Arrays and Strings

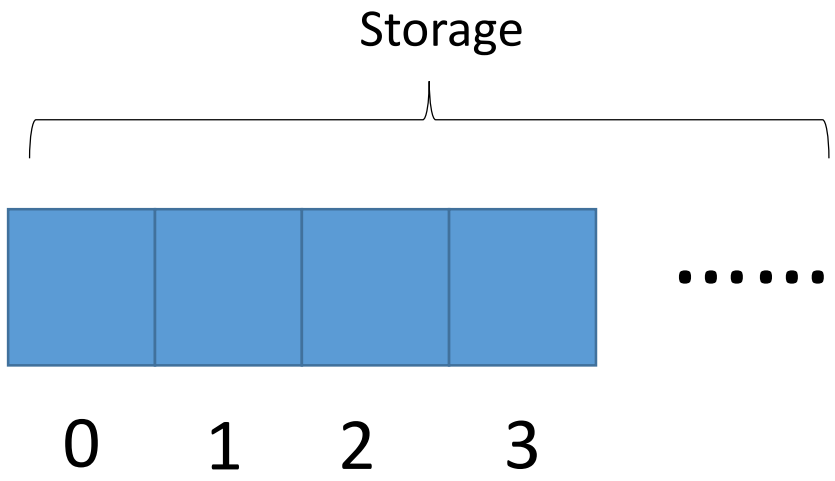
黃世強 (Sai-Keung Wong)

Intended Learning Outcomes

- Define arrays
- List some operations on arrays
- Define the linear index of a matrix

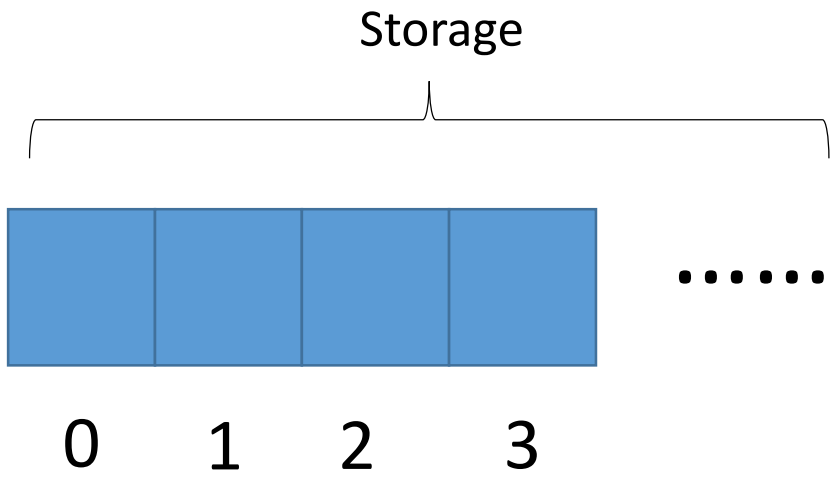
Example
Requirement Specification

1. Read one hundred numbers
2. Show them
3. Compute their average
4. Find out the number of numbers that are above the average.



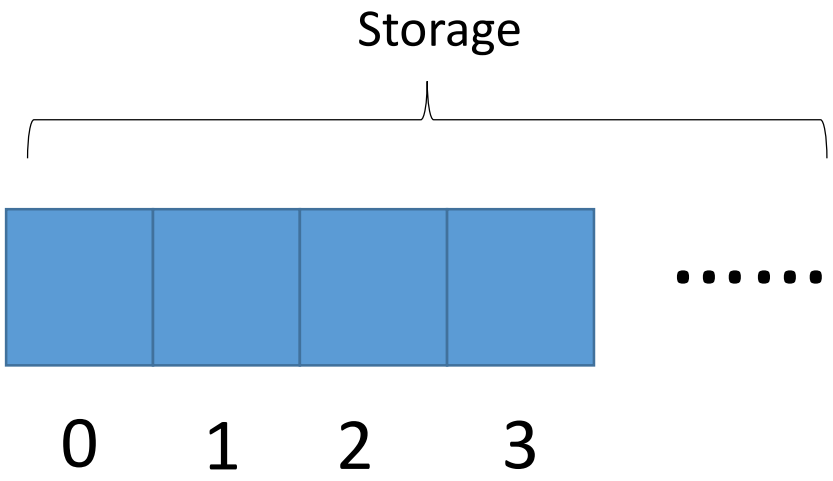
Example
Requirement Specification

1. Read one hundred numbers
2. Show them
3. Compute their average
4. Find out the number of numbers that are above the average.



Example Requirement Specification

1. Read one hundred numbers
2. Show them
3. Compute their average
4. Find out the number of numbers that are above the average.



data

Task 1

Task 2

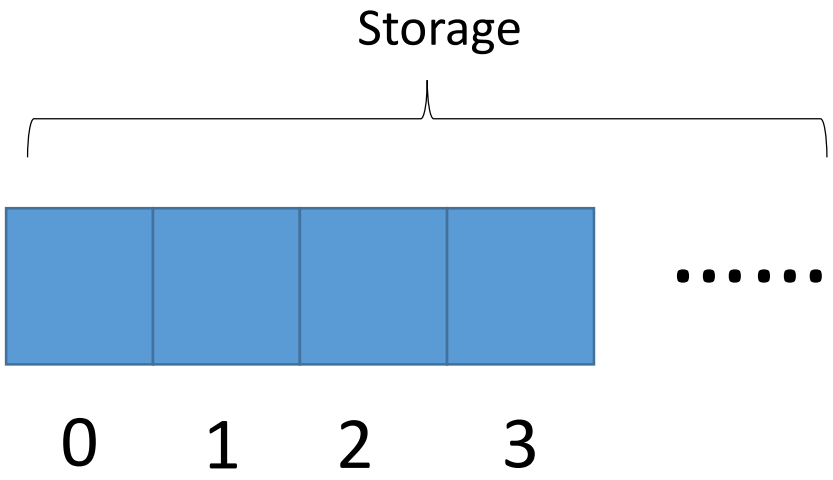
Task 3

Task 4

Task .....

Example
Requirement Specification

1. Read one hundred numbers
2. Show them
3. Compute their average
4. Find out the number of numbers that are above the average.



data

```
const int num = 100;
double numArr[ num ];                // array
```

Task 1

```
for ( int i; i < num; ++i ) {
   cin >> numArr[ i ];
}
```

Task 2

```
for ( int i; i < num; ++i ) {
   cout << numArr[ i ] << endl;
}
```

Task 3

```
double average = 0.0;
for ( int i; i < num; ++i ) {
   average += numArr[ i ]  / (double) num;
}
```

Task 4

```
int num_above_avg = 0;
for ( int i; i < num; ++i ) {
   if (numArr[ i ] > average ) num_above_avg++;
}
```

Arrays

Array is a data structure that represents a collection of the **same** **types** of **data**.

// Declaring, creating, and initializing

```
double numArr[6] = {1,3,7.5,-9,0,-2.1};
```

```
double *numPtr = numArr;
```
// numPtr is a pointer. It points to the first element of numArr.

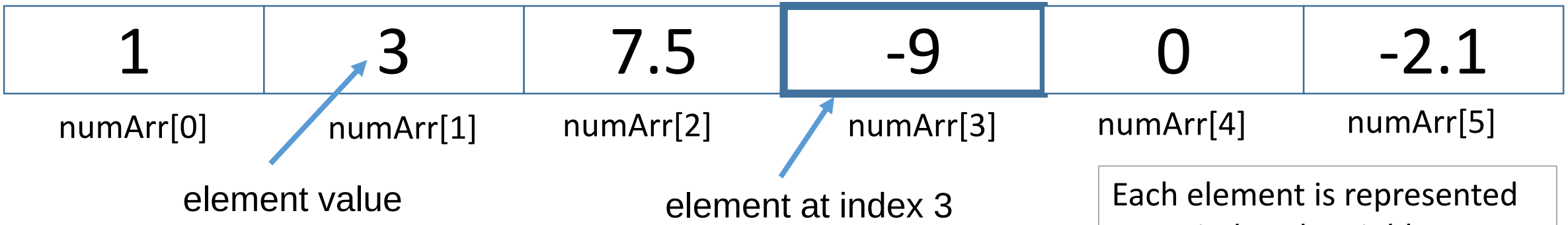
Arrays

Array is a data structure that represents a collection of the **same** **types** of **data**.

// Declaring, creating, and initializing

```
double numArr[6] = {1,3,7.5,-9,0,-2.1};
```

```
double *numPtr = numArr;
```
// numPtr is a pointer. It points to the first element of numArr.



Array indices are 0-based. They start from 0 to arraySize-1.

Each element is represented as an *indexed variable: numArr[index]. index must be an ordinal number.*

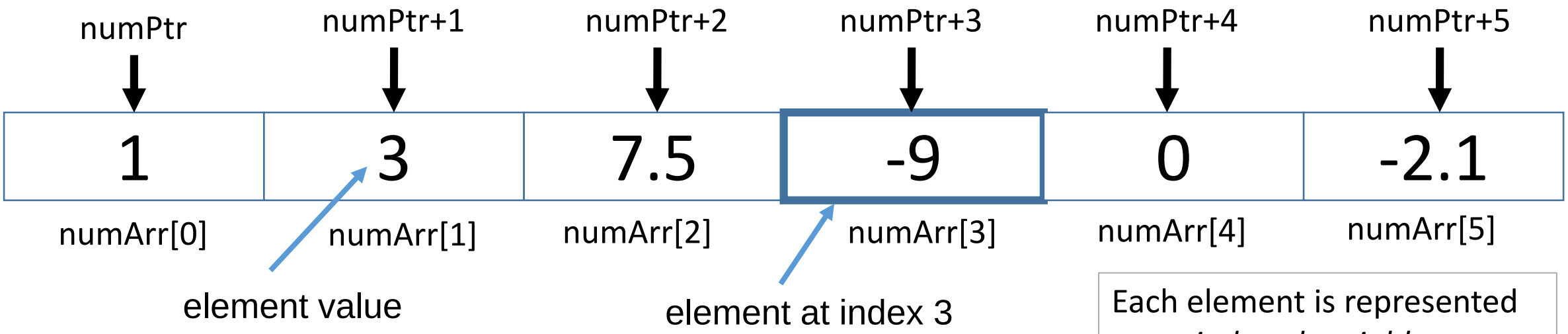
Arrays

Array is a data structure that represents a collection of the **same types** of **data**.

// Declaring, creating, and initializing

```
double numArr[6] = {1,3,7.5,-9,0,-2.1};
```

```
double *numPtr = numArr;
```
// numPtr is a pointer. It points to the first element of numArr.



Array indices are 0-based. They start from 0 to arraySize-1.

Each element is represented as an *indexed variable: numArr[index]. index must be an ordinal number.*

Declaring Array Variables

```
datatype arrayRefVar[arraySize];
```

A constant name

A constant size or a constant expression

Declaring Array Variables

```
datatype arrayRefVar[arraySize];
```


```
double myList[4];          // myList has 4 elements whose data type is double.

int vsize = 4;
double myList[vsize];      // This is illegal because vsize is not a constant.
                           // vsize is a variable whose value can be changed.

const int vsize = 4;       // vsize is a constant. Its value cannot
                           // be changed after initialization.
double myList[vsize];      // Correct
```

Arbitrary Initial Values

When an array is created, its elements are assigned with arbitrary values.

```
double myList[100];        // Declaration of myList.
                           // The elements of myList are not initialized.
```

Arbitrary Initial Values

When an array is created, its elements are assigned with arbitrary values.

```
double myList[100];        // Declaration of myList.
                           // The elements of myList are not initialized.
```

What is double[0]?
What is double[1]?
……
What is double[99]?

Arbitrary Initial Values

When an array is created, its elements are assigned with arbitrary values.

```
double myList[100];        // Declaration of myList.
                           // The elements of myList are not initialized.
What is double[0]?         // an arbitrary value. The first element
What is double[1]?          // an arbitrary value. The second element
……
What is double[99]?         // an arbitrary value. The 100-th element
```

Example

- Indexed variables are treated as regular variables.
- We use them to perform calculation.

Add the values of the second and third elements and then assign the result to the first element.

```
numArr[0] = numArr [1] + numArr[2];
```

The array boundary

C++ does not check array's boundary.

```
float numArr[6] = {1,3,7.5,-9,0,-2.1};
```

So numArr[-1] and numArr[7] do not cause syntax errors.

But a memory access violation may be reported at runtime.

1	3	7.5	-9	0	-2.1
numArr[0]	numArr[1]	numArr[2]	numArr[3]	numArr[4]	numArr[5]

The array boundary

C++ does not check array's boundary.

```
float numArr[6] = {1,3,7.5,-9,0,-2.1};
```

So numArr[-1] and numArr[7] do not cause syntax errors.

But a memory access violation may be reported at runtime.

???	1	3	7.5	-9	0	-2.1	???
numArr[-1]	numArr[0]	numArr[1]	numArr[2]	numArr[3]	numArr[4]	numArr[5]	numArr[6]

The array boundary

```
float a = 1;

float numArr[6] = {1,3,7.5,-9,0,-2.1};

float b = 9;

int i = 1; numArr[0-i] = 5; numArr[5+i] = 7;
```

???	1	3	7.5	-9	0	-2.1	???
numArr[-1]	numArr[0]	numArr[1]	numArr[2]	numArr[3]	numArr[4]	numArr[5]	numArr[6]

The array boundary

- `float a = 1;`

  `float numArr[6] = {1,3,7.5,-9,0,-2.1};`

- `float b = 9;`

  `int i = 1; numArr[0-i] = 5; numArr[5+i] = 7;`

???	1	3	7.5	-9	0	-2.1	???
numArr[-1]	numArr[0]	numArr[1]	numArr[2]	numArr[3]	numArr[4]	numArr[5]	numArr[6]

```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;









}
```

```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;




    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;









}
```

```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;




    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;




}
```

```
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
```

```
請按任意鍵繼續 . . .
```

```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;




    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;




}
```

```
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
```

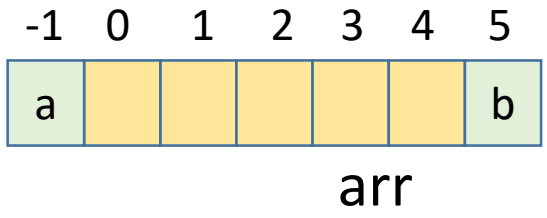
```
請按任意鍵繼續 . . .
```

```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;




    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```

```
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
```

```
請按任意鍵繼續 . . .
```

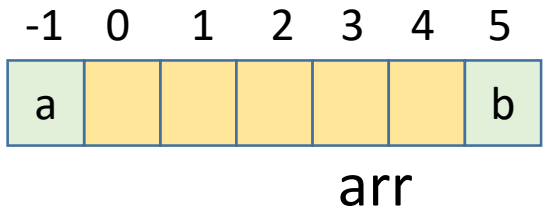
```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;



    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```



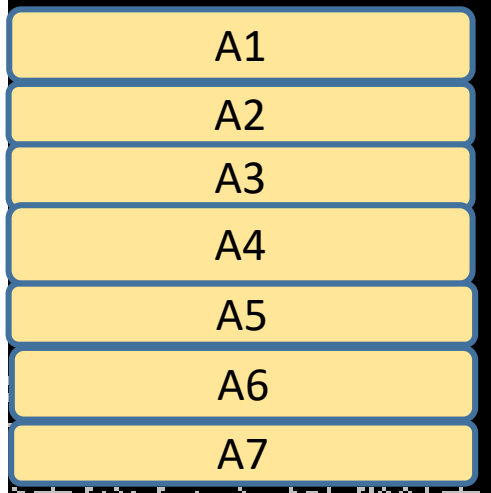

```
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
```

```
請按任意鍵繼續 . . .
```

```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;



    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```

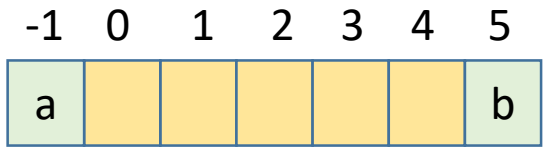



```
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
```



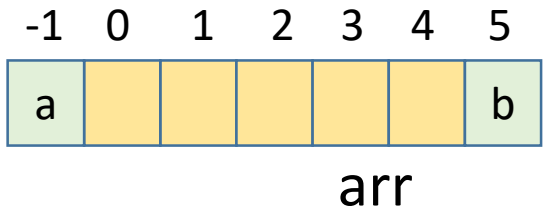

```
請按任意鍵繼續 . . .
```

```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;
    cout << "address of a:" << &a << endl;
    cout << "address of b:" << &b << endl;
    cout << "address of arr:" << arr << endl;

    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```



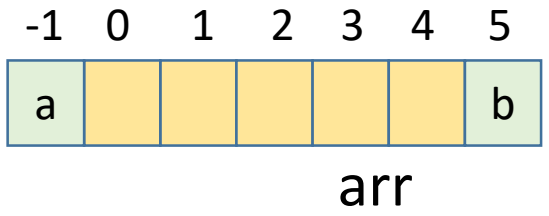

```
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
==============
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:7
請按任意鍵繼續 . . .
```

```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;
    cout << "address of a:" << &a << endl;
    cout << "address of b:" << &b << endl;
    cout << "address of arr:" << arr << endl;

    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```

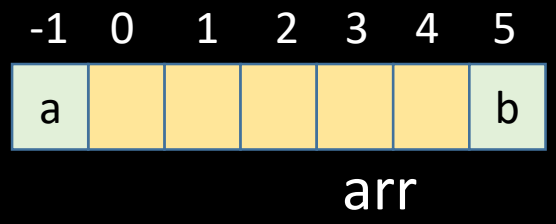



```
address of a:00B9FD8C
address of b:00B9FD90
address of arr:00B9FD94
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
==============
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:7
請按任意鍵繼續 . . .
```

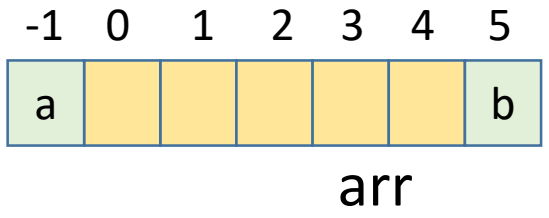
```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;
    cout << "address of a:" << &a << endl;
    cout << "address of b:" << &b << endl;
    cout << "address of arr:" << arr << endl;

    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```



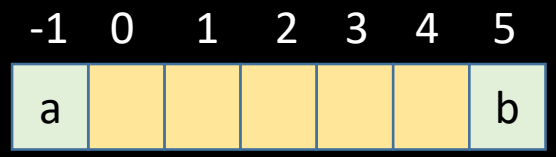

```
address of a:00B9FD8C
address of b:00B9FD90
address of arr:00B9FD94
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
==============
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:7
請按任意鍵繼續 . . .
```



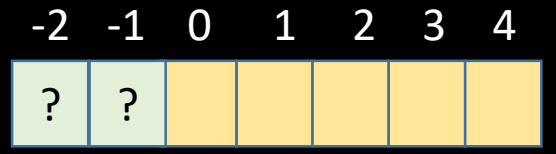
```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;
    cout << "address of a:" << &a << endl;
    cout << "address of b:" << &b << endl;
    cout << "address of arr:" << arr << endl;

    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```



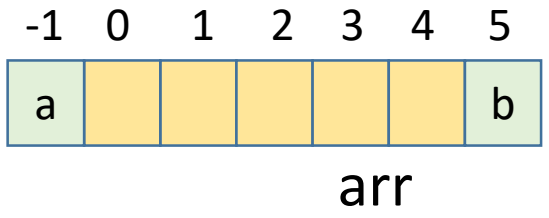

```
address of a:00B9FD8C
address of b:00B9FD90
address of arr:00B9FD94
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
==============
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:7
請按任意鍵繼續 . . .
```



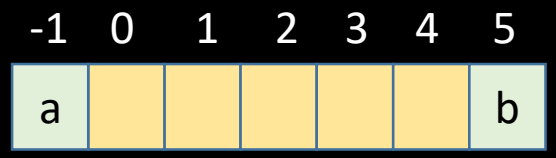
a?b?



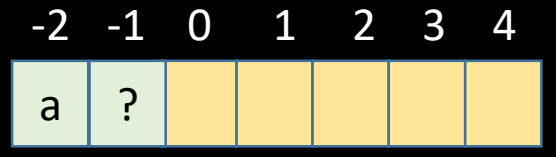
```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;
    cout << "address of a:" << &a << endl;
    cout << "address of b:" << &b << endl;
    cout << "address of arr:" << arr << endl;

    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```



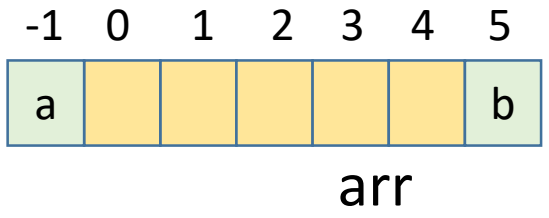

```
address of a:00B9FD8C
address of b:00B9FD90
address of arr:00B9FD94
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
==============
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:7
請按任意鍵繼續 . . .
```



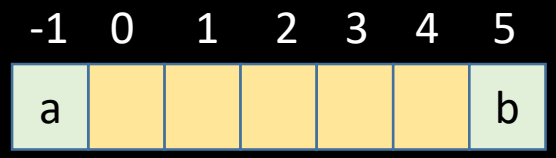
a?b?



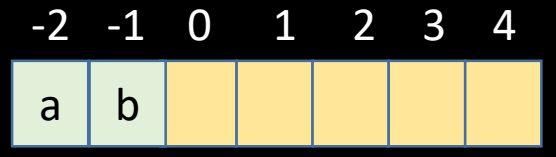
```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;
    cout << "address of a:" << &a << endl;
    cout << "address of b:" << &b << endl;
    cout << "address of arr:" << arr << endl;

    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```



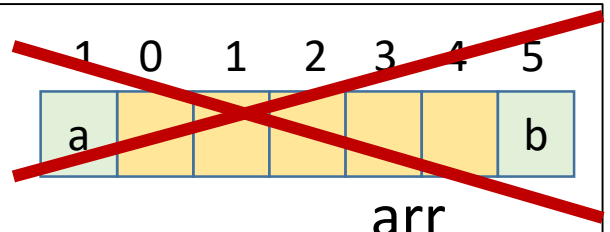

```
address of a:00B9FD8C
address of b:00B9FD90
address of arr:00B9FD94
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
==============
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:7
請按任意鍵繼續 . . .
```



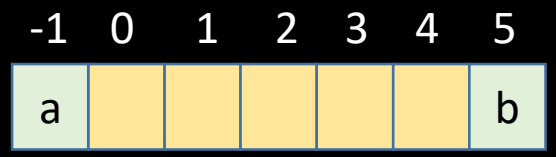
a?b?

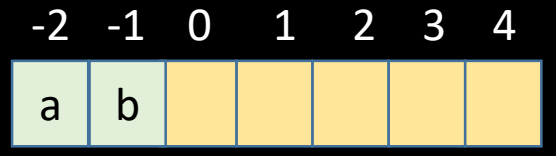


```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;
    cout << "address of a:" << &a << endl;
    cout << "address of b:" << &b << endl;
    cout << "address of arr:" << arr << endl;

    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```

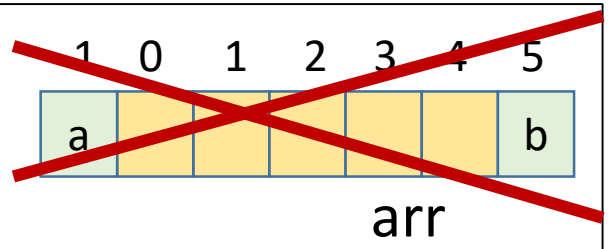



```
address of a:00B9FD8C
address of b:00B9FD90
address of arr:00B9FD94
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
==============
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:7
請按任意鍵繼續 . . .
```

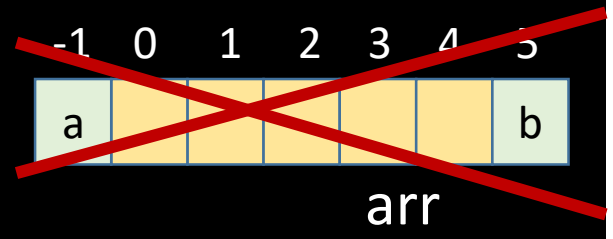


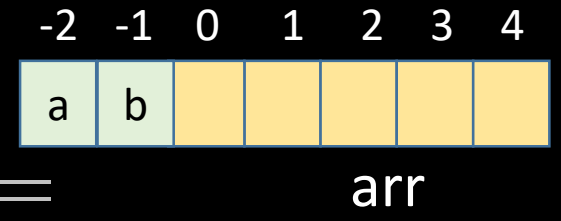


```
void test() {
    int a = 5;
    int arr[5] = { -1, -2, -3 };
    int b = 6;
    cout << "address of a:" << &a << endl;
    cout << "address of b:" << &b << endl;
    cout << "address of arr:" << arr << endl;

    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
    arr[-1] = 7;
    arr[5] = 8;
    cout << "==============" << endl;
    cout << "a:" << a << endl;
    for (int i = 0; i < 5; ++i) {
    cout << "arr[" << i << "]:" << arr[i] << endl;
    }
    cout << "b:" << b << endl;
}
```




```
address of a:00B9FD8C
address of b:00B9FD90
address of arr:00B9FD94
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:6
==============
a:5
arr[0]:-1
arr[1]:-2
arr[2]:-3
arr[3]:0
arr[4]:0
b:7
請按任意鍵繼續 . . .
```





Declaring, creating, and initializing an array

```
// Declaring, creating, and initializing
float numArr[6] = {1,3,7.5,-9,0,-2.1};

double numArr[6];
numArr[0] = 1;
numArr[1] = 3;
numArr[2] = 7.5;
numArr[3] = -9;
numArr[4] = 0;
numArr[5] = -2.1;
```

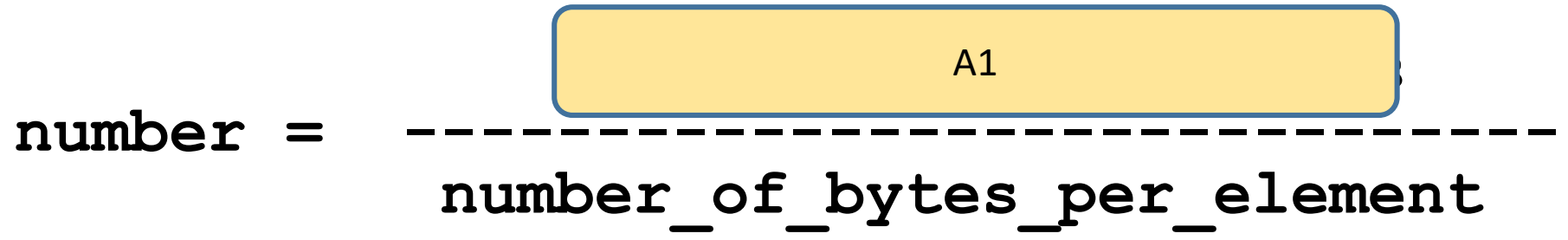
Declaring, creating, and initializing an array

```
// Declaring, creating, and initializing
float numArr[] = {1,3,7.5,-9,0,-2.1};
// Implicit size

// To get the number of elements of numArr
int num = sizeof(numArr)/sizeof(float);
```

$$\textbf{number} = \frac{\textbf{size\_of\_array\_in\_bytes}}{\textbf{number\_of\_bytes\_per\_element}}$$

Declaring, creating, and initializing an array

```
// Declaring, creating, and initializing
float numArr[] = {1,3,7.5,-9,0,-2.1};
// Implicit size

// To get the number of elements of numArr
int num = sizeof(numArr)/sizeof(float);
```



Partial Initialization

We can initialize the elements of the first portion in an array and the rest elements in the second portion are set to zero.

double myList[4] = {1.9, 2.9};

myList[0] =?

myList[1] =?

myList[2] =?

myList[3] =?

Partial Initialization

We can initialize the elements of the first portion in an array and the rest elements in the second portion are set to zero.

double myList[4] = {1.9, 2.9};

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 0;

myList[3] = 0;

Implement a program and check the result.
See the values of the elements of myList.

Initializing arrays with random values

Initialize the array myList with random values between 0 and 9:

```
for (int i = 0; i < ARRAY_SIZE; i++)
{
  myList[i] = _____?
}
```

Initializing arrays with random values

Initialize the array myList with random values between 0 and 9:

```
for (int i = 0; i < ARRAY_SIZE; i++)
{
  myList[i] =                    ;
}
```

Initializing arrays with random values

Initialize the array myList with random values between 0 and 9:

```
for (int i = 0; i < ARRAY_SIZE; i++)
{
  myList[i] = A1 % 10;
}
```

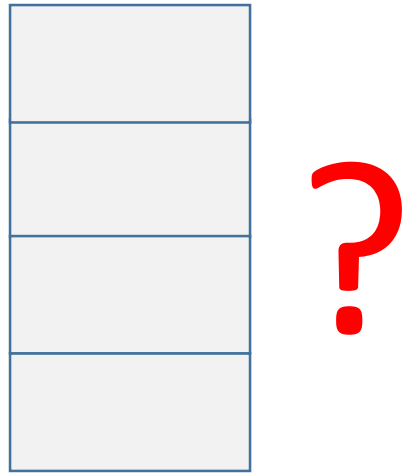
Dry Run

```
int main()
{
    int values[4];
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

Dry Run

```
int main()
{
    int values[4];
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

After the array is created



Dry Run

Declare an array with size 4, i.e., four elements.

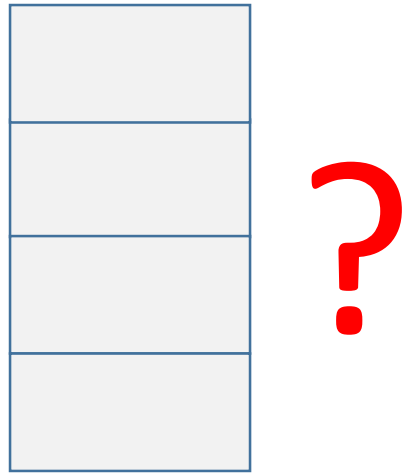
```
int main()
{
    int values[4];
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

Dry Run

Declare an array with size 4, i.e., four elements.

```
int main()
{
    int values[4];
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

After the array is created

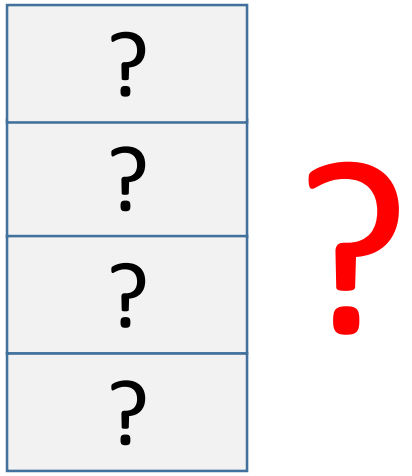


Dry Run

Declare an array with size 4, i.e., four elements.

```
int main()
{
    int values[4];
    for (int i = 1; i < 4; i++)
    {
        values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

They are arbitrary values!



No initialization

Dry Run

Declare and initialize an array with size 4, i.e., four elements.

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

After the array is created

0
0
0
0

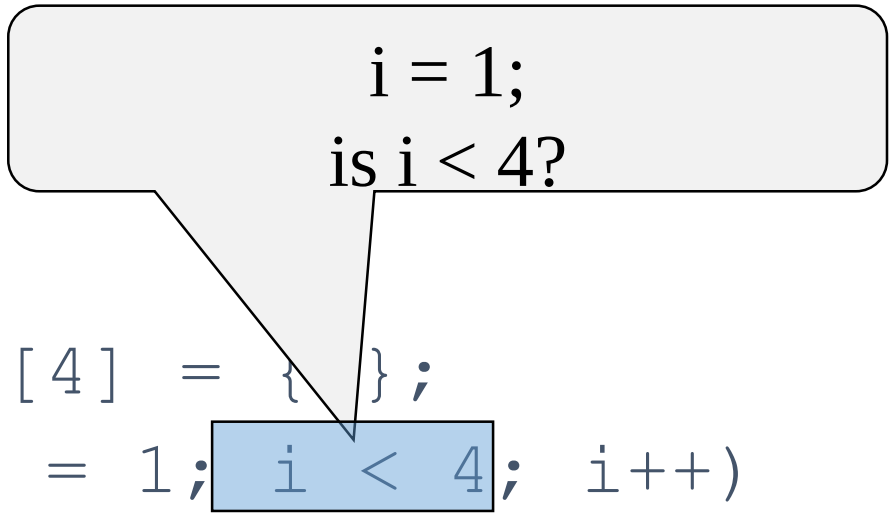
Dry Run

i = 1

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

0
0
0
0

Dry Run



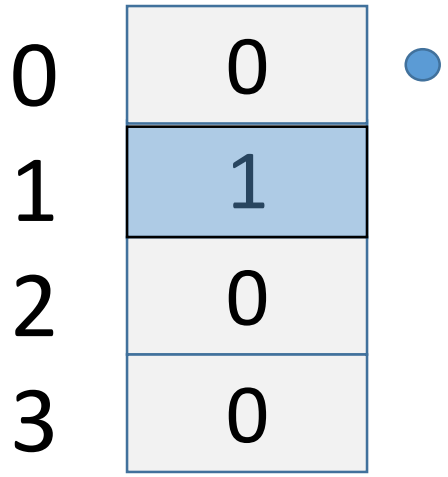

```
int main()
{
    int values[4] = { };
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

0
0
0
0

Dry Run

i = 1;
values[1] = 1 + values[0]

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
    {
        values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```



Dry Run

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

i = 1;
i ← i + 1
i = 2

0	0
1	1
2	0
3	0

Dry Run

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

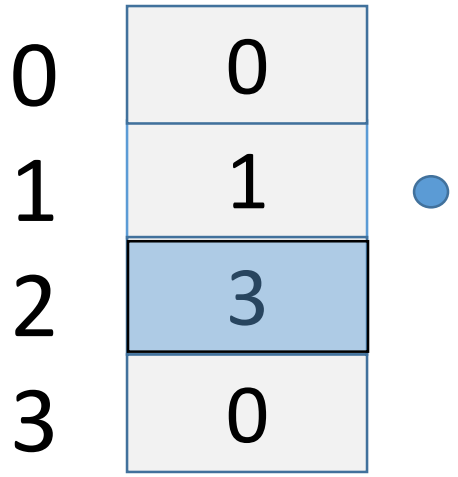
i = 2;
i < 4 ?

0	0
1	1
2	0
3	0

Dry Run

i = 2;
values[2] = 2 + values[1]

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
    {
        values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```



Dry Run

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

i = 2;
i ← i + 1
i = 3

0	0
1	1
2	3
3	0

Dry Run

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

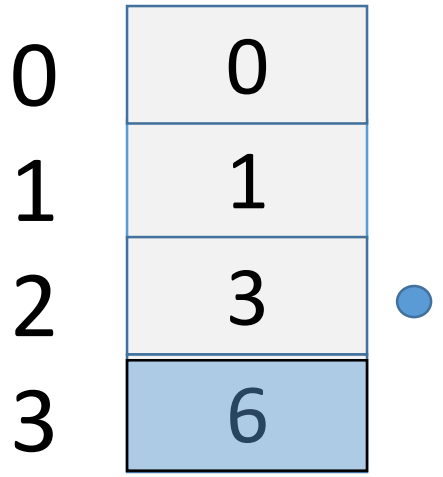
i = 3;
i < 4 ?

0	0
1	1
2	3
3	0

Dry Run

i = 3;
values[3] = 3 + values[2]

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
    {
        values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```



Dry Run

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

i = 3;
i ← i + 1
i = 4

0	0
1	1
2	3
3	6

Dry Run

i = 4;
i < 4 ?

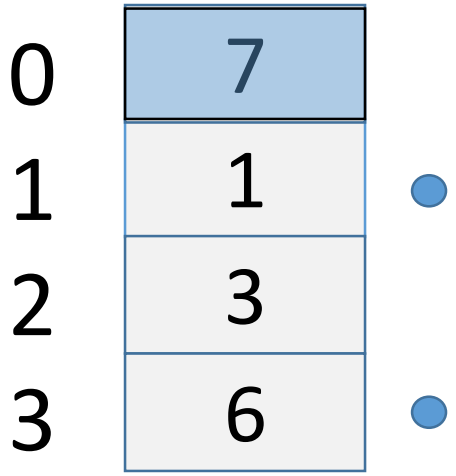
```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
   {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```

0	0
1	1
2	3
3	6

Dry Run

values[0] = values[1] + values[3]

```
int main()
{
    int values[4] = {0};
    for (int i = 1; i < 4; i++)
    {
        values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[3];
}
```



Printing arrays

```
const int ARRAY_SIZE = 3;

int myList[ARRAY_SIZE] = {1, 2, 3};

// print each element of an array

for (int i = 0; i < ARRAY_SIZE; i++)
{
  cout << ____________________;
}

cout << endl; // go to the next line
```

Printing arrays

```
const int ARRAY_SIZE = 3;

int myList[ARRAY_SIZE] = {1, 2, 3};

// print each element of an array

for (int i = 0; i < ARRAY_SIZE; i++)
{
  cout << myList[i] << " ";
}

cout << endl; // go to the next line
```

Copying Arrays

```
int list[100], myList[100];

…
```

Can you copy an array using a syntax like this?

```
        list = myList;          // not allowed in C++
                                // Note that list has a fixed value,
                                // i.e., the start address of the array.
```

Copy individual elements from one array to the other.

