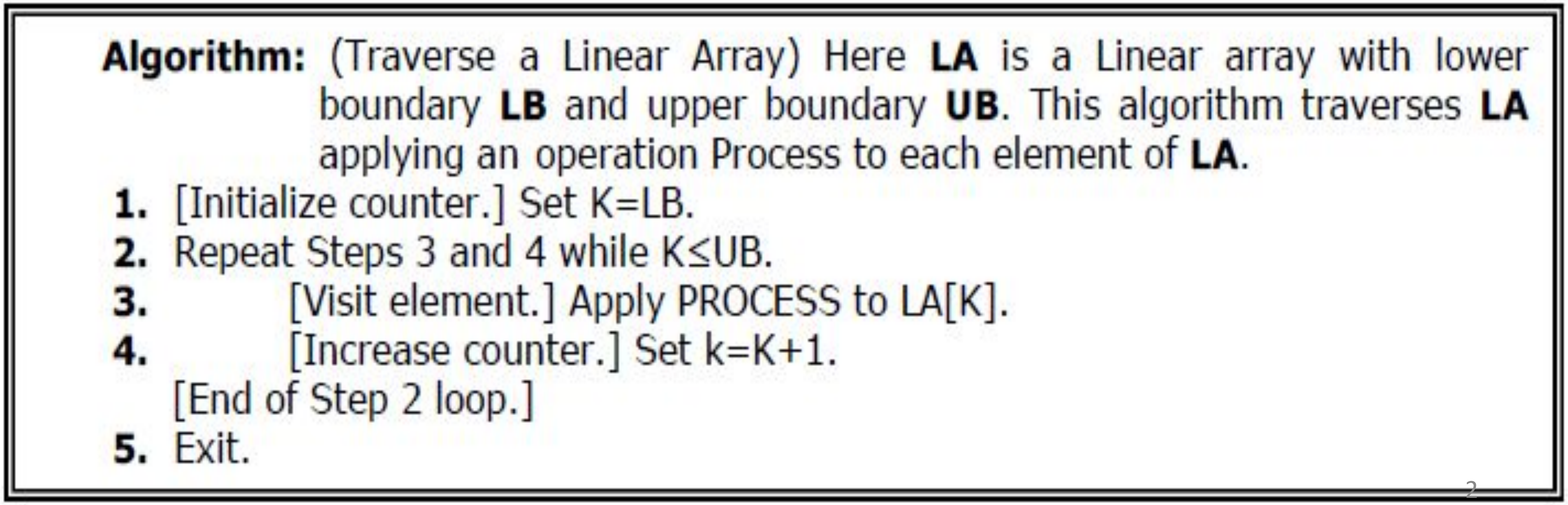
Structured Programming
CSE 103

Professor Dr. Mohammad Abu Yousuf

Traversing in Linear Array

- It means processing or visiting each element in the array exactly once;
- Let ‘A’ is an array stored in the computer’s memory. If we want to display the contents of *‘A’, it has to be traversed i.e. by accessing and processing each element* of *‘A’ exactly once.*



Algorithm: (Traverse a Linear Array) Here **LA** is a Linear array with lower boundary **LB** and upper boundary **UB**. This algorithm traverses **LA** applying an operation Process to each element of **LA**.

```
1. [Initialize counter.] Set K=LB.
2. Repeat Steps 3 and 4 while K≤UB.
3.         [Visit element.] Apply PROCESS to LA[K].
4.         [Increase counter.] Set k=K+1.
   [End of Step 2 loop.]
5. Exit.
```

Sorting in Linear Array

- Sorting an array is the ordering the array elements in ascending (increasing - from min to max) or descending (decreasing – from max to min) order.

