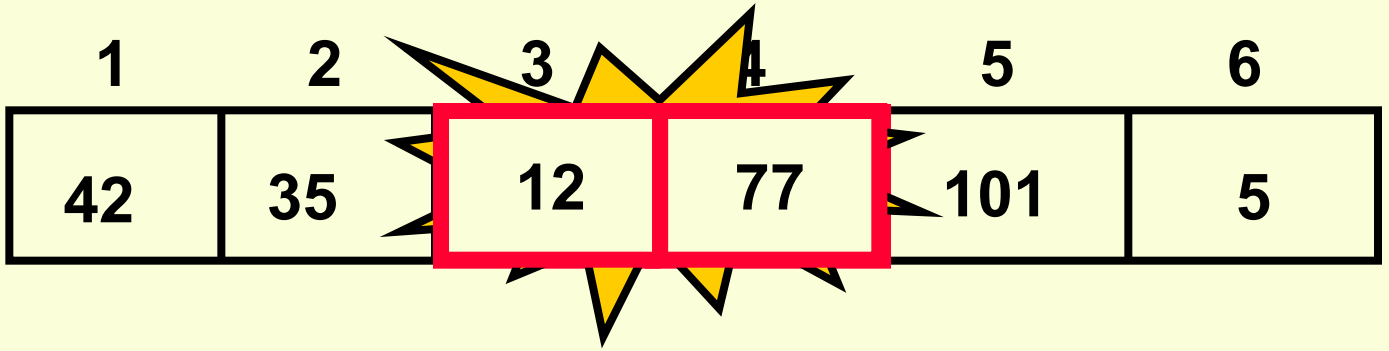
"Bubbling Up" the Largest Element

- 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



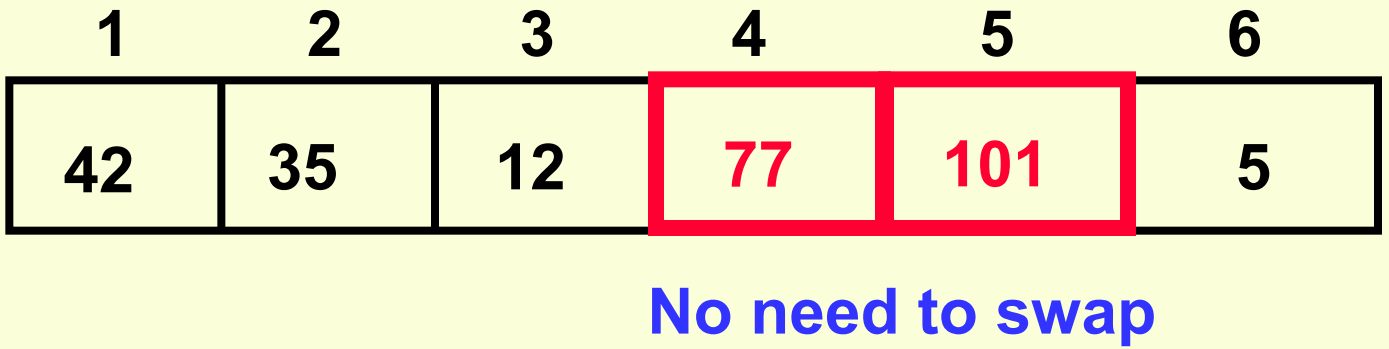
"Bubbling Up" the Largest Element

- 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



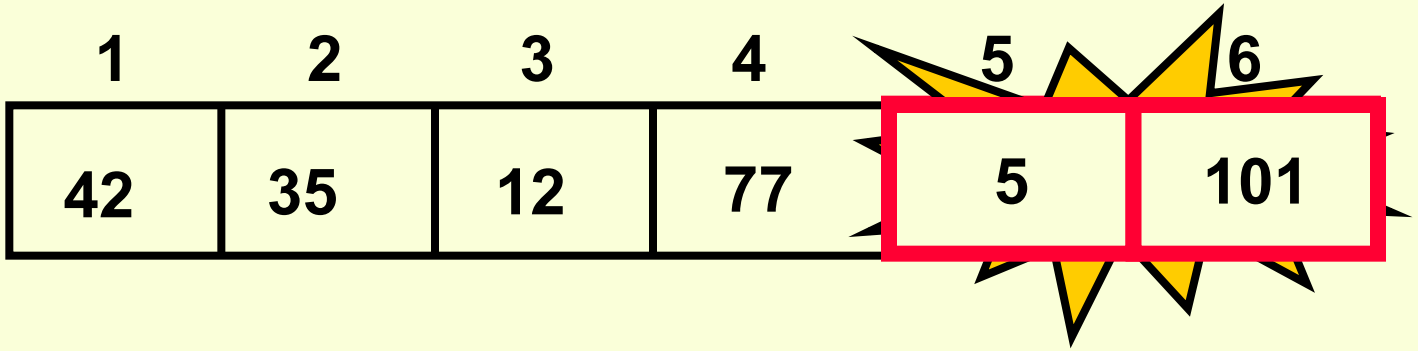
"Bubbling Up" the Largest Element

- 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



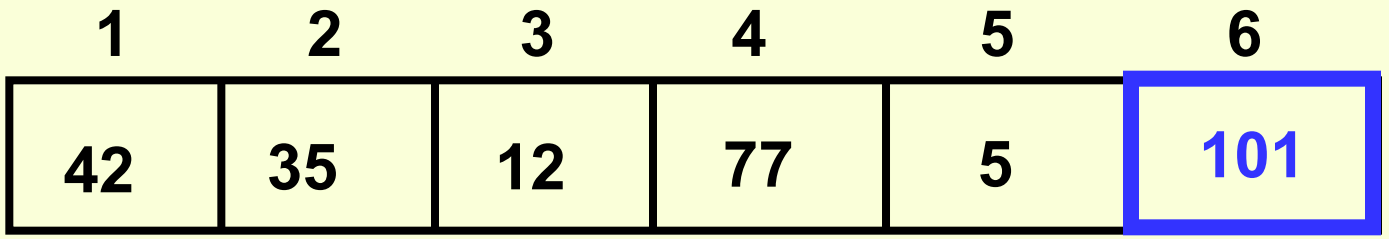
"Bubbling Up" the Largest Element

- 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



"Bubbling Up" the Largest Element

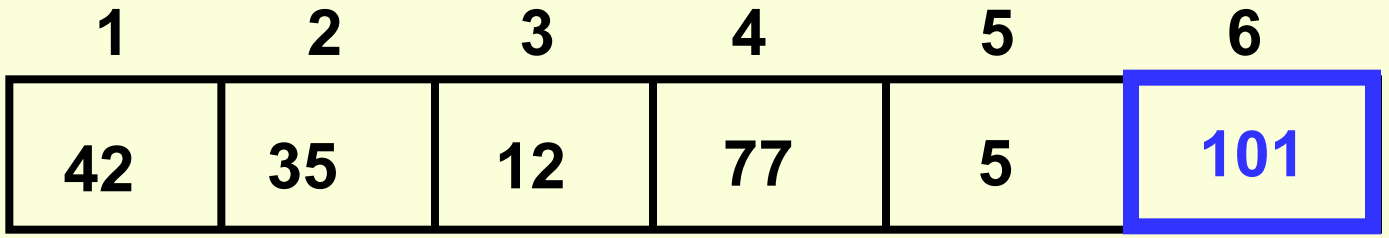
- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