```
for (int i = 0; i < ARRAY_SIZE; i++)
{
  list[i] = myList[i];   // copy the elements of myList to list
}
```

Summing All Elements

```
double total = 0;     // initialize the value of total

// run index over all the indices of an array,
// from 0 to (ARRAY_SIZE – 1)
for (int i = 0; i < ARRAY_SIZE; i++)
{
  total += ____L1____;
}
```

A) myList[i]
B) total + myList[i]

Summing All Elements

```
double total = 0;     // initialize the value of total

// run index over all the indices of an array,
// from 0 to (ARRAY_SIZE – 1)
for (int i = 0; i < ARRAY_SIZE; i++)
{
  total += myList[i];
}
```

A) myList[i]
B) total + myList[i]

Finding the Largest Element

1. Store the first element to a variable
2. Compare the variable with each remaining element of the array
3. If the current element is larger than the variable, set the value of the variable as the value of the element. Repeat the process until each element is checked.
4. Finally, return the result

```
double result = myList[0]; // initialize it to the first element
for (int i = 1; i < ARRAY_SIZE; i++)
{
  if ( myList[ i ] > max ) result = myList[ i ];
}
// return the result
```

$$\max \{a_0, a_1, \ldots, a_n\}$$

```
element: {1, 4, 5, 6, 9, 3, 9, 0}
  index: 0  1  2  3  4  5  6  7
```

Finding the smallest index of the largest element

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < ARRAY_SIZE; i++)
{
  if (myList[i] > max)
  {
    max = myList[i];
    indexOfMax = i;
  }
}
```

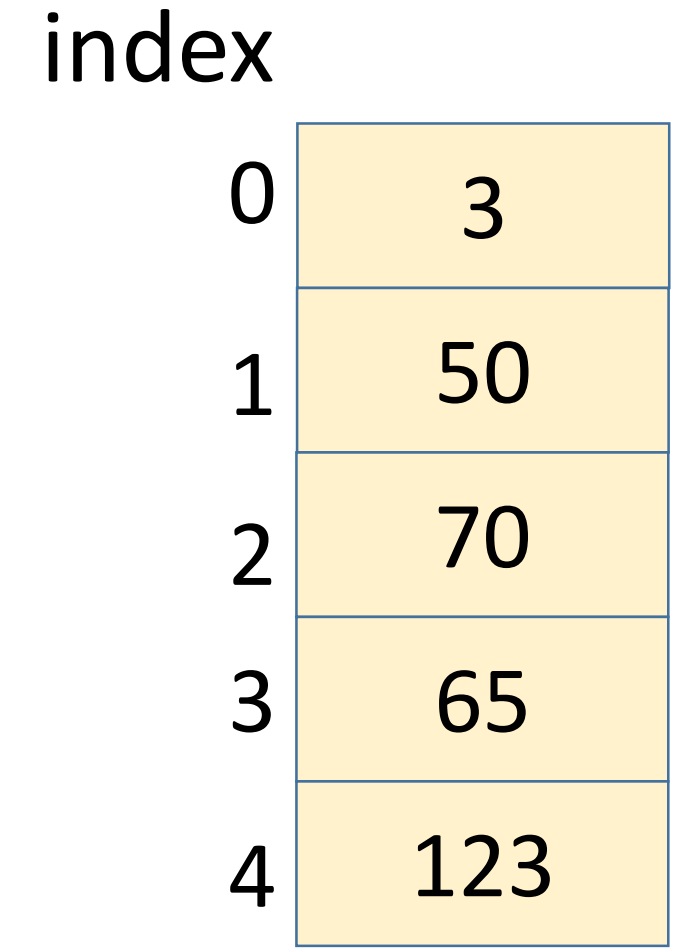
Book keeping

$$\arg\max_i (a_i)$$

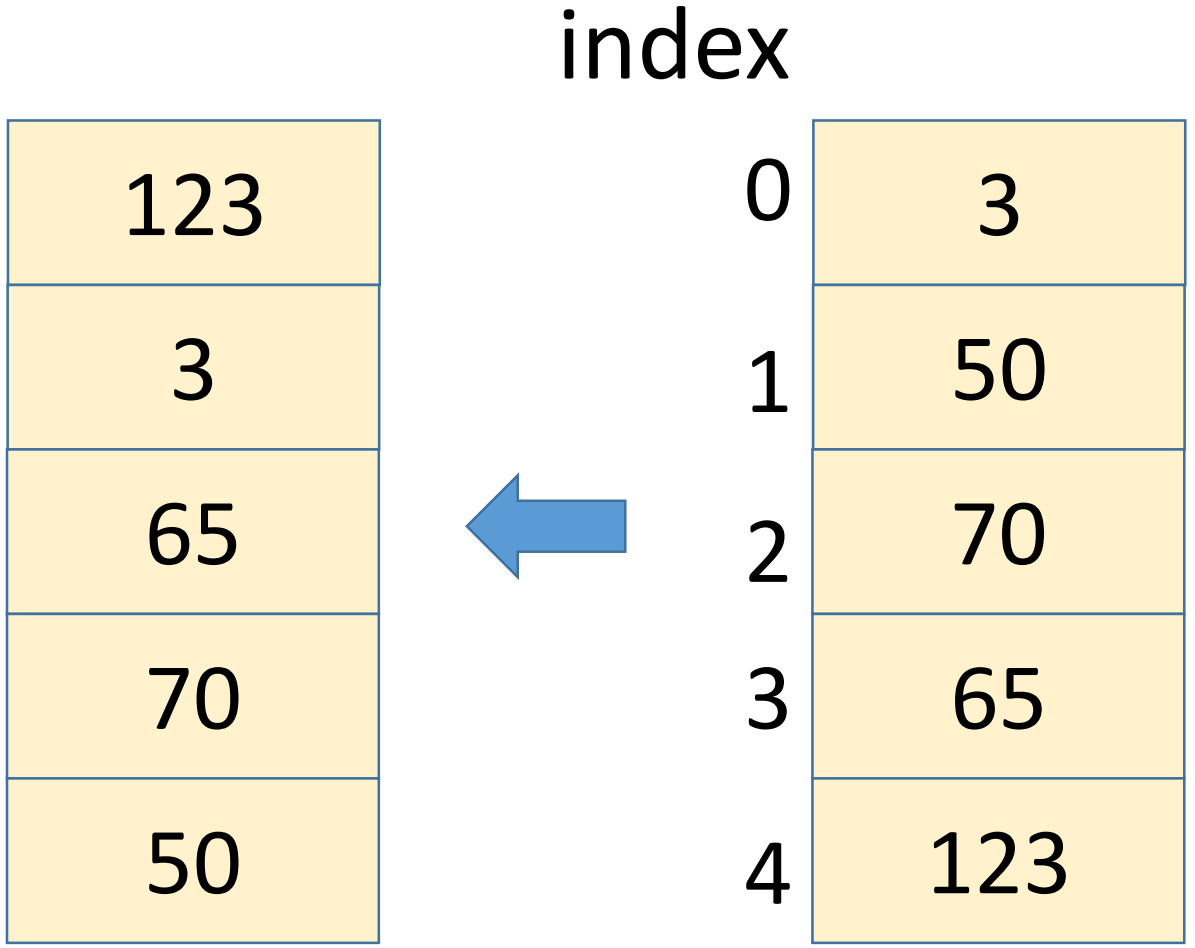
element: {1, 4, 5, 6, 9, 3, 9, 0}

index: 0 1 2 3 4 5 6 7

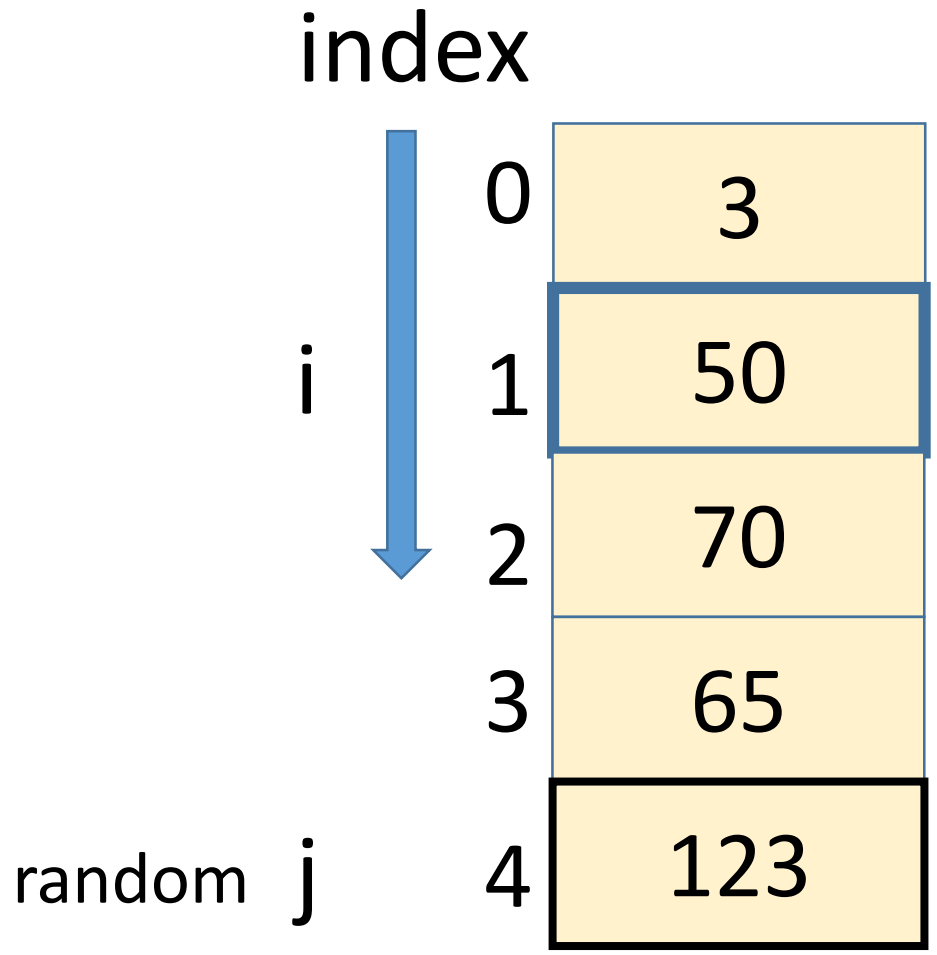
Random Shuffling



Random Shuffling



Random Shuffling

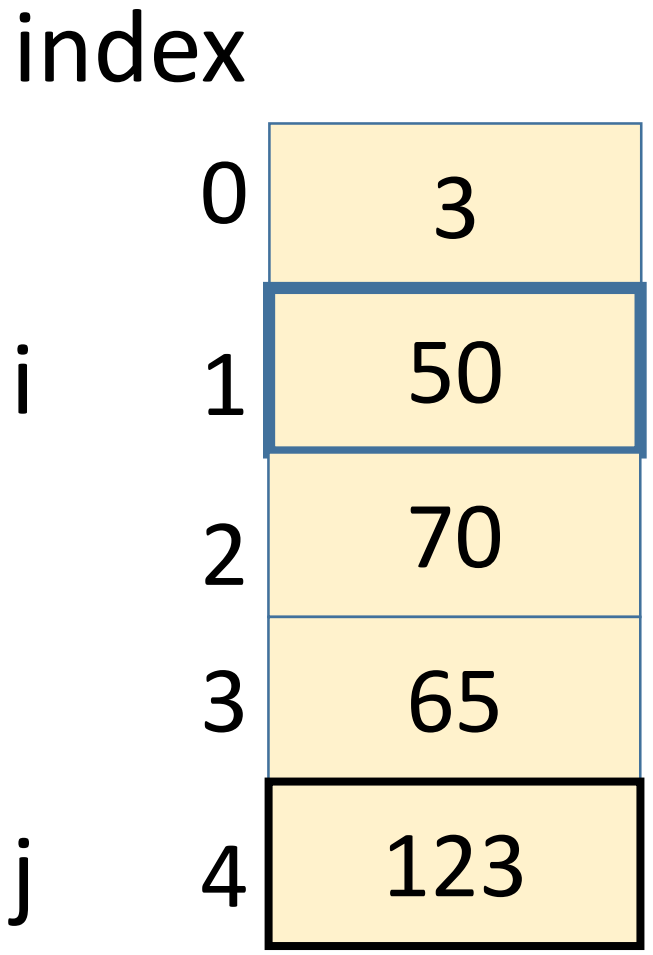


ARRAY_SIZE - 1

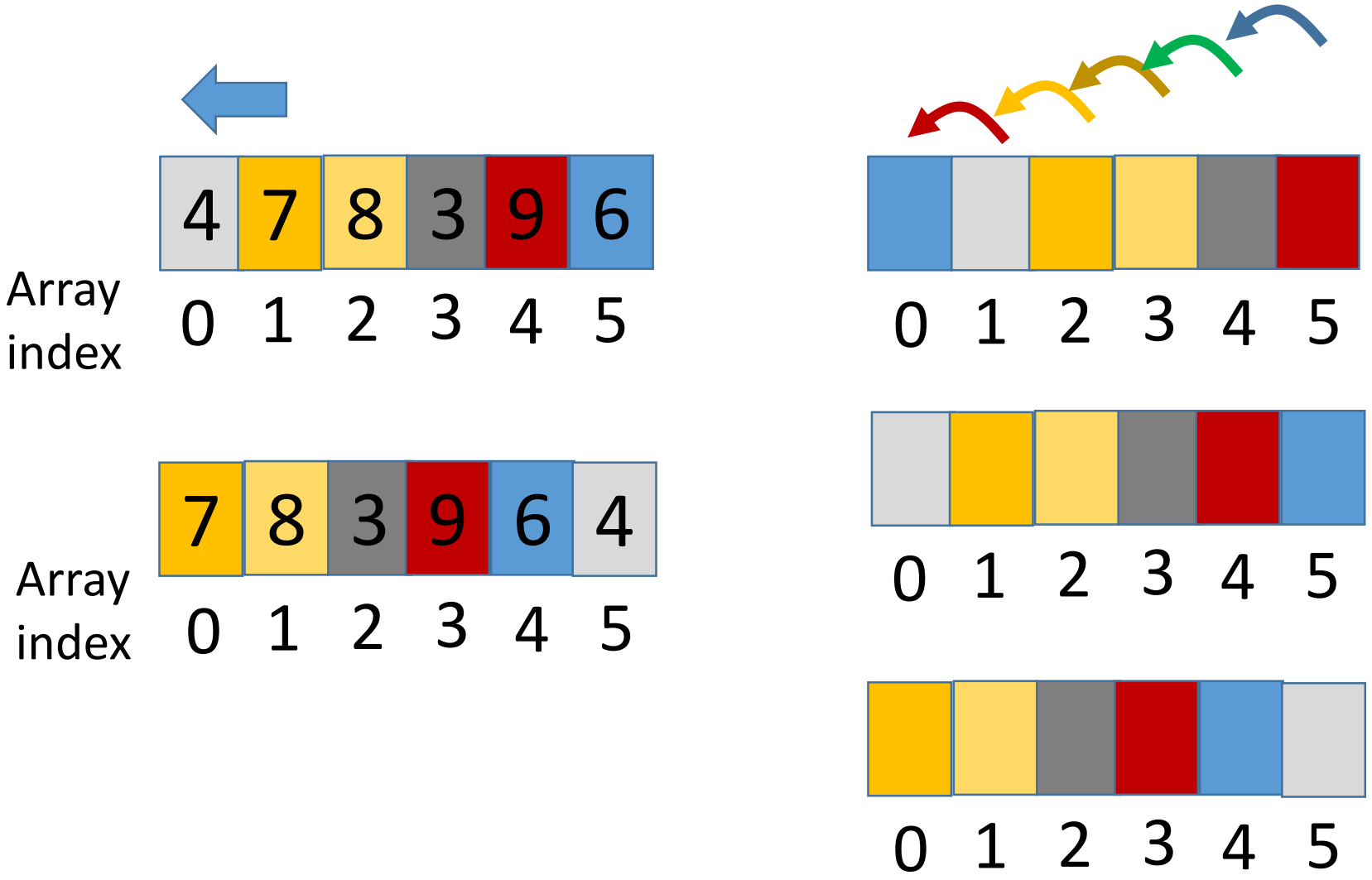
Random Shuffling

```
srand(time(0));
for (int i = 0; i < ARRAY_SIZE; i++)
{
  // Generate an index j randomly
  int j = rand() % ARRAY_SIZE;
  double temp = list[i];
  list[i] = list[ j ];
  list[ j ] = temp;
}
```

swap(list[i], list[j])

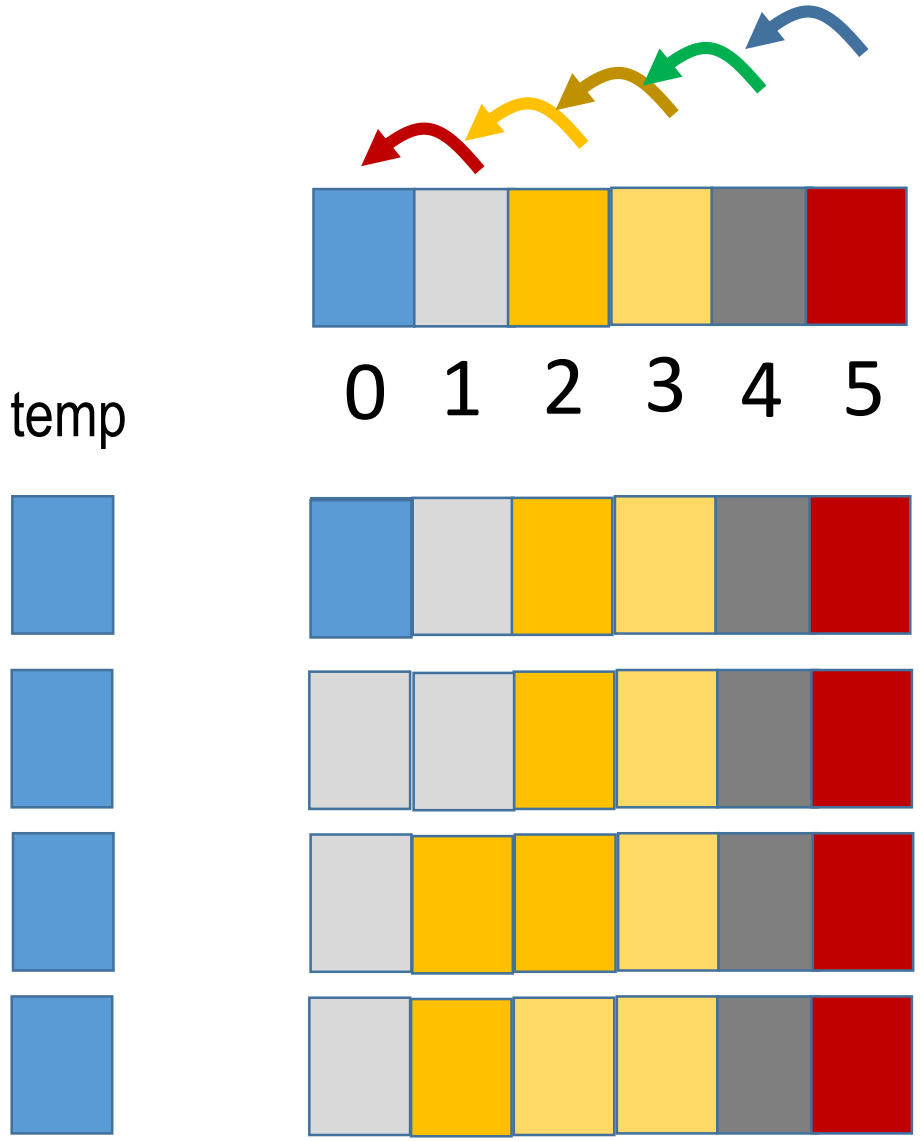


Shifting Elements : Shift to left with rotation



Shifting Elements : Shift to left with rotation

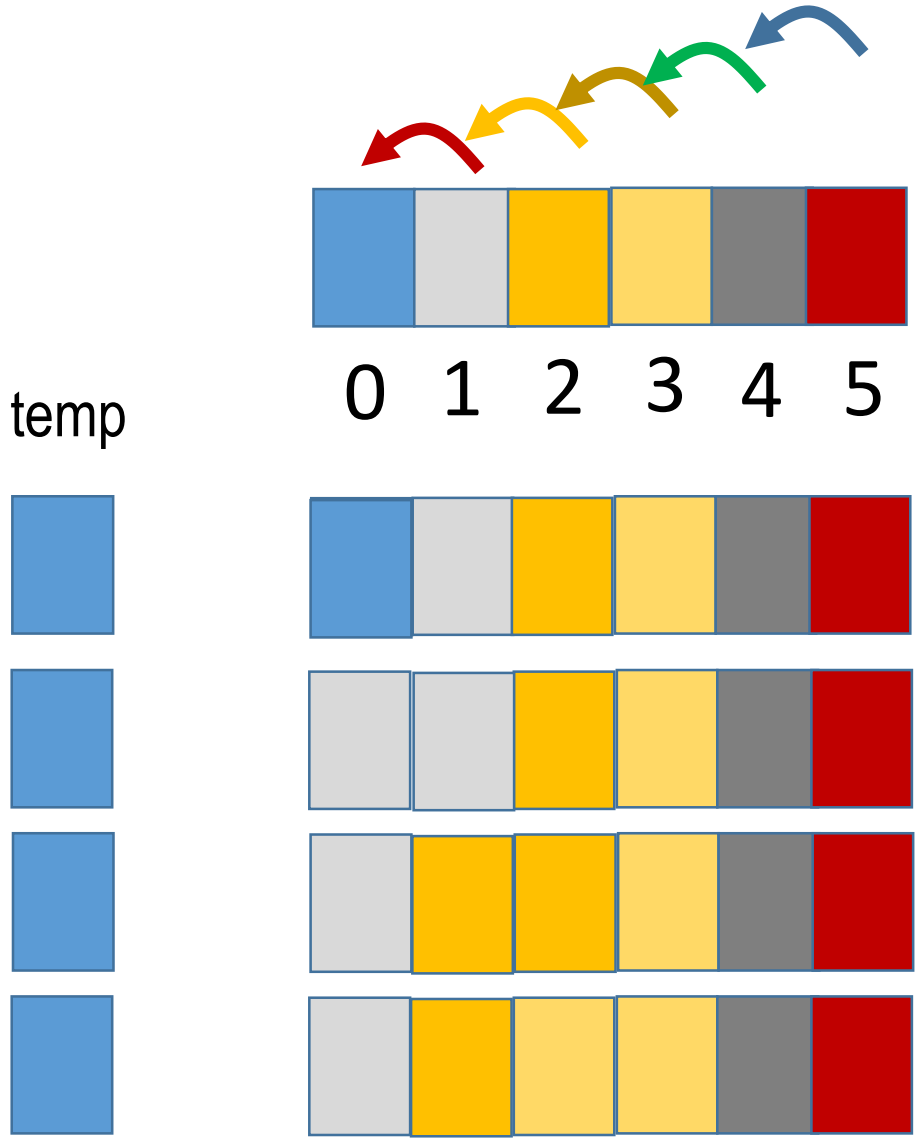
```
double temp = myList[0]; // Retain the first element
// Shift elements to left
for (int i = 1; i < myList.length(); i++)
{
  myList[i - 1] = myList[i];
}
// Move the first element to fill in the last position
myList[myList.length() - 1] = temp;
```



Shifting Elements : Shift to left with rotation

```
double temp = myList[0]; // Retain the first element
// Shift elements to left
for (int i = 1; i < myList.length(); i++)
{
  myList[i - 1] = myList[i];
}
// Move the first element to fill in the last position
myList[myList.length() - 1] = temp;
```

4	3
i - 1	i
3	3



Passing Arrays to Functions

We pass values, variables, and arrays to functions.

```
void foo( int a ) {
        int var = a;
}
void h( int &ref) {
        ref = 12;
}
```

```
void g( int *arr ) {
        arr[0] = 5;
        arr[5] = 7;
        *arr = 8;
}
void k( int arr[ ] ) {
        arr[0] = 1;
        arr[1] = 5;
}
```

Passing Arrays to Functions

We pass values, variables, and arrays to functions.

How do we know the number of elements in the array?

```
void g( int *arr ) {
        arr[0] = 5;
        arr[5] = 7;
        *arr = 8;
}
void k( int arr[ ] ) {
        arr[0] = 1;
        arr[1] = 5;
}
```

Passing Arrays to Functions

We pass an array and its size to a function.

We need to know the size of an array in the function.

Thus, we can run over all the elements.

```
void g( int *arr, int n ) {
}

void k(int arr[ ], int n ) {
        for (int i=0;i<n;++i) {
        }
}
```

```
void g( int *arr ) {
        arr[0] = 5;
        arr[5] = 7;
        *arr = 8;
}
void k( int arr[ ] ) {
        arr[0] = 1;
        arr[1] = 5;
}
```

Pass-by-Reference

When we pass an array to a function, the starting address of an array is passed.

```
void g( int *arr, int n ) {
}

void k(int arr[ ], int n ) {
        for (int i=0;i<n;++i) {
        }
}
```

```
void h( ) {
        int p[5] = {1, 2, 3, 4, 5};
        g( p, sizeof(p)/ size(int) );
}
```

actual parameters

//Formal parameters: arr and n.

//The array is passed by value. We cannot modify the array address.

//But the elements of the array are passed by reference.

const Parameters

const Parameters

```
//The elements of arr can be modified.
void g( int *arr, int n ) {
        if ( n <= 0 ) return;
        arr[ 0 ] = 10;
}
```

const Parameters

```
//The elements of arr can be modified.
void g( int *arr, int n ) {
        if ( n <= 0 ) return;
        arr[ 0 ] = 10;
}
```

```
//The elements of arr cannot be modified.
void g( const int *arr, int n ) {
        if ( n <= 0 ) return;
        arr[ 0 ] = 10; // not allowed
}
```

const Parameters

If an array is passed by value, all its elements must be copied into a new array. This can take a long time. Not good for performance.

For large arrays, it could take some time and additional memory space.

However, passing arrays by reference can lead to errors if our **function changes the array by mistakes**.

```
//The elements of arr can be modified.
void g( int *arr, int n ) {
        if ( n <= 0 ) return;
        arr[ 0 ] = 10;
}
```

```
//The elements of arr cannot be modified.
void g( const int *arr, int n ) {
        if ( n <= 0 ) return;
        arr[ 0 ] = 10; // not allowed
}
```

Returning an Array from a Function

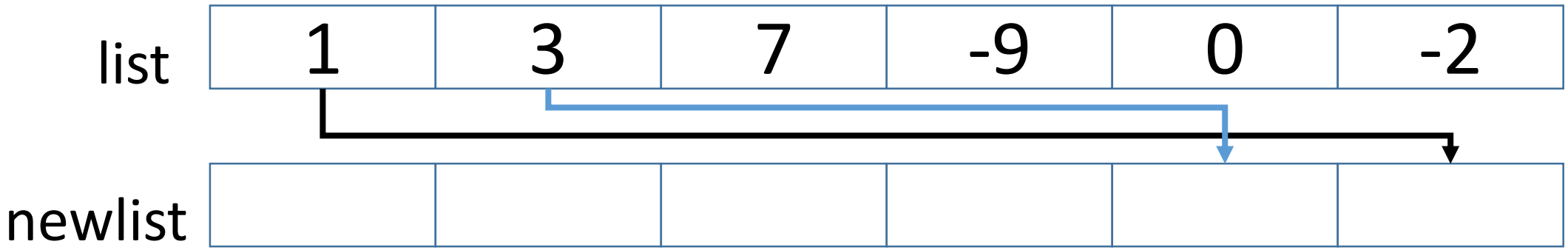
Define a function which returns a new array that is a reversal of an input array.

```
int * reverse(const int list[ ], int size)

int * reverse(const int *list, int size)
```

Handling multiple arrays

We can pass multiple arrays as arguments in the function

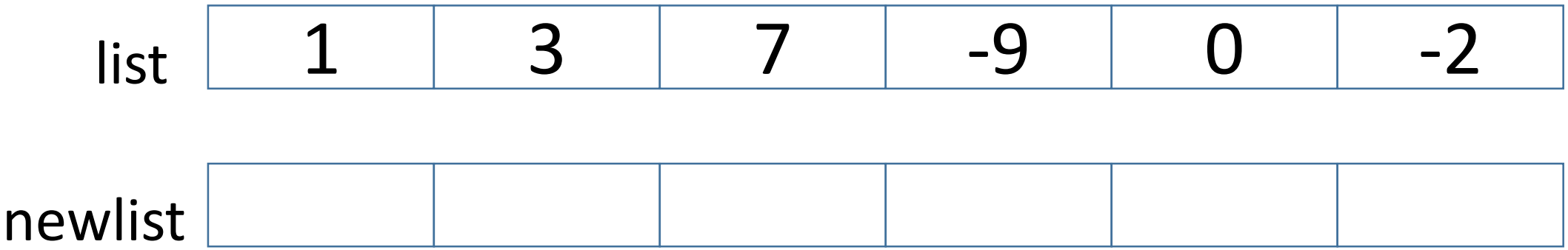
```
// newList is the reversal of list
void reverse(const int list[ ], int newList[ ], int size)
```



Handling multiple arrays

We can pass multiple arrays as arguments in the function

```
// newList is the reversal of list
void reverse(const int list[ ], int newList[ ], int size)
{
   for (int i = 0, j = size - 1; i < size; i++, j--)
  {
     newList[ j ] = list[ i ];
   }
}
```



Reversing elements

We can pass multiple arrays as arguments in the function

```
// newList is the reversal of list
void reverse(const int list[ ], int newList[ ], int size)
{
   for (int i = 0, j = size - 1; i < size; i++, j--)
  {
     newList[ j ] = list[ i ];
   }
}
```

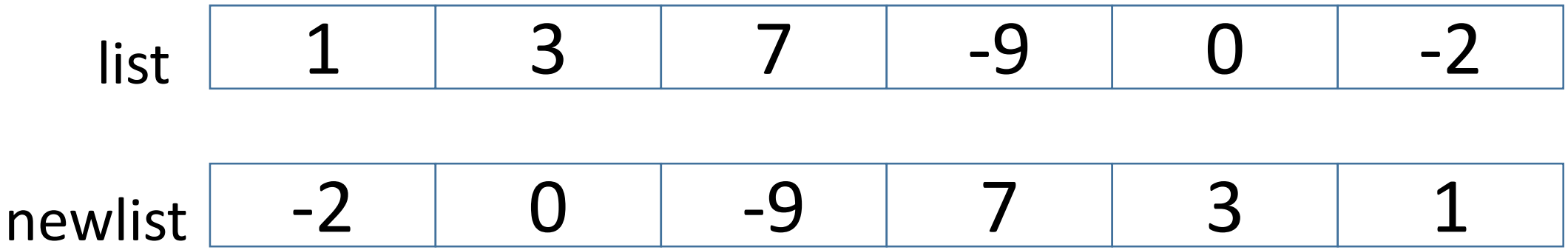
list	1	3	7	-9	0	-2

newlist	-2	0	-9	7	3	1

Reversing elements

Return a pointer (to an array).

```
int *reverse(const int list[ ],  int size)
```



Reversing elements

Return a pointer (to an array).

```
int *reverse(const int list[ ],  int size)
{
        _____(L1)_____;
   for (int i = 0, j = size - 1; i < size; i++, j--)
   {
        _____(L2)_____ ;
   }
   return newList;
}
```

(A) int *newList = new int[size]
(B) int newList[size]

list	1	3	7	-9	0	-2
newlist	-2	0	-9	7	3	1

Reversing elements

Return a pointer (to an array).

```
int *reverse(const int list[ ],  int size)
{
   int *newList = new int[ size ];
   for (int i = 0, j = size - 1; i < size; i++, j--)
   {
         ____(L2)_____ ;
   }
   return newList;
}
```

(A) int *newList = new int[size]
(B) int newList[size] **X**

list	1	3	7	-9	0	-2

newlist	-2	0	-9	7	3	1

Reversing elements

Return a pointer (to an array).

```
int *reverse(const int list[ ],  int size)
{
   int *newList = new int[ size ];
   for (int i = 0, j = size - 1; i < size; i++, j--)
   {
         ____(L2)_____ ;
   }
   return newList;
}
```

(C) newList[j] = list[i]
(D) list[j] = newList[i]

list	1	3	7	-9	0	-2

newlist	-2	0	-9	7	3	1

Reversing elements

Return a pointer (to an array).

```
int *reverse(const int list[ ],  int size)
{
   int *newList = new int[ size ];
   for (int i = 0, j = size - 1; i < size; i++, j--)
   {
      newList[ j ] = list[ i ];
   }
   return newList;
}
```

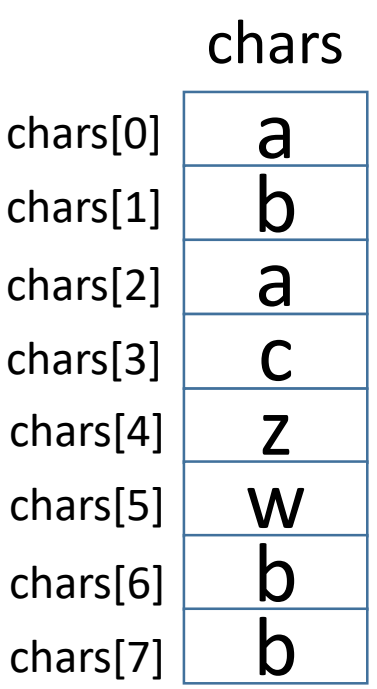
(C) newList[j] = list[i]
(D) list[j] = newList[i]

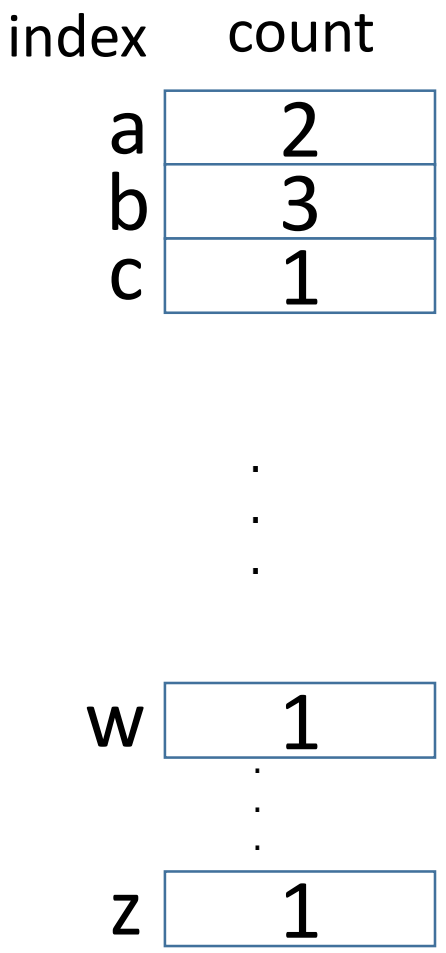
list	1	3	7	-9	0	-2

newlist	-2	0	-9	7	3	1

Problem: Counting Occurrence of an ordinal value

Ordinal data type: its values can be counted. The values can be mapped to the positive integers in an one-to-one manner.





Count the occurrence of each letter in the array.

Problem: Counting Occurrence of an ordinal value

Ordinal data type: its values can be counted. The values can be mapped to the positive integers in an one-to-one manner.

Mapping letters to non-negative integers
int index = letter – ‘a’;

Mapping letters to non-negative integers
index + ‘a’ = letter;

Mapping non-negative integers to letters
char letter = index + ‘a’;

mapping ‘a’ -> 0, ‘b’ -> 1, ‘c’ -> 2, …, ‘z’ -> 25

	chars
chars[0]	a
chars[1]	b
chars[2]	a
chars[3]	c
chars[4]	z
chars[5]	w
chars[6]	b
chars[7]	b

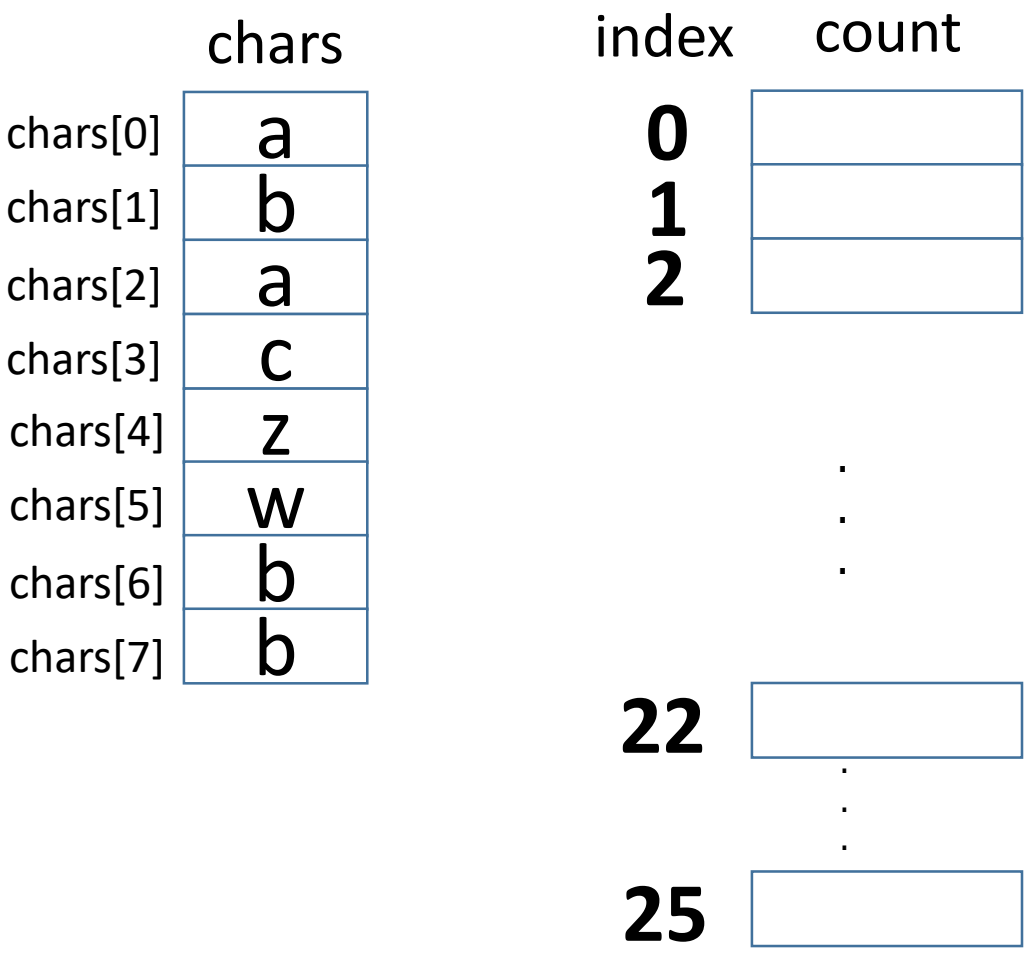
Count the occurrence of each letter in the array.

index		count
0	a	2
1	b	3
2	c	1
.		
.		
.		
22	w	1
.		
.		
.		
25	z	1

Problem: Counting Occurrence of an ordinal value

Ordinal data type: its values can be counted. The values can be mapped to the positive integers in an one-to-one manner.

Mapping letters to non-negative integers
int index = letter – ‘a’;



Problem:

- Generate 1000 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.

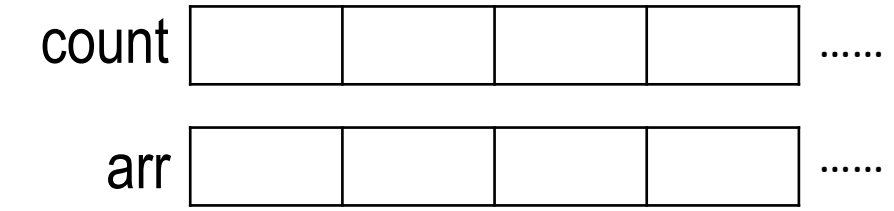
Problem: Counting Occurrence of an ordinal value

Ordinal data type: its values can be counted.
The values can be mapped to the positive integers in an one-to-one manner.

Mapping letters to non-negative integers
int index = letter – ‘a’;

Problem:

- Generate 1000 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.