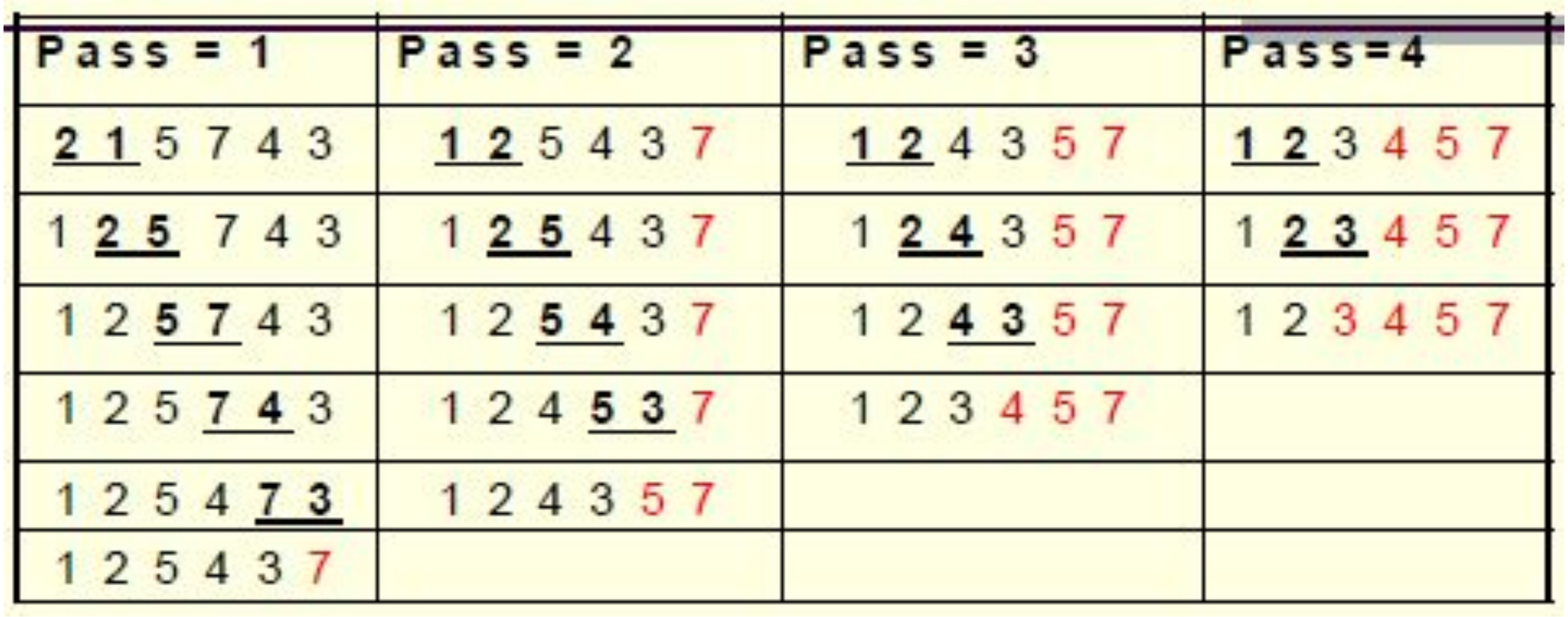
- **Example:**

  {2 1 5 7 4 3} →{1, 2, 3, 4, 5,7} ***ascending order***

  {2 1 5 7 4 3} →{7,5, 4, 3, 2, 1} ***descending order***

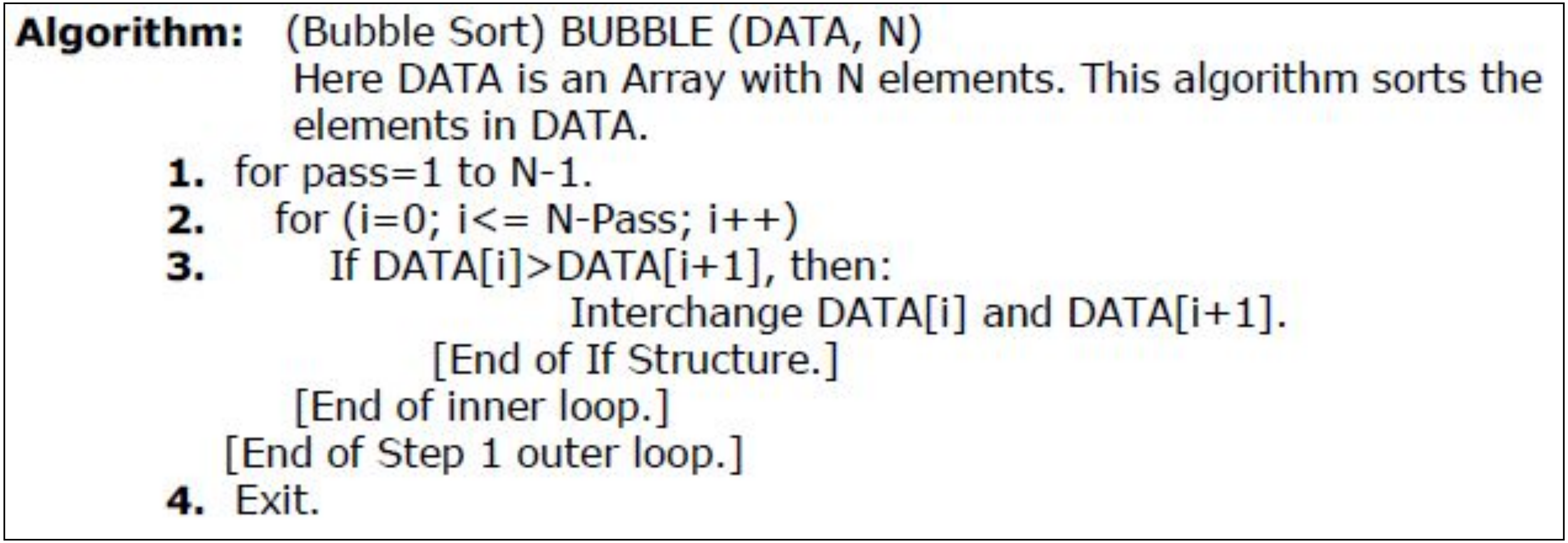
Bubble Sort

- Example:
- This sorting algorithm is comparison based algorithm in which each pair of adjacent elements is compared and elements are swapped if they are not in order.



Pass = 1	Pass = 2	Pass = 3	Pass=4
2 1 5 7 4 3	**1 2** 5 4 3 7	**1 2** 4 3 5 7	**1 2** 3 4 5 7
1 **2 5** 7 4 3	1 **2 5** 4 3 7	1 **2 4** 3 5 7	1 **2 3** 4 5 7
1 2 **5 7** 4 3	1 2 **5 4** 3 7	1 2 **4 3** 5 7	1 2 3 4 5 7
1 2 5 **7 4** 3	1 2 4 **5 3** 7	1 2 3 4 5 7	
1 2 5 4 **7 3**	1 2 4 3 5 7		
1 2 5 4 3 7			

