C Programming:

Cours 8:
ARRAYS

Dr. Chénche

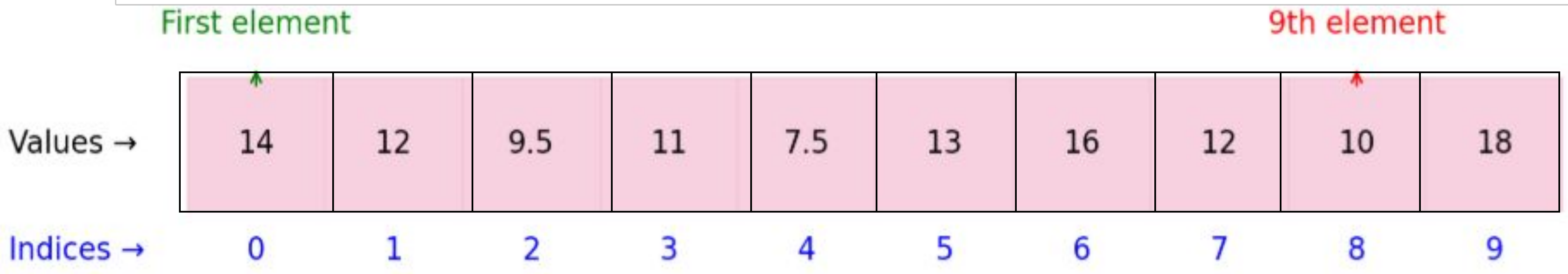
WHY DO WE USE ARRAYS ?

Write a program that allows to read the averages of all **100 students**, then determines how many of them are **above the class average**.

- To know whether a student's average is higher than the class average, we must first know the class average.
- But to calculate the class average, we need the sum of all the students’ averages.
- 1) Go through all the grades → calculate the sum → compute the class average
- 2) Go through them again → compare each grade with the class average

Definition-Arrays

- An array is a collection of several adjacent memory cells, called array elements, that are associated with a particular symbolic name.



- In C each array has: name, data type, size

One-Dimension Arrays

- **Declaration of one-dimension array**

Syntax:

```
atype  aname [ size ] ;
```

- **Uninitializing an array**

```
atype  aname [ size ] = { initialization list } ;
```

Where :

atype is any data type;

aname is the name given to the array;

size represents the number of elements in the array.

initialization list is the list of initial values given to the array.

Declaration of Arrays

Example 1 .

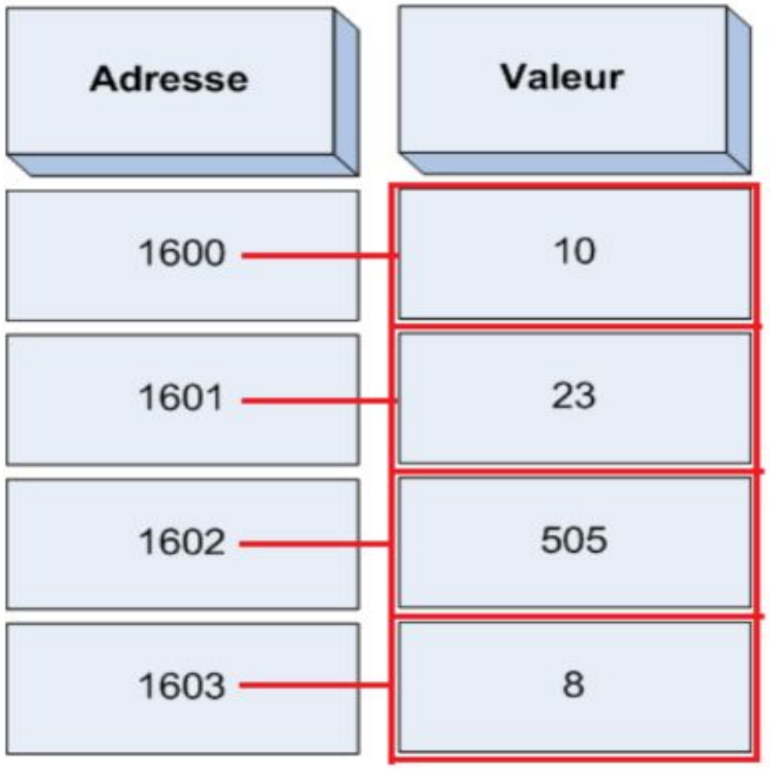
int y [4] ;

- This tells the compiler to associate **4 memory cells** with name x.

Example 2.

int y [4] ={10, 23, 505, 8};

- All elements of an array are of the same type. Thus, an array of int will contain only int values, and nothing else.



Declaration of Arrays

- More than one array can be declared on a line

```
int age [10] , height [10] , names [20] ;
```

- Mix declaration of variables with declaration of arrays

```
int i , j , age [10] ;
```