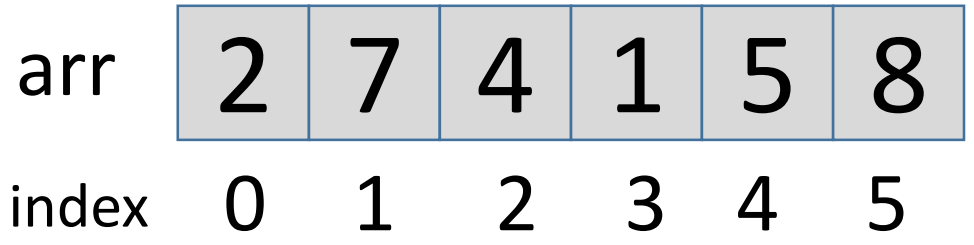


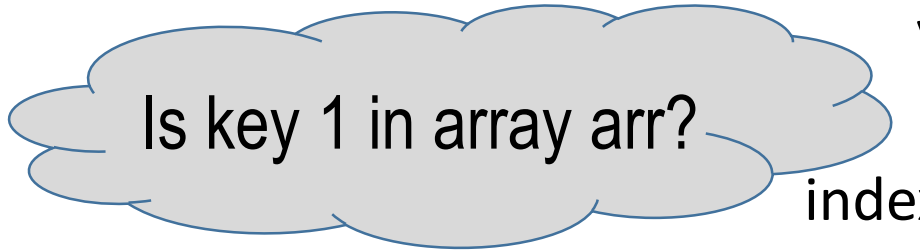
Set each element of count to zero.
Generate the letters in array arr

For index variable i, i runs over from 0 to 999

```
// convert arr[ i ] into an integer inside [0,25]
char letter = arr[ i ] ;
int index = letter – ‘a’;

// increase the counter for the letter
// that is mapped to index.
++count[index];
```

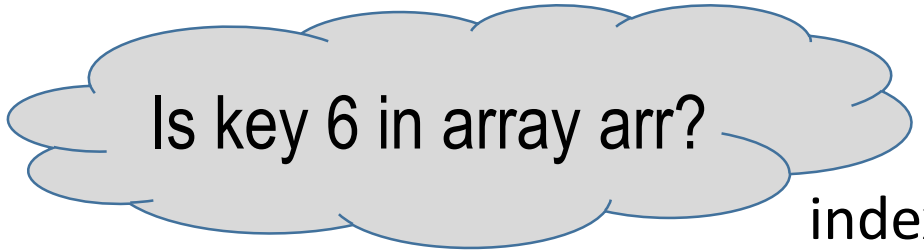
Searching Arrays





yes

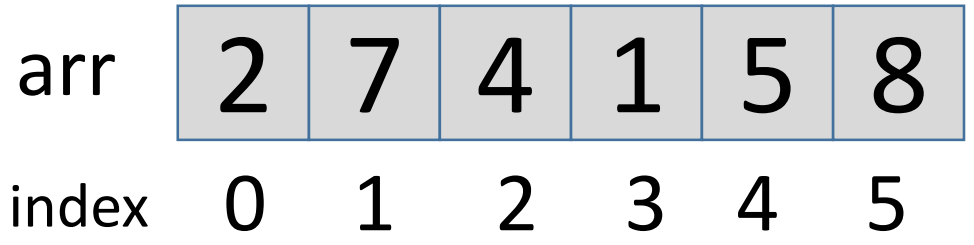
index = 3

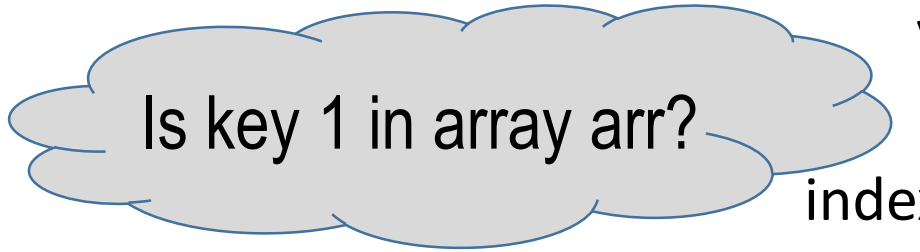


no

index = -1

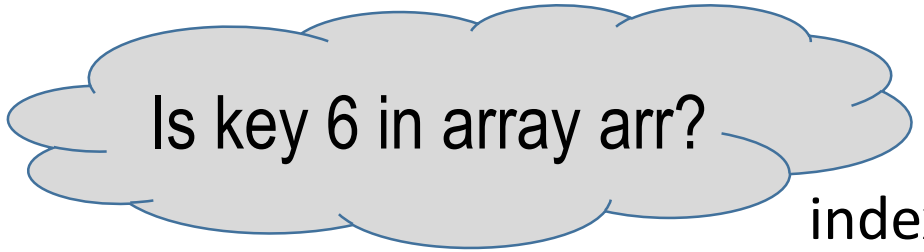
Searching Arrays





yes

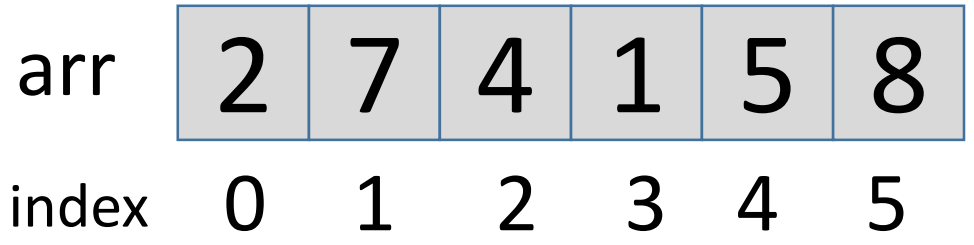
index = 3

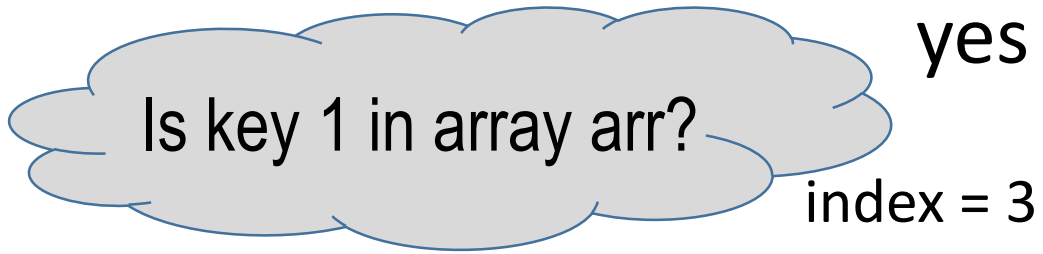


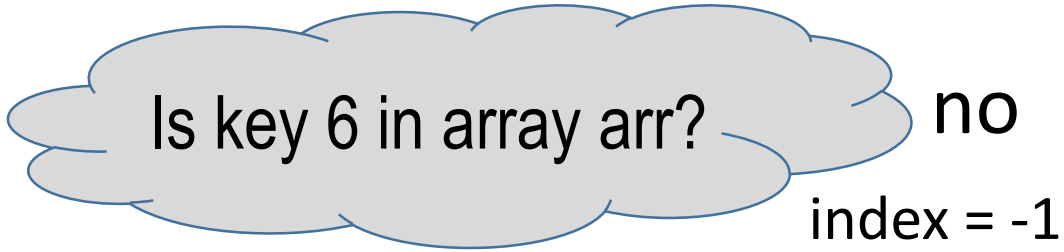
no

index = -1

Searching Arrays



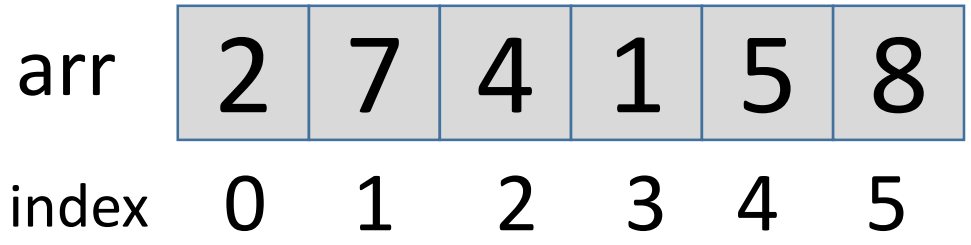


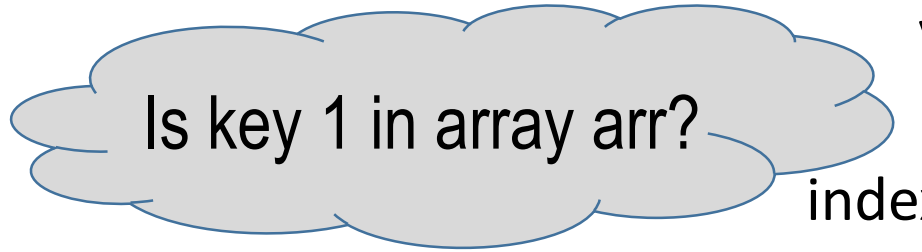


Note: the index variable is non-negative.

Searching Arrays

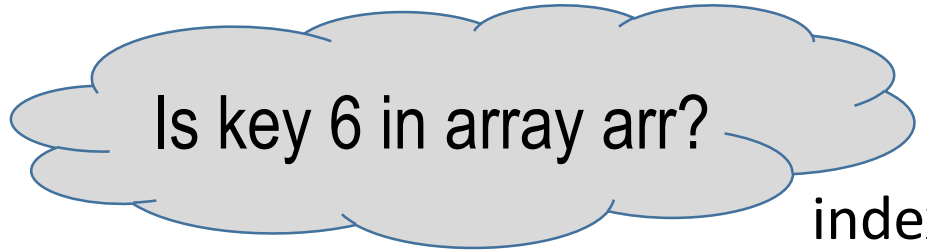
- Look for a specific element in an array.
- Linear search: search for an element by comparing elements one by one.
- Binary search: search for an element while discarding a half of the elements each time. This is achieved by comparing the middle element. The elements must be sorted beforehand.





yes

index = 3



no

index = -1

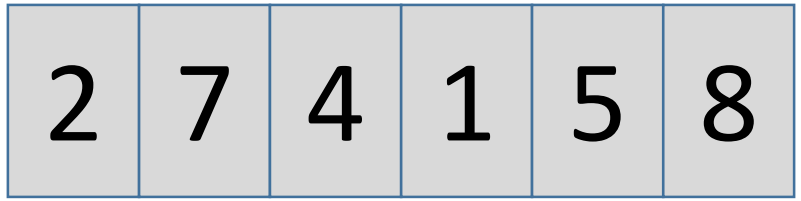
Note: the index variable is non-negative.

Linear Search

The linear search approach compares key *sequentially* with each element of arr.

```
int linearSearch(
const int arr[ ]
, int key
, int arraySize ) {
        int index = -1;
        for ( int i = 0; i < arraySize; ++i) {
                if (arr[ i ] != key) continue;
                index = i;
                break;
        }
        return index;
}
```

index = linearSearch(myArr, 5, 6)



Best case: the first element is the key.

Worst case: the array does not have the key. In this case, all the elements are checked.

Binary Search

The elements in the array must already be ordered (sorted).

The binary search first compares the key with the middle element in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:

1 2 4 8 12 13 43 51 71

The following elements are sorted in **descending order**:

71 51 43 13 12 8 4 2 1

Binary Search

The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:

Binary Search

The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:

1	2	4	8	12	13	43	51	71

Binary Search

The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:

1	2	4	8	12	13	43	51	71

key = 51

Binary Search

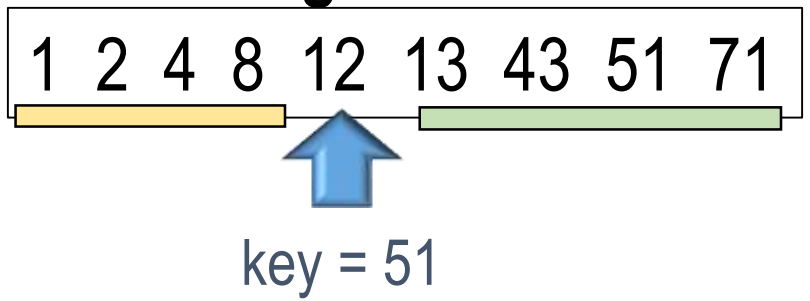
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

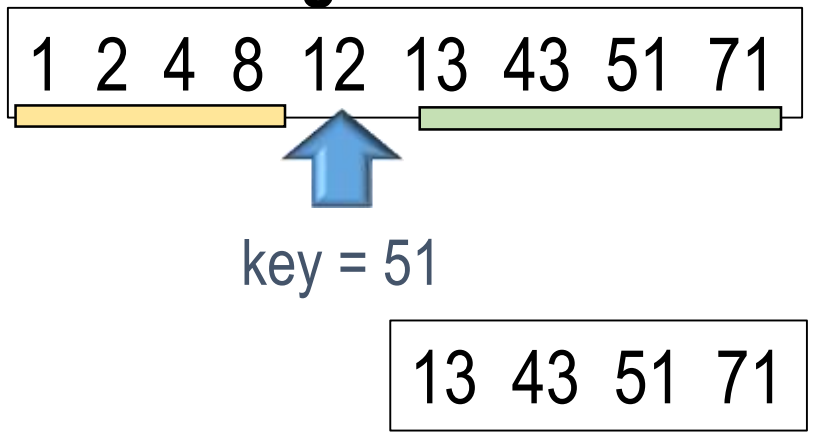
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

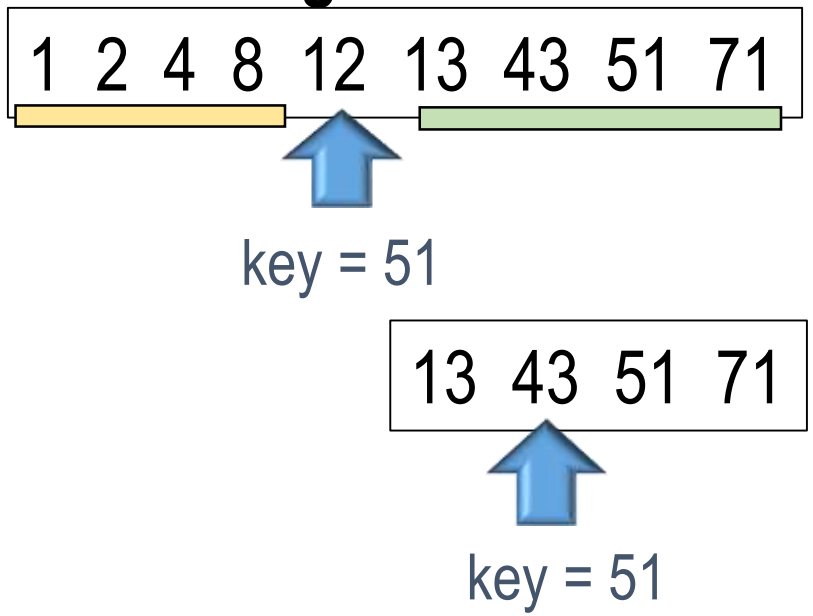
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

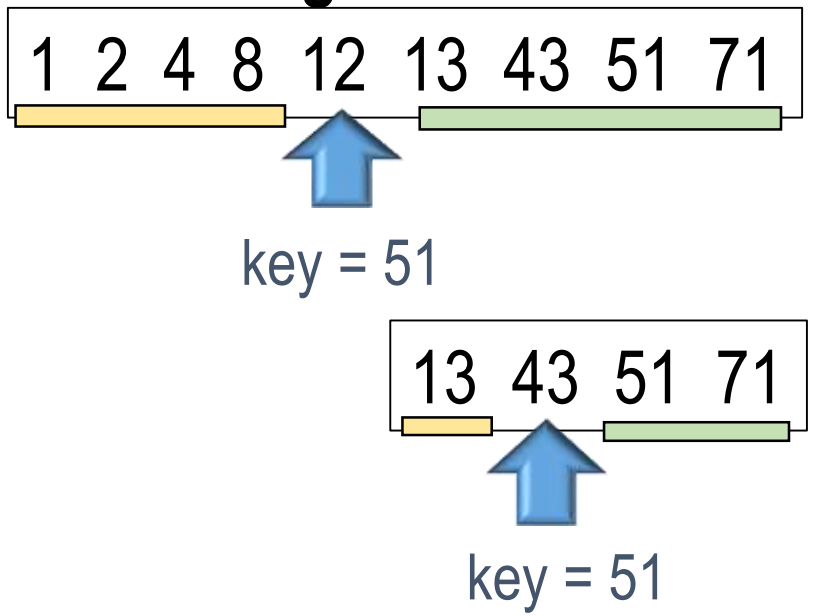
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

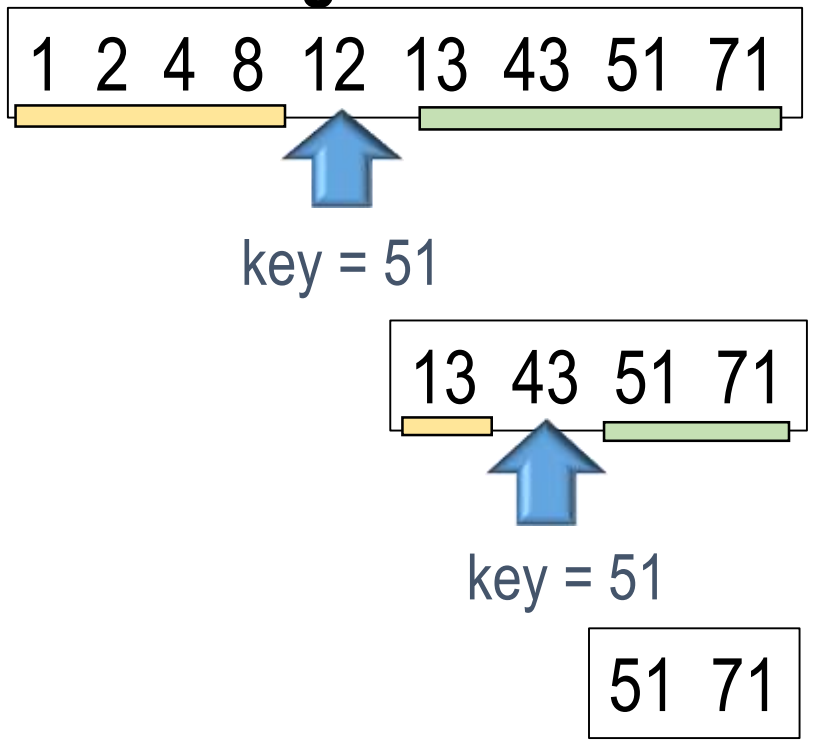
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

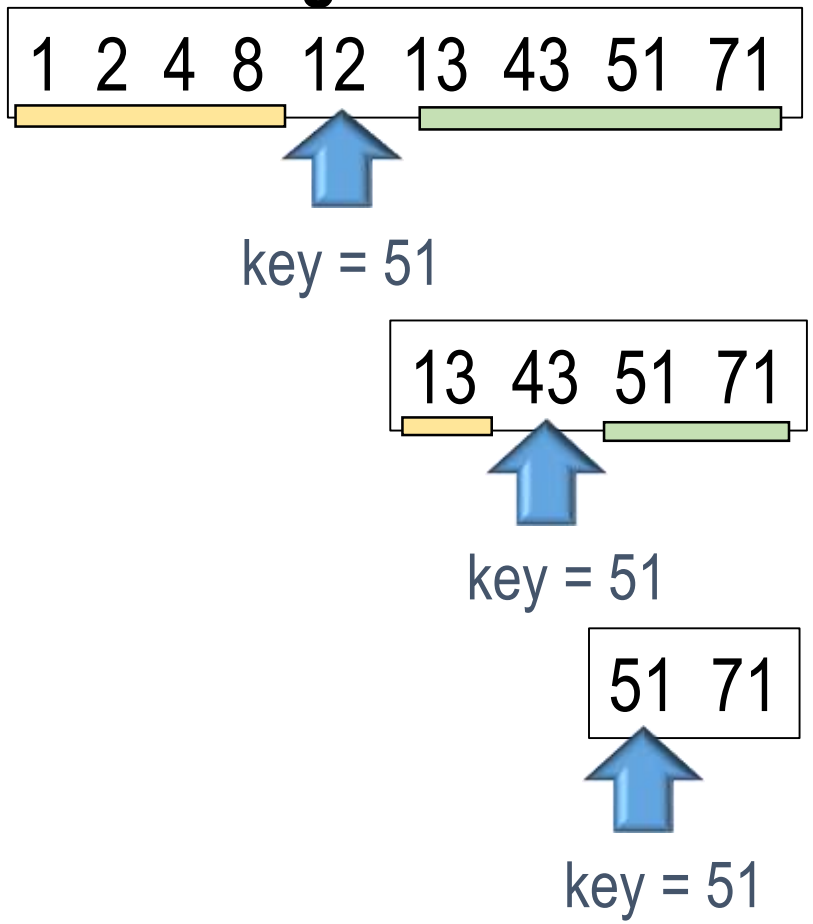
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

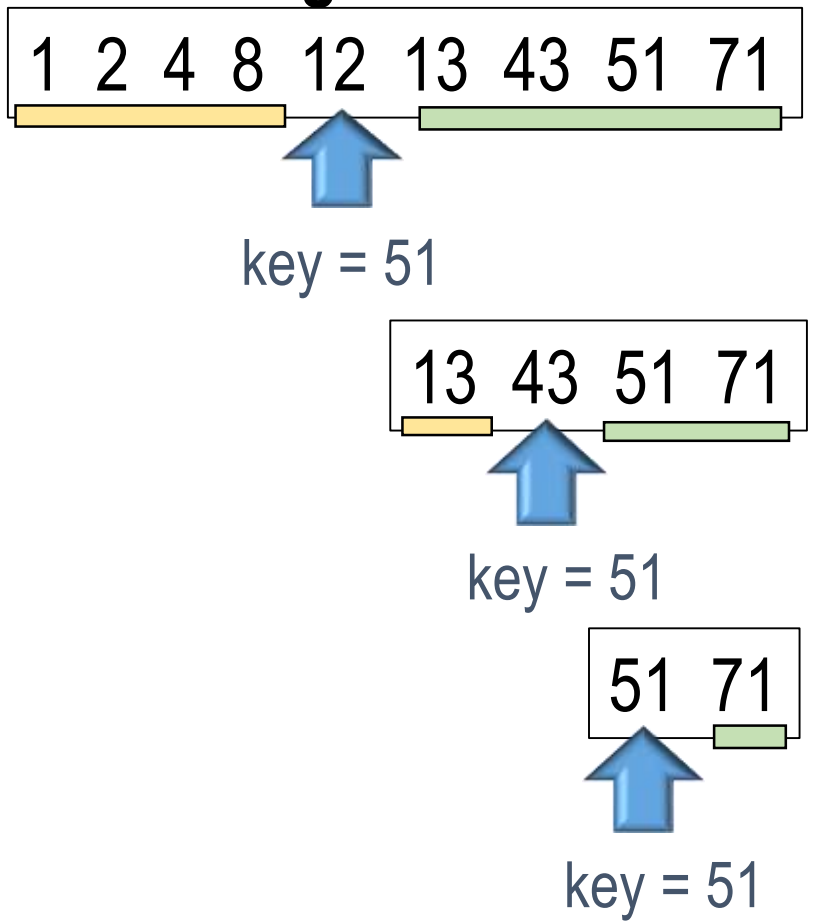
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:

1	2	4	8	12	13	43	51	71

Binary Search

The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:

1	2	4	8	12	13	43	51	71

key = 3

Binary Search

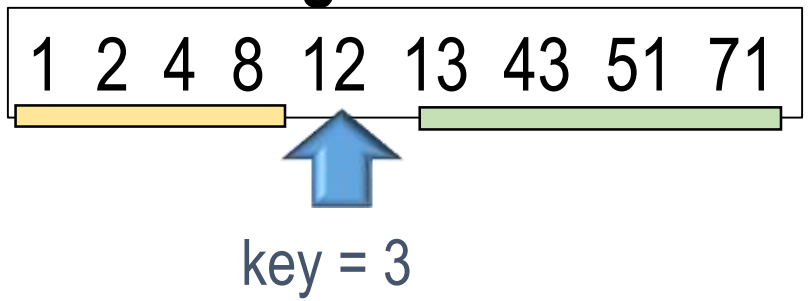
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

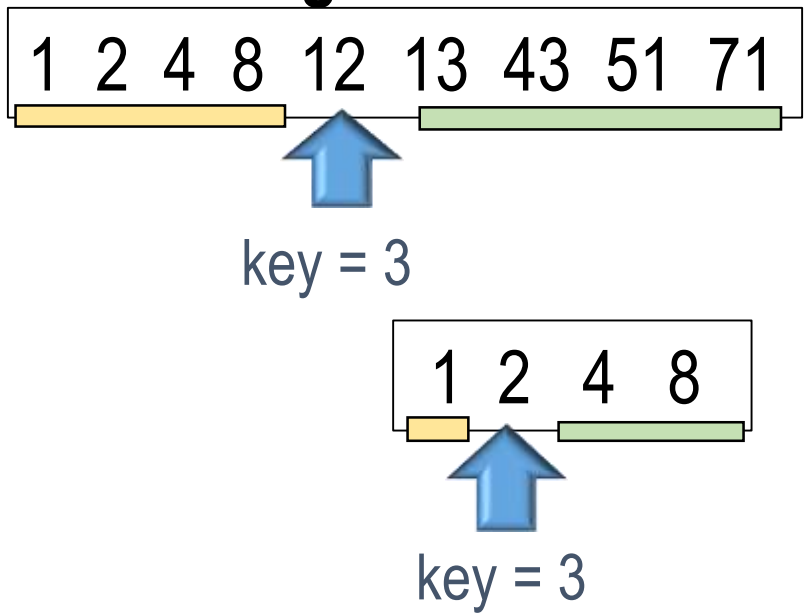
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

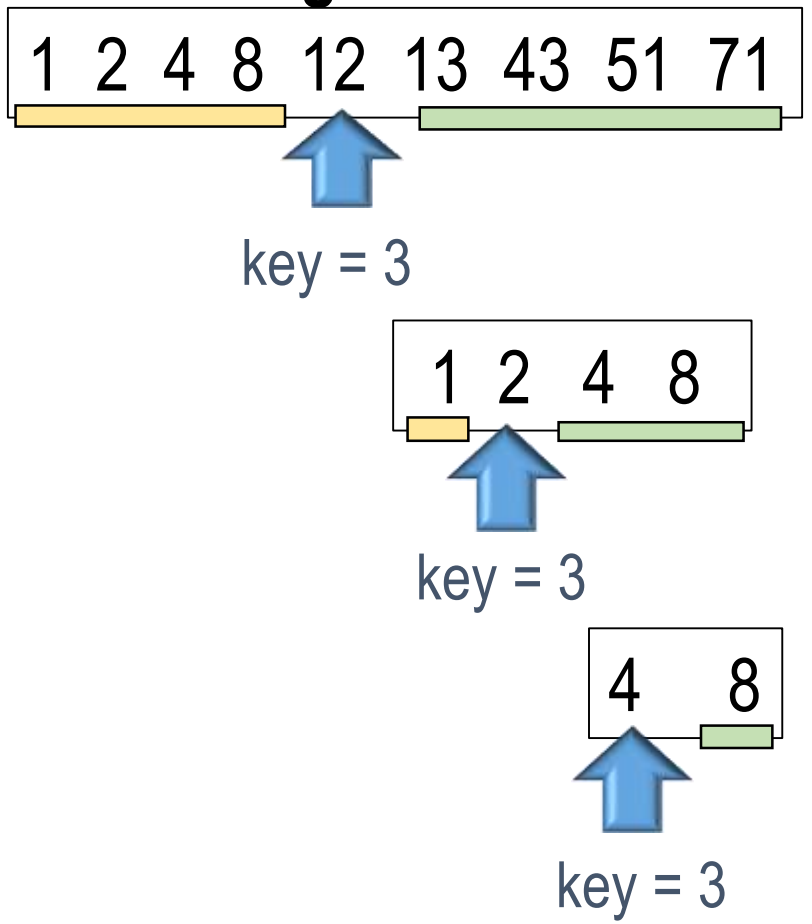
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Binary Search

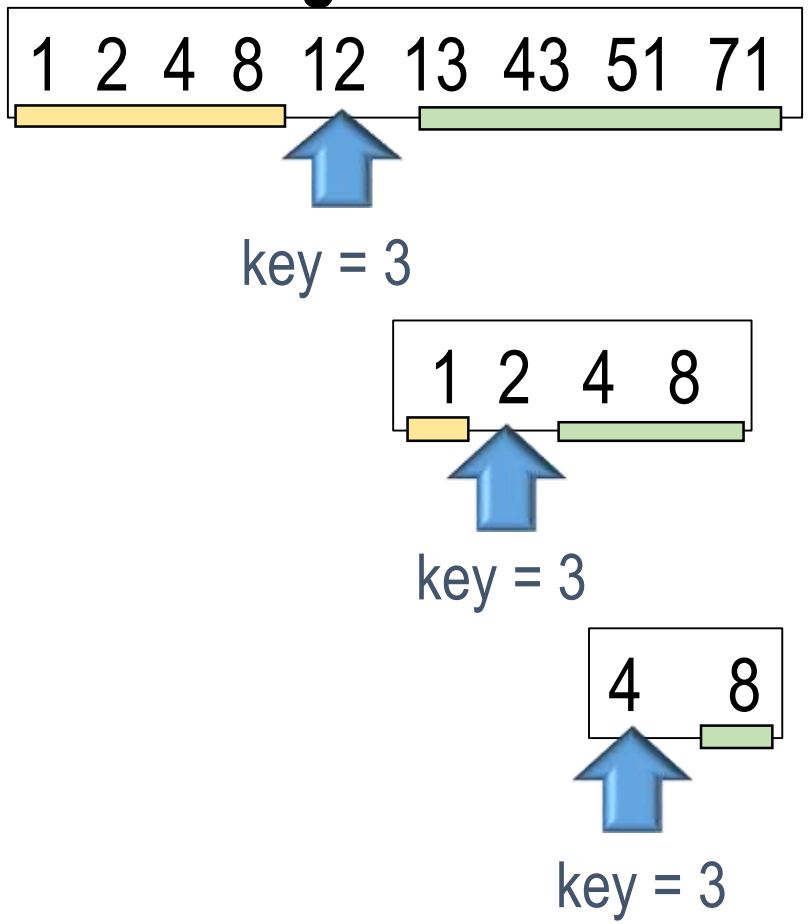
The elements in the array must already be ordered (sorted).

The binary search first compares the key with the **middle element** in the array.

Middle element

index = (n – 1) / 2

The following elements are sorted in **ascending order**:



Sorting Arrays

A sorting algorithm puts elements in a certain order.

- Unsorted: 4 3 1 2 5
- Ascending order: 1 2 3 4 5
- Descending order: 5 4 3 2 1

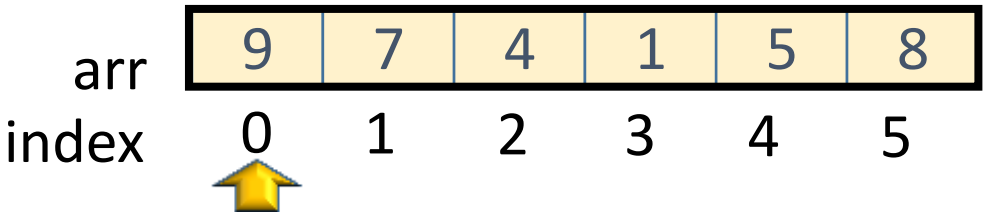
Learn two sorting methods:

1. Selection sort

2. Insertion sort

Selection Sort

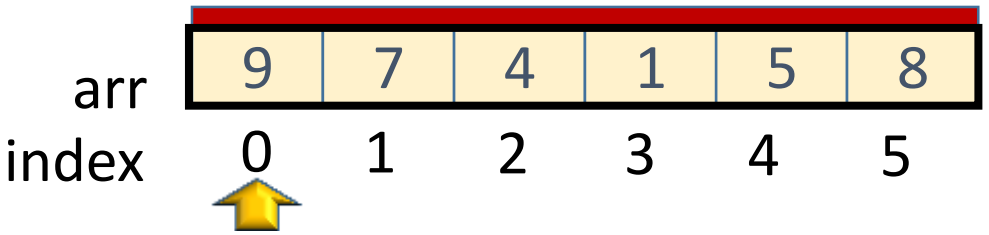
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

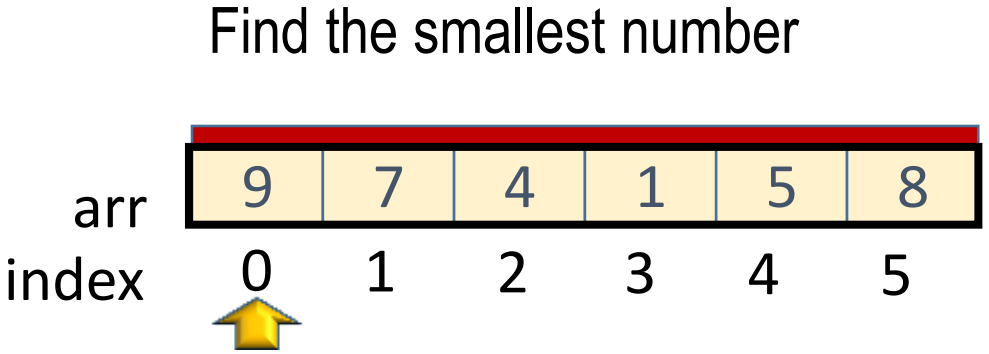
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

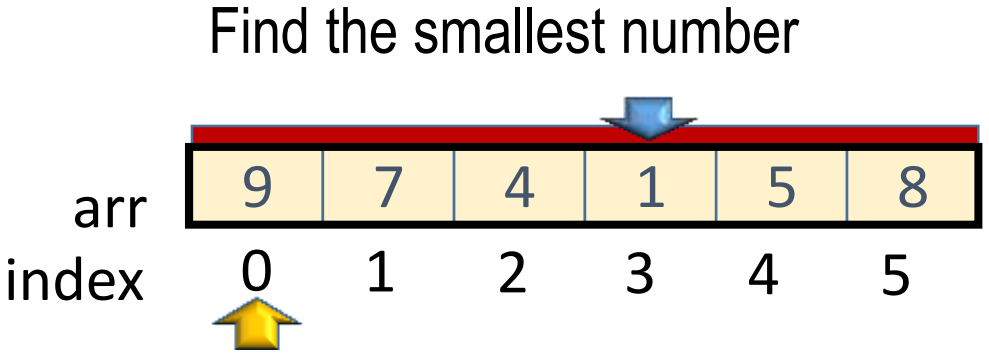
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

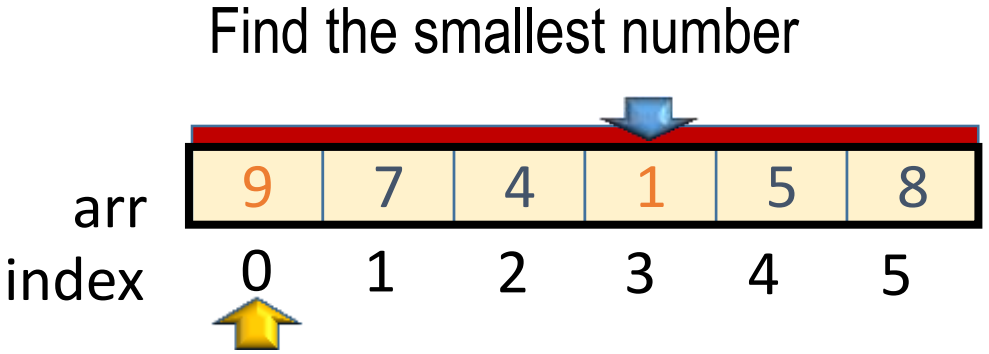
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.