Bubble Sort



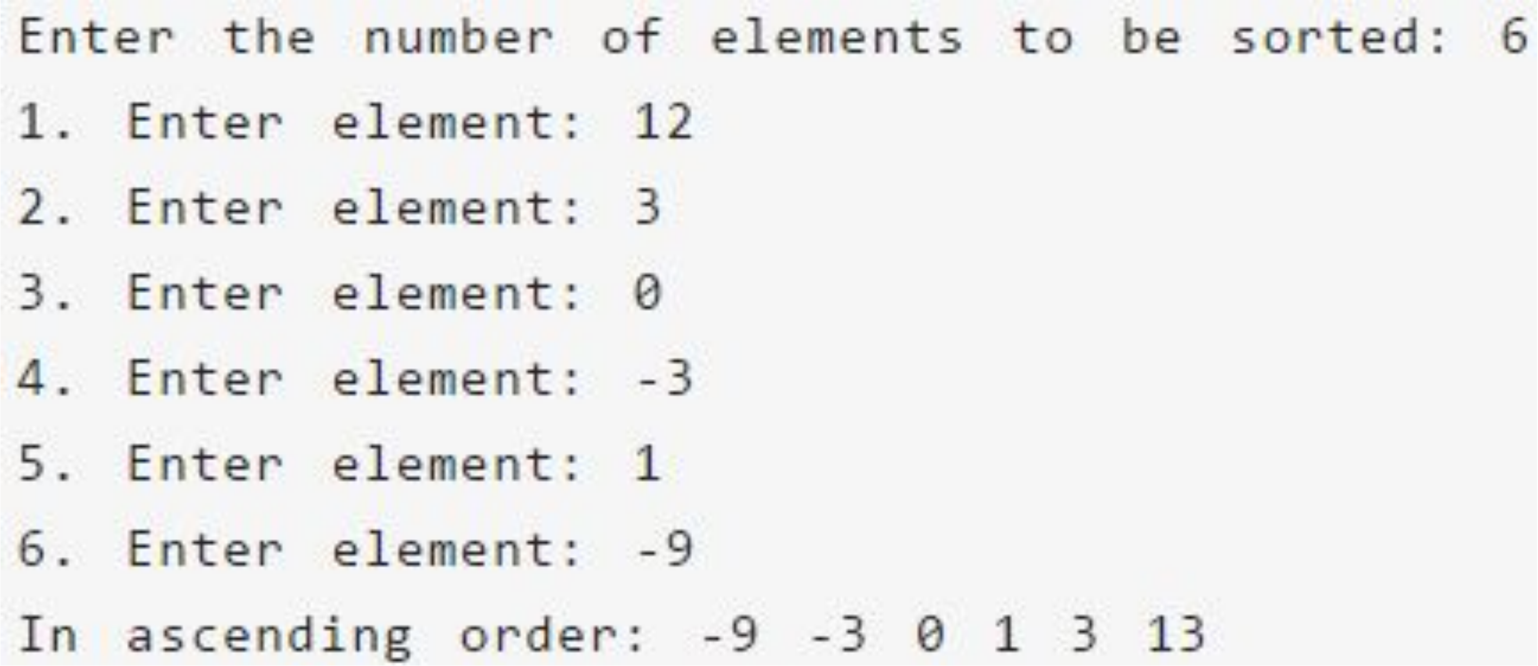
```
Algorithm:   (Bubble Sort) BUBBLE (DATA, N)
              Here DATA is an Array with N elements. This algorithm sorts the
              elements in DATA.
      1.  for pass=1 to N-1.
      2.     for (i=0; i<= N-Pass; i++)
      3.         If DATA[i]>DATA[i+1], then:
                         Interchange DATA[i] and DATA[i+1].
                   [End of If Structure.]
              [End of inner loop.]
          [End of Step 1 outer loop.]
      4.  Exit.
```

Bubble sort

```c
#include <stdio.h>
int main()
{
    int data[100],i,n,step,temp;
    printf("Enter the number of elements to be sorted: ");
    scanf("%d",&n);
    for(i=0;i<n;++i)
    {   printf("%d. Enter element: ",i+1);
        scanf("%d",&data[i]);
    }
    for(step=1;step<n;++step)
    for(i=0;i<n-step;++i)
    {
        if(data[i]>data[i+1])
        {
            temp=data[i];
            data[i]=data[i+1];
            data[i+1]=temp;
        }
    }
    printf("In ascending order: ");
    for(i=0;i<n;++i)
         printf("%d  ",data[i]);
    return 0;
```