Initializing an Array

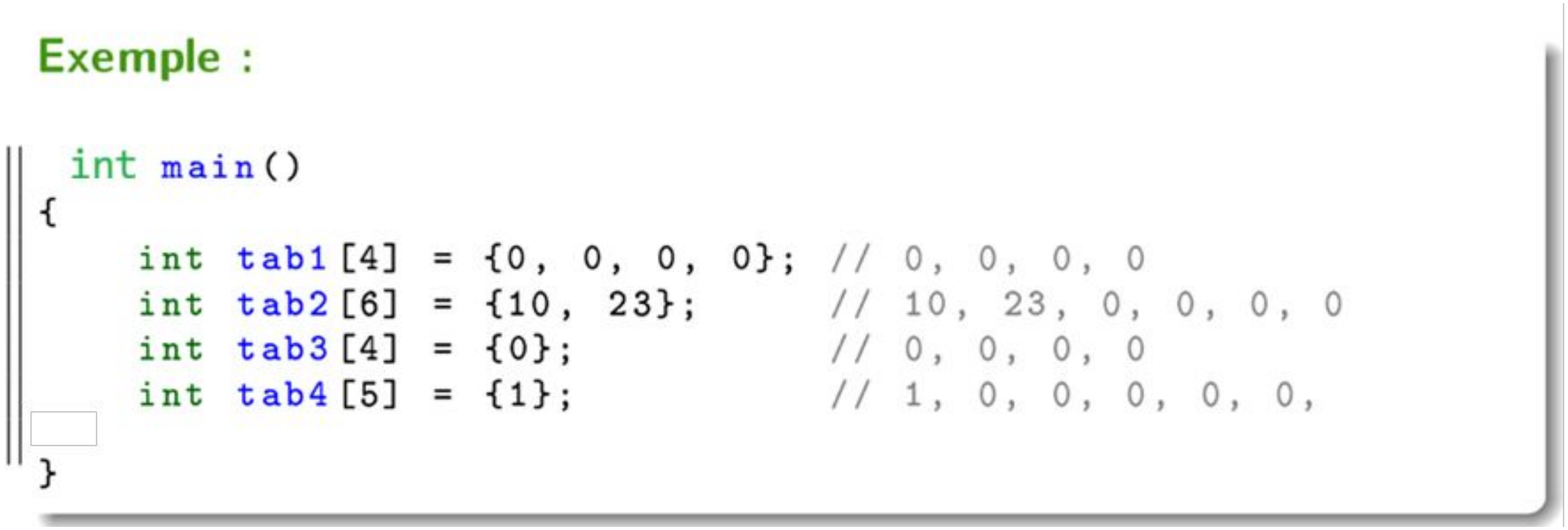
Example 3:

```
int age [ 10 ] = { 0,0,0,0,0,0,0,0,0,0 } ;

int age[ 10 ] = { 0 } ;
```

0	0	0	0	0	0	0	0	0	0

Initializing an Array



Exemple :

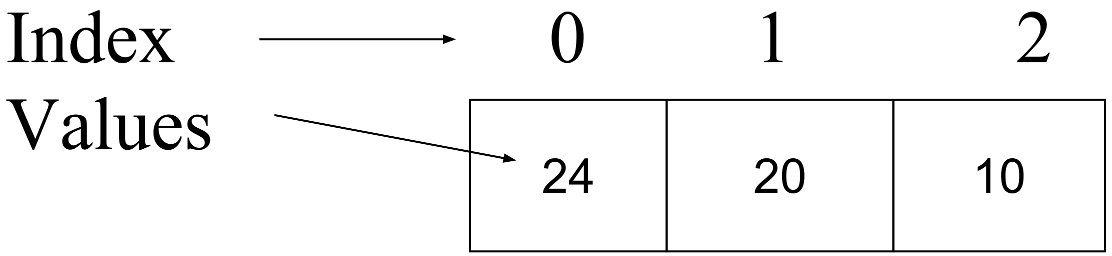
```
 int main()
{
    int tab1[4] = {0, 0, 0, 0}; // 0, 0, 0, 0
    int tab2[6] = {10, 23};     // 10, 23, 0, 0, 0, 0
    int tab3[4] = {0};          // 0, 0, 0, 0
    int tab4[5] = {1};          // 1, 0, 0, 0, 0, 0,

}
```

Attention: In the array tab4, not all elements are initialized **to 1**: only the first element will be **1**, and all the others will be **0**.

Accessing elements in One-Dimensional Array

Array x in memory:



Position (Rank) = index +1

Accessing the array:

x [0] to access the first element of x

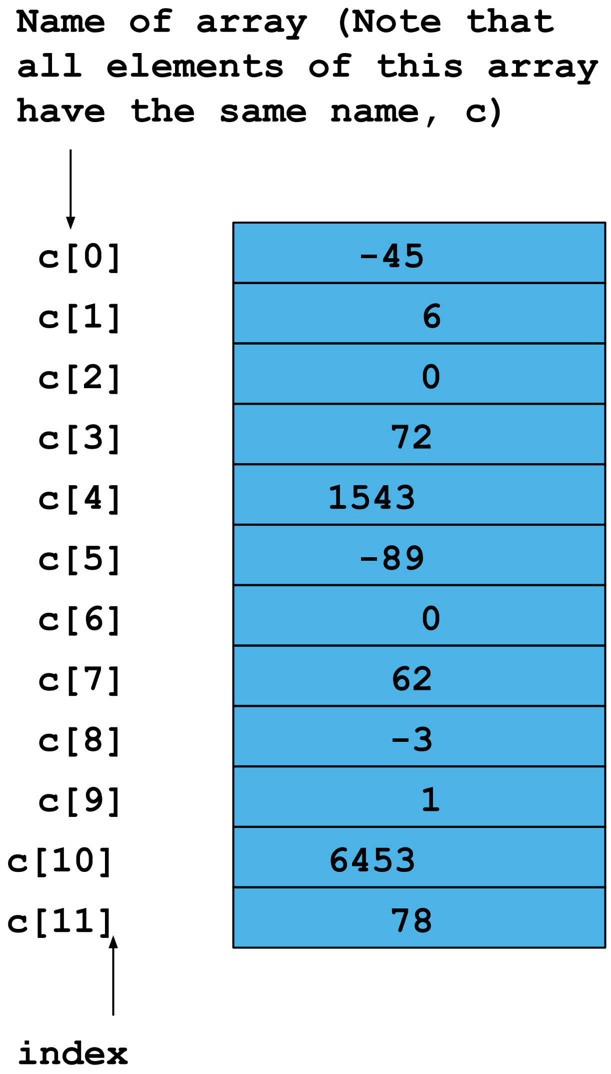
x [1] to access the second element of x

x [2] to access the third element of x

Storage of an array in memory

- The size =…..
- The index goes from … until ….
- The position 5 is the index
- The element number four is c[…]

and its value is ……



Accessing elements in One-Dimensional Array

How to process the data stored in an array?