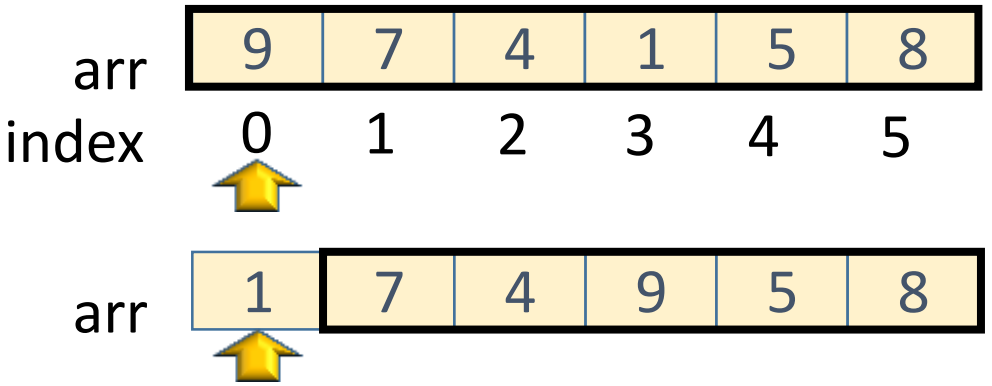

```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

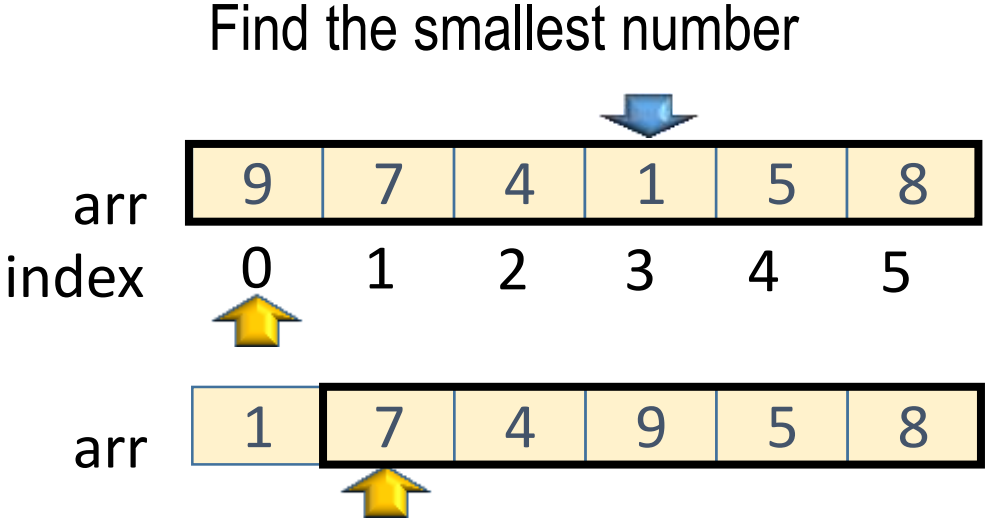
```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
    ⇨   i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Find the smallest number



Selection Sort

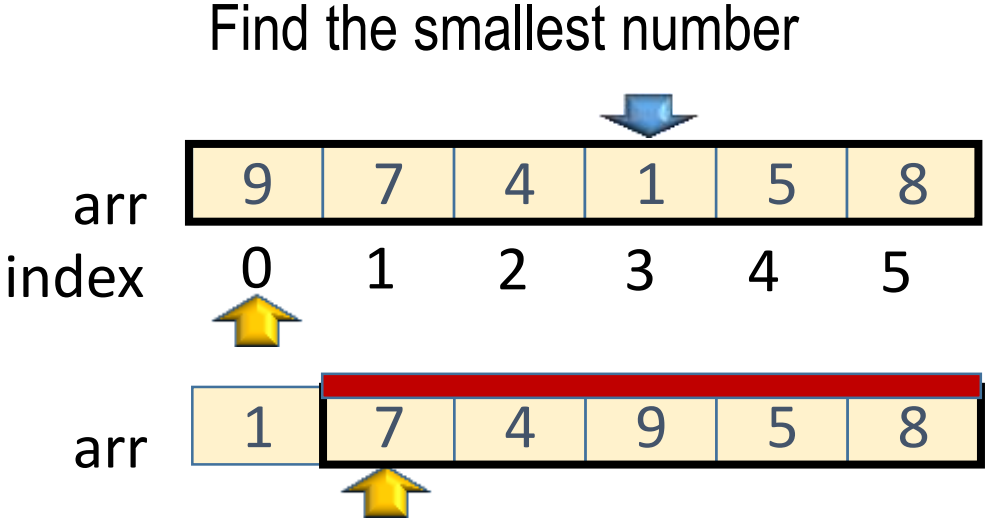
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
    ⇨   i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

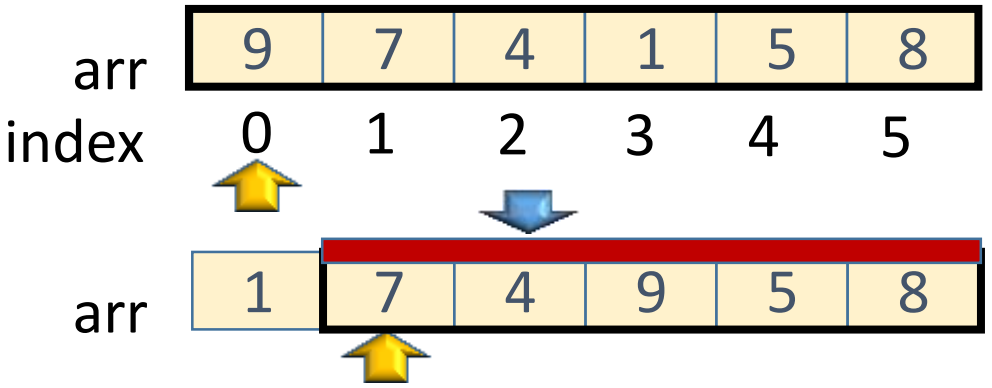



```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
    ⇨   i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

Find the smallest number

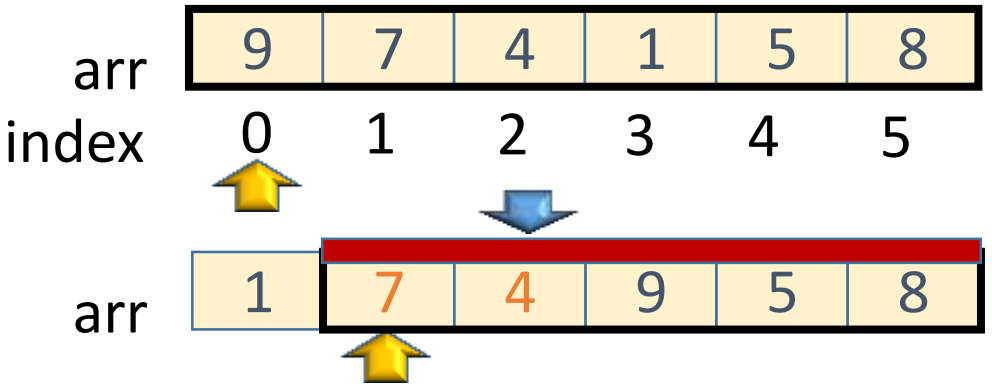



```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
    ⇨   i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

Find the smallest number

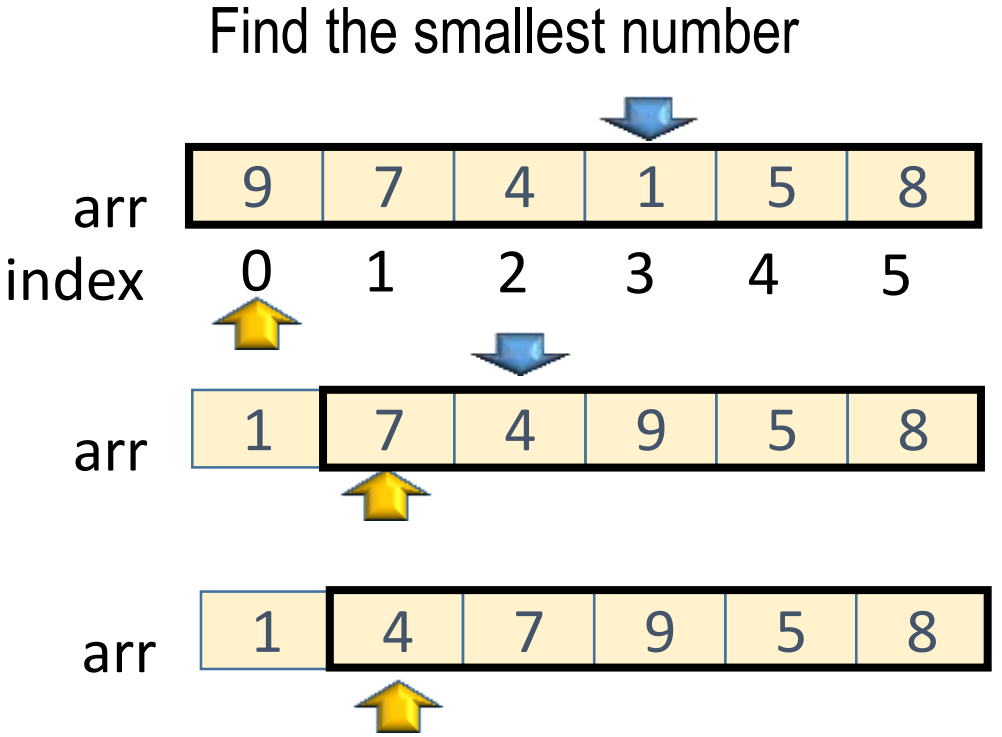



```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
    ⇨   i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

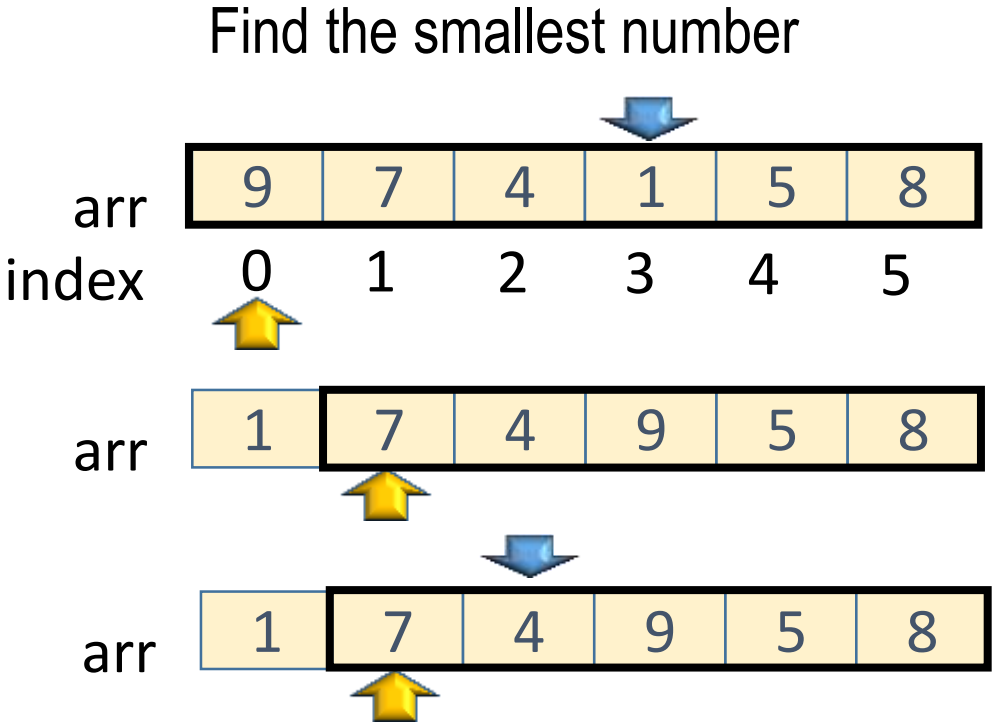
```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```



Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

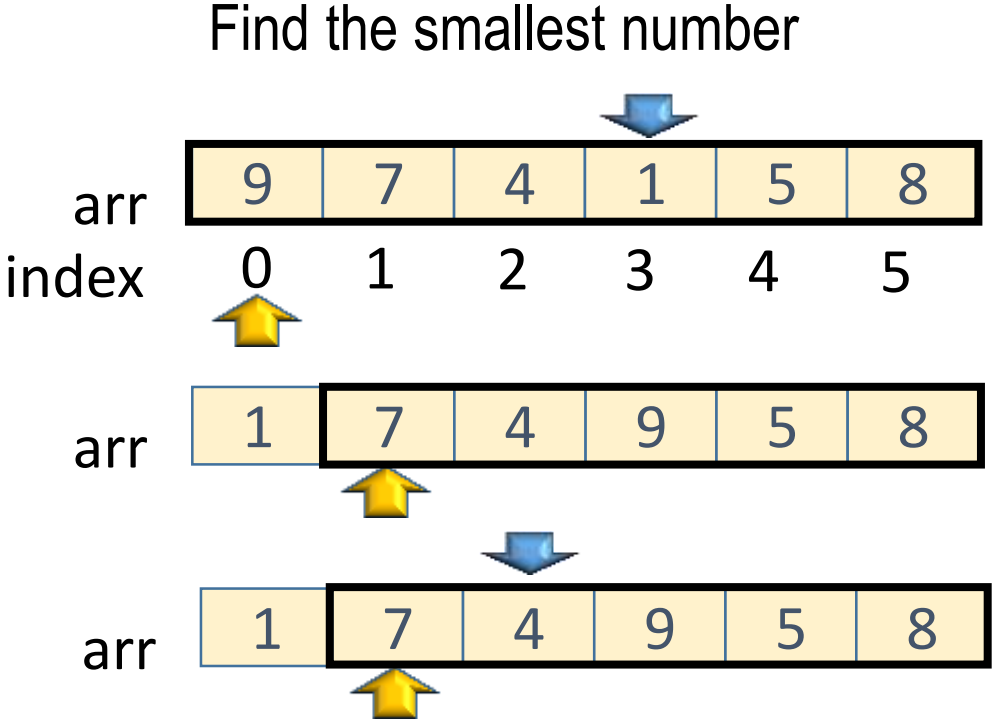
```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```



Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

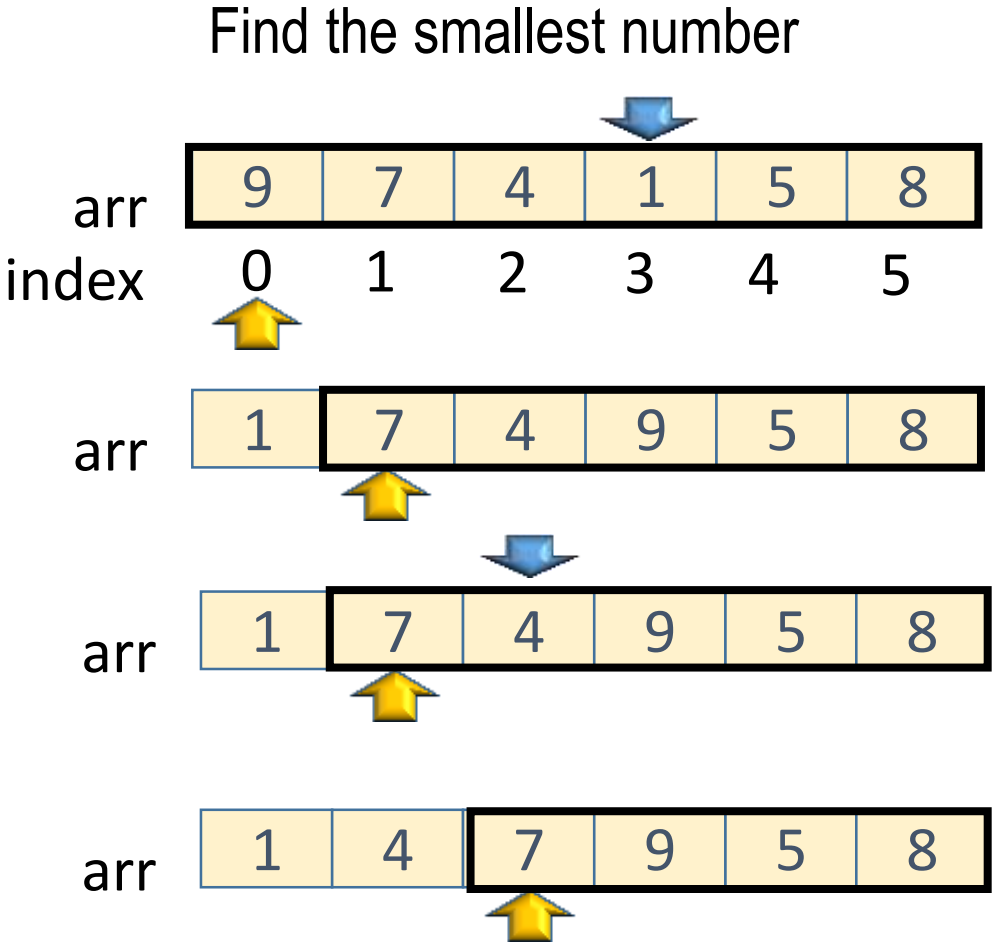
```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
    ⇨   i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```



Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

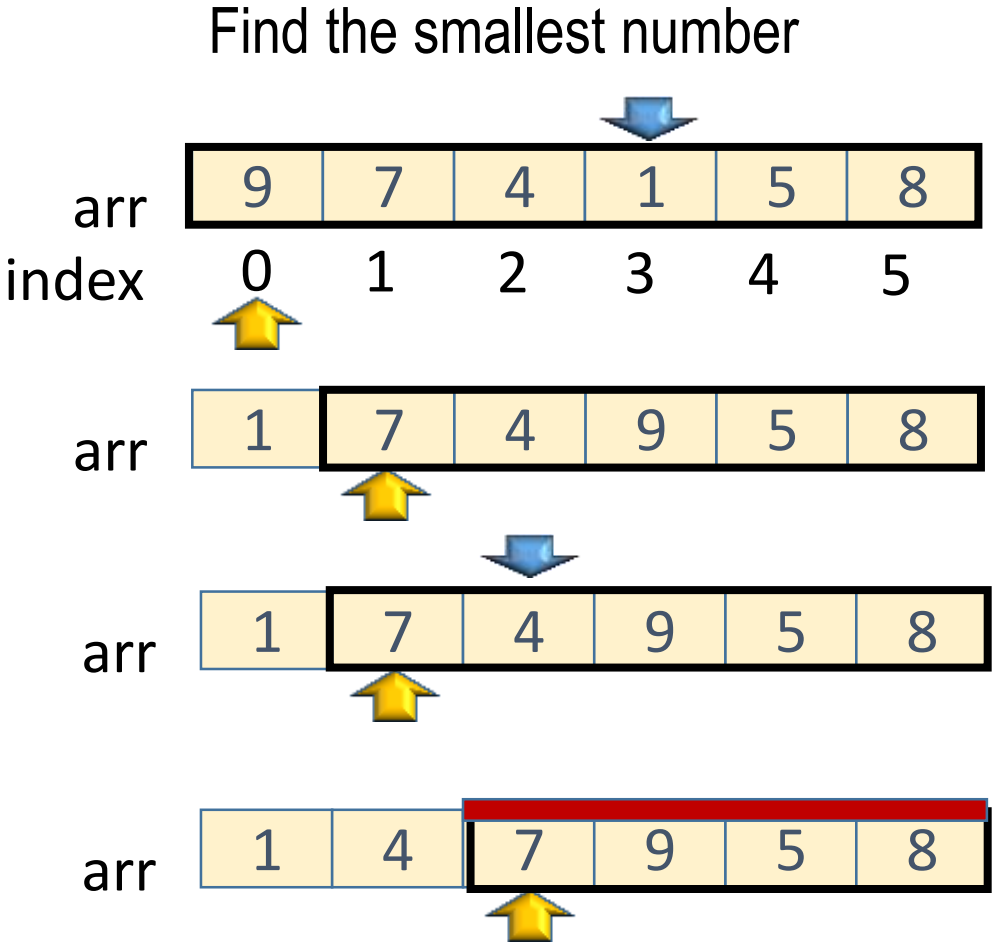
```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```



Selection Sort

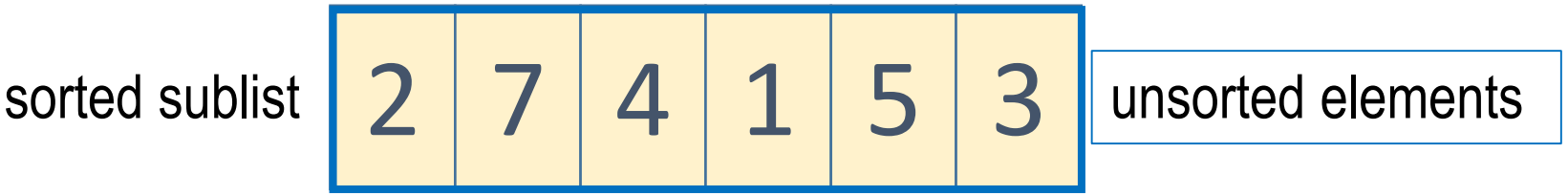
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.

```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i ← i + 1
Repeat if  i != (NUM-1)
Here, NUM is the number of the elements.
```



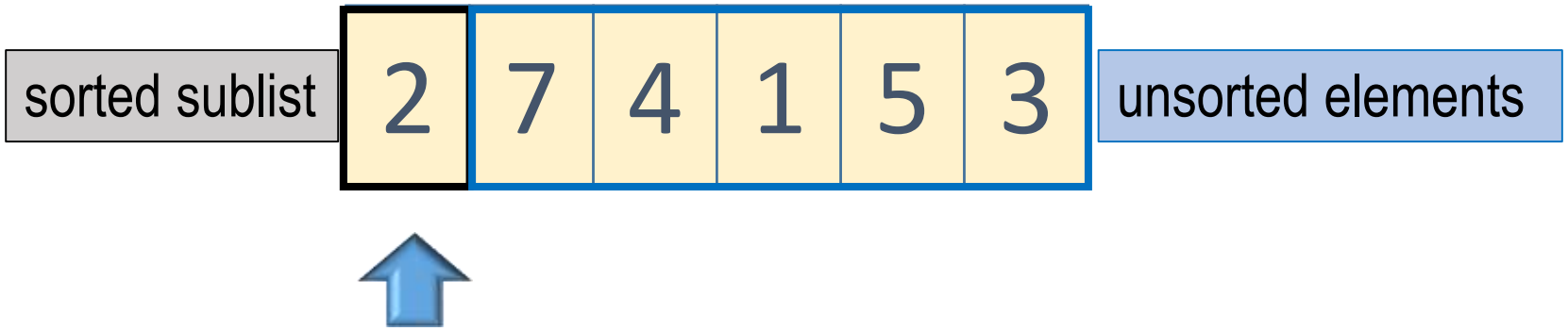
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



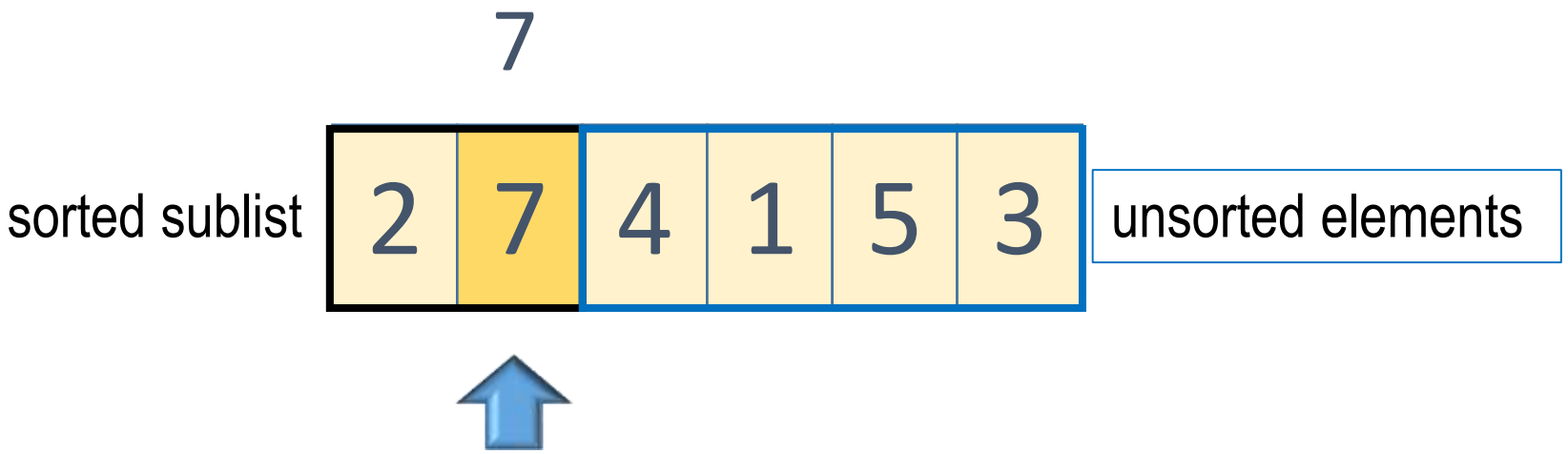
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



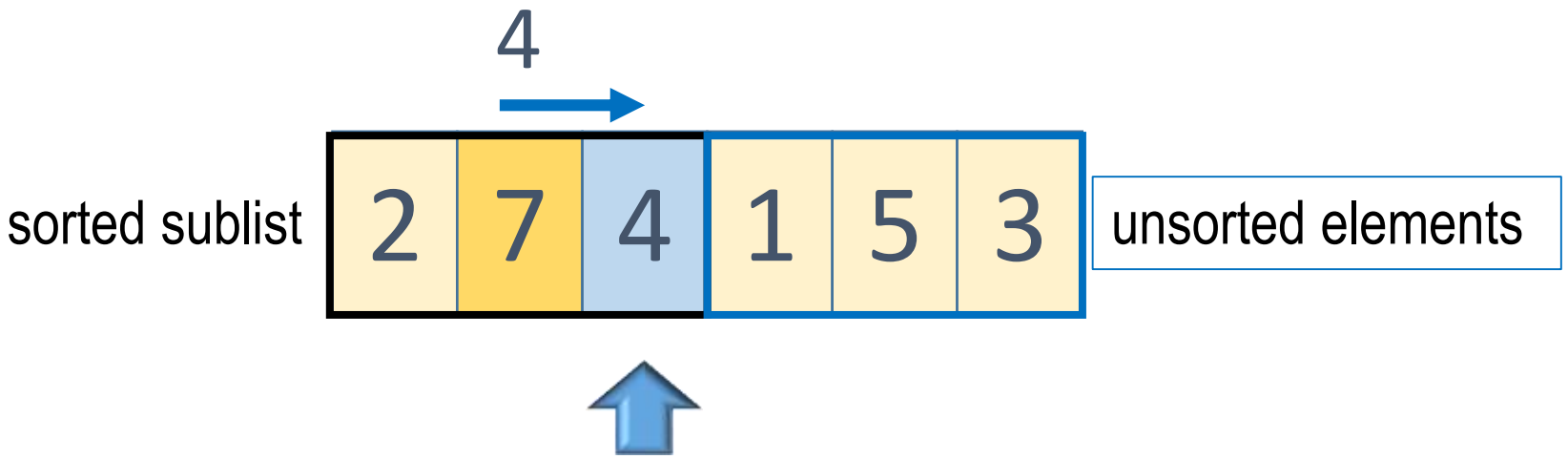
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



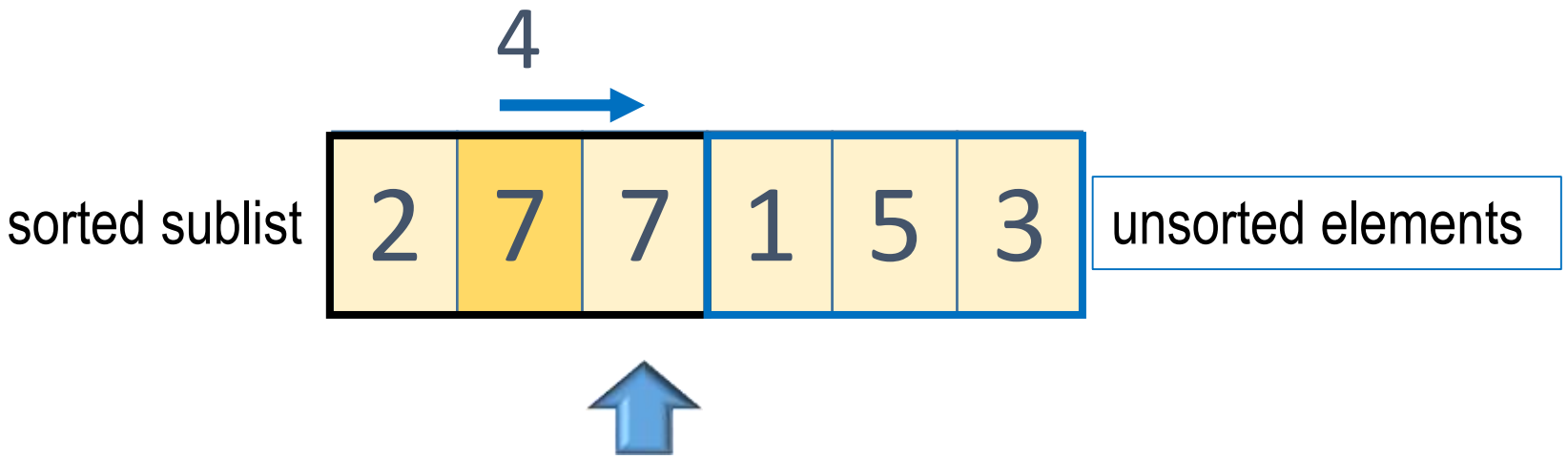
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



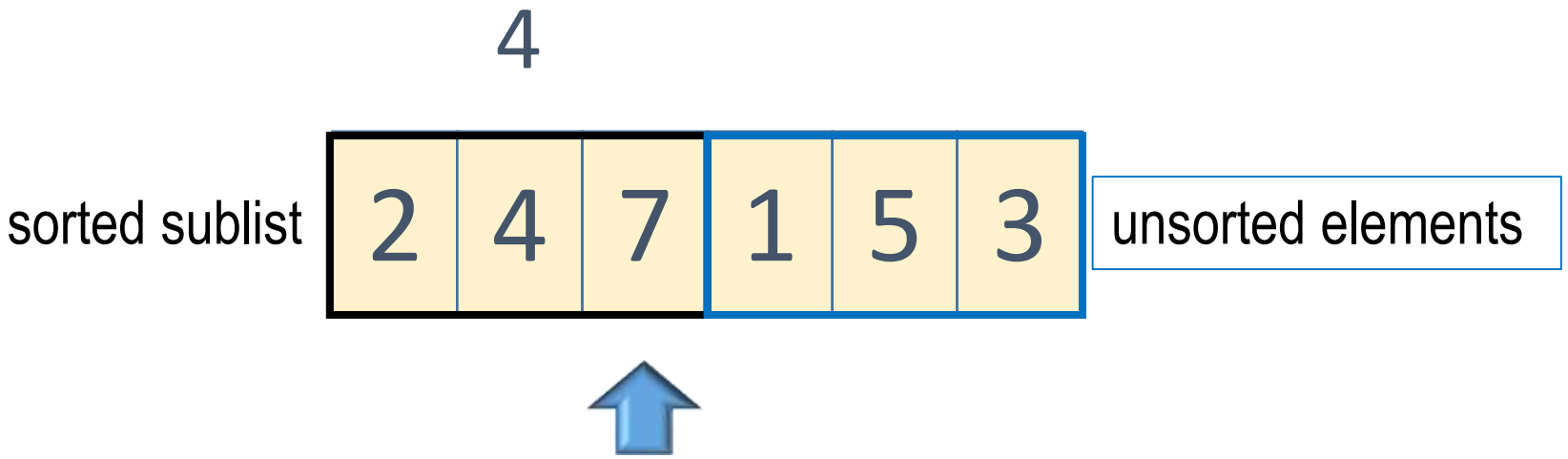
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



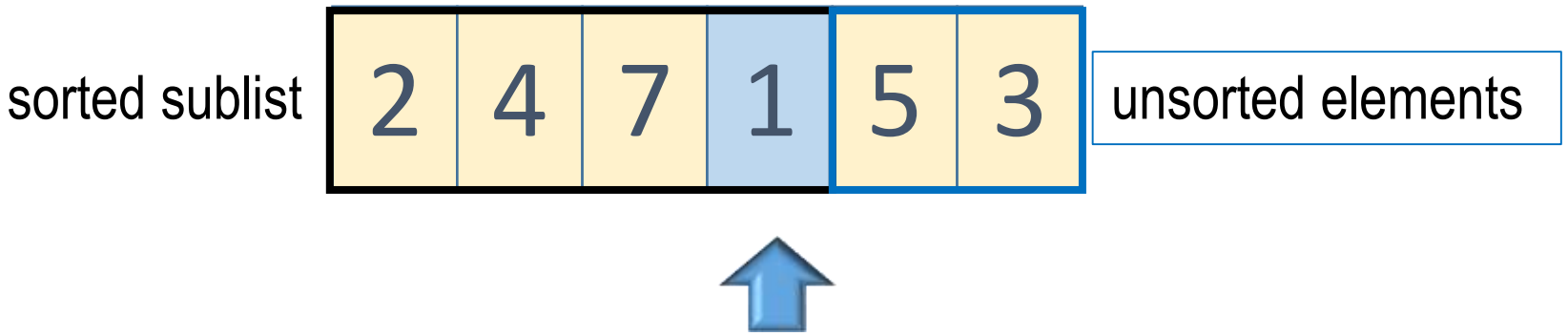
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



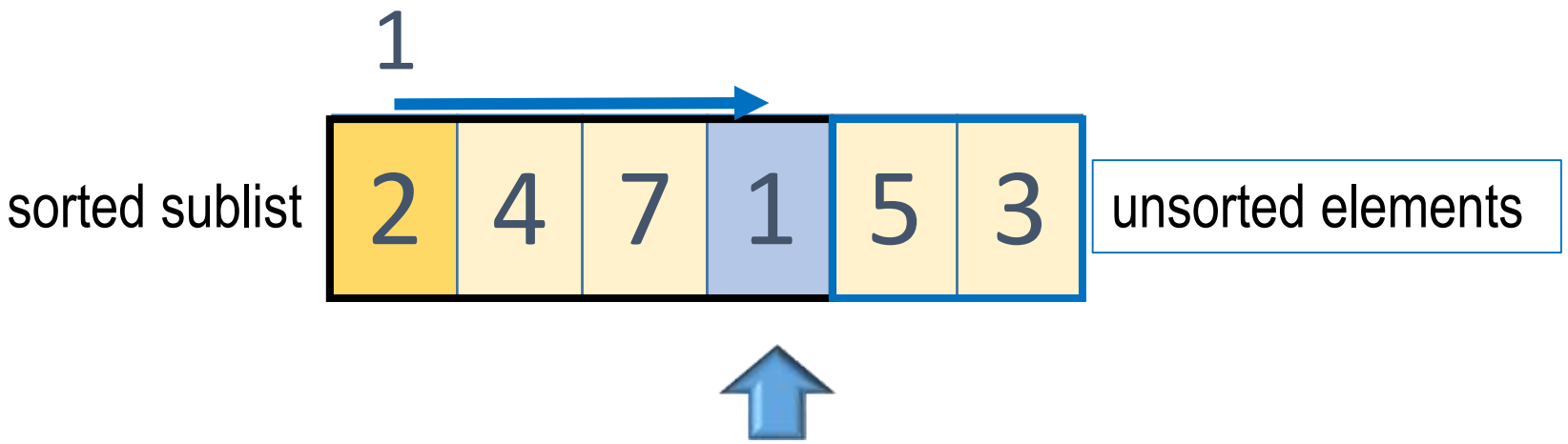
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



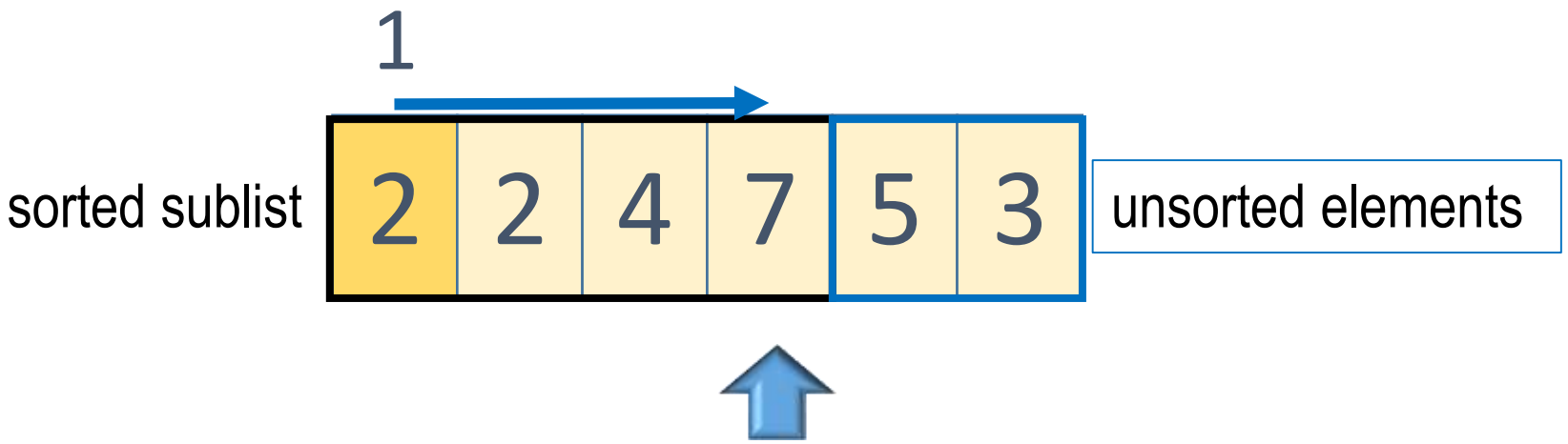
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



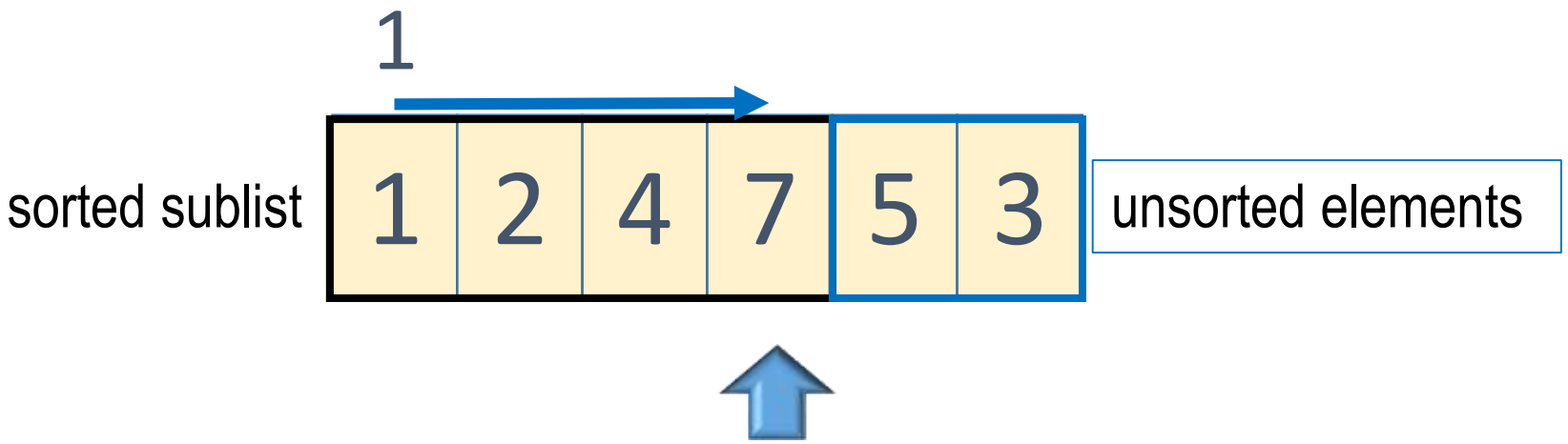
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



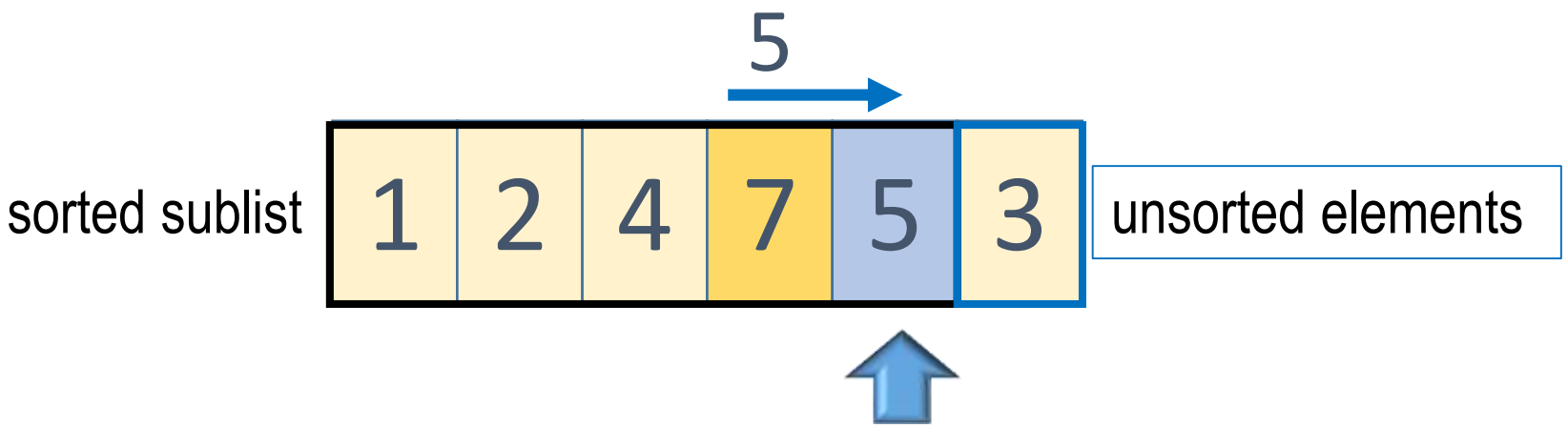
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



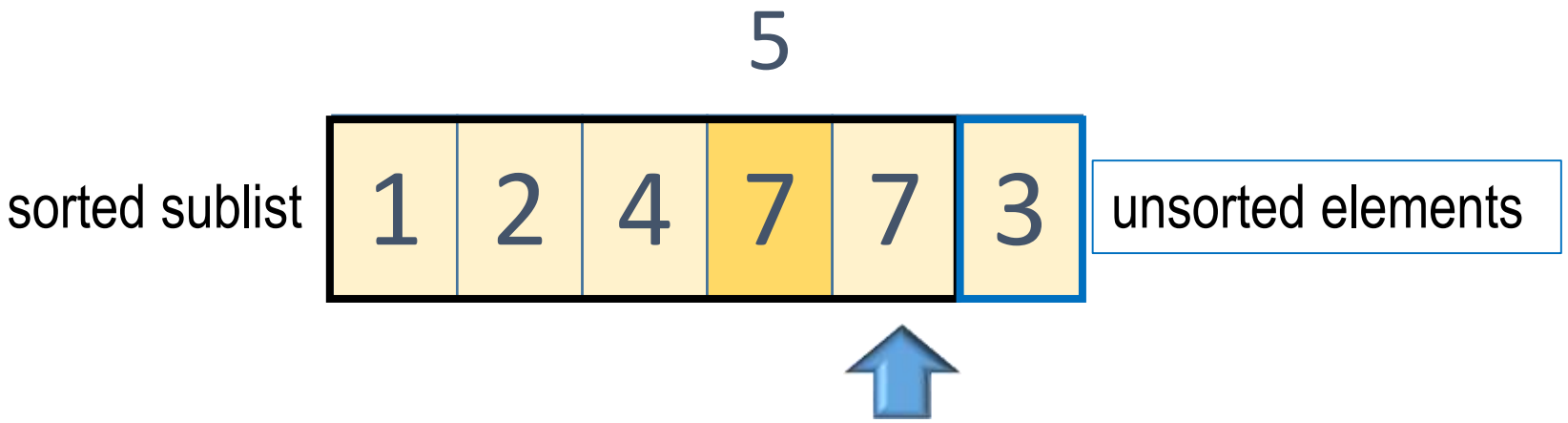
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



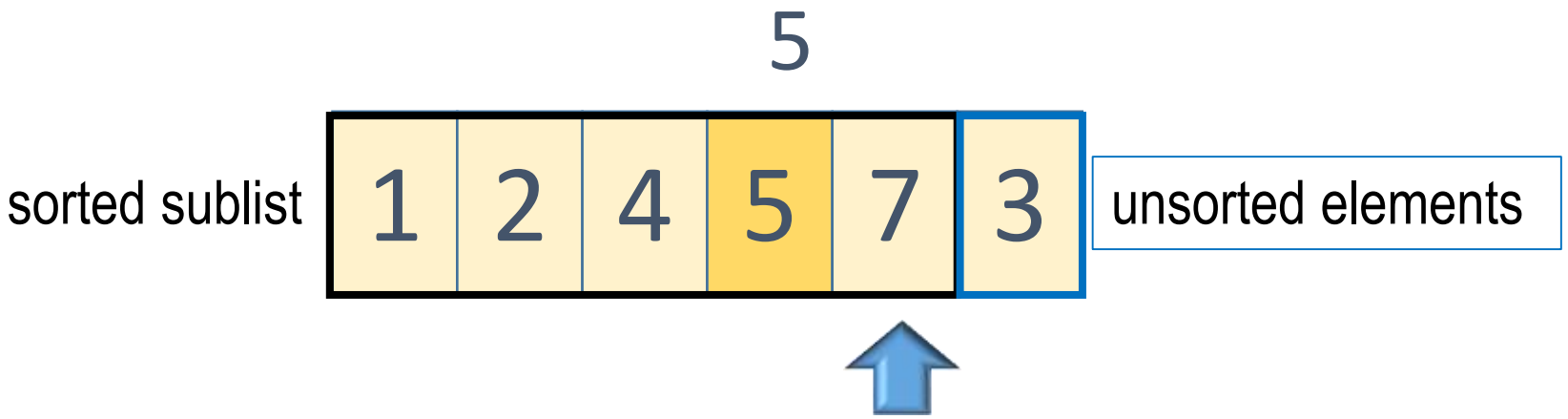
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



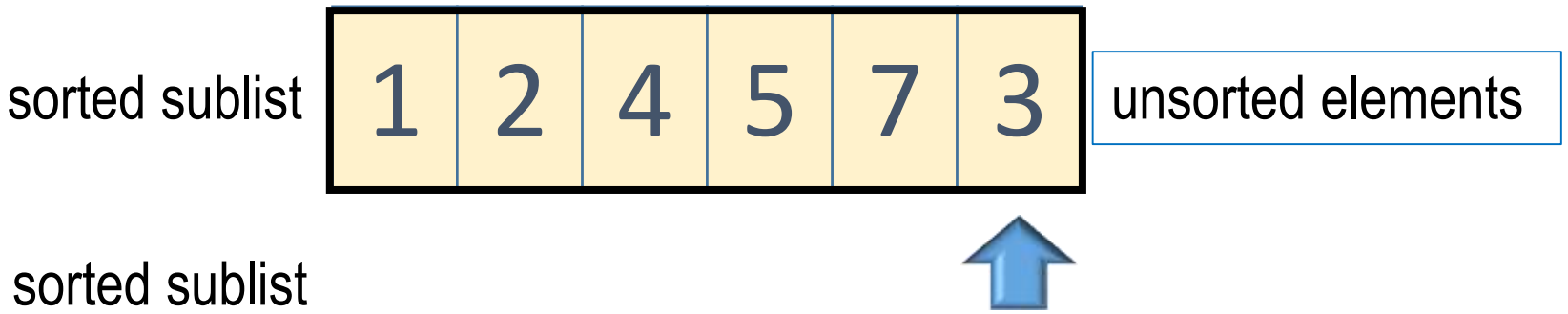
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



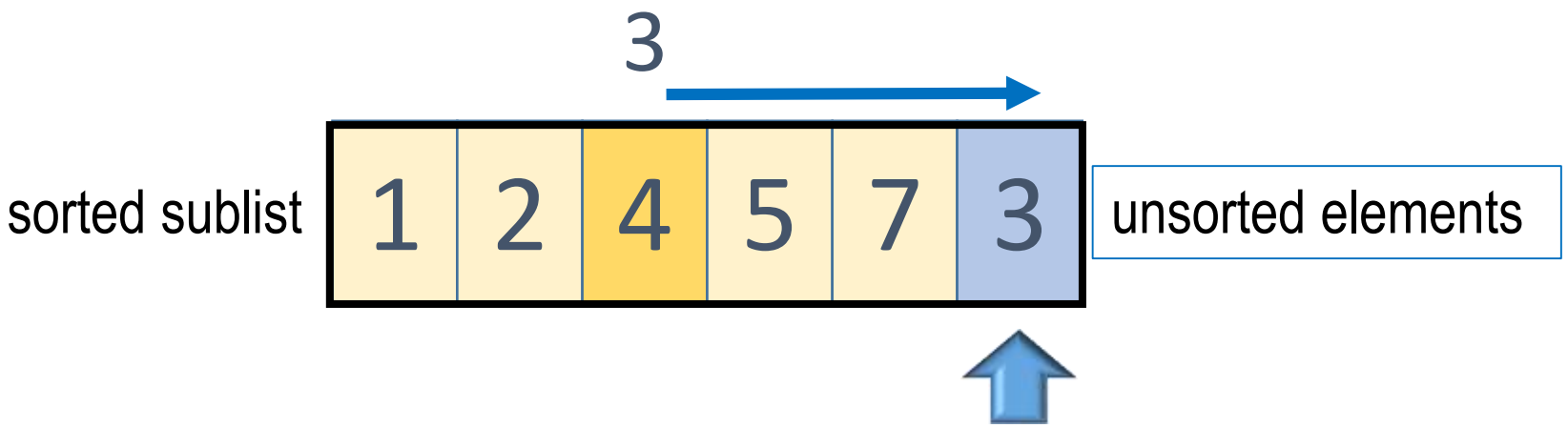
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



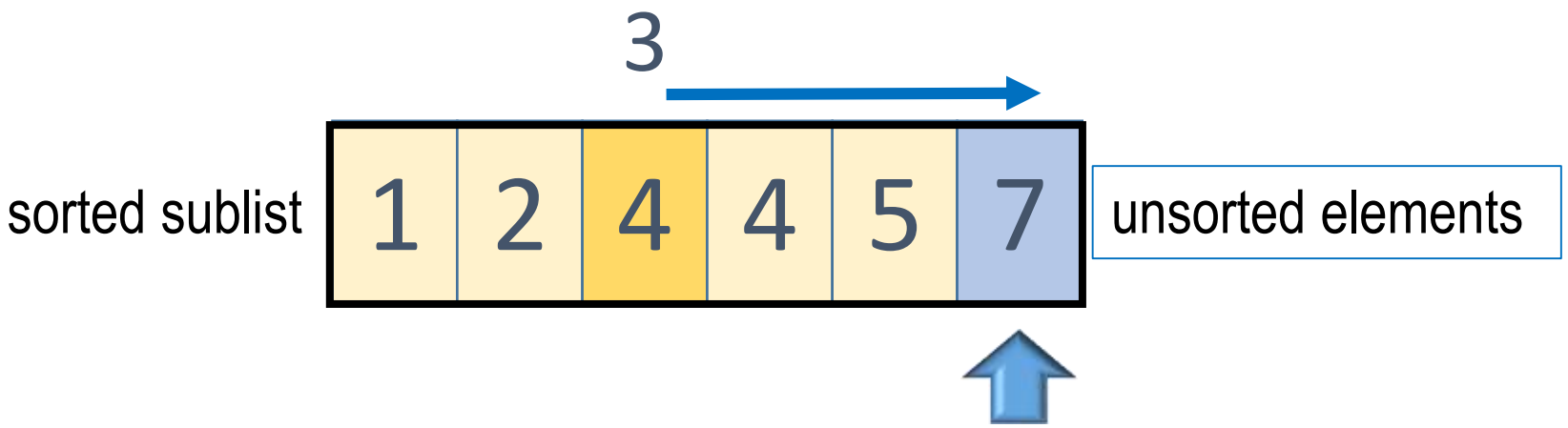
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



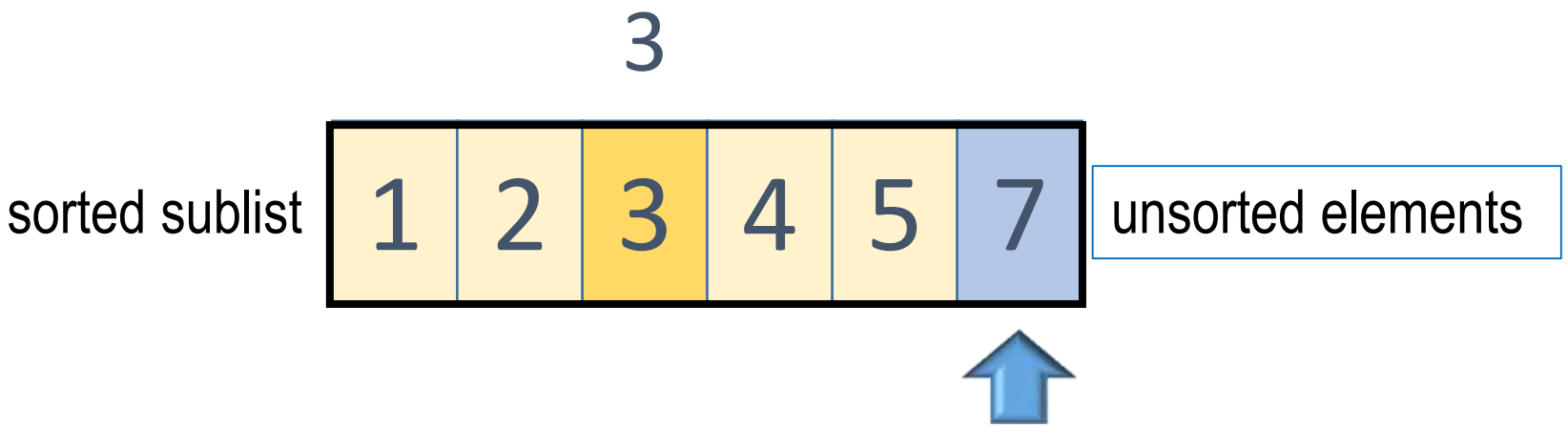
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



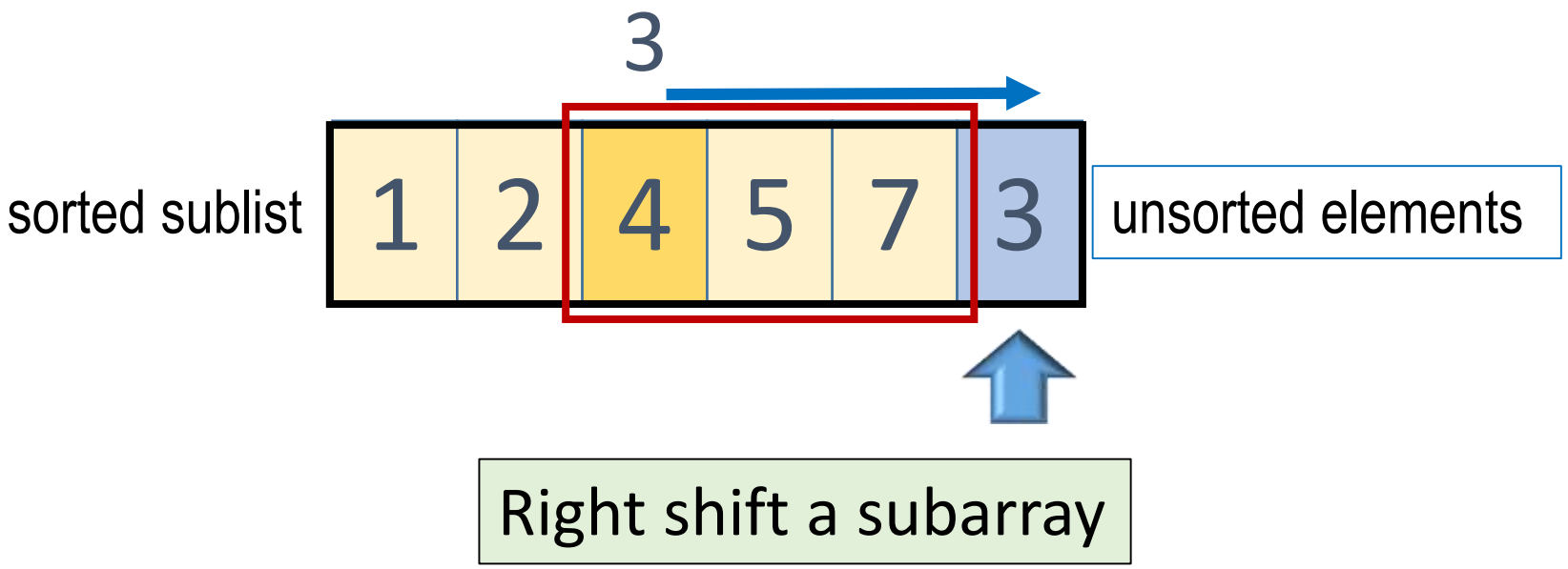
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



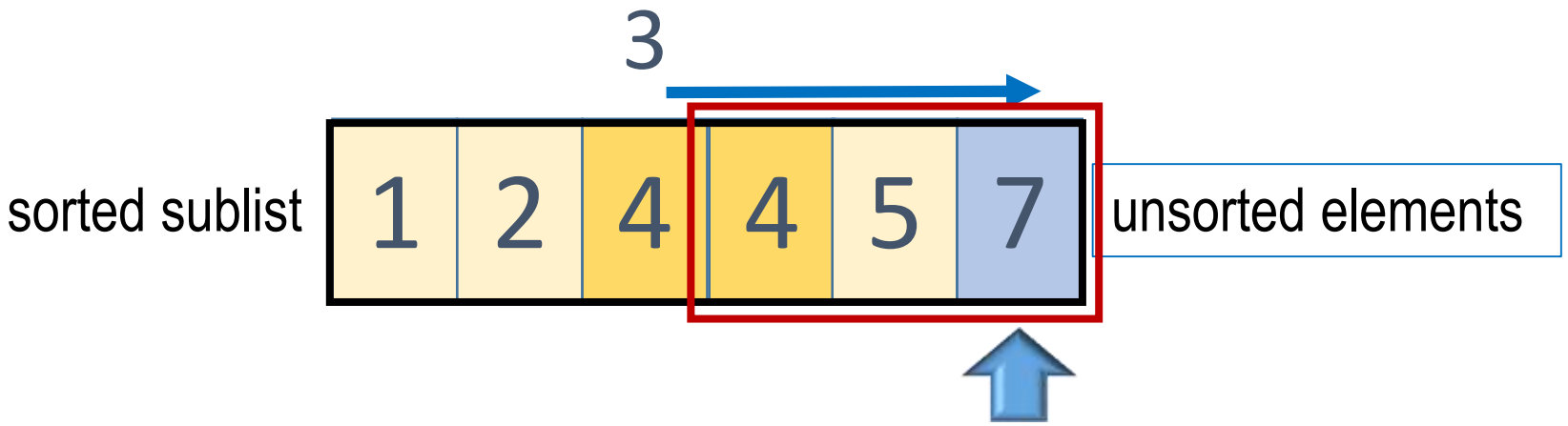
Insertion sort: Summary

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



Insertion sort : Summary

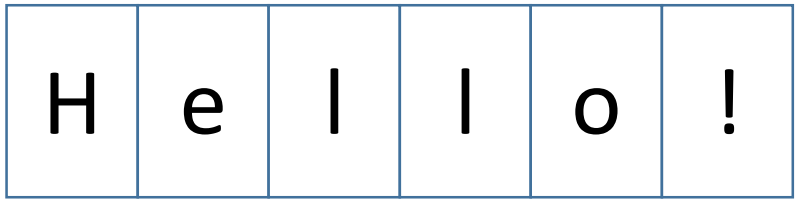
1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



Initializing Character Arrays

Initializing Character Arrays

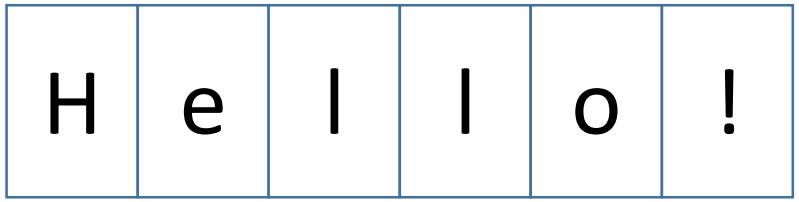
char msg[ ] = {‘H', ‘e', 'l', 'l', ‘o', ‘!'}; // not include \0

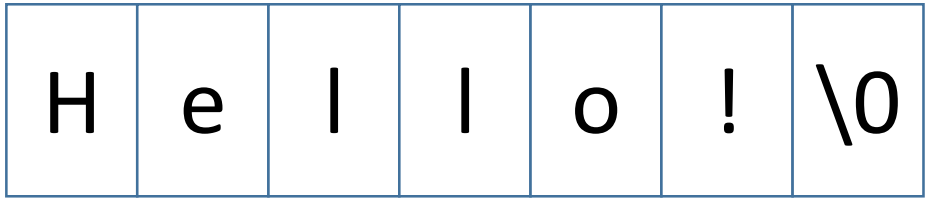


Initializing Character Arrays

char msg[ ] = {‘H', ‘e', 'l', 'l', ‘o', ‘!'}; // not include \0

char msg[] = “Hello!"; //include \0



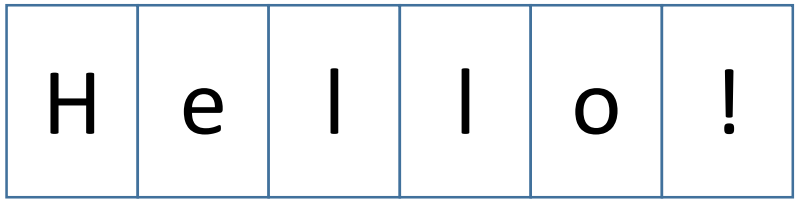


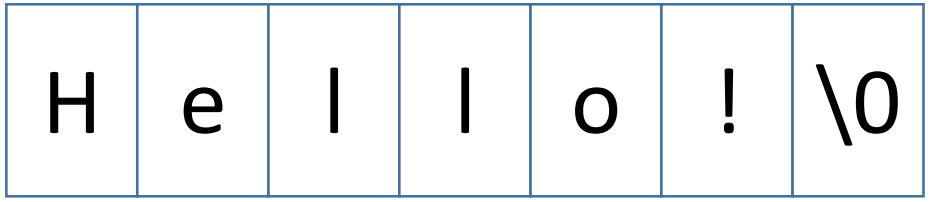
Initializing Character Arrays

char msg[ ] = {‘H', ‘e', 'l', 'l', ‘o', ‘!'}; // not include \0

char msg[] = “Hello!"; //include \0

- This statement is equivalent to the preceding statement, except that C++ adds the character '\0‘.
- \0: the *null terminator*. Indicate the end of the string.
- A character that begins with the back slash (\) is an escape character.





Exercise 1

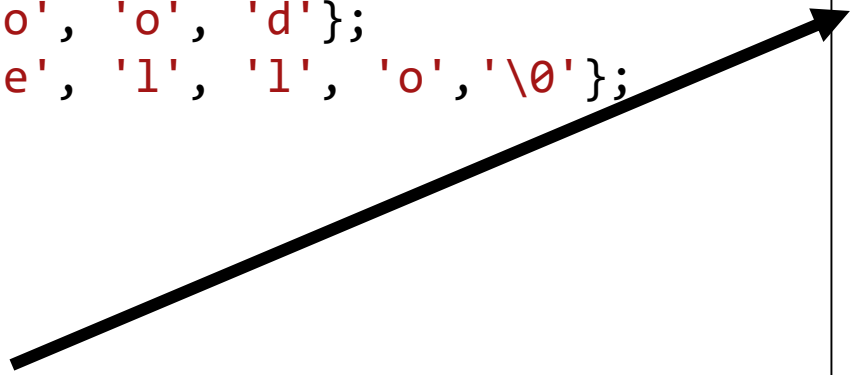
```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};