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

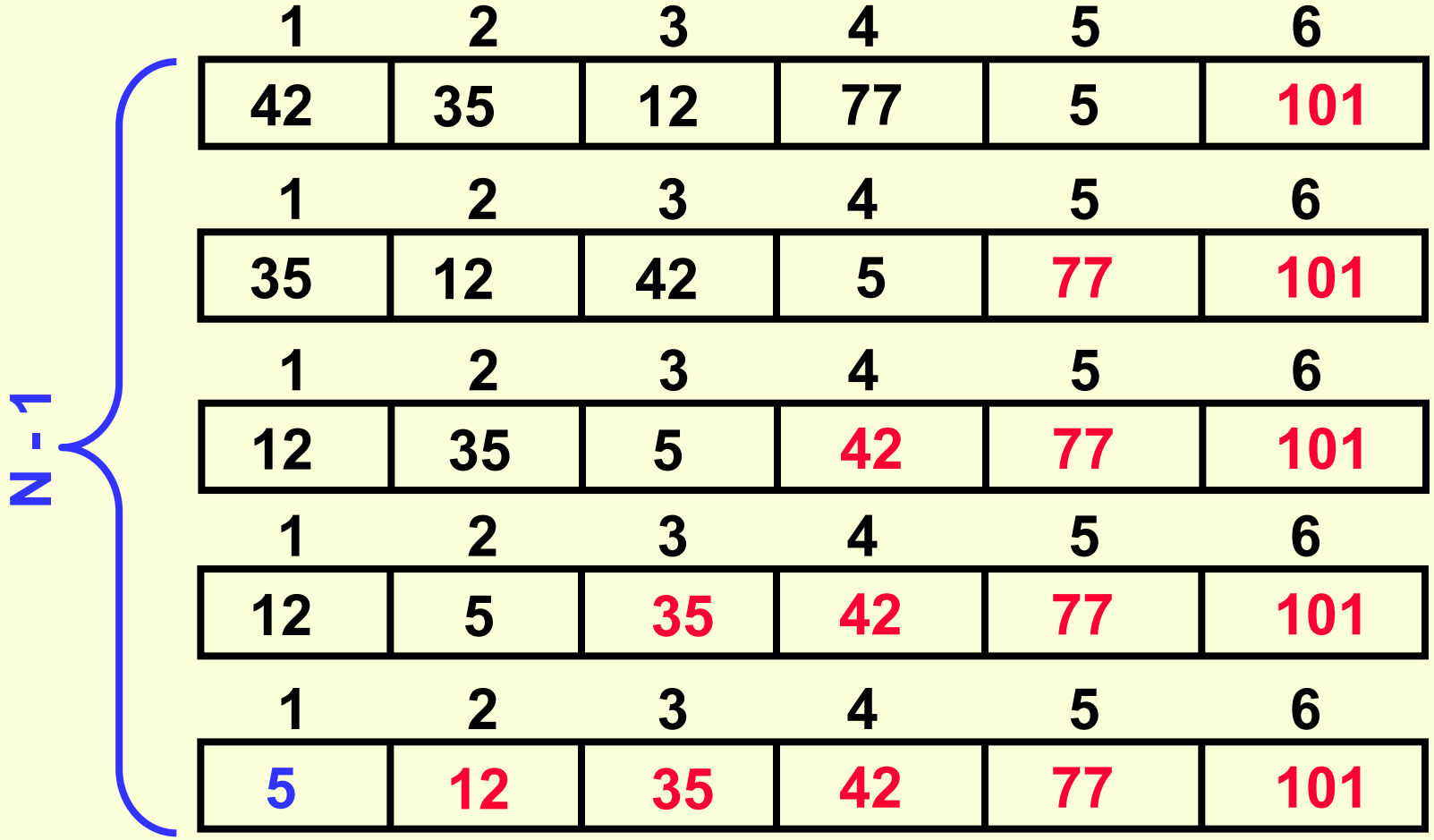