Syntax:

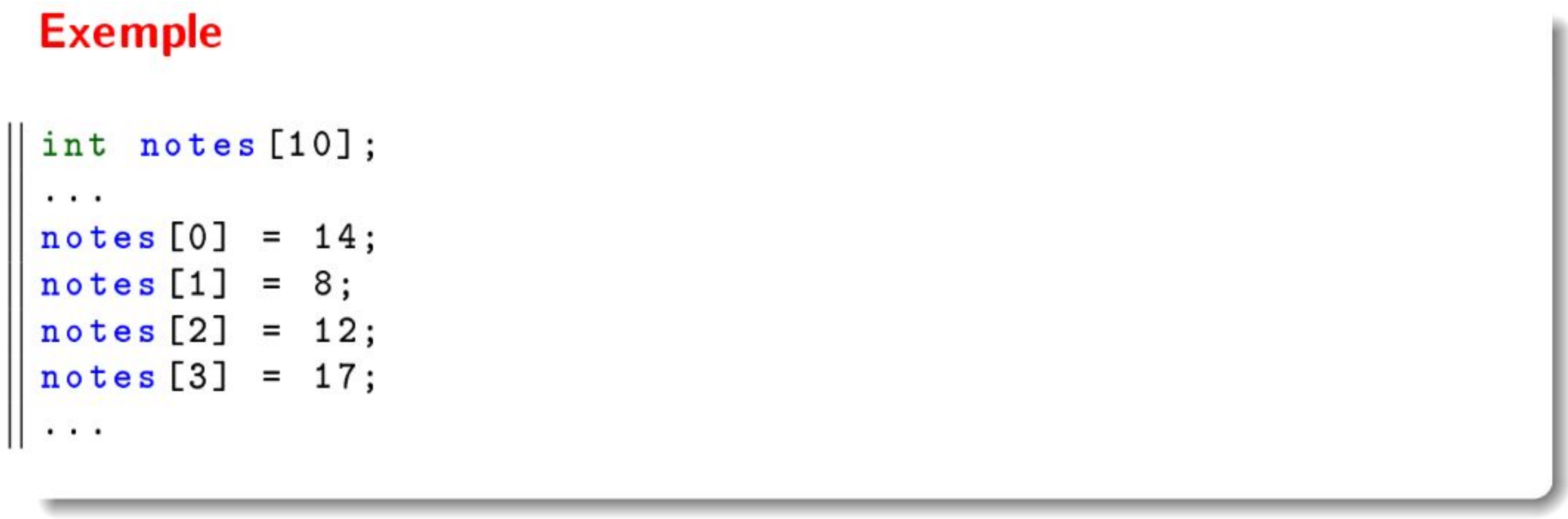
aname [**index**]

- **index** is the **subscript** that is used to reference the desired element.

The array **index** starts from **0** until the fixed **size -1**.

Fill in an array :

If we want to assign values to the elements of an array, we can use direct assignment:

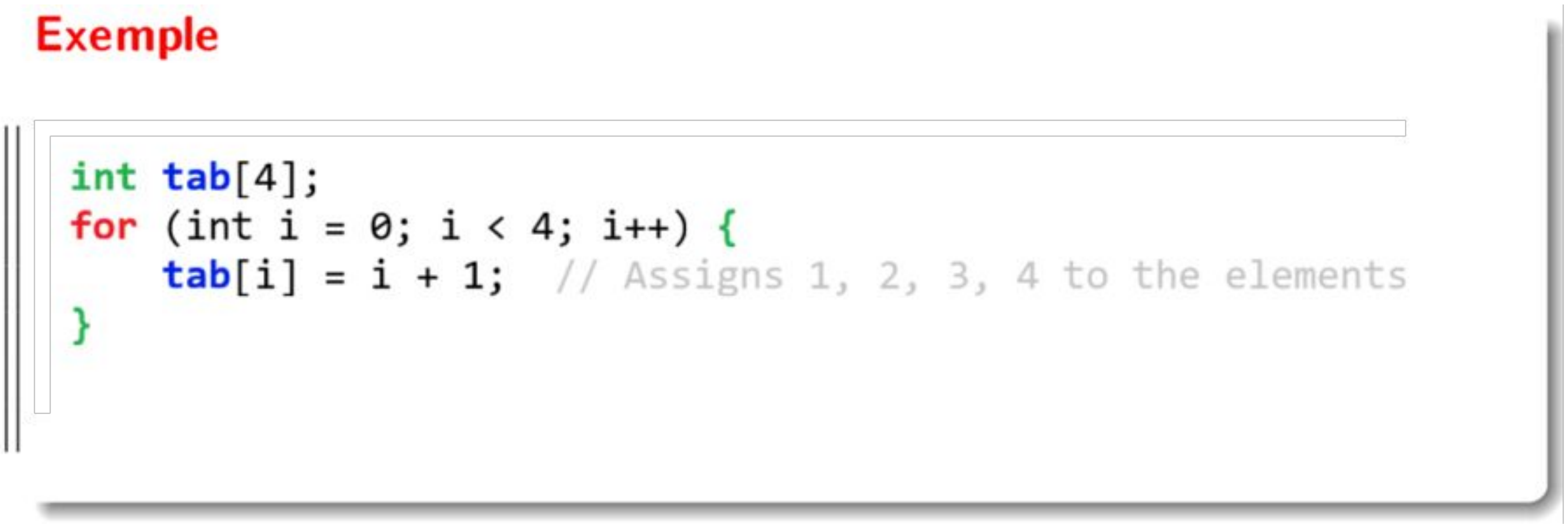


Exemple

```
int notes[10];
...
notes[0] = 14;
notes[1] = 8;
notes[2] = 12;
notes[3] = 17;
...
```