    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};




    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};




    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```



Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};




    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};




    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};




    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};




    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};




    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

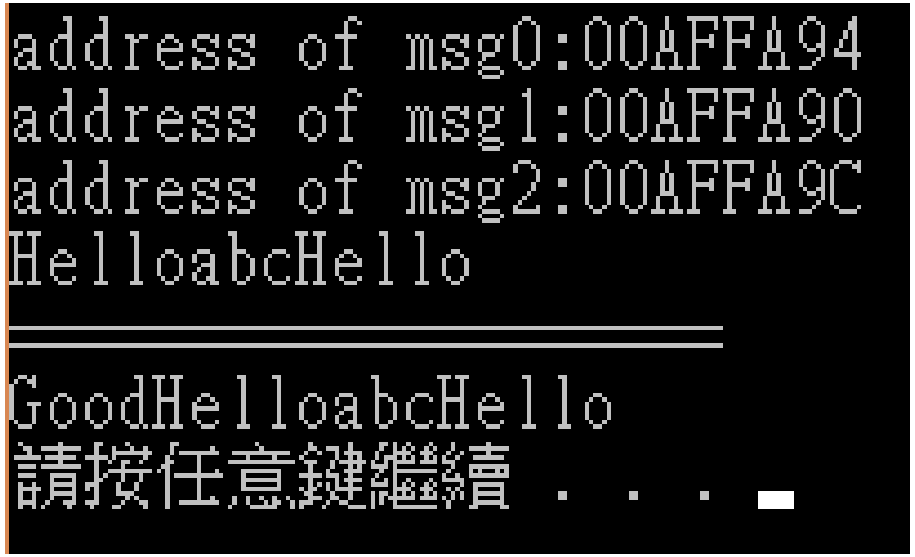
```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```



Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

```
address of msg0:00AFFA94
address of msg1:00AFFA90
address of msg2:00AFFA9C
HelloabcHello
====================
GoodHelloabcHello
請按任意鍵繼續 . . .
```

?	?	?

Exercise 1

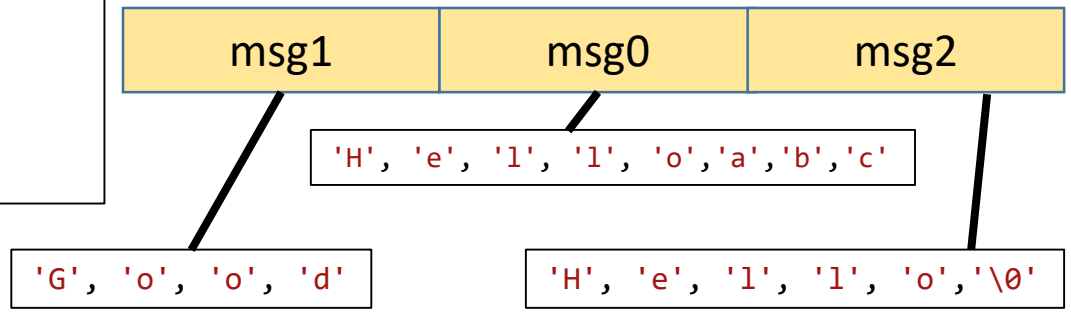
```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

```
address of msg0:00AFFA94
address of msg1:00AFFA90
address of msg2:00AFFA9C
HelloabcHello
====================
GoodHelloabcHello
請按任意鍵繼續 . . .
```

Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

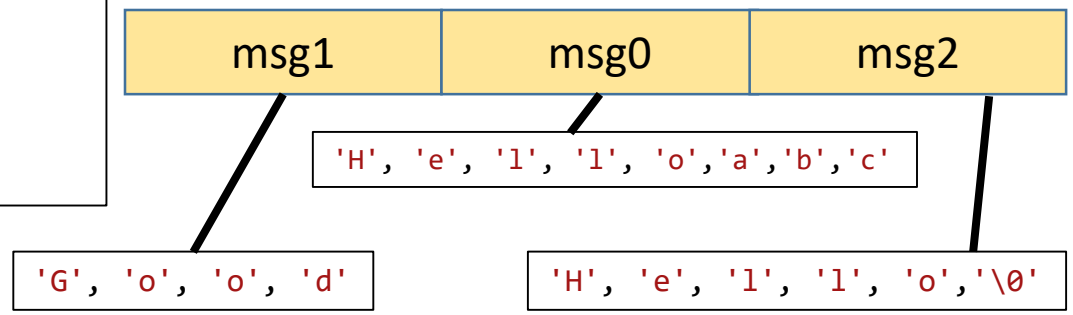
```
address of msg0:00AFFA94
address of msg1:00AFFA90
address of msg2:00AFFA9C
HelloabcHello
====================
GoodHelloabcHello
請按任意鍵繼續 . . .
```



Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

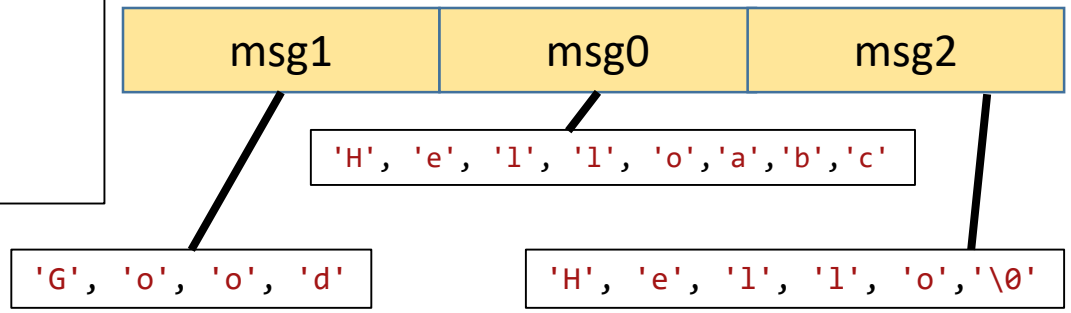
```
address of msg0:00AFFA94
address of msg1:00AFFA90
address of msg2:00AFFA9C
HelloabcHello
====================
GoodHelloabcHello
請按任意鍵繼續 . . .
```



Exercise 1

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o','a','b','c'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

```
address of msg0:00AFFA94
address of msg1:00AFFA90
address of msg2:00AFFA9C
HelloabcHello
====================
GoodHelloabcHello
請按任意鍵繼續 . . .
```



Exercise 2

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

Exercise 2

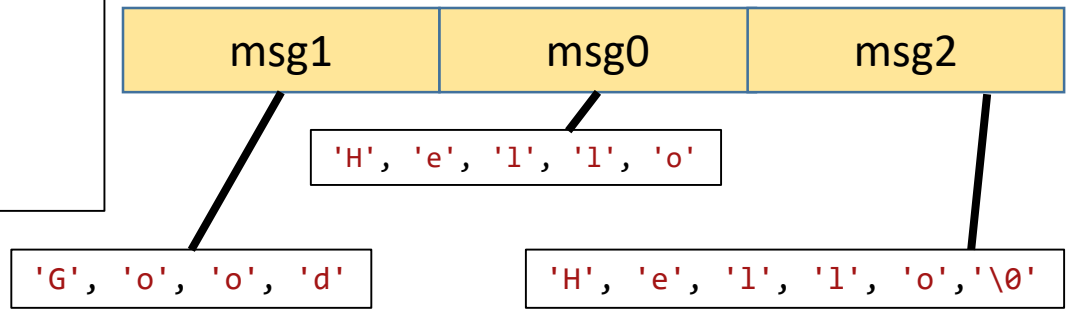
```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

```
address of msg0:009FFEC4
address of msg1:009FFEC0
address of msg2:009FFECC
Hello4肆Hello
====================
GoodHello4肆Hello
請按任意鍵繼續 . . .
```

Exercise 2

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

```
address of msg0:009FFEC4
address of msg1:009FFEC0
address of msg2:009FFECC
Hello4肆Hello
====================
GoodHello4肆Hello
請按任意鍵繼續 . . .
```

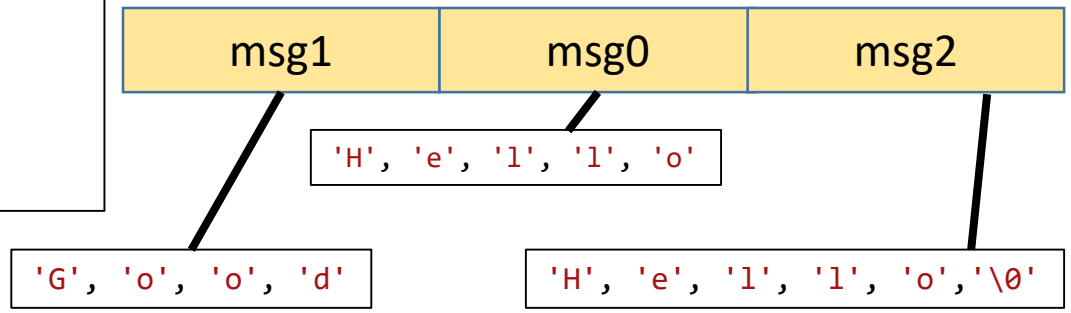


Exercise 2

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

04 = 4 (DEC)
0C = 12(DEC)
Difference = 12 – 4 = 8

```
address of msg0:009FFEC4
address of msg1:009FFEC0
address of msg2:009FFECC
Hello4肆Hello
====================
GoodHello4肆Hello
請按任意鍵繼續 . . .
```

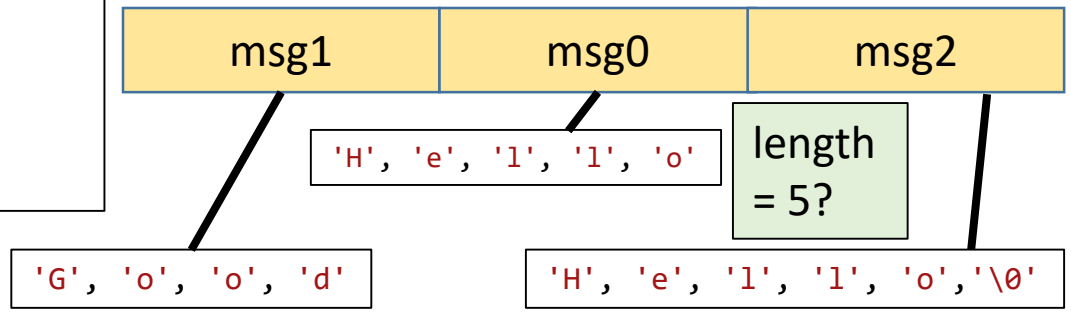


Exercise 2

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

04 = 4 (DEC)
0C = 12(DEC)
Difference = 12 – 4 = 8

```
address of msg0:009FFEC4
address of msg1:009FFEC0
address of msg2:009FFECC
Hello4肆Hello
====================
GoodHello4肆Hello
請按任意鍵繼續 . . .
```

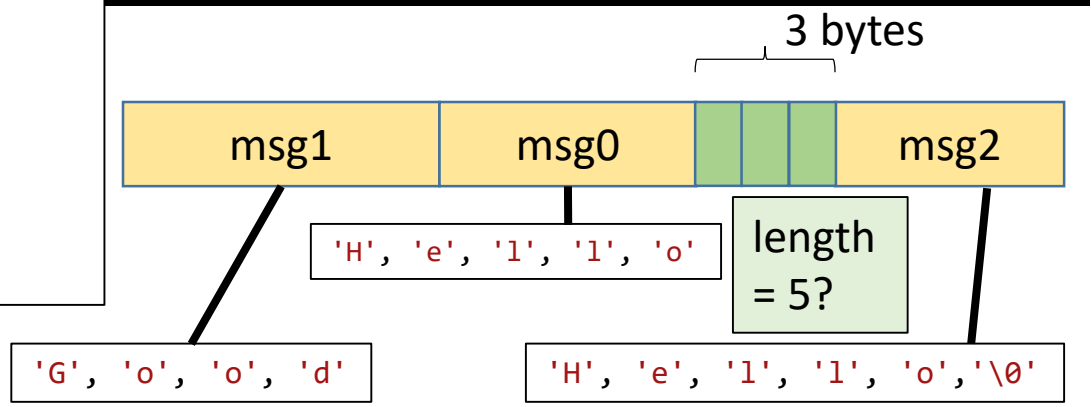


Exercise 2

```
void test00() {
    int a = 12345678;
    char b[] = { 'W','o', 'r','l', 'd' };
    char msg0[] = {'H', 'e', 'l', 'l', 'o'};
    char msg1[] = { 'G', 'o', 'o', 'd'};
    char msg2[] = { 'H', 'e', 'l', 'l', 'o','\0'};

    cout << "address of msg0:" << &msg0 << endl;
    cout << "address of msg1:" << &msg1 << endl;
    cout << "address of msg2:" << &msg2 << endl;
    cout << msg0 << endl;
    cout << "====================" << endl;
    cout << msg1 << endl;
}
```

04 = 4 (DEC)
0C = 12(DEC)
Difference = 12 – 4 = 8

```
address of msg0:009FFEC4
address of msg1:009FFEC0
address of msg2:009FFEBC
Hello4肆Hello
====================
GoodHello4肆Hello
請按任意鍵繼續 . . .
```



3 bytes are padded
to achieve 4-byte alignment

Reading C-Strings

Read a string from the keyboard using the cin object.