Largest value correctly placed

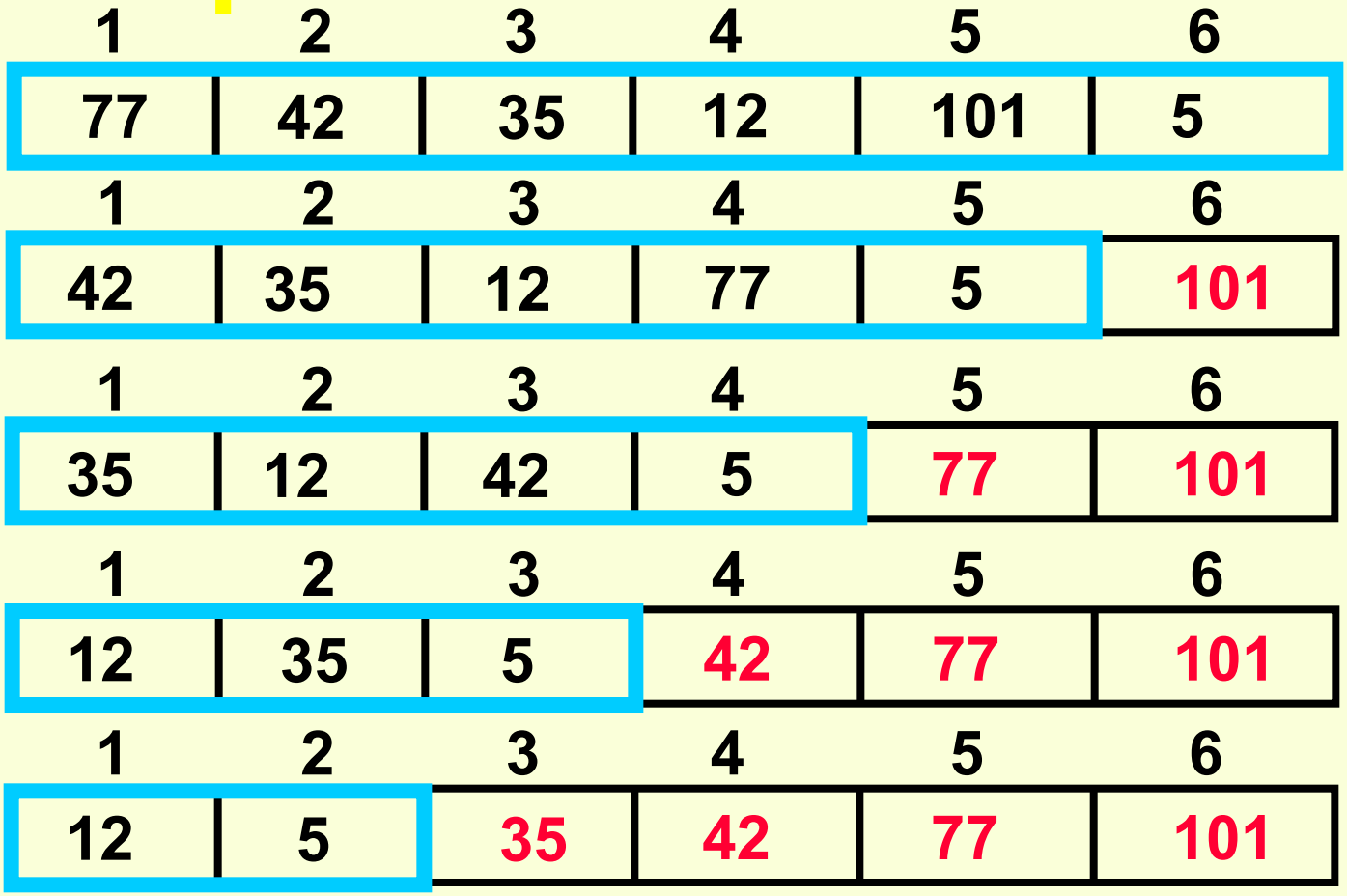
Repeat “Bubble Up” How Many Times?