Fill in an array :

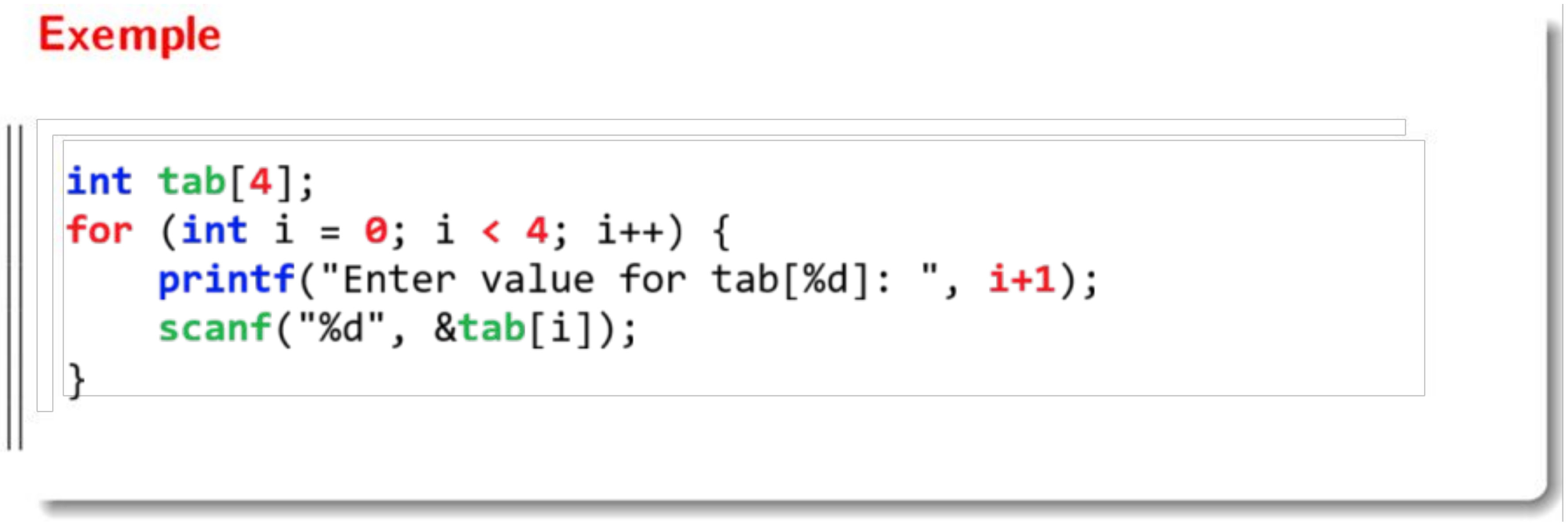
- Common method to assign values to an array is using a **for loop**.


Exemple

```
int tab[4];
for (int i = 0; i < 4; i++) {
    tab[i] = i + 1;  // Assigns 1, 2, 3, 4 to the elements
}
```

Fill in an array :

- Common method to assign values to an array is using a **for loop**.

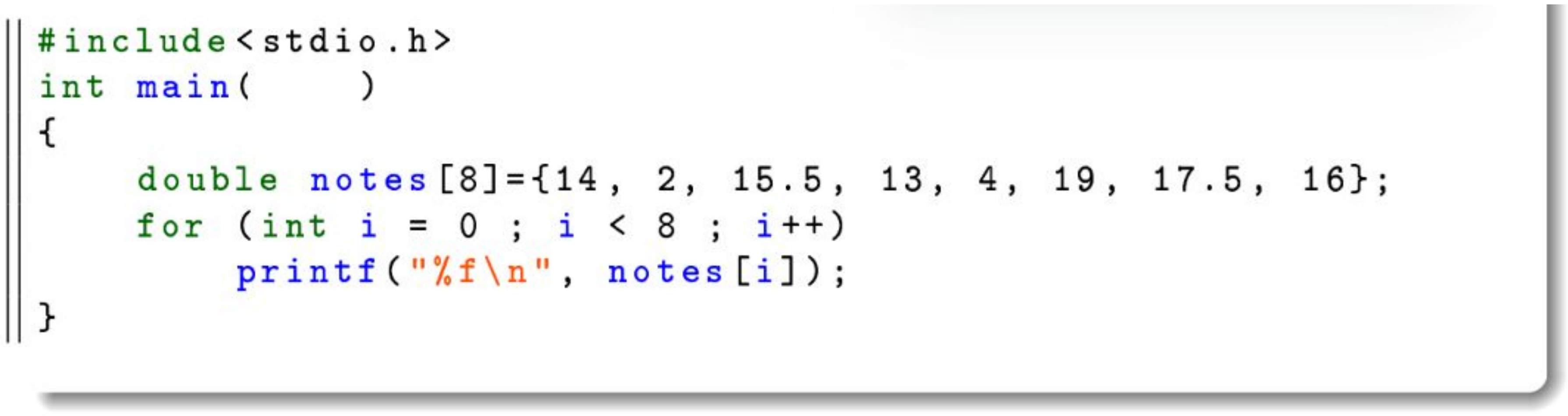


Exemple

```
int tab[4];
for (int i = 0; i < 4; i++) {
    printf("Enter value for tab[%d]: ", i+1);
    scanf("%d", &tab[i]);
}
```

Traverse an array :