```
char place[10];
cout << "Enter a place: ";
cin >> place;                    // read to array, place

// note: what is added to the end of the input?
cout << "You entered: " << place<< endl;

//Type: abcd
//Then press ENTER
```

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;
```

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;
```

Each xx is a byte.
ASCII code of ‘a’ is 0x61
ASCII code of ‘b’ is 0x62

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;
```

Each xx is a byte.
ASCII code of ‘a’ is 0x61
ASCII code of ‘b’ is 0x62

```
abcd                    << ENTER
Message:abcd
61 62 63 64 0 22 2b 0 2f 22
String:
61 62 63 64 0 0 0 0 0 0
```

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;
```

Each xx is a byte.
ASCII code of ‘a’ is 0x61
ASCII code of ‘b’ is 0x62

```
abcd                    << ENTER
Message:abcd
61 62 63 64 0 22 2b 0 2f 22
String:
61 62 63 64 0 0 0 0 0 0
```

The null terminator is appended after ENTER is pressed

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;

    cout << "String:" << endl;
    char str[10] = {'a','b','c','d'};
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) str[i] << " ";
    }
```

Each xx is a byte.
ASCII code of ‘a’ is 0x61
ASCII code of ‘b’ is 0x62

```
abcd                    << ENTER
Message:abcd
61 62 63 64 0 22 2b 0 2f 22
String:
61 62 63 64 0 0 0 0 0 0
```

The null terminator is appended at compilation time

The null terminator is appended after ENTER is pressed

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;

    cout << "String:" << endl;
    char str[10] = {'a','b','c','d'};
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) str[i] << " ";
    }
```

Each xx is a byte.
ASCII code of ‘a’ is 0x61
ASCII code of ‘b’ is 0x62

```
abcd                    << ENTER
Message:abcd
61 62 63 64 0 22 2b 0 2f 22
String:
61 62 63 64 0 0 0 0 0 0
```

The null terminator is appended after ENTER is pressed

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;

    cout << "String:" << endl;
    char str[10] = {'a','b','c','d'};
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) str[i] << " ";
    }
```

Each xx is a byte.
ASCII code of ‘a’ is 0x61
ASCII code of ‘b’ is 0x62

```
abcd                    << ENTER
Message:abcd
61 62 63 64 0 22 2b 0 2f 22
String:
61 62 63 64 0 0 0 0 0 0
```

The null terminator is appended at compilation time

The null terminator is appended after ENTER is pressed

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;

    cout << "String:" << endl;
    char str[] = {'a','b'};
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) str[i] << " ";
    }
```

```
abcd                    << ENTER
Message:abcd
61 62 63 64 0 22 3c 1 2f 22
String:
61 62 3c 1 61 62 63 64 0 22
```

The null terminator is appended after ENTER is pressed

Reading C-Strings

```
char msg[10];
    cin >> msg;
    cout << "Message:" << msg << endl;
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) msg[i] << " ";
    }
    cout << endl;

    cout << "String:" << endl;
    char str[] = {'a','b'};
    for (int i =0; i < 10; ++i) {
        cout << hex << (unsigned int) str[i] << " ";
    }
```

```
abcd                    << ENTER
Message:abcd
61 62 63 64 0 22 3c 1 2f 22
String:
61 62 3c 1 61 62 63 64 0 22
```

getline

Read a string into an array.

cin.getline(char array[], int size, char delimitChar)

- The function stops reading characters when the delimiter character is encountered or when the size - 1 number of characters are read.
- The last character in the array is reserved for the null terminator ('\0').
- If the delimiter is encountered, it is read, but not stored in the array.
- The third argument delimitChar has a default value ('\n').

Reading C-Strings Using getline

cin.getline(char array[], int size, char delimitChar)

char array[256];

cin.getline(array, 256, ‘w’);

cout << “array:” << array << endl;

Input: ThereWeGo!

What does array store?

The program waits for input… Because ‘w’ is not encountered.
Press ENTER does not terminate the input process.

Reading C-Strings Using getline

cin.getline(char array[], int size, char delimitChar)

char array[256];

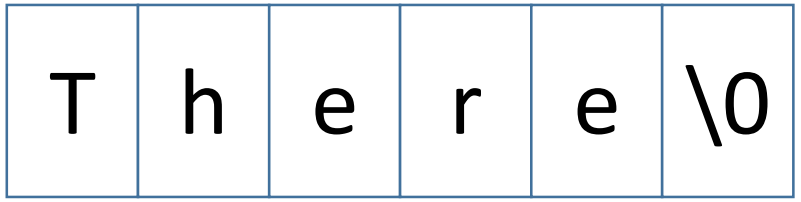
cin.getline(array, 256, ‘w’);

cout << “array:” << array << endl;

Input: ThereweGo!

What does array store?

There (followed with \0 at the end)



C-String Functions

Functions		
strlen	atoi	
strcpy	atof	
strncpy	atol	
strcat	itoa	
strncat		
strcmp		
strncmp		

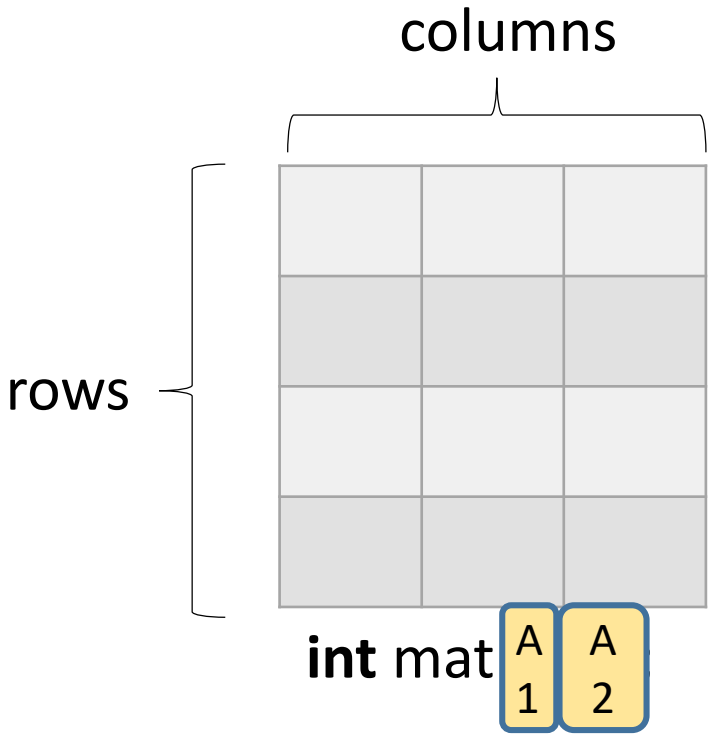
Read the manual of C-String

Multidimensional Arrays

Two-dimensional Arrays

elementType arrayName[rowSize][columnSize]; `// Declare a 2D array`

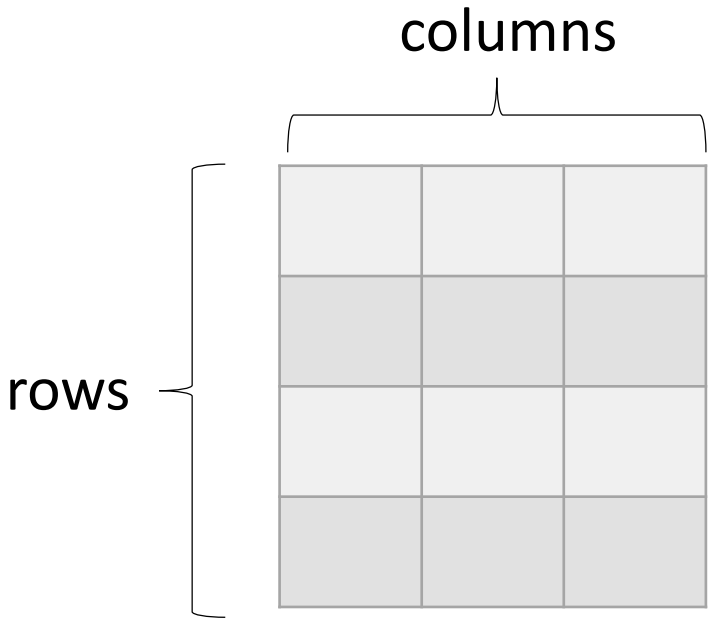
int mat[4][3]; // 4 rows and 3 columns



Two-dimensional Arrays

elementType arrayName[rowSize][columnSize]; `// Declare a 2D array`

int mat[4][3]; // 4 rows and 3 columns



int mat[4][3];

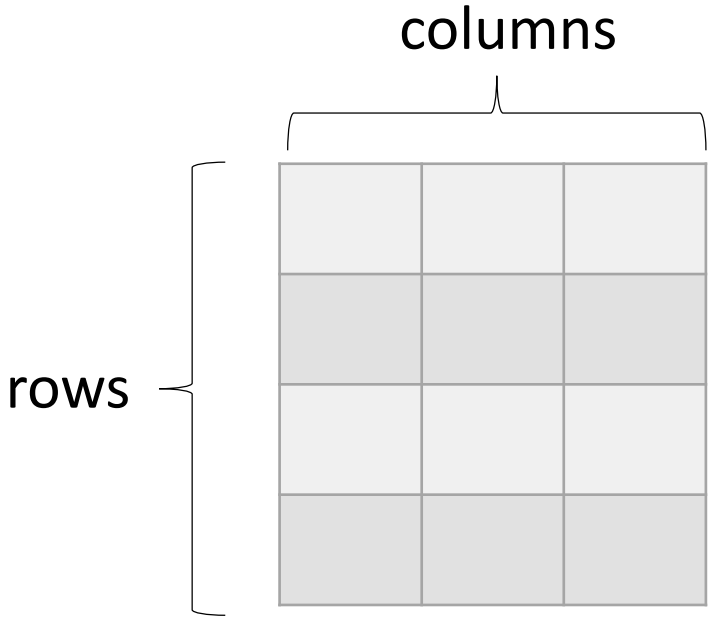
Element at the third row and second column

mat[A1][A2] = 6;

Two-dimensional Arrays

elementType arrayName[rowSize][columnSize]; `// Declare a 2D array`

int mat[4][3]; // 4 rows and 3 columns



int mat[4][3];

	6	

Element at the third row and second column
mat[2][1] = 6;

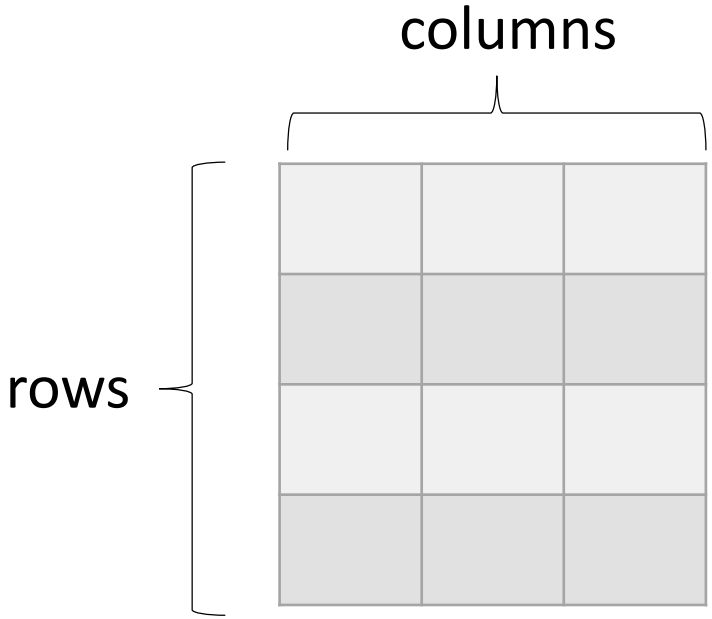
1	2	3
4	5	6
7	8	9
10	11	12

```
int mat[4][3] = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9},
                {10, 11, 12}
};
```

Two-dimensional Arrays

elementType arrayName[rowSize][columnSize]; `// Declare a 2D array`

int mat[4][3]; // 4 rows and 3 columns



int mat[4][3];

	6	

Element at the third row and second column
mat[2][1] = 6;

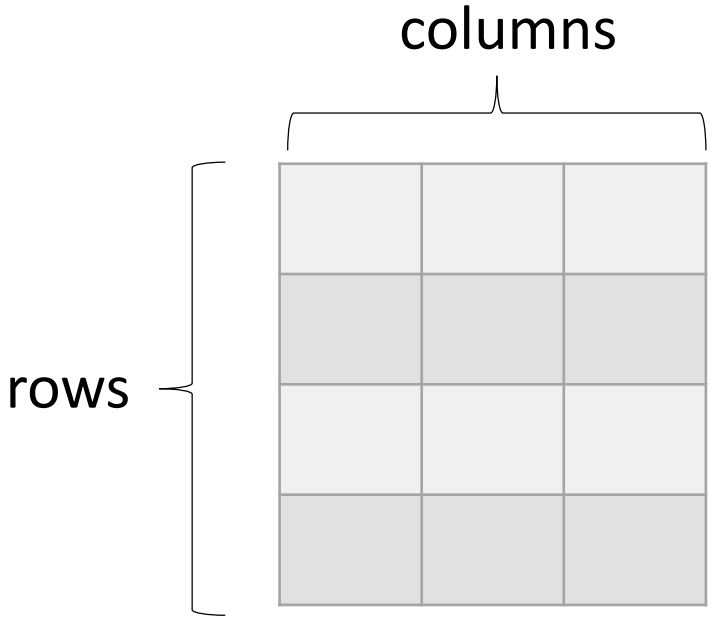
1	**2**	**3**
4	5	6
7	8	9
10	11	12

```
int mat[4][3] = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9},
            {10, 11, 12}
};
```

Two-dimensional Arrays

elementType arrayName[rowSize][columnSize]; `// Declare a 2D array`

int mat[4][3]; // 4 rows and 3 columns



int mat[4][3];

	6	

Element at the third row and second column
mat[2][1] = 6;

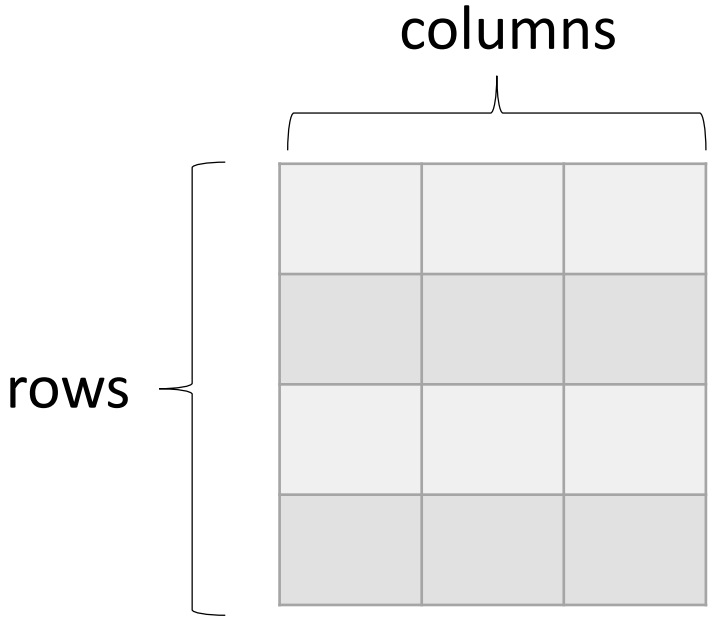
1	2	3
4	**5**	**6**
7	8	9
10	11	12

```
int mat[4][3] = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9},
                {10, 11, 12}
};
```

Two-dimensional Arrays

elementType arrayName[rowSize][columnSize]; `// Declare a 2D array`

int mat[4][3]; // 4 rows and 3 columns



int mat[4][3];

	6	

Element at the third row and second column
mat[2][1] = 6;

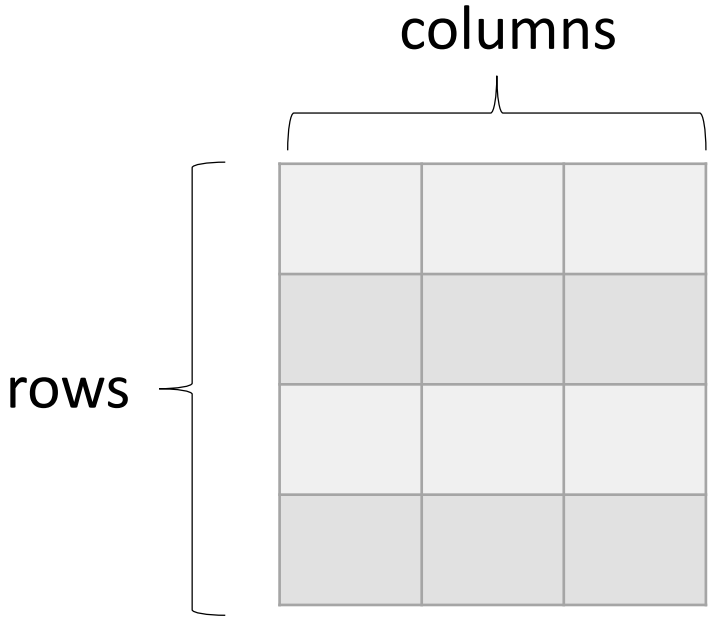
1	2	3
4	5	6
7	**8**	**9**
10	11	12

```
int mat[4][3] = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9},
                {10, 11, 12}
};
```

Two-dimensional Arrays

elementType arrayName[rowSize][columnSize]; `// Declare a 2D array`

int mat[4][3]; // 4 rows and 3 columns



int mat[4][3];

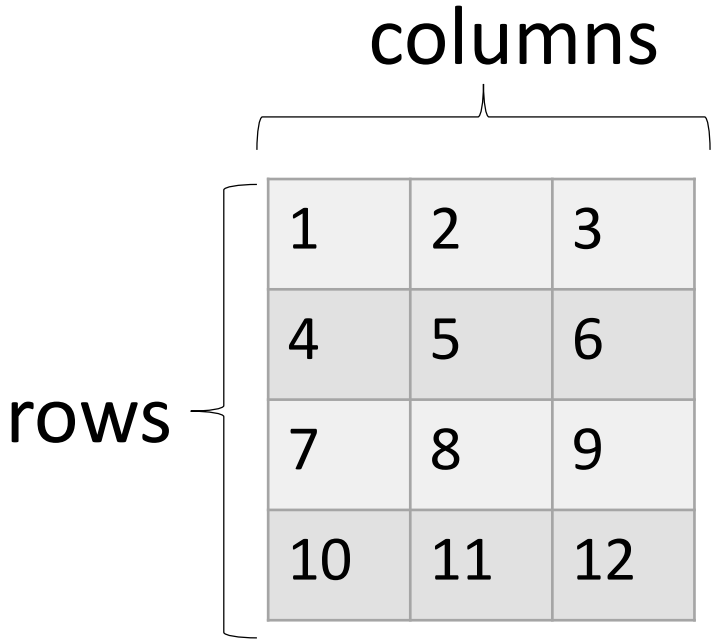
	6	

Element at the third row and second column
mat[2][1] = 6;

1	2	3
4	5	6
7	8	9
10	**11**	**12**

```
int mat[4][3] = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9},
                {10, 11, 12}
};
```

Initializing Arrays with Random Values



The following loop initializes the array with random values: 0 to 999:

```
for (int row = 0; row < rowSize; row++)
{
  for (int column = 0; column < columnSize; column++)
  {
    matrix[row][column] = rand() % 1000;
  }
}
```

Printing Arrays

To print a two-dimensional array:

```
for (int row = 0; row < rowSize; row++)
{
  for (int column = 0; column < columnSize; column++)
  {
    cout << matrix[row][column] << “\t"; // tab between two elements
  }
  cout << endl;    // go to next line after one row is printed.
}
```

1	2	3
4	5	6
7	8	9
10	11	12

rowSize = 4
columnSize = 3

Summing All Elements

```
int sum = 0;
for (int row = 0; row < rowSize; row++)
{
  for (int column = 0; column < columnSize; column++)
  {
    sum += matrix[row][column];
  }
}
cout << “Sum:” << sum << endl;
```

1	2	3
4	5	6
7	8	9
10	11	12

rowSize = 4
columnSize = 3

Summing Elements by Column

For each column, use a variable, named total, to store its sum.

1	2	3
4	5	6
7	8	9
10	11	12

rowSize = 4
columnSize = 3

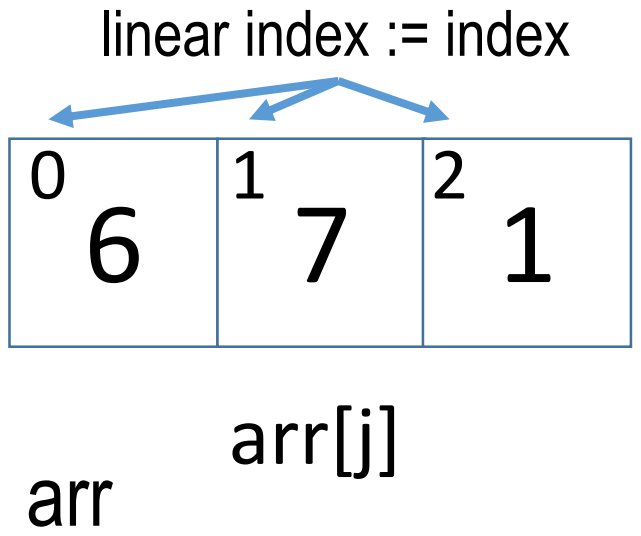
```
for (int column = 0; column < columnSize; column++)
{
  int total = 0;
  for (int row = 0; row < rowSize; row++) {
       total += matrix[row][column];
  }
  cout    << "Sum for column "
          << column << " is " << total << endl;
}
```

Linear index of an element

- The linear index of an element of a 1-D array is the same as the index of the element.

ElementType arrayName[columnSize];

int arr[3] = {6, 7, 1};



Assume using row-major to store the elements of array arr.

The linear index element arr(i, j) = arr[j]
= i * columnSize + j

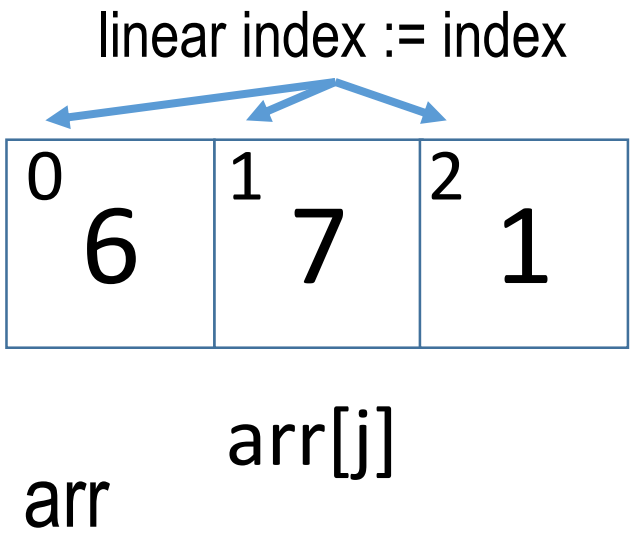
As there is only one row, i = A 1
So the linear index = A 2 same as the index of the element.

Linear index of an element

- The linear index of an element of a 1-D array is the same as the index of the element.

ElementType arrayName[columnSize];

int arr[3] = {6, 7, 1};



Assume using row-major to store the elements of array arr.

The linear index element arr(i, j) = arr[j]
= i * columnSize + j

As there is only one row, i = 0.
So the linear index = j, same as the index of the element.

Linear index of an element

- A 2D array can be treated as a 1-D array.
- Convert the coordimates (i,j) of element[i][j] to a linear index.

ElementType arrayName[rowSize][columnSize];

int arr[4][3] = …… ;

Assume using row-major to store the elements of array arr.

The linear index L of element arr(i, j)
L = i * columnSize + j

6	7	1
2	3	5
9	10	6
4	8	-1

arr

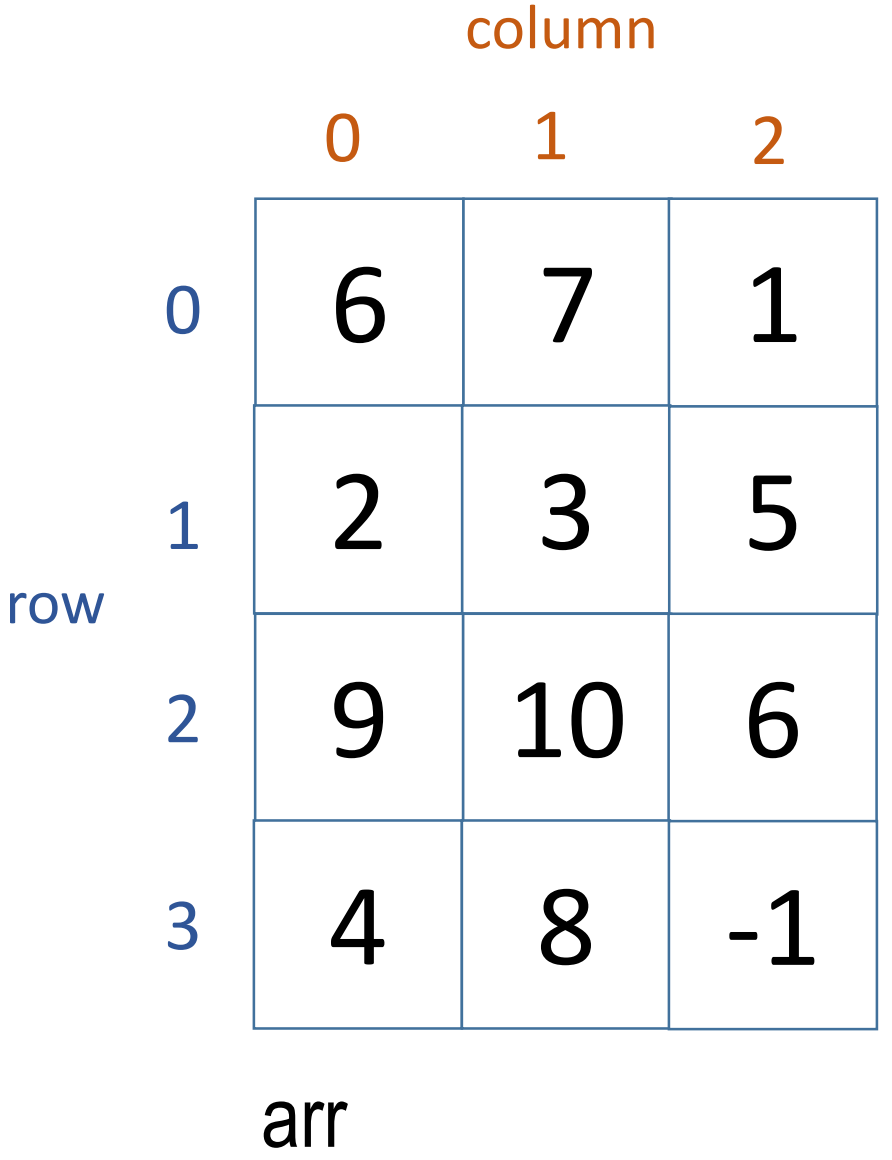
Linear index of an element

- A 2D array can be treated as a 1-D array.
- Convert the coordimates (i,j) of element[i][j] to a linear index.

ElementType arrayName[rowSize][columnSize];

Assume using row-major to store the elements of array arr.

The linear index L of element arr(i, j)
L = i * columnSize + j



arr

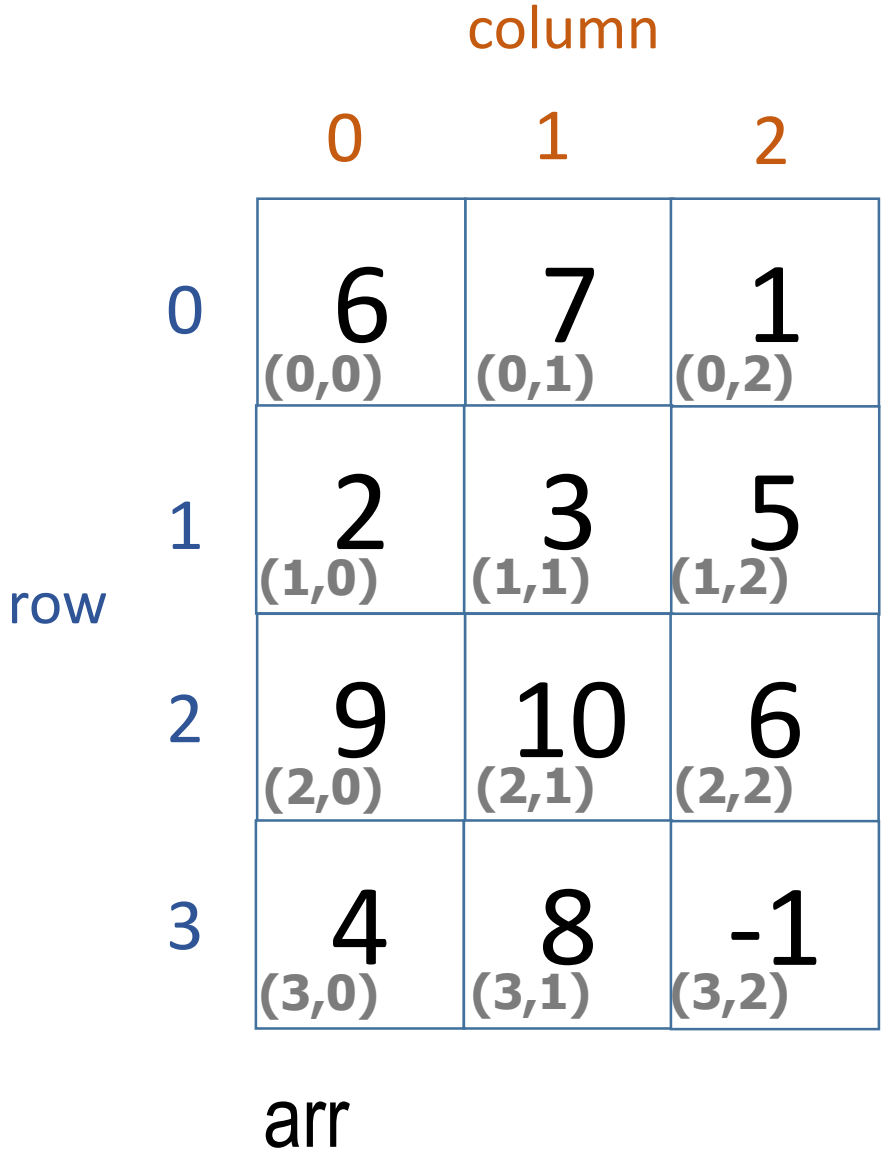
Linear index of an element

- A 2D array can be treated as a 1-D array.
- Convert the coordimates (i,j) of element[i][j] to a linear index.

ElementType arrayName[rowSize][columnSize];

Assume using row-major to store the elements of array arr.

The linear index L of element arr(i, j)
L = i * columnSize + j



arr

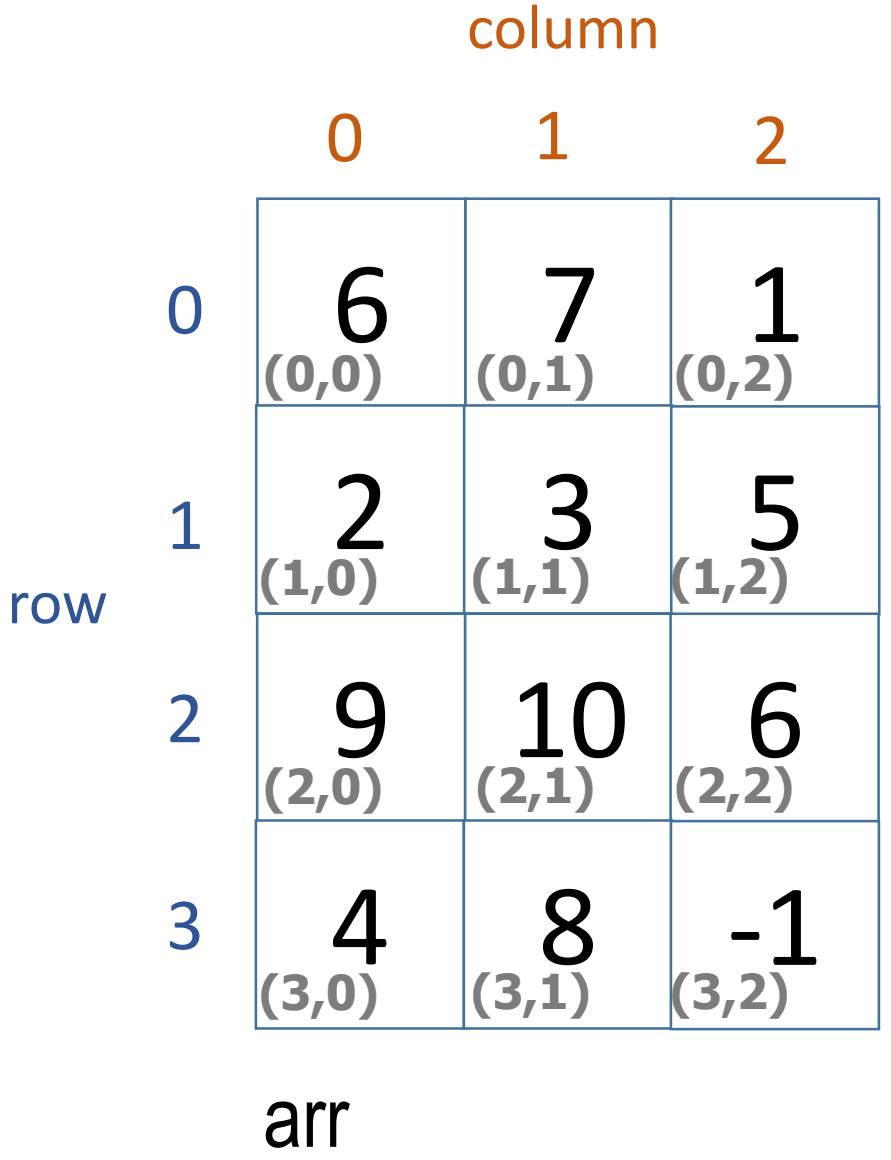
Linear index of an element

- A 2D array can be treated as a 1-D array.
- Convert the coordimates (i,j) of element[i][j] to a linear index.

ElementType arrayName[rowSize][columnSize];

Assume using row-major to store the elements of array arr.

The linear index L of element arr(i, j)
L = i * columnSize + j



arr

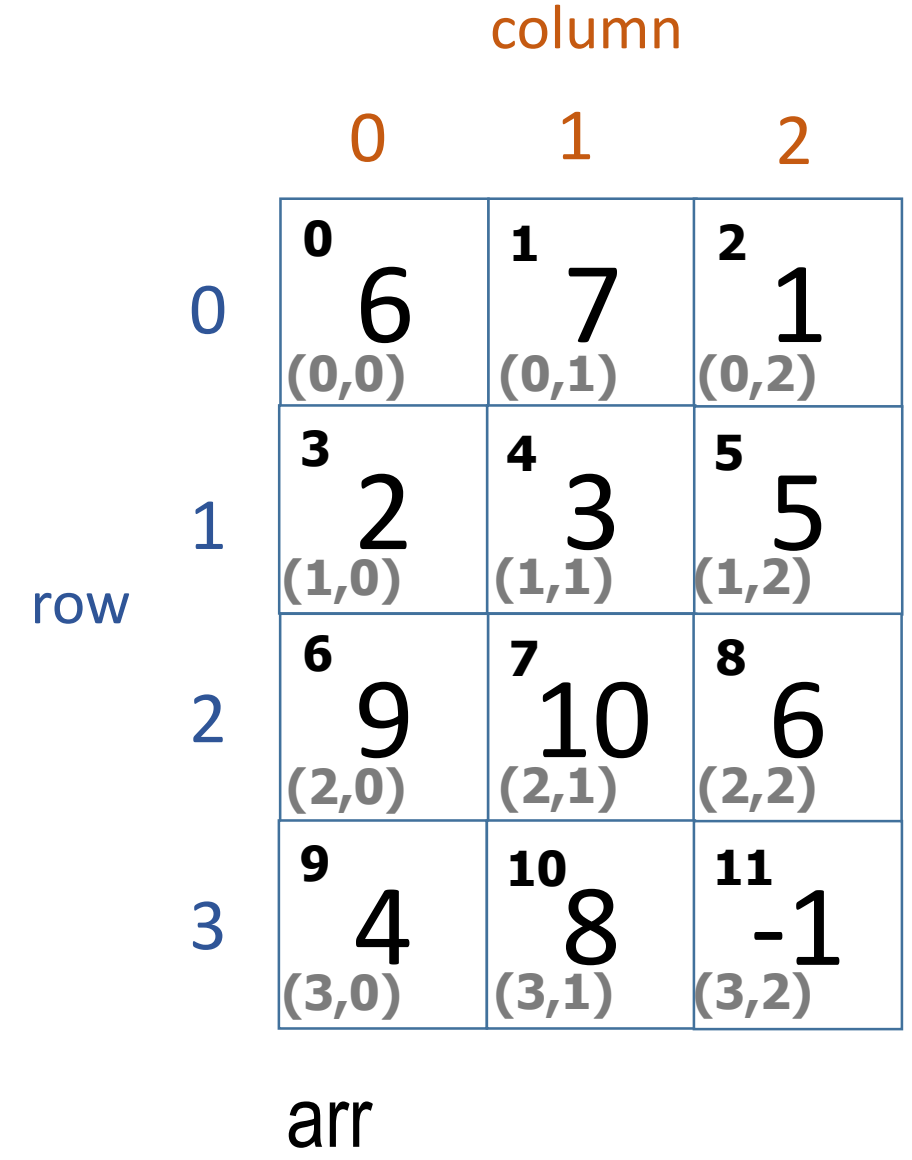
Linear index of an element

- A 2D array can be treated as a 1-D array.
- Convert the coordimates (i,j) of element[i][j] to a linear index.

ElementType arrayName[rowSize][columnSize];

Assume using row-major to store the elements of array arr.

The linear index L of element arr(i, j)
L = i * columnSize + j



arr

Linear index of an element

- A 2D array can be treated as a 1-D array.
- Convert the coordimates (i,j) of element[i][j] to a linear index.

ElementType arrayName[rowSize][columnSize];

Assume using row-major to store the elements of array arr.

The linear index L of element arr(i, j)
L = i * columnSize + j

arr

0 6 (0,0)	1 7 (0,1)	2 1 (0,2)
3 2 (1,0)	4 3 (1,1)	5 5 (1,2)
6 9 (2,0)	7 10 (2,1)	8 6 (2,2)
9 4 (3,0)	10 8 (3,1)	11 -1 (3,2)

Linear index of an element

- A 2D array can be treated as a 1-D array.
- Convert the coordimates (i,j) of element[i][j] to a linear index.

ElementType arrayName[rowSize][columnSize];

Assume using row-major to store the elements of array arr.

The linear index L of element arr(i, j)
L = i * columnSize + j

Thus, we have
i = L / columnSize // integer devision
j = L% columnSize // remainder

arr

0 6 (0,0)	1 7 (0,1)	2 1 (0,2)
3 2 (1,0)	4 3 (1,1)	5 5 (1,2)
6 9 (2,0)	7 10 (2,1)	8 6 (2,2)
9 4 (3,0)	10 8 (3,1)	11 -1 (3,2)

Linear index of an element

- A 2D array can be treated as a 1-D array.
- Convert the coordimates (i,j) of element[i][j] to a linear index.

ElementType arrayName[rowSize][columnSize];

arr

0 6 (0,0)	1 7 (0,1)	2 1 (0,2)
3 2 (1,0)	4 3 (1,1)	5 5 (1,2)
6 9 (2,0)	7 10 (2,1)	8 6 (2,2)
9 4 (3,0)	10 8 (3,1)	11 -1 (3,2)

Assume using row-major to store the elements of array arr.

The linear index L of element arr(i, j)
L = i * columnSize + j

Thus, we have
i = L / columnSize // integer devision
j = L% columnSize // remainder

Example. L = 10
i = L / 3 = 3
j = 10%3 = 1

Passing Two-Dimensional Arrays to Functions

Pass a two-dimensional array to a function.

ElementType arrayName[rowSize][columnSize];

1	2	3
4	5	6
7	8	9
10	11	12

The compiler needs to know how to compute the linear address of an element of the array. Thus, the number of columns must be passed.