- 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

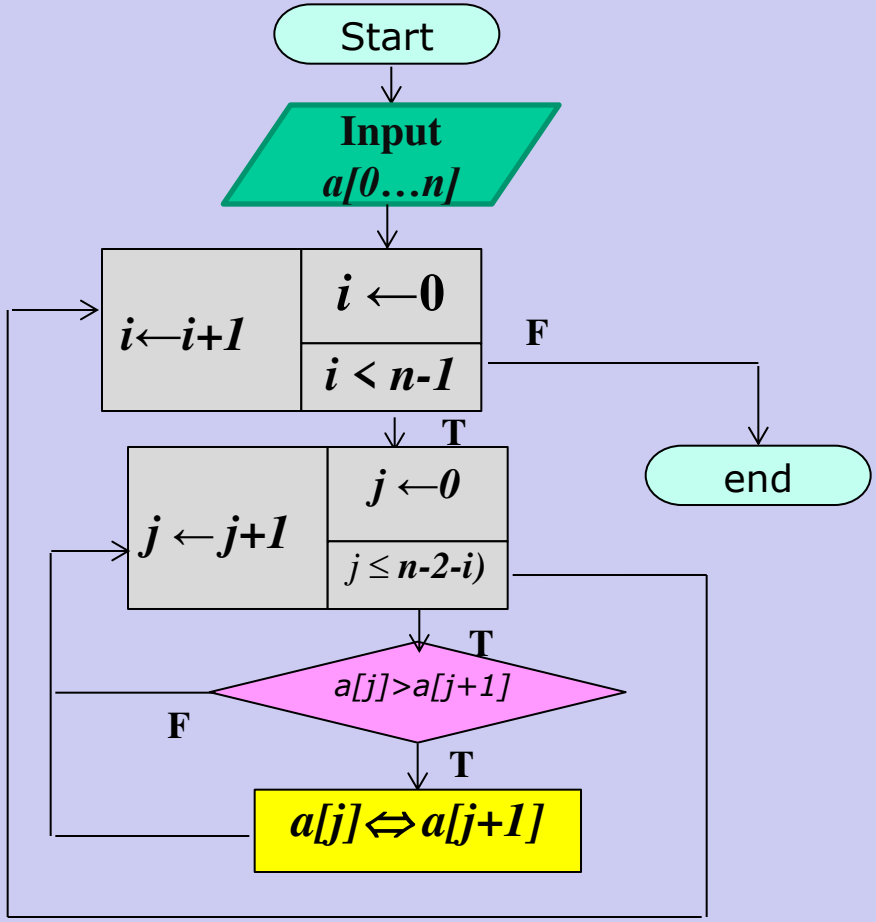


Reducing the Number of Comparisons



Bubble Sort

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



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

Complexity Analysis of Bubble Sort

In Bubble Sort, n-1 comparisons will be done in the 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on. So the total number of comparisons will be,

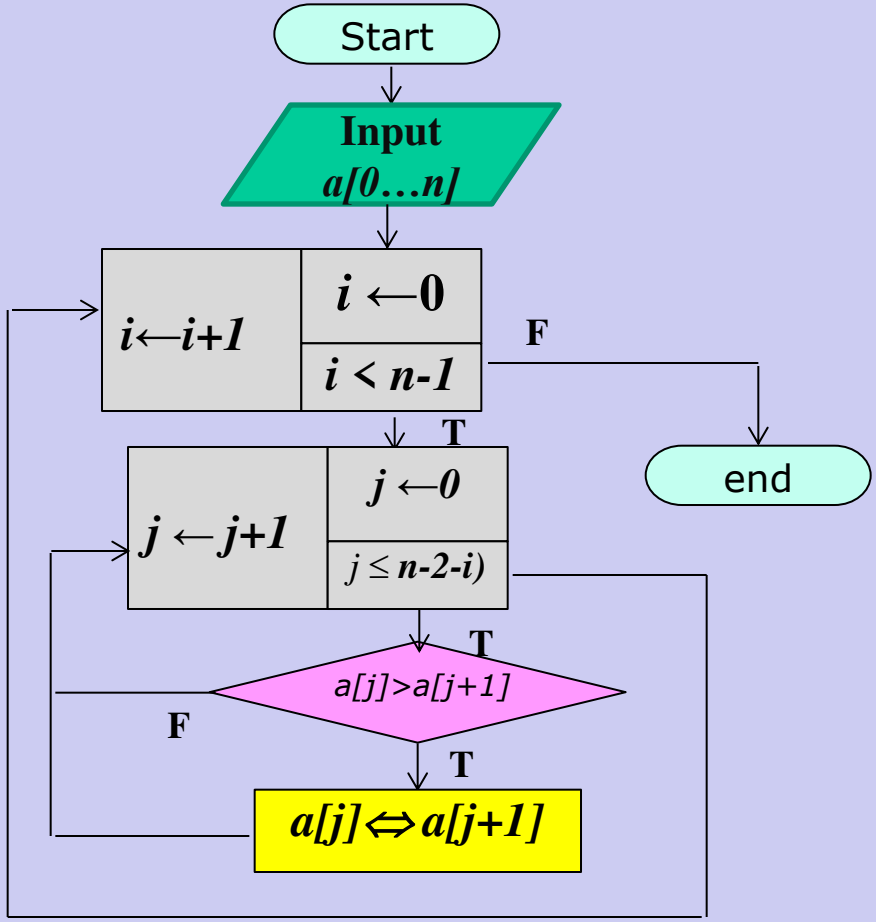
Output:

Sum=(n-1) + (n-2) + (n-3) + ... + 3 + 2 + 1

Sum = n(n-1)/2 i.e **$O(n^2)$**

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

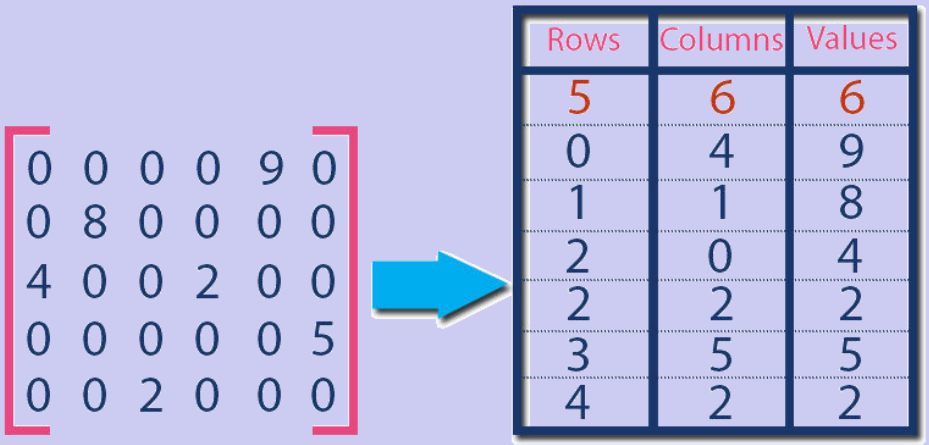
Representation

➔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 0^{th} 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.