- To traverse an array" means to visit each element of the array, one by one, usually to print or modify its value



```
#include<stdio.h>
int main(    )
{
    double notes[8]={14, 2, 15.5, 13, 4, 19, 17.5, 16};
    for (int i = 0 ; i < 8 ; i++)
        printf("%f\n", notes[i]);
}
```

Searching

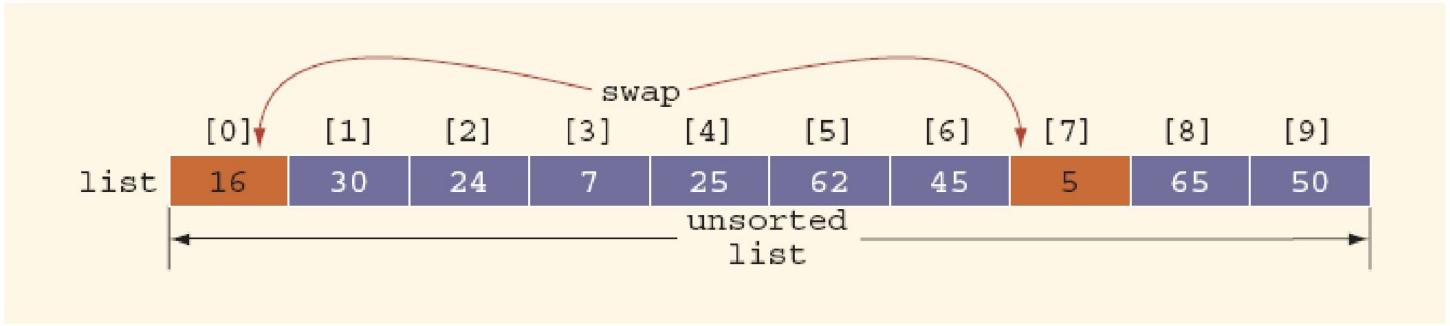
- To search an array, you need:
  - The array contents
  - Array length
  - Item to be found
- After the search is completed
  - If found:
    - Report “success”
    - Location where the item was found
  - If not found, report “failure”

Sorting

- Common problem: sort an array of values, starting from lowest to highest (resp. Highest to lowest).
  - List of exam scores
  - Words of dictionary in alphabetical order
  - Students names listed alphabetically
  - Student records sorted by ID#

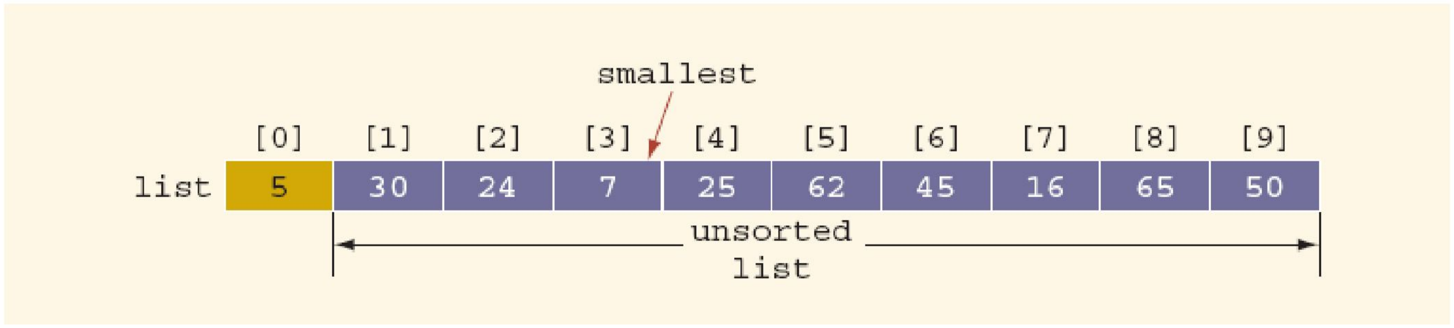
Selection Sort

- Selection sort: rearrange array by selecting an element and moving it to its proper position
- Find the smallest (or largest) element and move it to the beginning (end) of the list



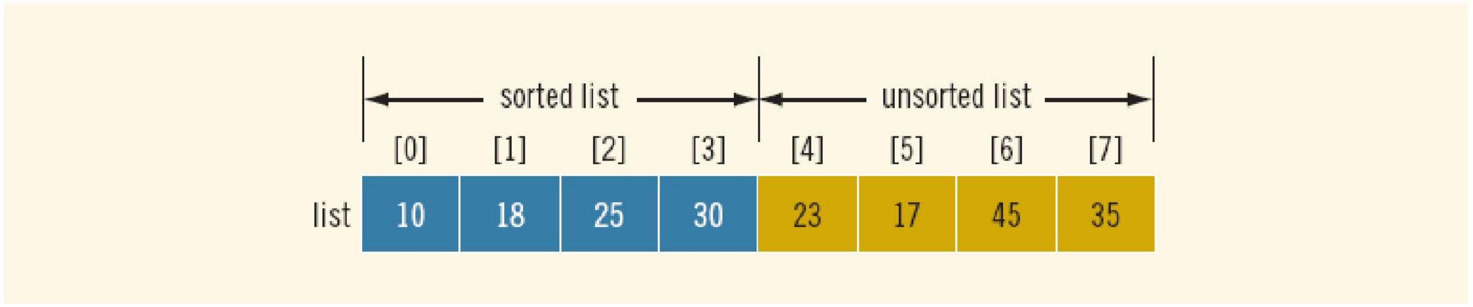
Selection Sort

- On successive passes, locate the smallest item in the array starting from the next element