```
void foo( int array[][100] ) {
        ……
}
```

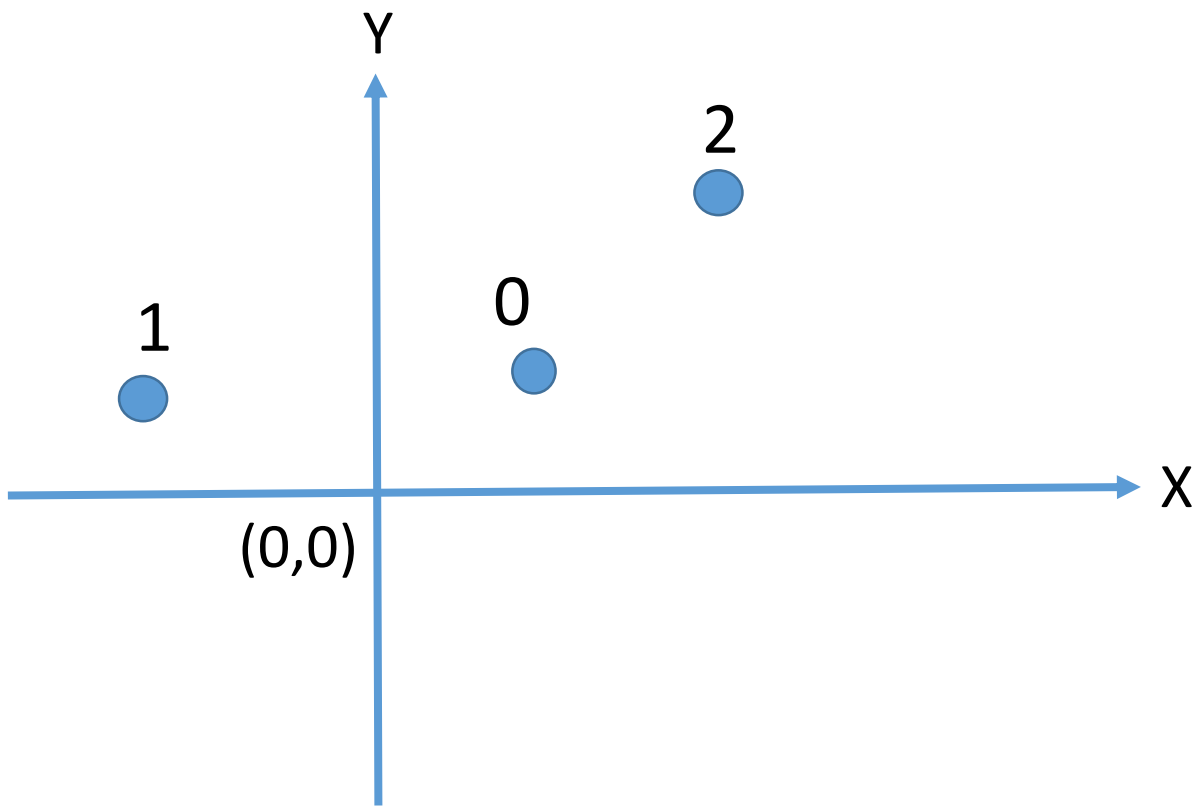
Element: array[i][j]
Assume using row-major

The address of an element [i][j]
= elementSize*(i*numberOfColumns + j)

Representation of points on a Cartesian coordinate system

```
double p0[2];     // p0[0] is the x-coordinate
                  // p0[1] is the y-coordinate

double p1[2];     // p1[0] is the x-coordinate
                  // p1[1] is the y-coordinate

double p2[2];     // p2[0] is the x-coordinate
                  // p2[1] is the y-coordinate
```

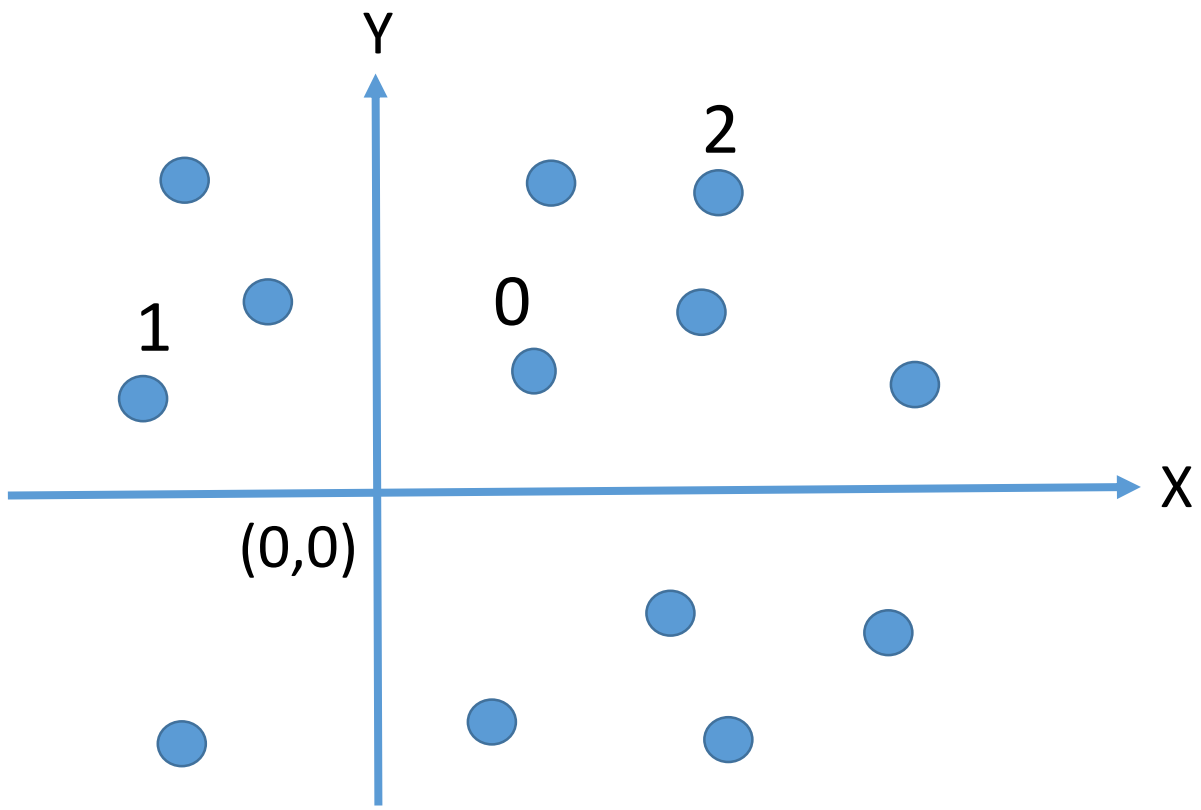
What should you do if you have n points?
n can be very large, e.g., n > 10^6



Representation of points on a Cartesian coordinate system

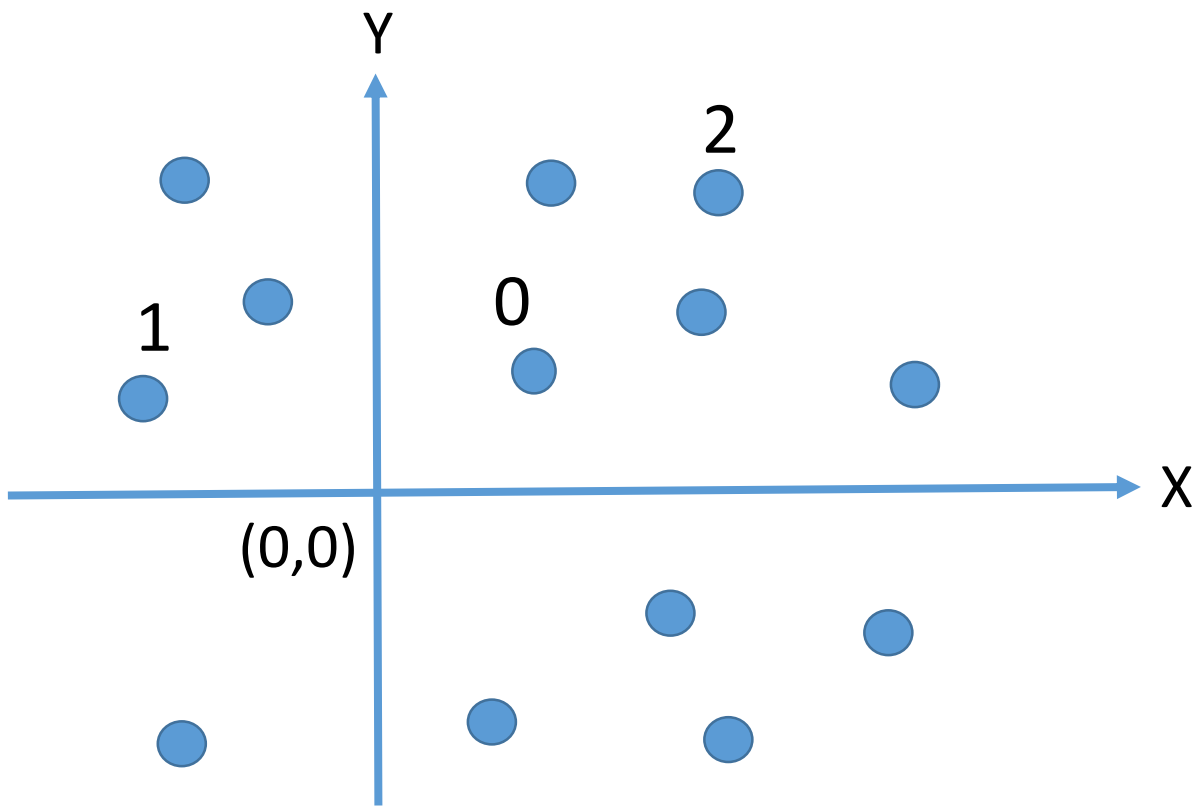
```
double p0[2];     // p0[0] is the x-coordinate
                  // p0[1] is the y-coordinate

double p1[2];     // p1[0] is the x-coordinate
                  // p1[1] is the y-coordinate

double p2[2];     // p2[0] is the x-coordinate
                  // p2[1] is the y-coordinate
```

What should you do if you have n points?
n can be very large, e.g., n > 10^6



Representation of points on a Cartesian coordinate system

```
double p0[2];     // p0[0] is the x-coordinate
                  // p0[1] is the y-coordinate

double p1[2];     // p1[0] is the x-coordinate
                  // p1[1] is the y-coordinate

double p2[2];     // p2[0] is the x-coordinate
                  // p2[1] is the y-coordinate
```



What should you do if you have n points?
n can be very large, e.g., n > 10^6

```
double p[numOfPoints][2]; // [2] for x and y coordinates.
double p[numOfPoints][Dimension]; // for higher dimension
```

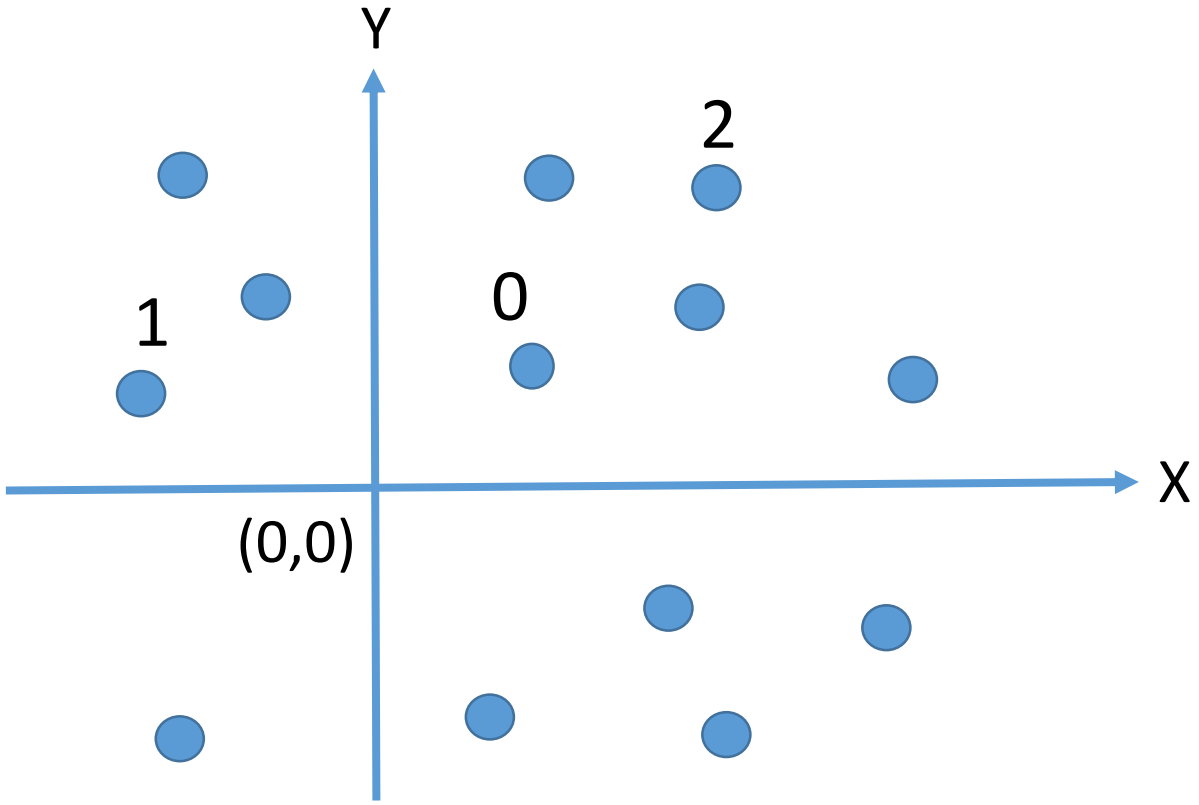
Problem: Finding Two Points Nearest to Each Other

Given n points
double p[n][2]

System specification

Implement a method to find two points which are nearest to each other.

void find2Points(int &ni, int &nj)



Problem: Finding Two Points Nearest to Each Other

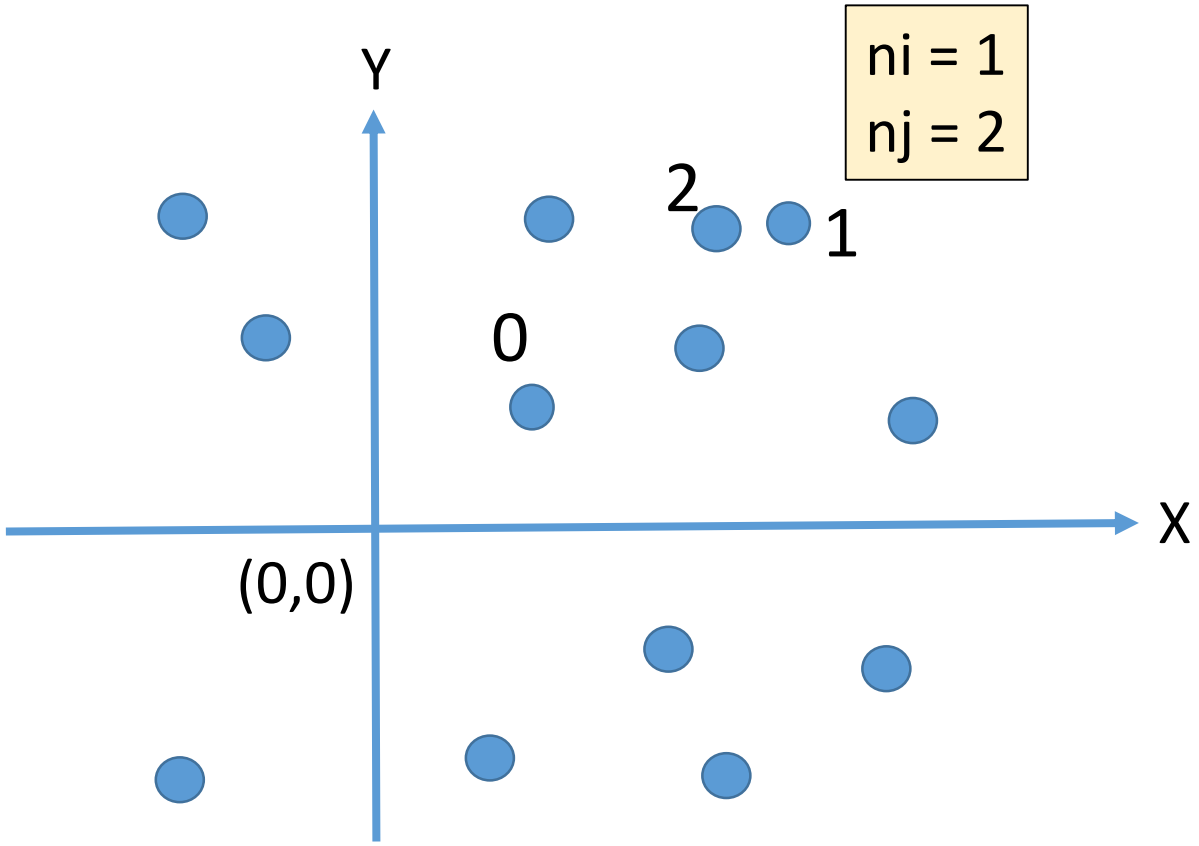
Given n points
double p[n][2]

System specification

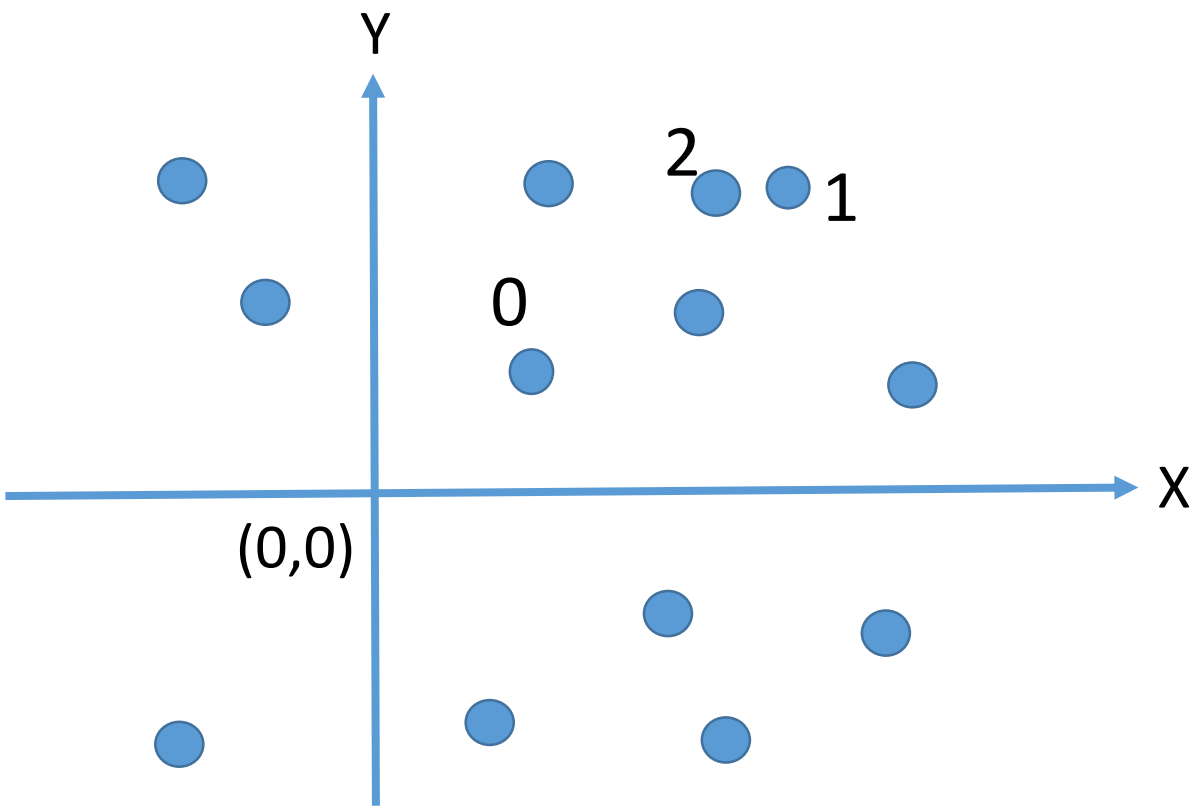
Implement a method to find two points which are nearest to each other.

double find2Points(int &ni, int &nj)

ni = 1
nj = 2



Problem: Finding Two Points Nearest to Each Other



Set the min distance as the first point points

loop through all pairs of points

Update the min distance
Update indices ni and nj

Remark: do not use double d = sqrt(d2);

Problem: Finding Two Points Nearest to Each Other

```
double void find2Points(int &ni, int &nj) {
    double minDistance2 = distance2(0, 1, p);
    ni = 0; nj = 1;
    for ( int i = 0; i < nP; ++i  )  {
        for ( int j = i + 1; j < nP; ++j  )  {
            double dx = p[i][0] – p[j][0];
            double dy = p[i][1] – p[j][1];
            double d2 = dx*dx+dy*dy;
            if ( minDistance > d2 ) {
                ni = i; nj = j;
                minDistance = d2;
            }
        } // for j
    } // for i
    return minDistance;
}
```

Set the min distance as the first point points

loop through all pairs of points

Update the min distance
Update indices ni and nj

Remark: do not use double d = sqrt(d2);

Problem: Finding Two Points Nearest to Each Other

```
double void find2Points(int &ni, int &nj) {
    double minDistance2 = distance2(0, 1, p);
    ni = 0; nj = 1;
    for ( int i = 0; i < nP; ++i  )  {
        for ( int j = i + 1; j < nP; ++j  )  {
            double dx = p[i][0] – p[j][0];
            double dy = p[i][1] – p[j][1];
            double d2 = dx*dx+dy*dy;
            if ( minDistance > d2 ) {
                ni = i; nj = j;
                minDistance = d2;
            }
        } // for j
    } // for i
    return minDistance;
}
```

```
double distance2( int i, int j) {
        double dx = p[i][0] – p[j][0];
        double dy = p[i][1] – p[j][1];
        double d2 = dx*dx+dy*dy;
        return d2;
}
```

Remark: do not use double d = sqrt(d2);

Problem: Finding Two Points Nearest to Each Other

```
double void find2Points(int &ni, int &nj) {
    double minDistance2 = distance2(0, 1, p);
    ni = 0; nj = 1;
    for ( int i = 0; i < nP; ++i  )  {
        for ( int j = i + 1; j < nP; ++j  )  {
            double dx = p[i][0] – p[j][0];
            double dy = p[i][1] – p[j][1];
            double d2 = dx*dx+dy*dy;
            d2 = distance2(i, j);
            if ( minDistance > d2 ) {
                ni = i; nj = j;
                minDistance = d2;
            }
        } // for j
    } // for i
    return minDistance;
}
```

```
double distance2( int i, int j) {
        double dx = p[i][0] – p[j][0];
        double dy = p[i][1] – p[j][1];
        double d2 = dx*dx+dy*dy;
        return d2;
}
```

Remark: do not use double d = sqrt(d2);

Problem: Finding Two Points Nearest to Each Other

```
double void find2Points(int &ni, int &nj) {
    double minDistance2 = distance2(0, 1, p);
    ni = 0; nj = 1;
    for ( int i = 0; i < nP; ++i  )  {
        for ( int j = i + 1; j < nP; ++j  )  {
            d2 = distance2(i, j);
            if ( minDistance > d2 ) {
                ni = i; nj = j;
                minDistance = d2;
            }
        } // for j
    } // for i
    return minDistance;
}
```

```
double distance2( int i, int j) {
        double dx = p[i][0] – p[j][0];
        double dy = p[i][1] – p[j][1];
        double d2 = dx*dx+dy*dy;
        return d2;
}
```

Modular programming ☺

Code simplification

```
double void find2Points(int &ni, int &nj) {
    double minDistance2 = distance2(0, 1, p);
    ni = 0; nj = 1;
    for ( int i = 0; i < nP; ++i  )  {
        for ( int j = i + 1; j < nP; ++j  )  {
            d2 = distance2(i, j);
            if ( minDistance > d2 ) {
                ni = i; nj = j;
                minDistance = d2;
            }
        } // for j
    } // for i
    return minDistance;
}
```

Code simplification

```
double void find2Points(int &ni, int &nj) {
    double minDistance2 = distance2(0, 1, p);
    ni = 0; nj = 1;
    for ( int i = 0; i < nP; ++i  )  {
        for ( int j = i + 1; j < nP; ++j  )  {
            d2 = distance2(i, j);
            if ( minDistance > d2 ) {
                ni = i; nj = j;
                minDistance = d2;
            }
        } // for j
    } // for i
    return minDistance;
}
```

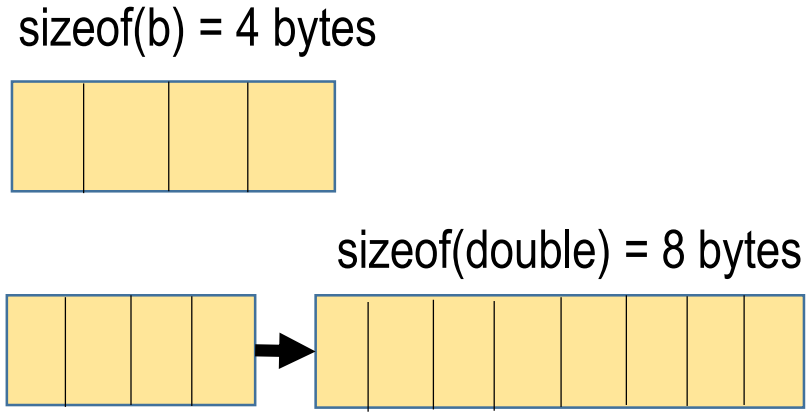
```
double void find2Points(int &ni, int &nj) {
    double minDistance2 = distance2(0, 1, p);
    ni = 0; nj = 1;
    for ( int i = 0; i < nP; ++i  )  {
        for ( int j = i + 1; j < nP; ++j  )  {
            d2 = distance2(i, j);
            if ( minDistance <= d2 ) continue;
            ni = i; nj = j;
            minDistance = d2;
        } // for j
    } // for i
    return minDistance;
}
```

Multidimensional Arrays

double scores[10][5][2];

One-dimensional arrays

```
double *b;

b = new double;
```

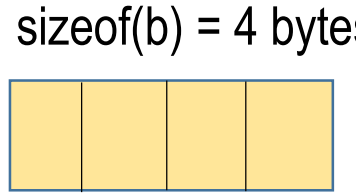


Memory allocation at runtime

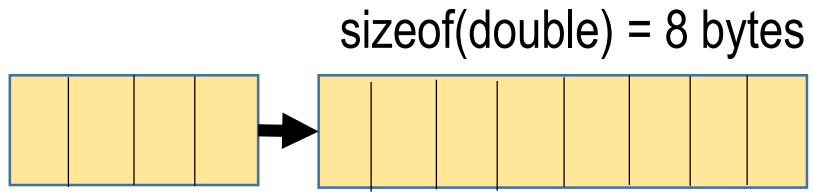
8 bytes

One-dimensional arrays

double *b;

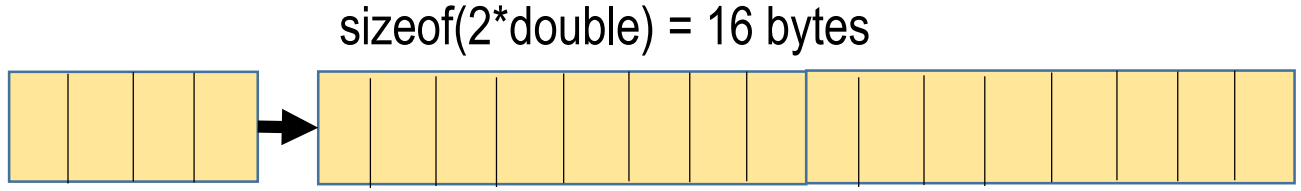


b = new double;



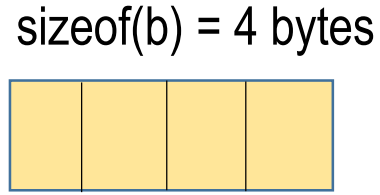
double *c;

c = new double[2];

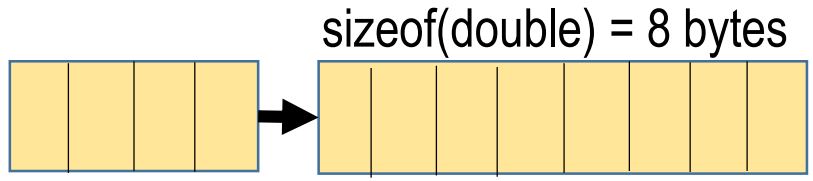


One-dimensional arrays

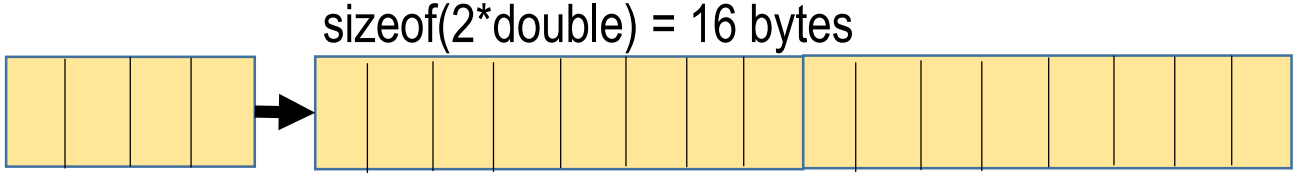
```
double *b;
```



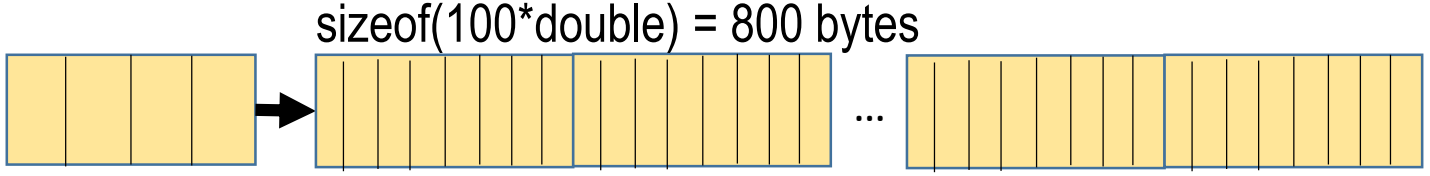

```
b = new double;
```




```
double *c;
c = new double[2];
```

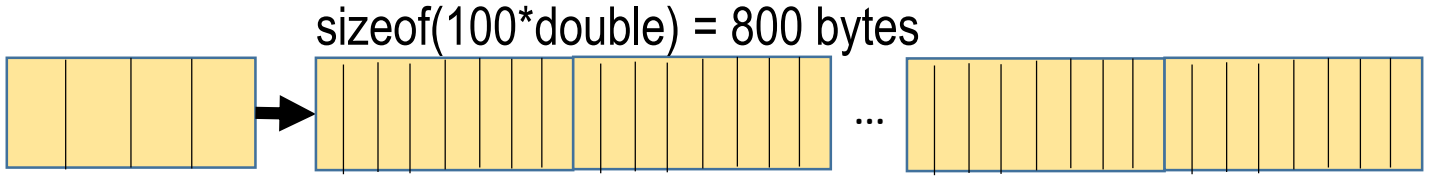



```
double *d;
d = new double[100];
```



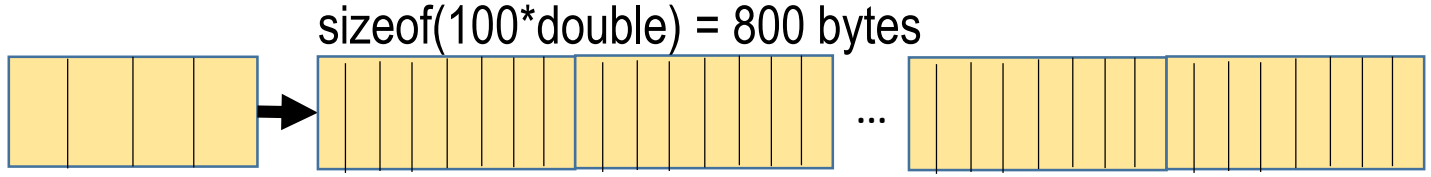
Two-dimensional arrays

```
double *d;
d = new double[100];
```

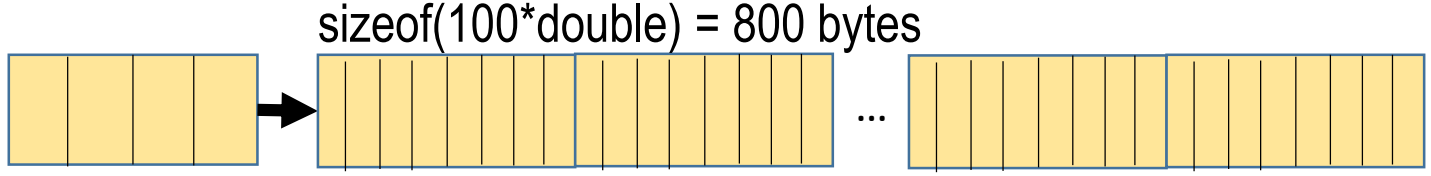


Two-dimensional arrays

```
double *d1;
d1 = new double[100];
```

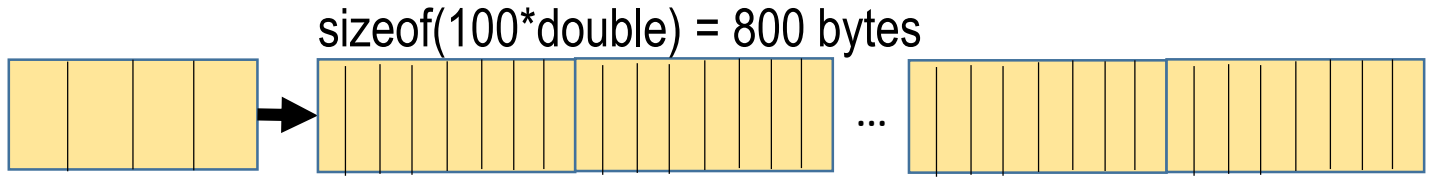



```
double *d0;
d0 = new double[100];
```

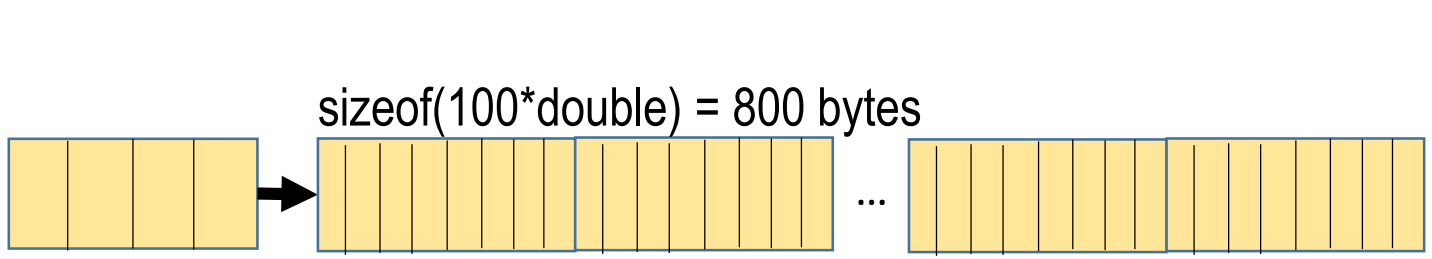


Two-dimensional arrays

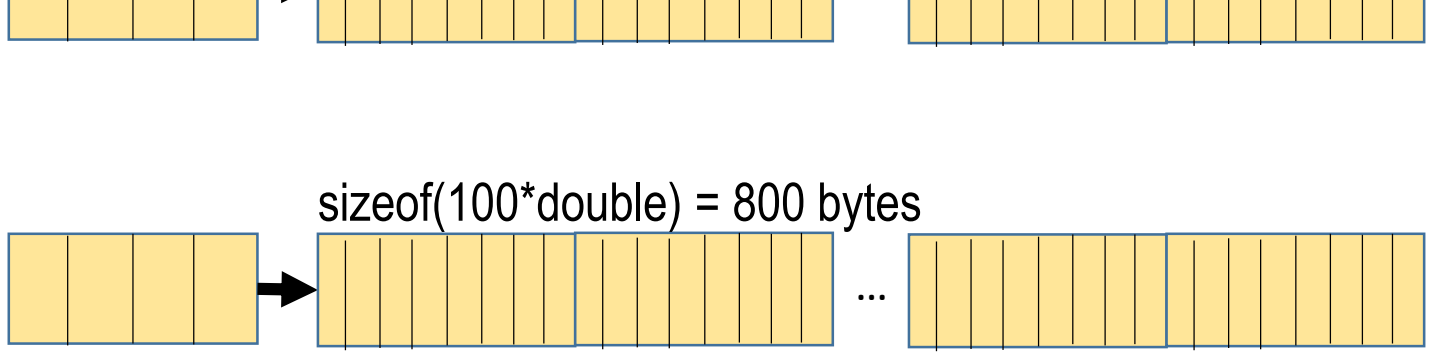
```
double *d2;
d2 = new double[100];
```




```
double *d1;
d1 = new double[100];
```



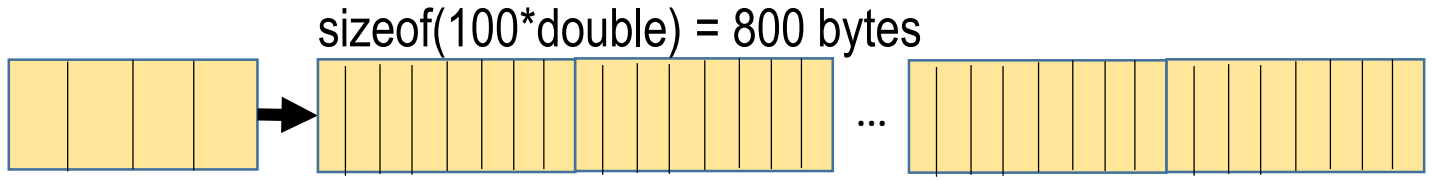

```
double *d0;
d0 = new double[100];
```



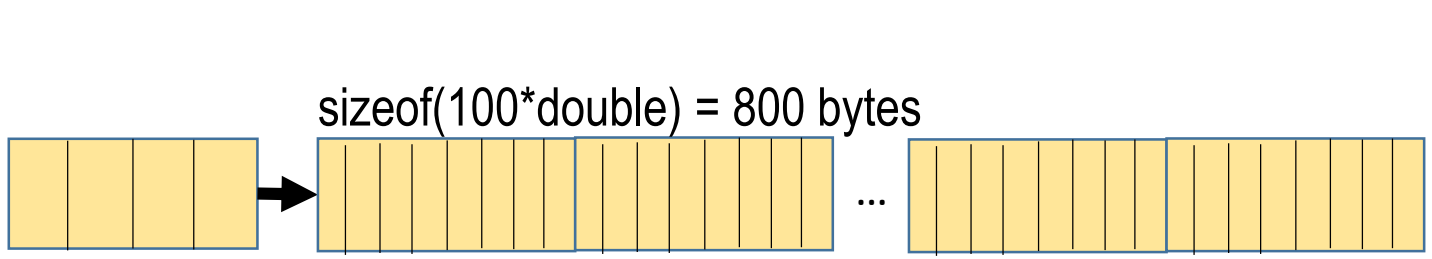
Two-dimensional arrays

.
.
.

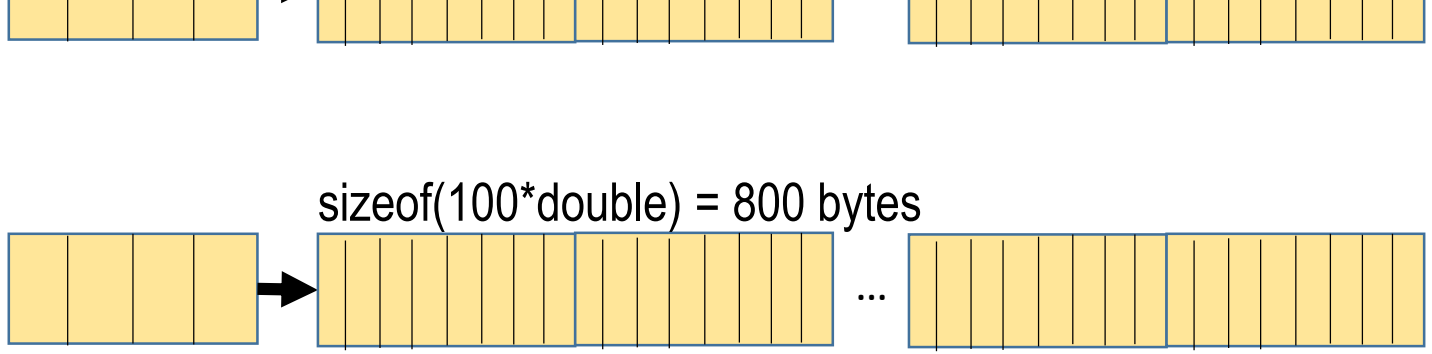
```
double *d2;
d2 = new double[100];
```




```
double *d1;
d1 = new double[100];
```

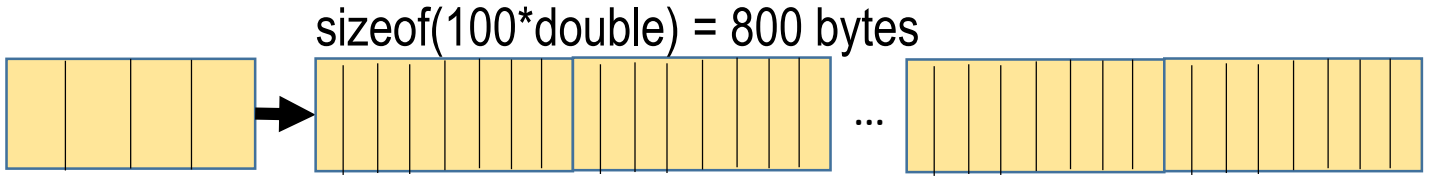



```
double *d0;
d0 = new double[100];
```

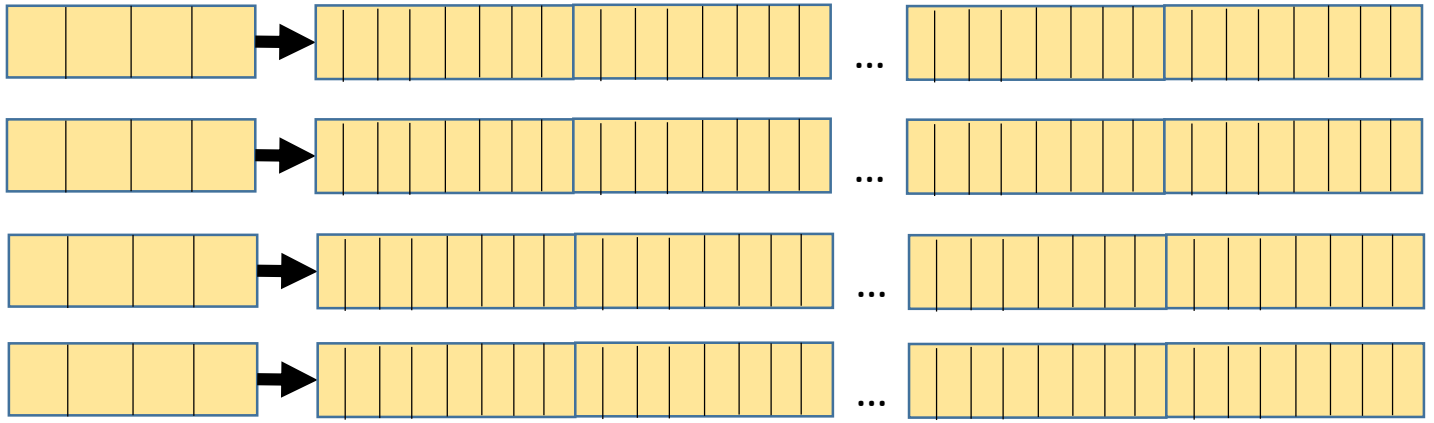


Two-dimensional arrays

```
double *d0;
d0 = new double[100];
```



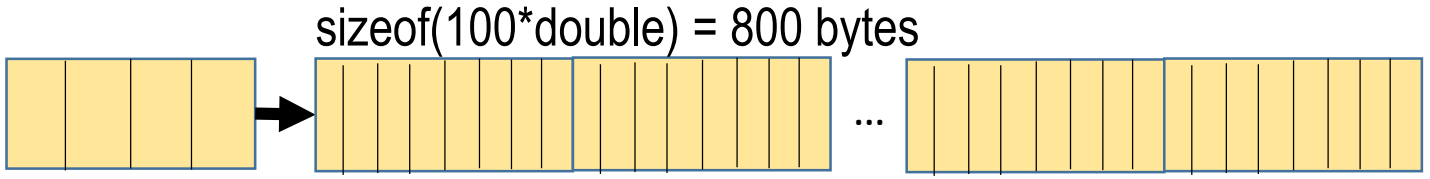

```
double *arr[4];
arr[0] = new double[100];
arr[1] = new double[100];
arr[2] = new double[100];
arr[3] = new double[100];
```



Use an array of pointers.
Each pointer points to an 1-D array.

Two-dimensional arrays

```
double *d0;
d0 = new double[100];
```




```
double *arr[4];
arr[0] = new double[100];
arr[1] = new double[100];
arr[2] = new double[100];
arr[3] = new double[100];
```

Use an array of pointers.
Each pointer points to an 1-D array.