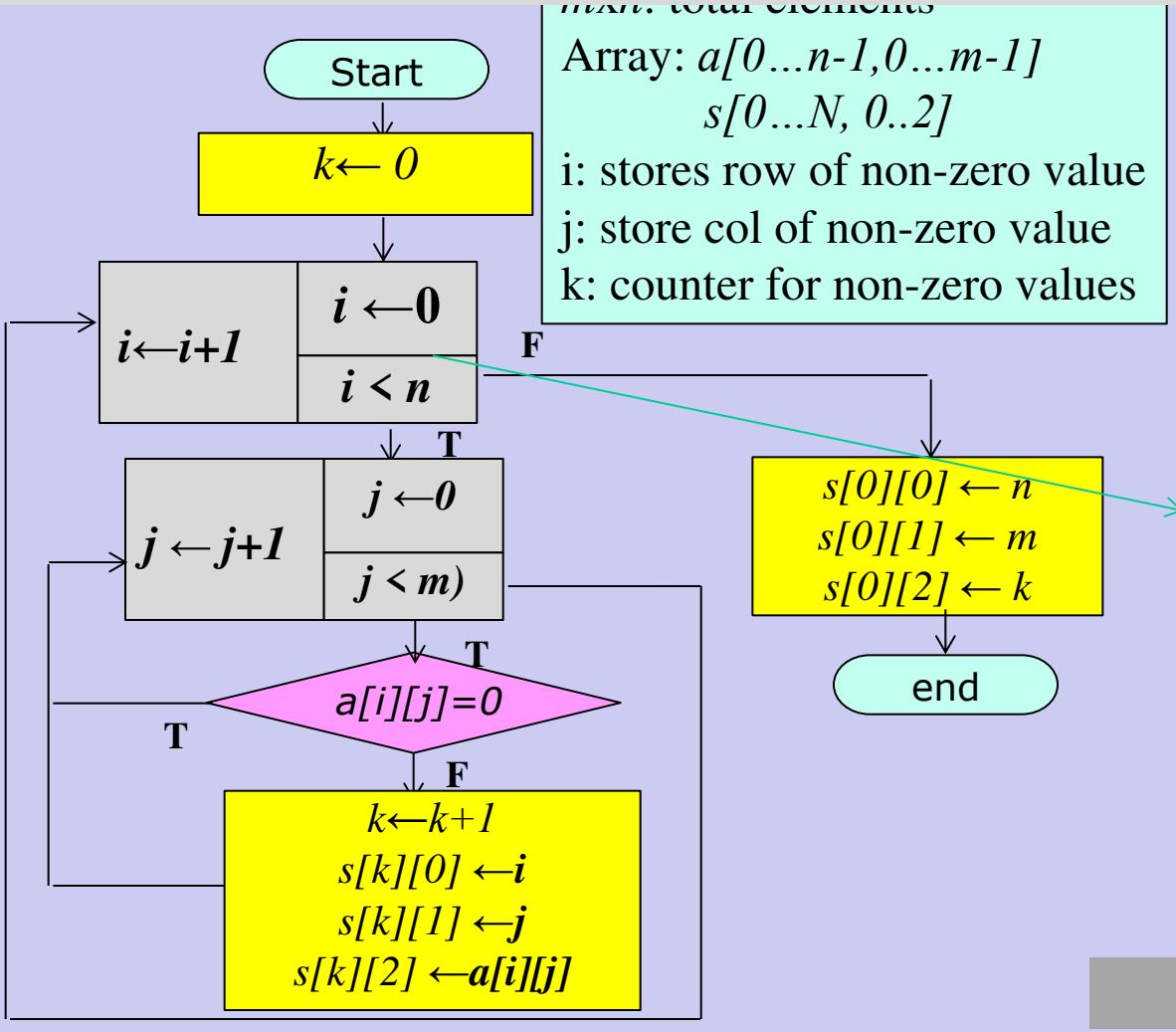


0	0	0	0	9	0
0	8	0	0	0	0
4	0	0	2	0	0
0	0	0	0	0	5
0	0	2	0	0	0

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



Array: *a[0…n-1,0…m-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

```
Int main(){
 int a[5][6], s[100][3];
 int k=0,n=5,m=6,i,j;
 for(i=0;i<n;i++){
    for(j=0;j<m;j++)
       if(a[i][j]!=0){
          k++;
          s[k][0]=I;
           s[k][1]=j;
           s[k][2]=a[i][j];
   }
}
s[0][0]=n;  s[0][1]=m; s[0][2]=k;
}
```

Corresponding C program

Searching

Search: locate an item in a list of data/information.

Two approaches will be discussed…

1. **Linear or Sequential Search:**
   - **Searches sequentially for an element.**
   - **Starts from the first element.**
2. **Binary Search:**
   - **Searches an element by dividing the sorted elements in a list into two sub-list**
   - **Starts with the middle element.**