Bubble Sort

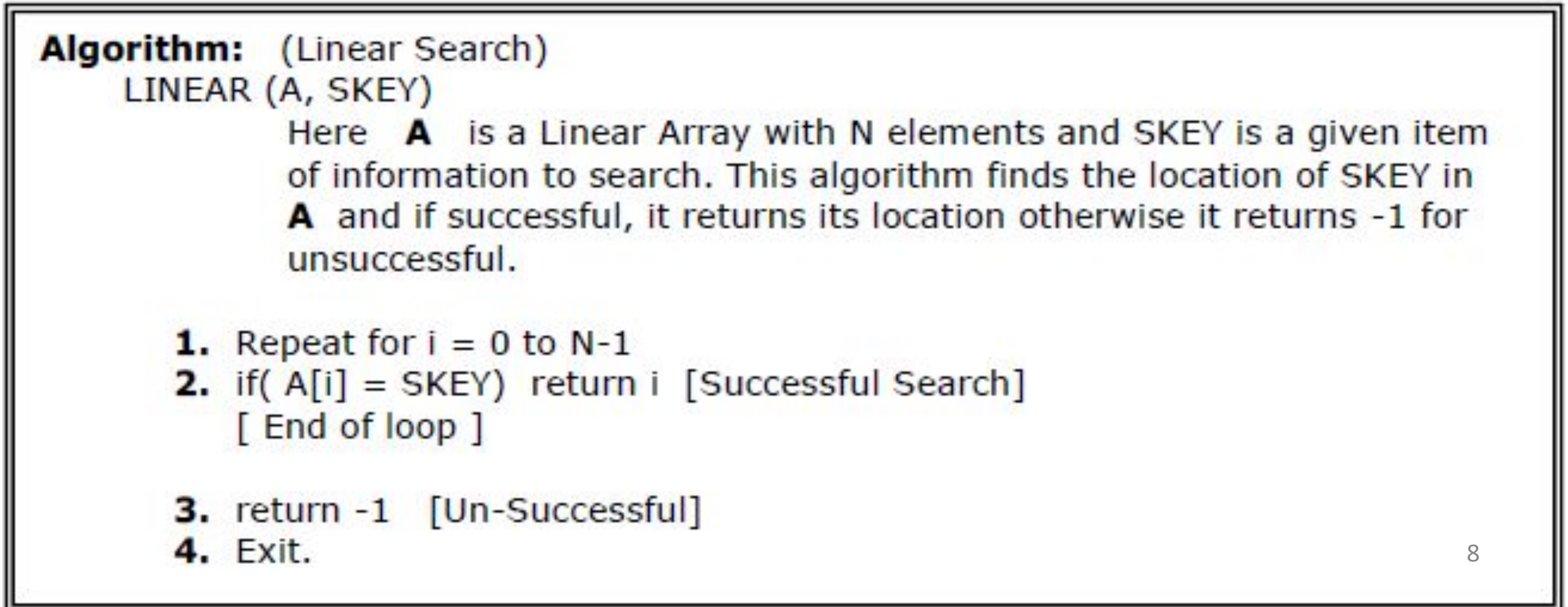
- Output of previous program:



```
Enter the number of elements to be sorted: 6
1. Enter element: 12
2. Enter element: 3
3. Enter element: 0
4. Enter element: -3
5. Enter element: 1
6. Enter element: -9
In ascending order: -9 -3 0 1 3 13
```

Searching in Linear Array