```
double **arr;
arr = new double*[4];
for (int i = 0; i < 4; ++i) {
        arr[i] = new double[100];
}
// same as arr[4][100]
```

Intended Learning Outcomes

- Define arrays
- List some operations on arrays
- Define the linear index of a matrix

Supplemental Materials

Mapping between a one-dimensional array and a two-dimensional array

0	1	2
3	4	5
6	7	8
9	10	11

Mapping between a one-dimensional array and a two-dimensional array

0	1	2
3	4	5
6	7	8
9	10	11

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

Mapping between a one-dimensional array and a two-dimensional array

Fact: The elements of the arrays are stored one by one.

Consider a row-major two-dimensional array.

The elements of a row-major 2D array is stored row by row.

So they are stored as follows:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Thus, they can be treated as being stored in a one-dimensional array

0	1	2
3	4	5
6	7	8
9	10	11

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

Mapping between a one-dimensional array and a two-dimensional array

Fact: The elements of the arrays are stored one by one.

Consider a row-major two-dimensional array.

The elements of a row-major 2D array is stored row by row.

So they are stored as follows:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Given an element a[y][x], what is its index in an one-dimensional array?

index = y*Columns + x

0	1	2
3	4	5
6	7	8
9	10	11

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

Mapping between a one-dimensional array and a two-dimensional array (row-major)

Given an element a[y][x], what is its index in an one-dimensional array?

index = y*Columns + x

Given an index of an element of a 1D array, convert it into a pair of indices of the element of a 2D array.

x = index%Columns;

y = index/Columns; // integer division

0	1	2
3	4	5
6	7	8
9	10	11

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

Mapping between a one-dimensional array and a two-dimensional array (row-major)

Given an element a[y][x], what is its index in an one-dimensional array?

index = y*Columns + x

Given an index of an element of a 1D array, convert it into a pair of indices of the element of a 2D array.

index = y*Columns + x
(index%Columns) = (y*Columns + x)%Columns
index%Columns = x

x = index%Columns;

y = index/Columns; // integer division

0	1	2
3	4	5
6	7	8
9	10	11

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

Mapping between a one-dimensional array and a two-dimensional array (row-major)

Given an element a[y][x], what is its index in an one-dimensional array?

index = y*Columns + x

0	1	2
3	4	5
6	7	8
9	10	11

x = 10%3 = 1
y = 10/3 = 3

Given an index of an element of a 1D array, convert it into a pair of indices of the element of a 2D array.

index = y*Columns + x
(index%Columns) = (y*Columns + x)%Columns
index%Columns = x

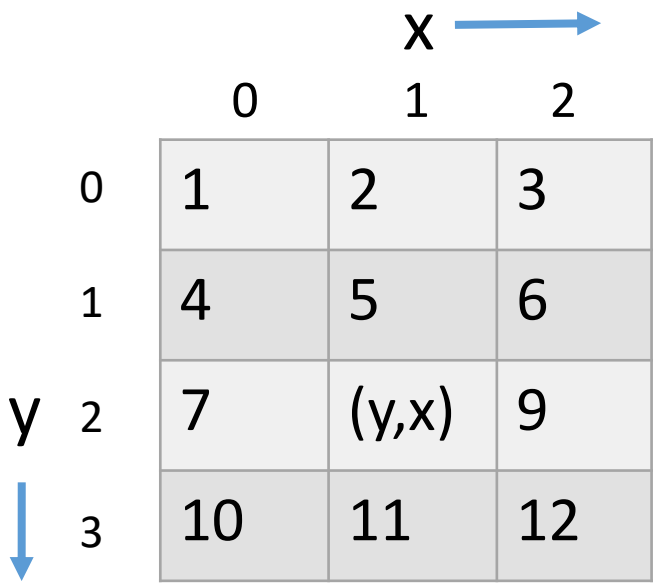
x = index%Columns;

y = index/Columns; // integer division

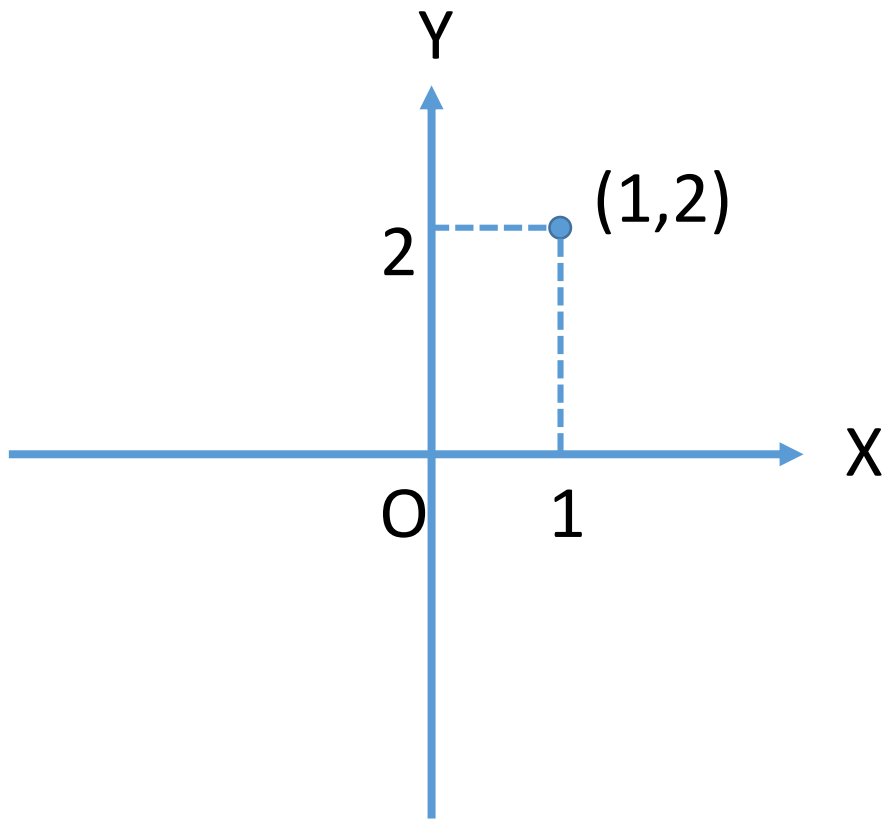
(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

Elements in matrices and coordinates in 2D Space

- 2D Matrix indices and 2D coordinates
- Assume that we use the row-major convention
- An matrix element Coordinates
- a[y][x] vs (x, y)



A 2D Coordinate System

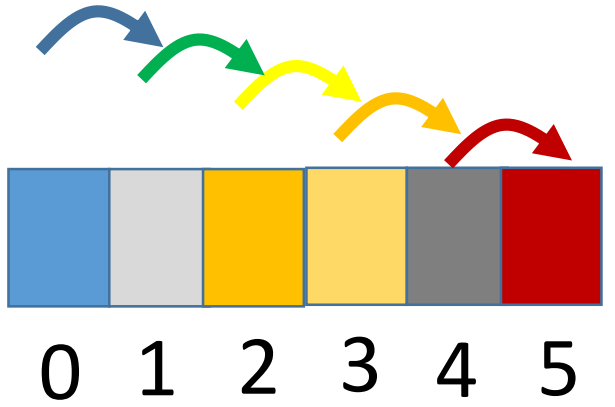


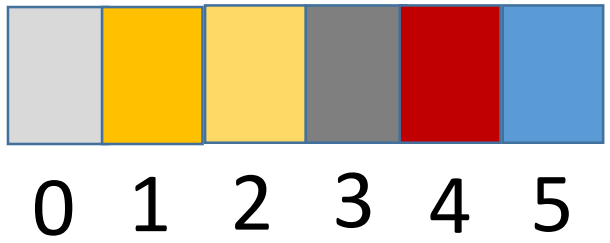
Learn the following topics

- Arrays
- Shuffling
- Shifting
- Searching
- Sorting
- Finding

Shifting Elements : Shift to right

```
n = myList.length();
double temp = myList[n-1]; // Retain the last element
// Shift elements to right
for (int i = 1; i < n; i++)
{
  myList[n- i] = myList[n-i-1];
}
// Move the last element to fill in the first position
myList[0] = temp;
```





Binary Search

- If the key is less than the middle element, we search the key in the first half of the array.
- If the key is equal to the middle element, we find the key in the array.
- If the key is greater than the middle element, we search the key in the second half of the array.

1 2 4 8 13 12 43 51 71

Binary Search

- If the key is less than the middle element, we search the key in the first half of the array.
- If the key is equal to the middle element, we find the key in the array.
- If the key is greater than the middle element, we search the key in the second half of the array.

Example: Search for key = 4

```
Element: 1   2   4   8   13   12   43   51
```

Binary Search

- If the key is less than the middle element, we search the key in the first half of the array.
- If the key is equal to the middle element, we find the key in the array.
- If the key is greater than the middle element, we search the key in the second half of the array.

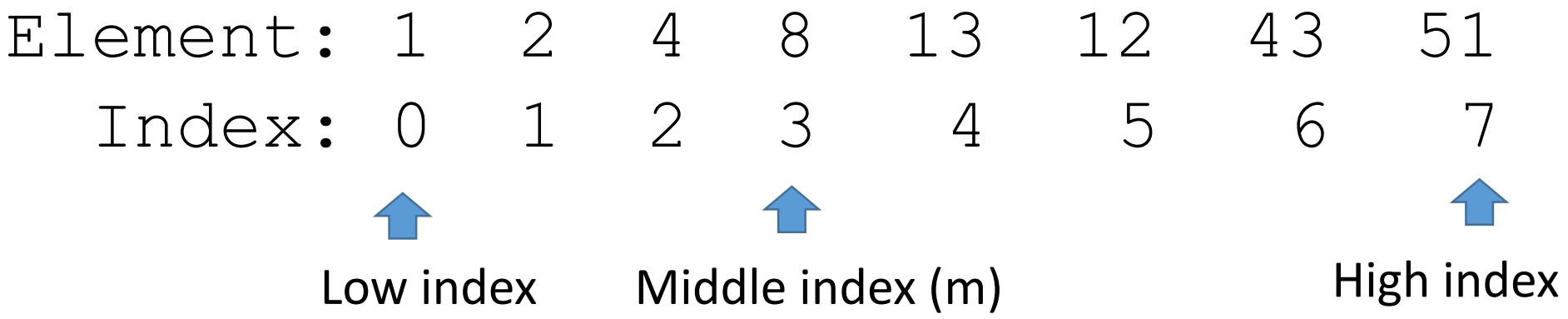
Example: Search for key = 4. m = middle_element

```
Element: 1   2   4   8   13   12   43   51
```

Binary Search

- If the key is less than the middle element, we search the key in the first half of the array.
- If the key is equal to the middle element, we find the key in the array.
- If the key is greater than the middle element, we search the key in the second half of the array.

Example: Search for key = 4. m = middle_element

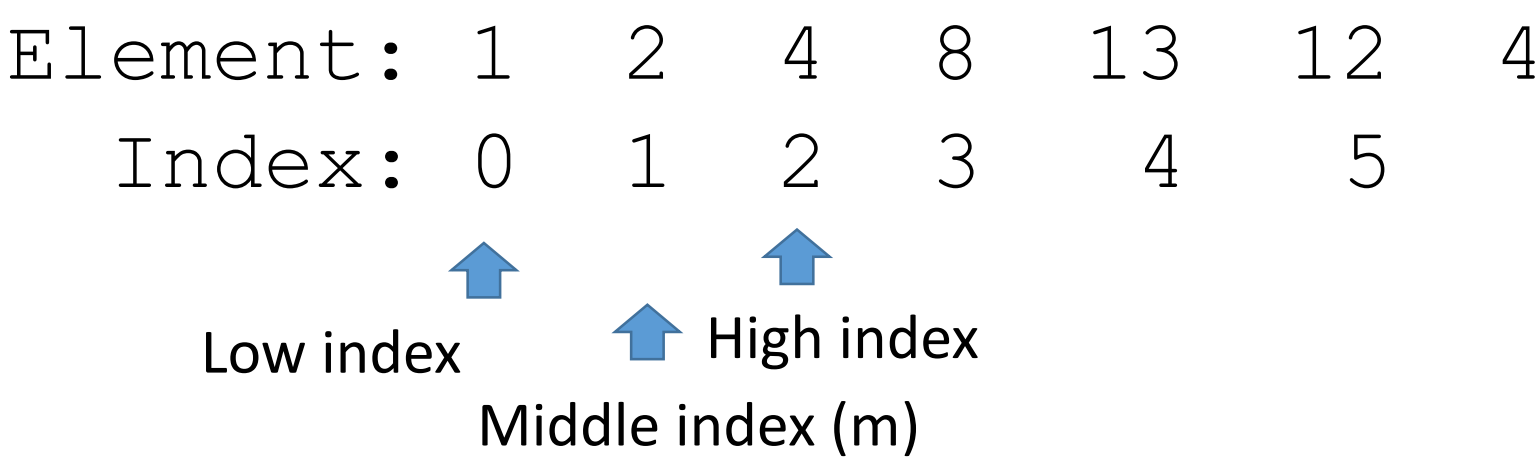


Low index = 0
High index = 7
m = (0+7)/2 = 3

Binary Search

- If the key is less than the middle element, we search the key in the first half of the array.
- If the key is equal to the middle element, we find the key in the array.
- If the key is greater than the middle element, we search the key in the second half of the array.

Example: Search for key = 4. m = middle_element



Low index = 0
High index = 2
m = (0+2)/2 = 1

Binary Search

- If the key is less than the middle element, we search the key in the first half of the array.
- If the key is equal to the middle element, we find the key in the array.
- If the key is greater than the middle element, we search the key in the second half of the array.

Example: Search for key = 4. m = middle_element

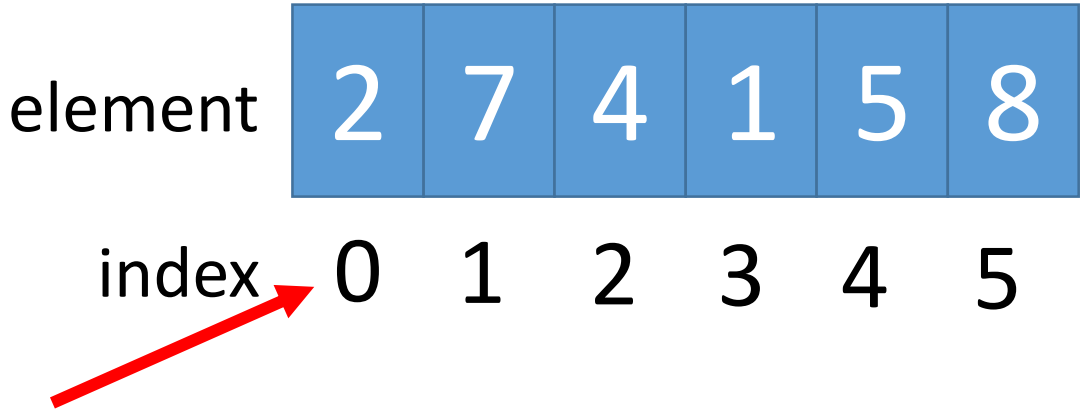
```
Element:  1   2   4   8   13   12   43   51
  Index:  0   1   2   3    4    5    6    7
```

Middle index (m)

Low index = 2
High index = 2
m = (2+2)/2 = 2

Selection Sort

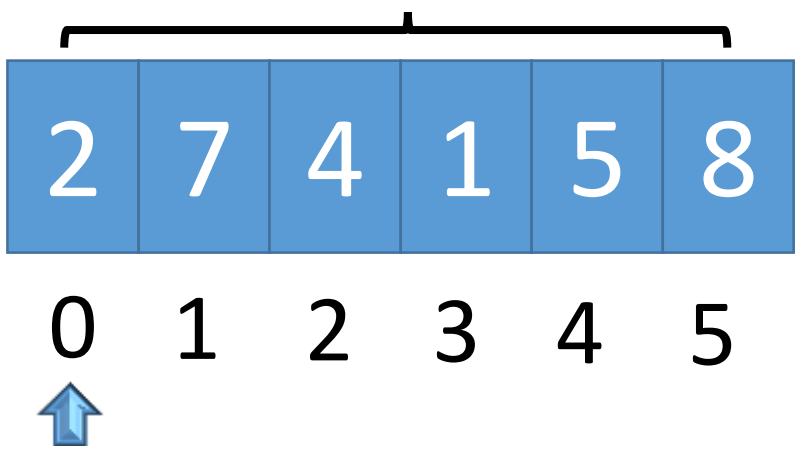
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.



i = 0
Loop
    The remaining elements: from index i to (NUM-1)
    Find the smallest number from remaining elements
    Swap the smallest number to the position i
    i <- i + 1
Repeat if i != (NUM-1)

Selection Sort

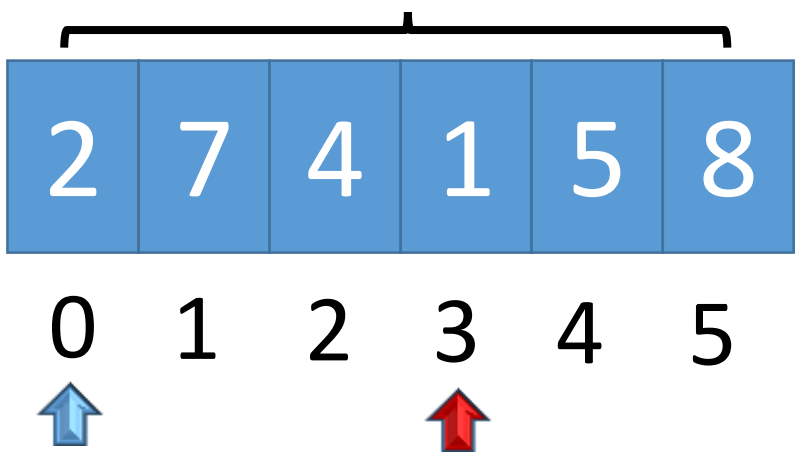
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.



i = 0
Loop
    The remaining elements: from index i to (NUM-1)
    Find the smallest number from remaining elements
    Swap the smallest number to the position i
    i <- i + 1
Repeat if i != (NUM-1)

Selection Sort

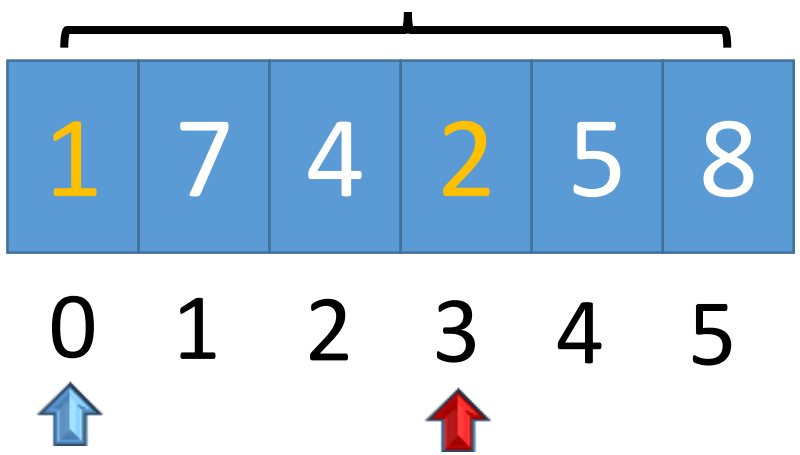
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i <- i + 1
Repeat if  i != (NUM-1)
```

Selection Sort

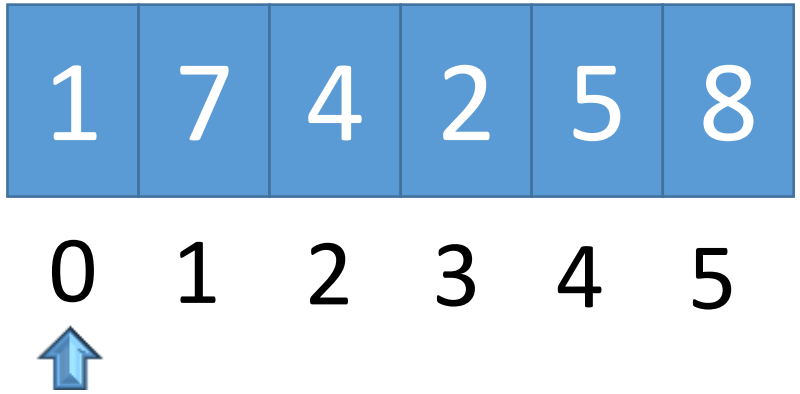
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i <- i + 1
Repeat if  i != (NUM-1)
```

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.



i = 0

Loop

The remaining elements: from index i to (NUM-1)

Find the smallest number from remaining elements

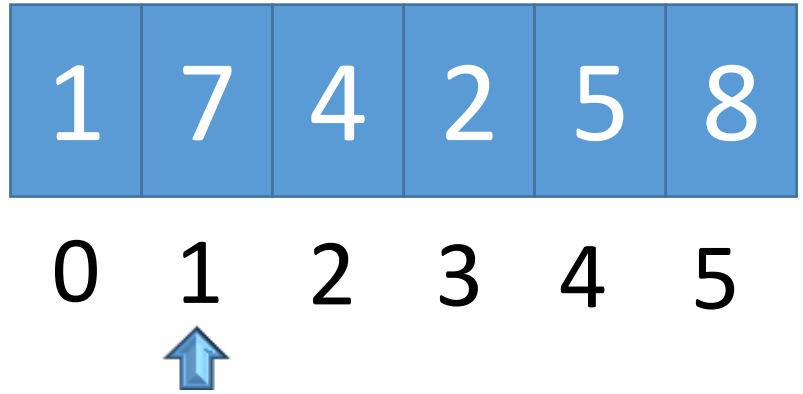
Swap the smallest number to the position i

i <- i + 1

Repeat if i != (NUM-1)

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.



i = 0

Loop

The remaining elements: from index i to (NUM-1)

Find the smallest number from remaining elements

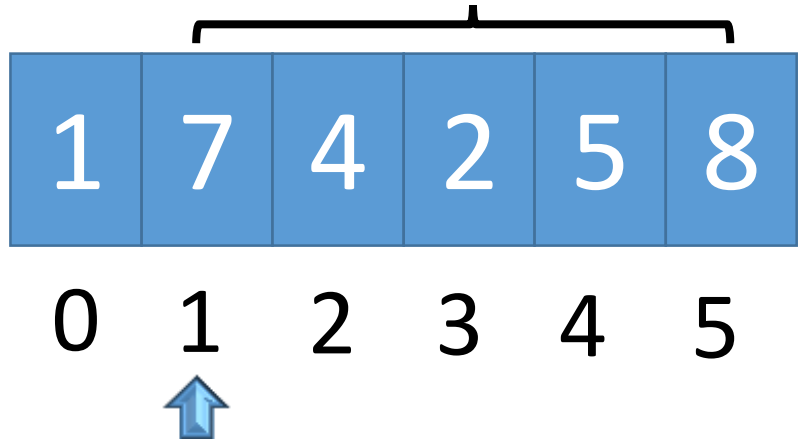
Swap the smallest number to the position i

i <- i + 1

Repeat if i != (NUM-1)

Selection Sort

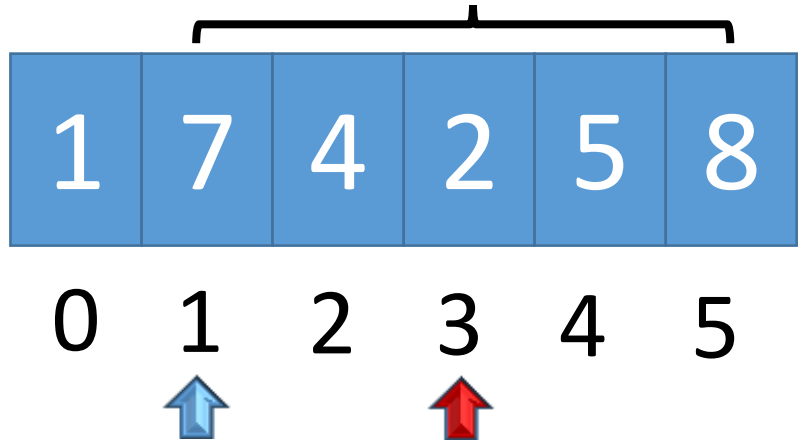
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i <- i + 1
Repeat if  i != (NUM-1)
```

Selection Sort

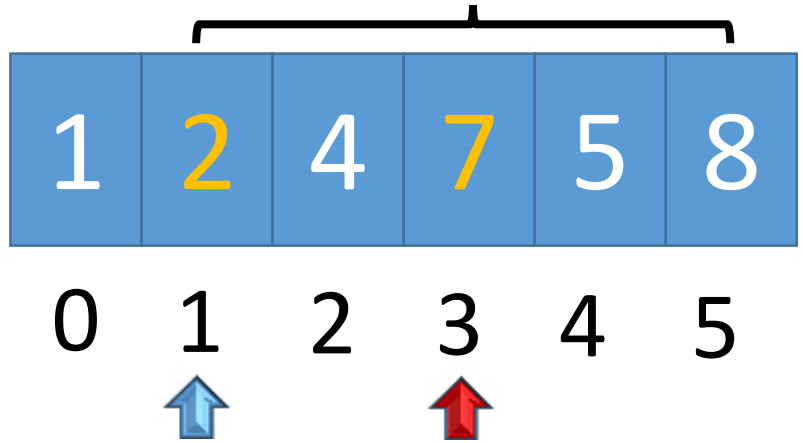
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i <- i + 1
Repeat if  i != (NUM-1)
```

Selection Sort

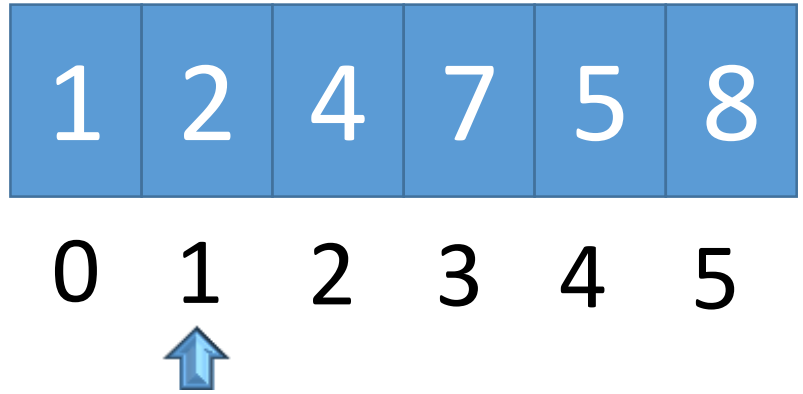
1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.




```
i = 0
Loop
        The remaining elements: from index i to (NUM-1)
        Find the smallest number from remaining elements
        Swap the smallest number to the position i
        i <- i + 1
Repeat if  i != (NUM-1)
```

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.



i = 0

Loop

The remaining elements: from index i to (NUM-1)

Find the smallest number from remaining elements

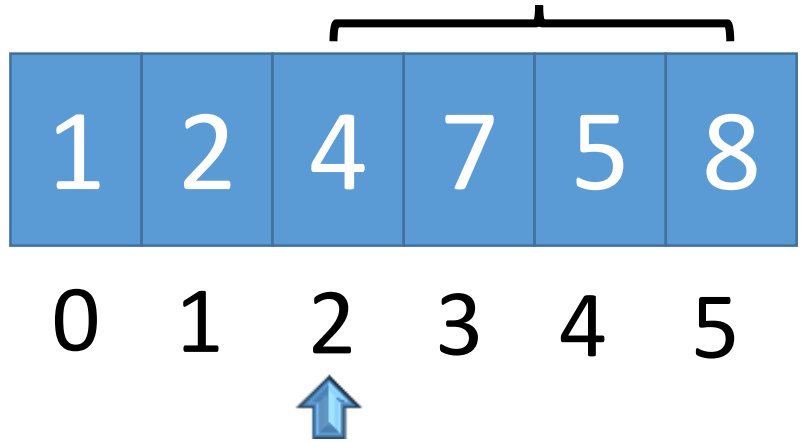
Swap the smallest number to the position i

i <- i + 1

Repeat if i != (NUM-1)

Selection Sort

1. Find the smallest number in the list and places it first.
2. Then it finds the smallest remaining number and places it next to first.
3. Repeat until the list contains only a single number.



i = 0
Loop
    The remaining elements: from index i to (NUM-1)
    Find the smallest number from remaining elements
    Swap the smallest number to the position i
    i <- i + 1
Repeat if i != (NUM-1)

Insertion sort

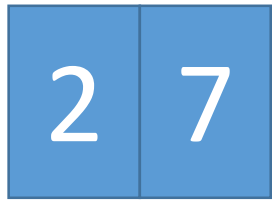
1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.

Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.

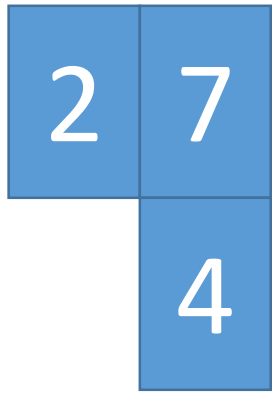
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.

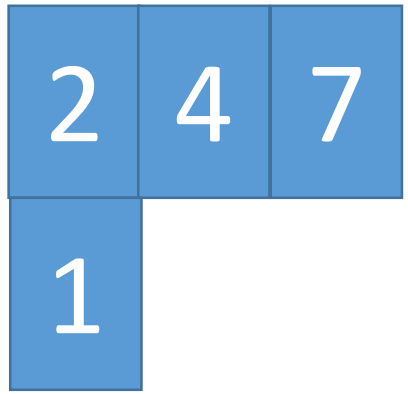


Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.

Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.

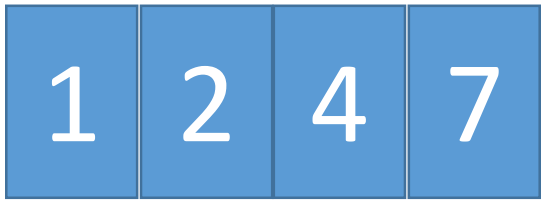


Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.

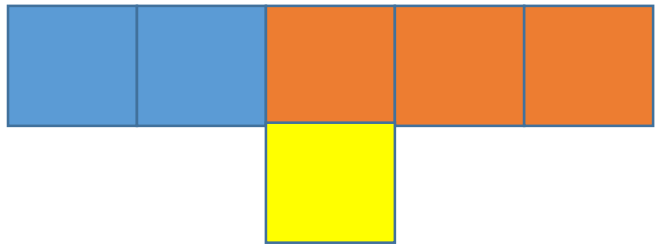
Insertion sort

1. Sort a list of values by repeatedly inserting an unsorted element into a sorted sublist.
2. Repeat the process until the whole list is sorted.



Inserting an element into an array

```
int a[100];
int numElements = 50;
……
int new_element = 34;
int index = 2;
// insert new_element at a[index]
```



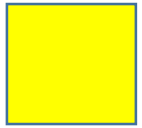
……

Inserting an element into an array

```
int a[100];
int numElements = 50;
……
int new_element = 34;
int index = 2;
// insert new_element at a[index]
```

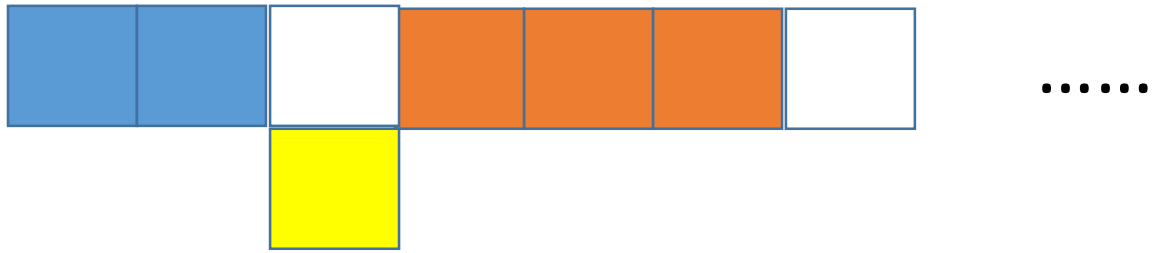
……

Not used yet



Inserting an element into an array

```
int a[100];
int numElements = 50;
……
int new_element = 34;
int index = 2;
// insert new_element at a[index]
```



……

Inserting an element into an array

```
int a[100];
int numElements = 50;
……
int new_element = 34;
int index = 2;
// insert new_element at a[index]
```

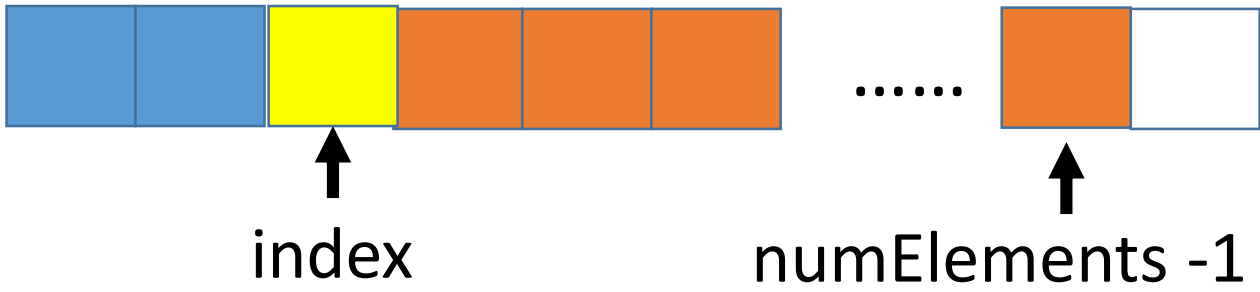
……

Inserting an element into an array

1) Move the elements from index to numElements the right.
2) set a[index] = new_element;
3) Increase numElements by 1

```
for ( int i = numElements; i > index; --i) {
        a[i] = a[i-1];
}
a[index] = new_element;
++ numElements;
```

```
int a[100];
int numElements = 50;
……
int new_element = 34;
int index = 2;
// insert new_element at a[index]
```



Example

```
string myString;
cin >> myString;
```

Input:

This is a good program!

What is stored in myString?

Example

```
string myString;
cin.getline(myString, 256, '\n');
```

Input:

This is a good program!

What is stored in myString?

Example

```
string myString;
cin >> myString;
```

Input:

This is a good program!

What is stored in myString?

```
string myString;
cin.getline(myString, 256, '\n');
```

Input:

This is a good program!

What is stored in myString?

srand(10); // seed

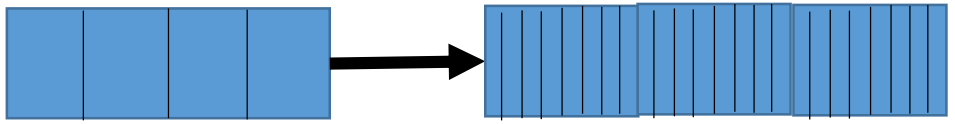
```
int s = time(0);          // save the seed. So that we can reuse it for
                          // checking/debugging our program.

srand(s); // seed
int a = rand( );

rand();

                                          // Use the same seed.
1, 5, 4, 3, 2, 7, 9, 6, 5, 4             // We get the same sequence of numbers.
; simulation
```

Create a two-dimensional array

double *a[5];

a[0] = new double;

a[1] = new double[100];



……

a[2] = new double[100];

……

a[0][0] =

a[1][99] =

```
int **p;

int NR = 100;
p = new int*[NR];
int NC = 64;

for (int i = 0; i < NR; ++i) {
        p[i] = new int[NC];
}
```

```
int **p;

int NR = 100;
p = new int*[NR];   // create a set of pointers to integers
int NC = 64;

for (int i = 0; i < NR; ++i) {
        p[i] = new int[NC];
}

p[r][c] = 10;
```

Passing Arrays to Functions

We pass values, variables, and arrays to functions.

What is the meaning of *arr = 8?

How do we know the number of elements in the array?

```
void g( int *arr ) {
        arr[0] = 5;
        arr[5] = 7;
        *arr = 8;
}
void k( int arr[ ] ) {
        arr[0] = 1;
        arr[1] = 5;
}
```