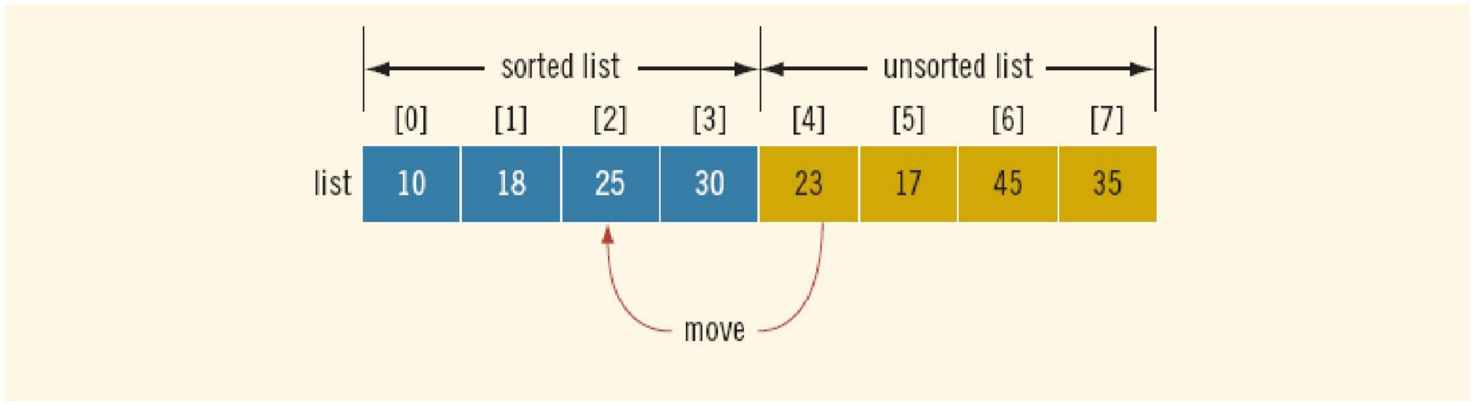


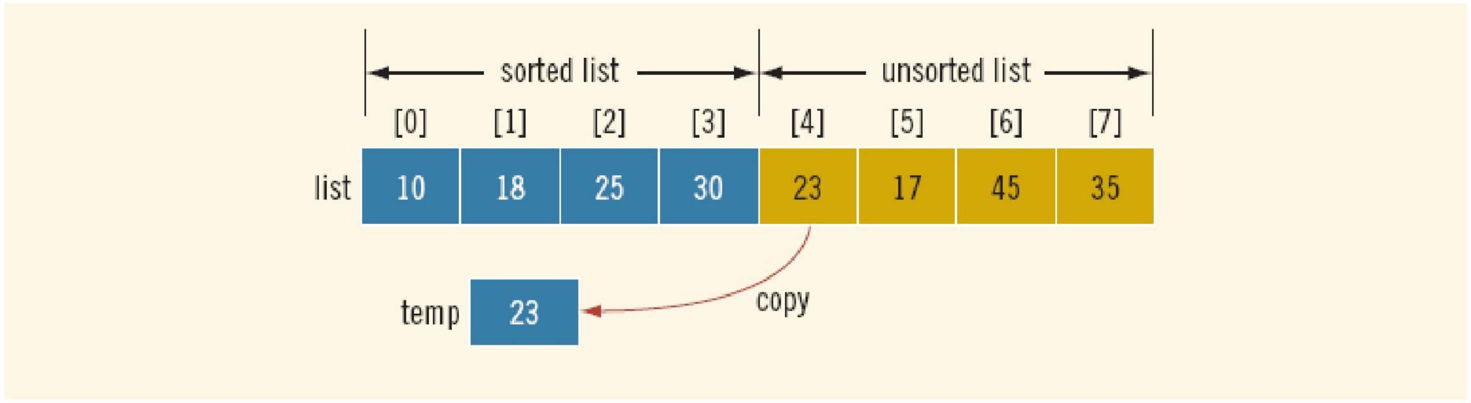
Insertion Sort

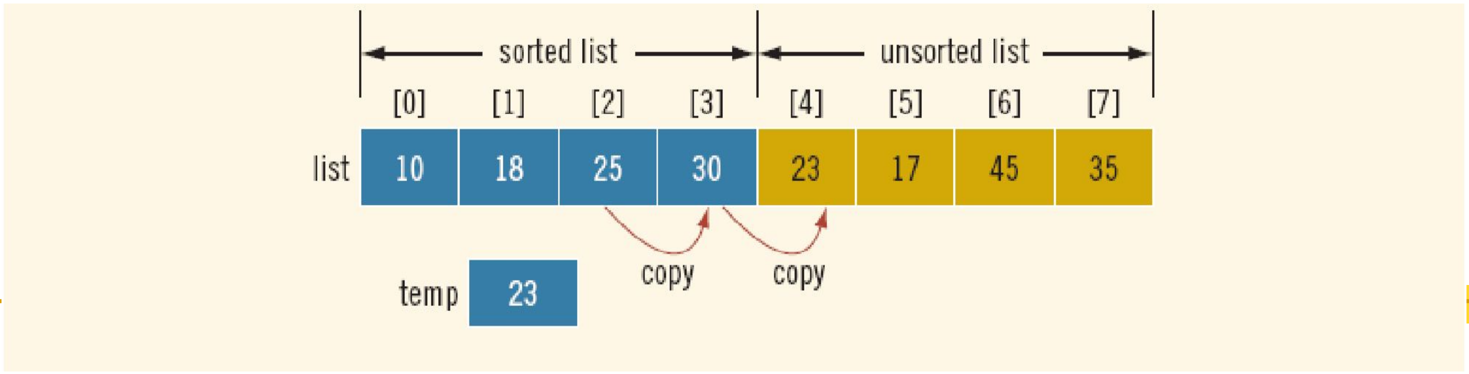
- The insertion sort algorithm sorts the array by moving each element to its proper place