- **Linear Search:**
- The linear search compares each element of the array with the ***search key*** until the search key is found. To determine that a value is not in the array, the program must compare the search key to every element in the array.



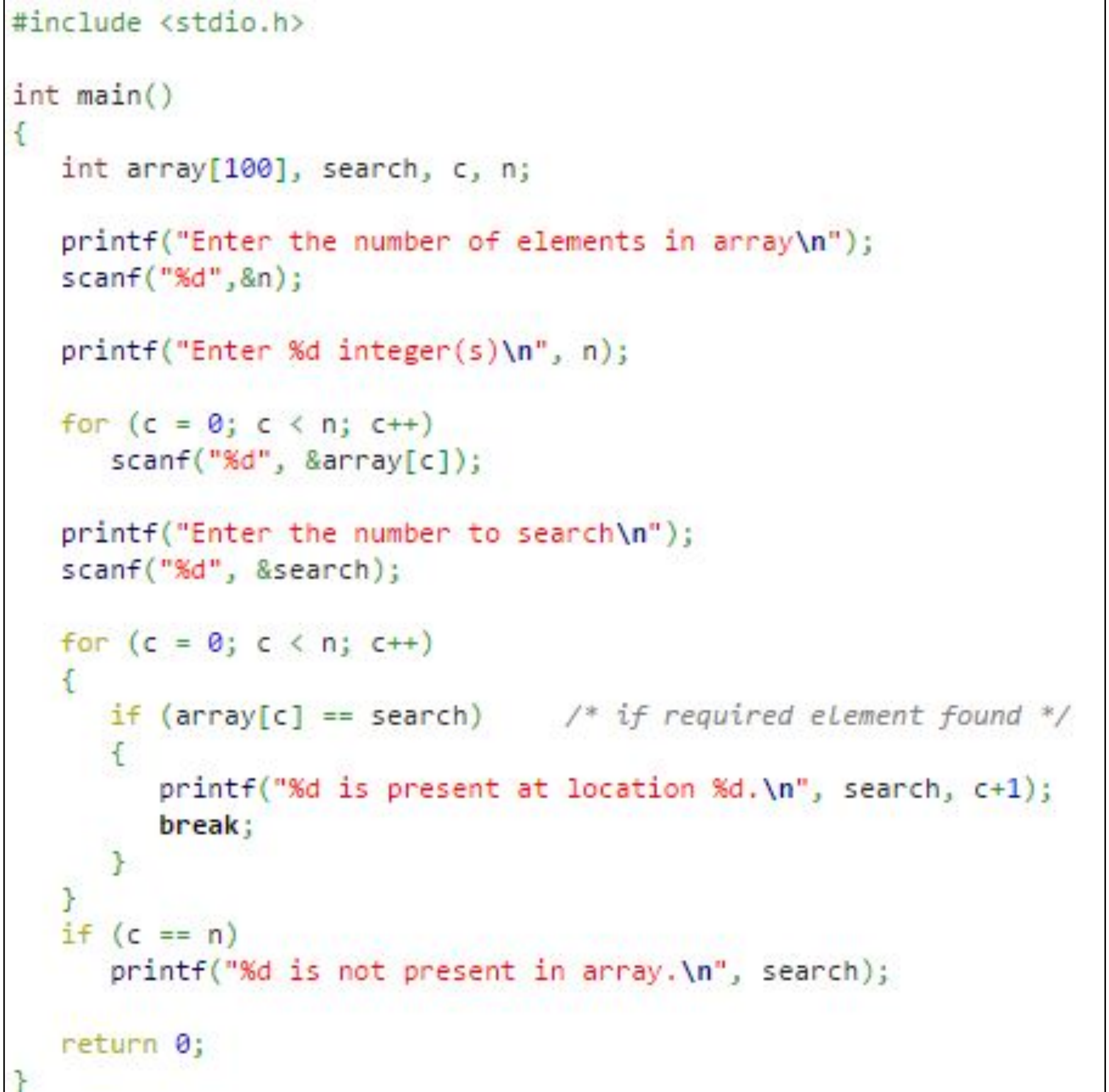
Algorithm: (Linear Search)