Linear Search

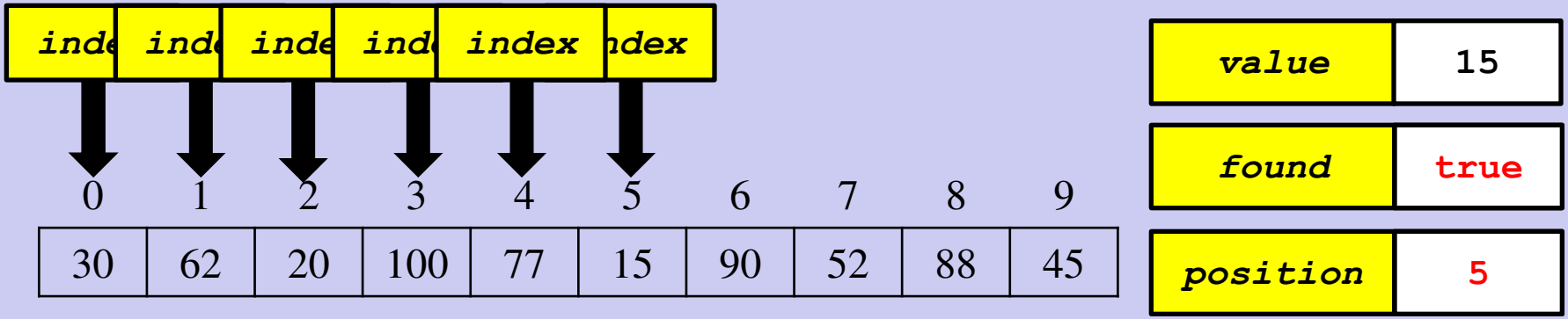
Algorithm:

Input: Array, #elements, item (to search)

Start with the element at index = 0

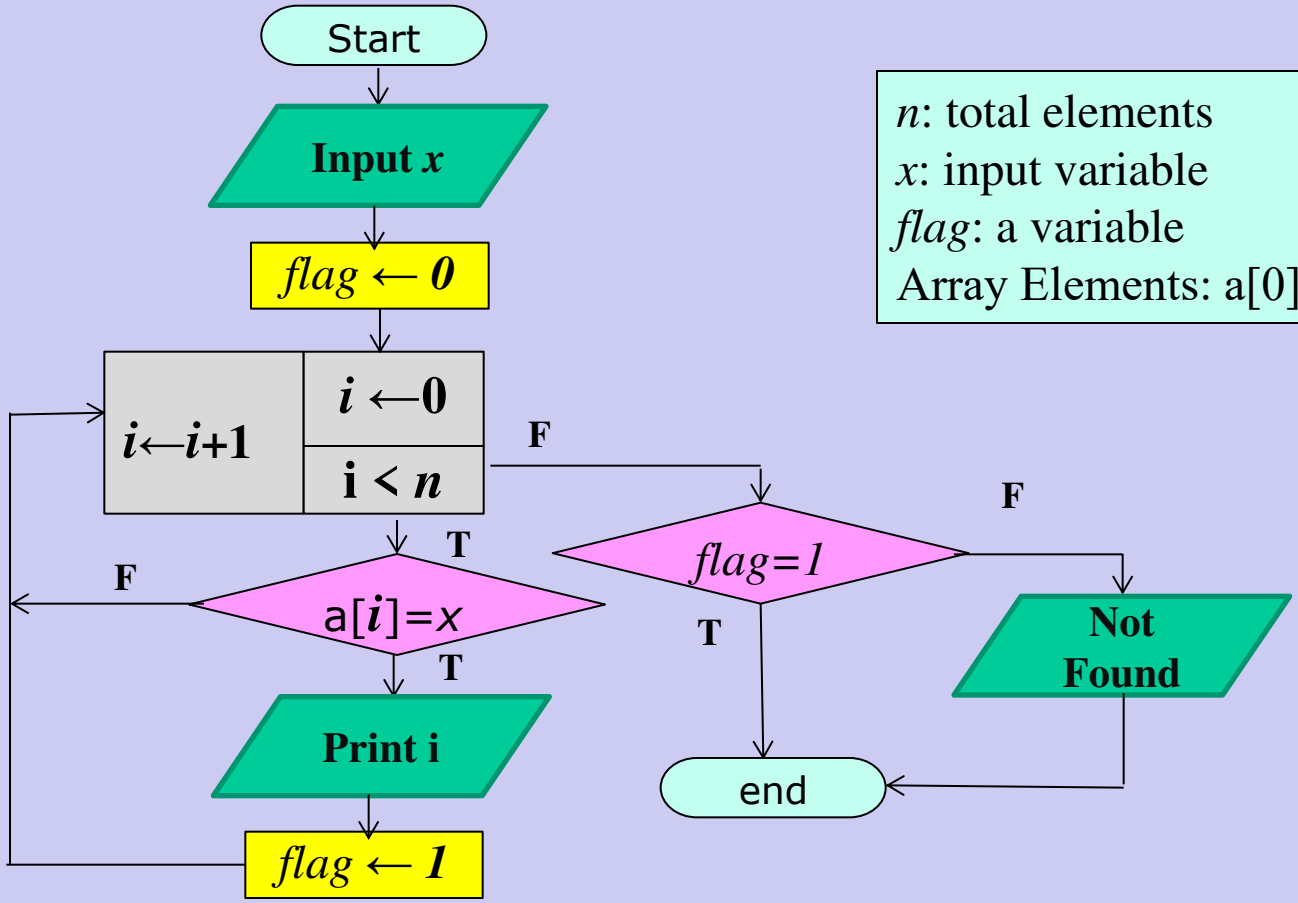
Step 1: Compare the element at *index* with item. If its equal to item then return *index* with status "Found" otherwise go to step 2.

Step 2: Increase index by 1. If index is less than #elements go to step 1 otherwise return -1 with status "Not found".