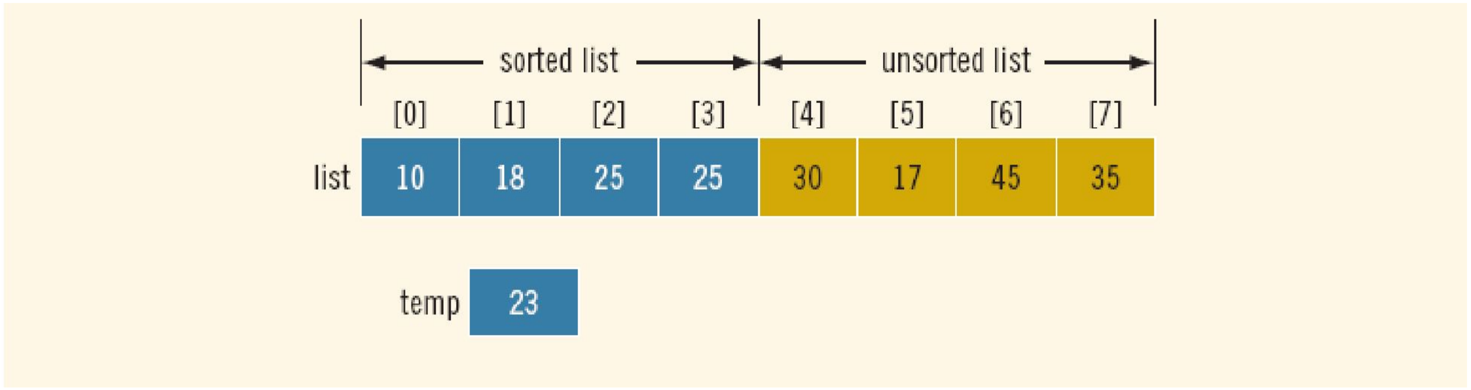


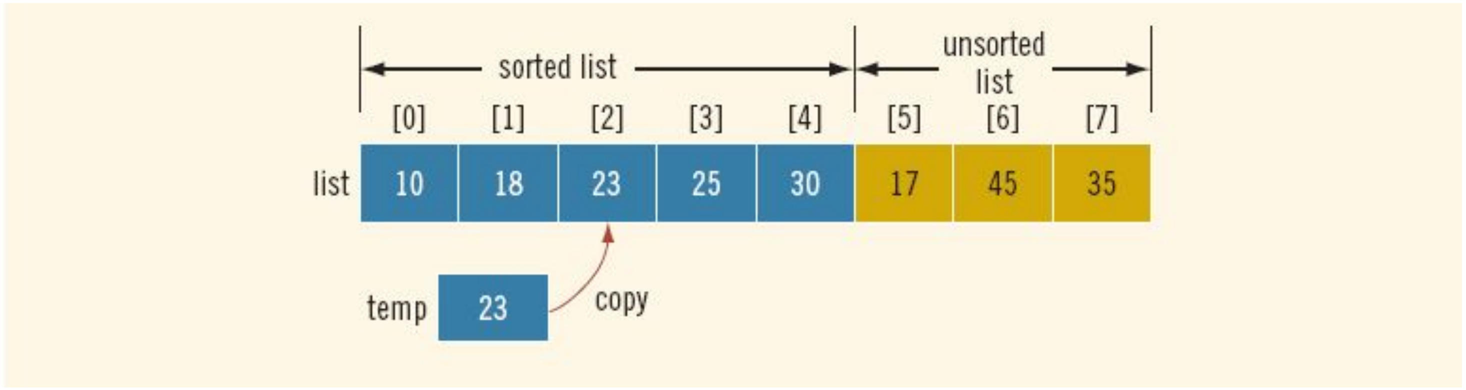
Insertion Sort





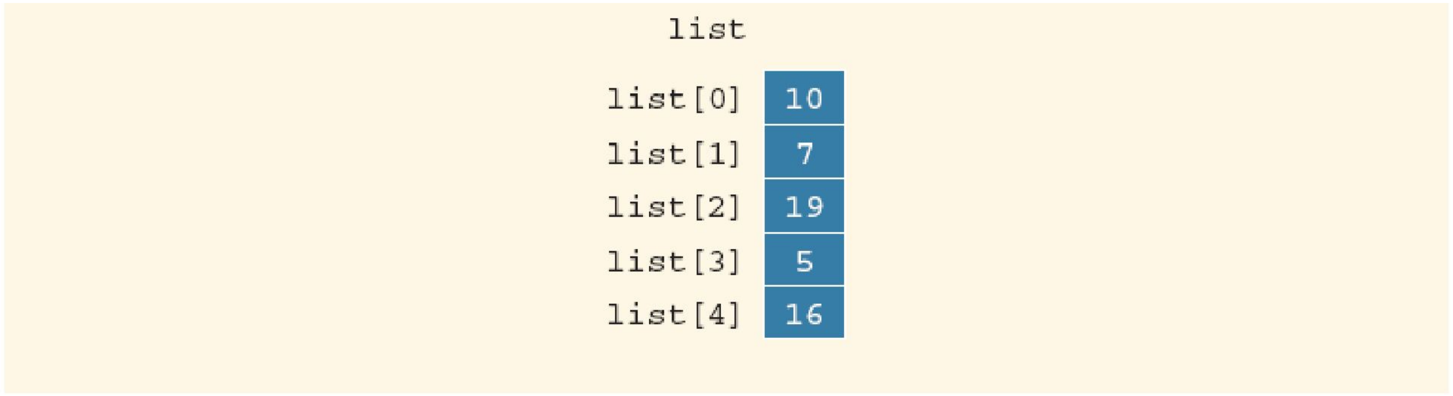

sorted list
unsorted list
[0] [1] [2] [3] [4] [5] [6] [7]
list 10 18 25 30 23 17 45 35
copy
copy
temp 23


sorted list
unsorted list
[0] [1] [2] [3] [4] [5] [6] [7]
list 10 18 25 25 30 17 45 35
temp 23