LINEAR (A, SKEY)

Here **A** is a Linear Array with N elements and SKEY is a given item of information to search. This algorithm finds the location of SKEY in **A** and if successful, it returns its location otherwise it returns -1 for unsuccessful.

```
1. Repeat for i = 0 to N-1
2. if( A[i] = SKEY)  return i  [Successful Search]
   [ End of loop ]

3. return -1   [Un-Successful]
4. Exit.
```

Linear search



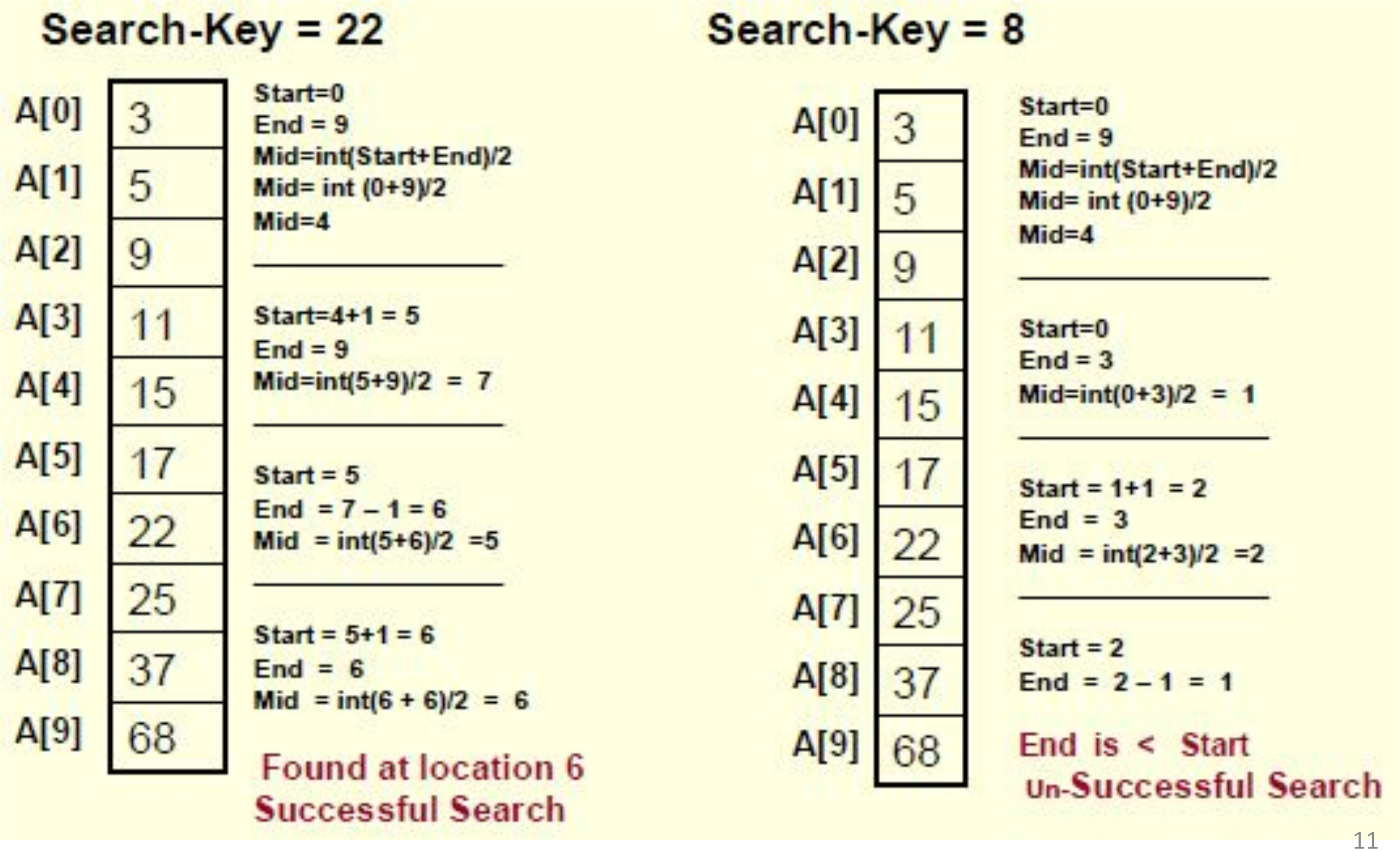
```c
#include <stdio.h>

int main()
{
   int array[100], search, c, n;

   printf("Enter the number of elements in array\n");
   scanf("%d",&n);

   printf("Enter %d integer(s)\n", n);

   for (c = 0; c < n; c++)
      scanf("%d", &array[c]);

   printf("Enter the number to search\n");
   scanf("%d", &search);

   for (c = 0; c < n; c++)
   {
      if (array[c] == search)     /* if required element found */
      {
         printf("%d is present at location %d.\n", search, c+1);
         break;
      }
   }
   if (c == n)
      printf("%d is not present in array.\n", search);

   return 0;
}
```

Binary Search

- It is useful for the large sorted arrays. The binary search algorithm can only be used with sorted array and eliminates one half of the elements in the array being searched after each comparison.

Binary Search

- **Example:**



Search-Key = 22

A[0]	3
A[1]	5
A[2]	9
A[3]	11
A[4]	15
A[5]	17
A[6]	22
A[7]	25
A[8]	37
A[9]	68

Start=0
End = 9
Mid=int(Start+End)/2
Mid= int (0+9)/2
Mid=4

Start=4+1 = 5
End = 9
Mid=int(5+9)/2 = 7

Start = 5
End = 7 – 1 = 6
Mid = int(5+6)/2 =5

Start = 5+1 = 6
End = 6
Mid = int(6 + 6)/2 = 6

Found at location 6
Successful Search

Search-Key = 8

A[0]	3
A[1]	5
A[2]	9
A[3]	11
A[4]	15
A[5]	17
A[6]	22
A[7]	25
A[8]	37
A[9]	68

Start=0
End = 9
Mid=int(Start+End)/2
Mid= int (0+9)/2
Mid=4

Start=0
End = 3
Mid=int(0+3)/2 = 1

Start = 1+1 = 2
End = 3
Mid = int(2+3)/2 =2

Start = 2
End = 2 – 1 = 1

End is < Start
Un-Successful Search

```c
// Binary Search in C

#include <stdio.h>

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

  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 n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
```

Thank you