Linear Searching

Topic 3: Modifications of previous algorithm



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

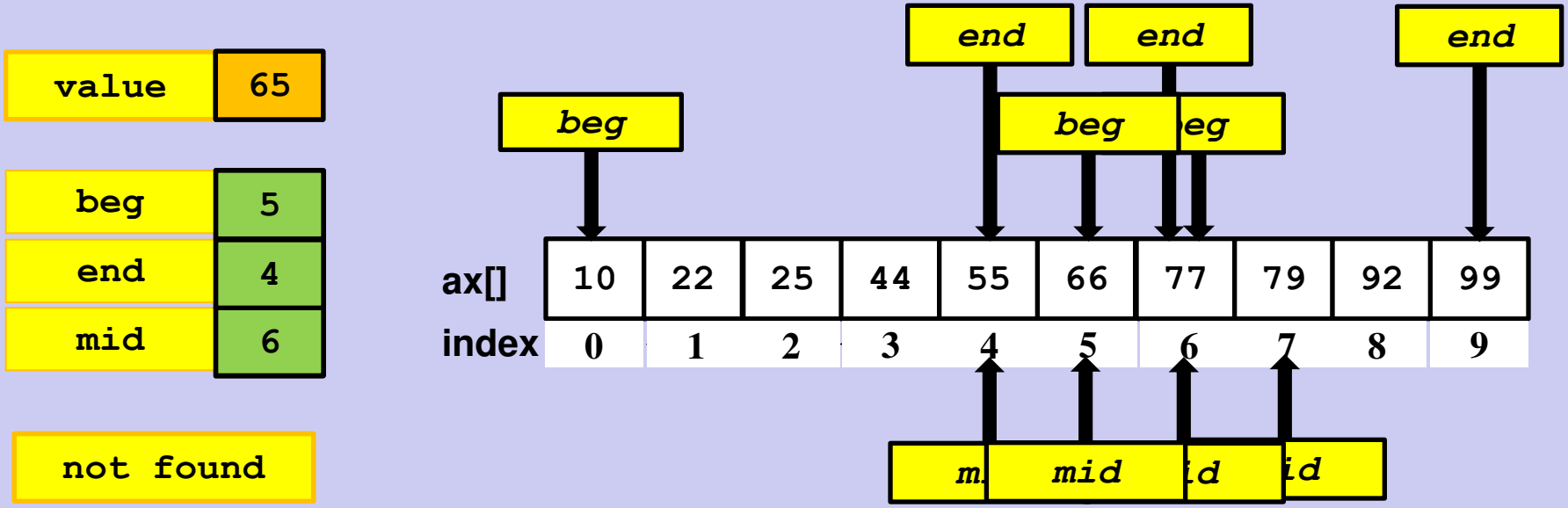
Complexity

O(1) for best-case

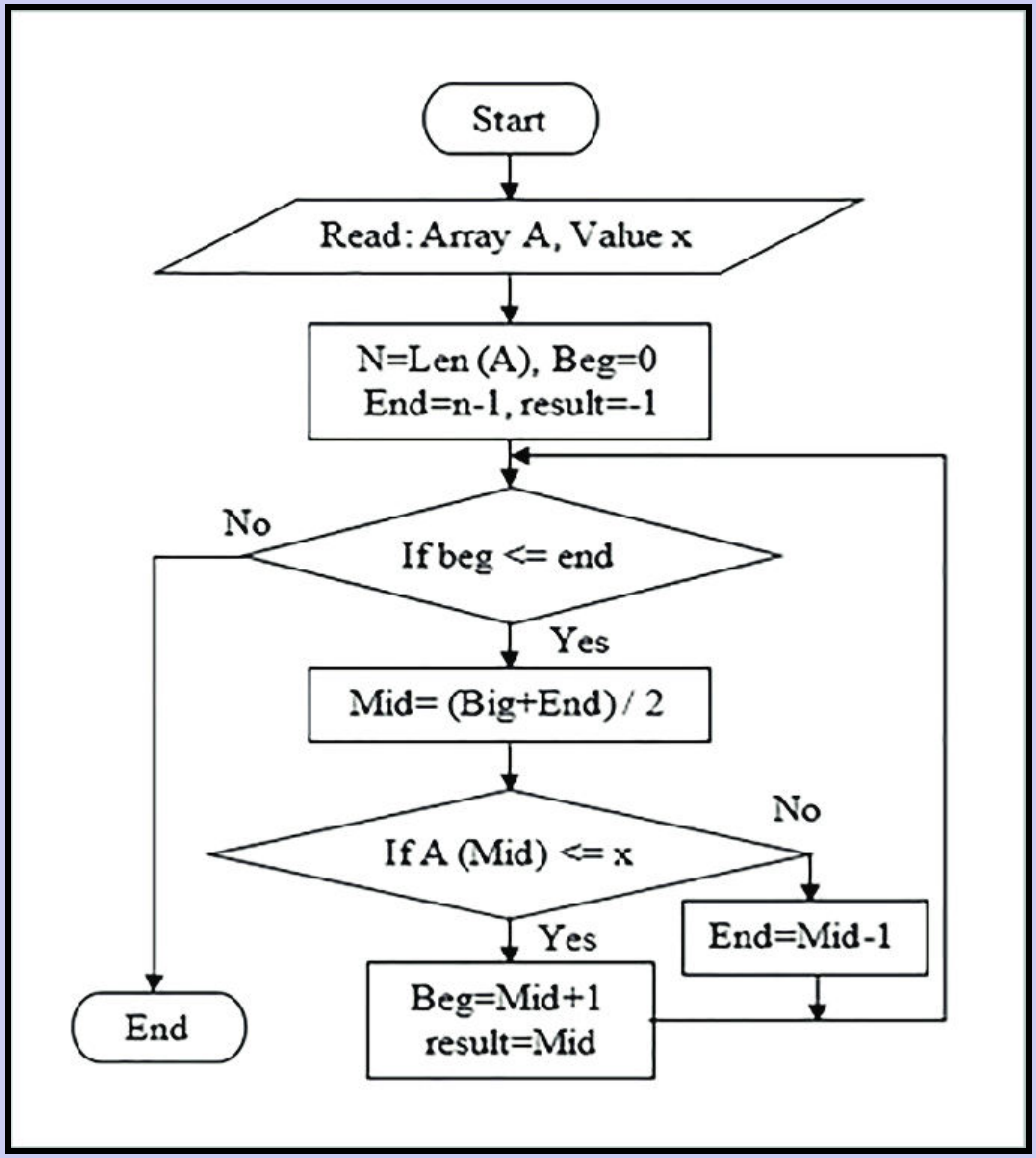
O(n) for worst-case

Binary Search

Suppose an array ax[] has 10 elements



Flowchart: Binary Search



At Iteration 1:

Length of array = ***n***

At Iteration 2:

Length of array = ***n/2***

At Iteration 3:

Length of array = ***(n/2)/2 = n/2^2***

Therefore, after Iteration k:

Length of array = ***n/2^k***

Also, we know that after **k iterations**, the **length of the array becomes 1** Therefore, the Length of the array

$n/2^k = 1$

$\Rightarrow n = 2^k$

Applying log function on both sides:

$\Rightarrow \log_2 n = \log_2 2^k$

$\Rightarrow \log_2 n = k * \log_2 2$

As $(\log_a (a) = 1)$ *Therefore,* $k = \log_2(n)$

Time Complexity=O($\log_2 n$)

Binary Search Program

```
#include <iostream>

using namespace std;

int binarySearch(int array[], int x, int low, int high) {

// Repeat until the pointers low and high meet each other

  while (low <= high) {

    int mid = low + (high - low) / 2;

    if (array[mid] == x)

      return mid;

    if (array[mid] < x)

      low = mid + 1;

    else

      high = mid - 1;

  }

  return -1;

}
```

```
int main(void) {

  int array[] = {3, 4, 5, 6, 7, 8, 9};

  int x = 7;

  //int n = sizeof(array) / sizeof(array[0]);

  n=7;

  int result = binarySearch(array, x, 0, n - 1);

  if (result == -1)

    printf("Not found");

  else

    printf("Element is found at index %d", result);

}
```

Quiz Time

Let's have some fun!

Q1： An array is what kind of data structure?

A Linear

B Non-linear

C Sorted

D Sequential

30

Q2: If a new element is needed to be stored at index 5 of 100 elements array then how many elements should be shifted?

A 5

B 100

C 95

D 96

30

Q3: If an element is needed to be deleted with index 10 of 100 elements array then how many elements should be shifted?

A 10

B 90

C 89

D 110

30

Q4: How many times bubble up is repeated in Bubble Sort algorithm with 10 elements?

A 10

B 11

C 9

D 12

30

Q5: The complexity of Bubble Sort algorithm is _____?

A O (n)

B O (n-1)

C O (2n)

D O (n^2)

30

Q6: The sparse matrix has more _____ values?

A zero

B non-zero

C positive

D negative

30

Q7: Which searching can be performed with unsorted data?

A Binary Search

B Linear Search

C

D

30

Q9: The complexity of Binary search is _____?

A. O(n)

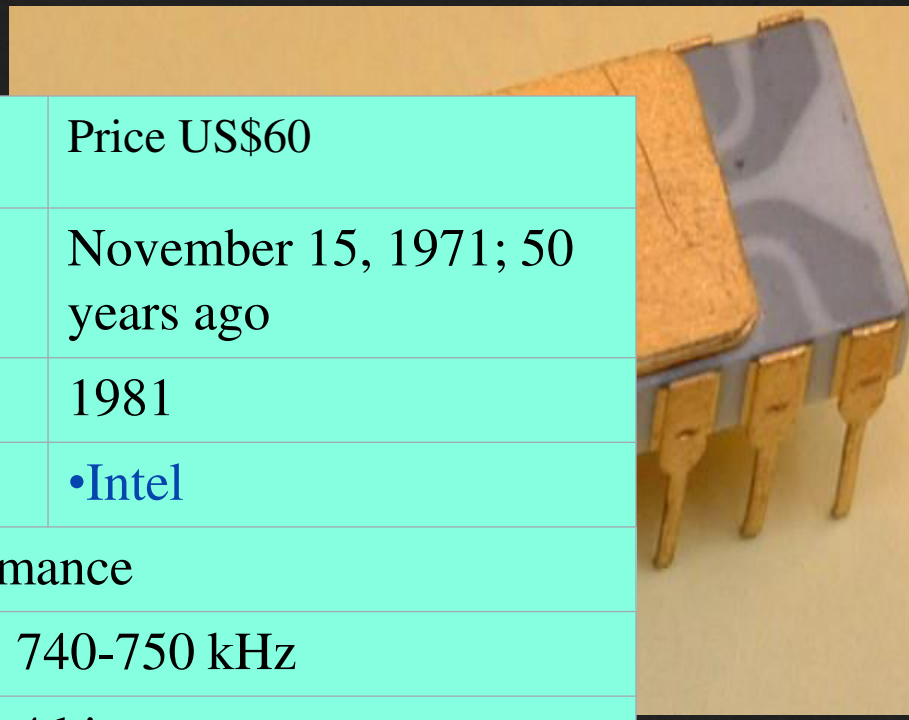
B. $O(2^n)$

C. $O(n^2)$

D. $O(\log_2 n)$

30

Q10: Identify the following _____.?



First Commercial Processor	Price US$60
Launched	November 15, 1971; 50 years ago
Discontinued	1981
Common manufacturer(s)	•Intel
Performance	
Max. CPU clock rate	740-750 kHz
Data width	4 bits
Address width	12 bits (multiplexed)

A G

B M

C R

D Di

30

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