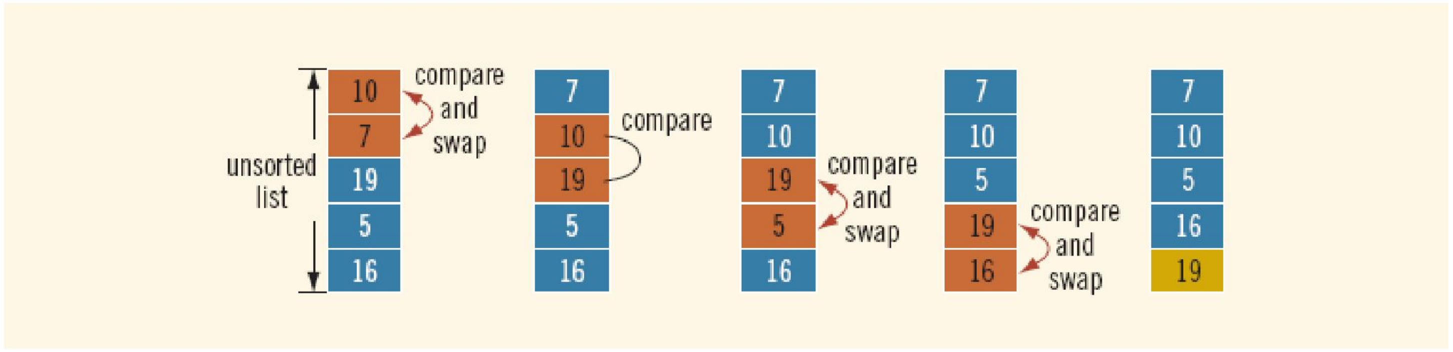

sorted list
unsorted list
[0] [1] [2] [3] [4] [5] [6] [7]
list 10 18 23 25 30 17 45 35
temp 23
copy

Bubble Sort

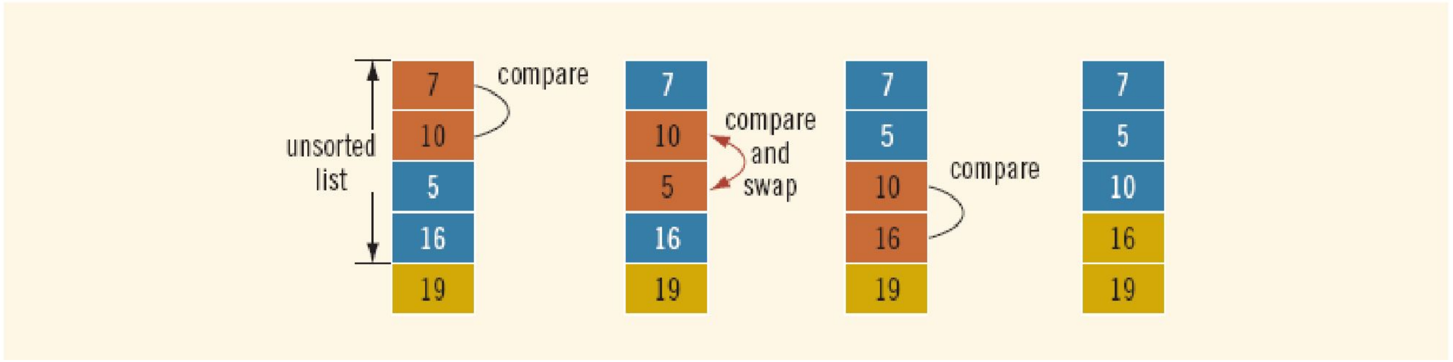
- Suppose `list` is an array of `n` elements
- In `n-1` iterations compare elements `list[index]` and `list[index + 1]`
- If `list[index]` > `list[index + 1]`, then swap them



Bubble Sort

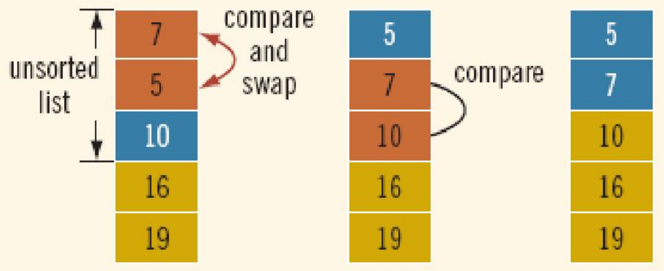


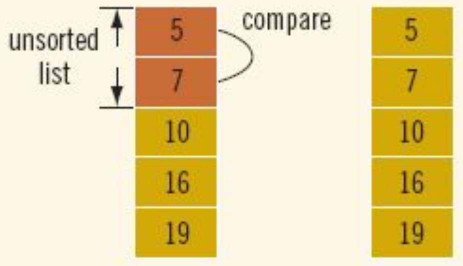
unsorted list
10
7
19
5
16
compare and swap
7
10
19
5
16
compare
7
10
19
5
16
compare and swap
7
10
5
19
16
compare and swap
7
10
5
16
19



unsorted list
7
10
5
16
19
compare
7
10
5
16
19
compare and swap
7
5
10
16
19
compare
7
5
10
16
19

Bubble Sort


unsorted list
7
5
10
16
19
compare and swap
5
7
10
16
19
compare
5
7
10
16
19


unsorted list
5
7
10
16
19
compare
5
7
10
16
19