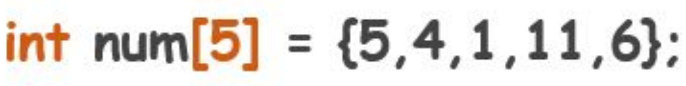
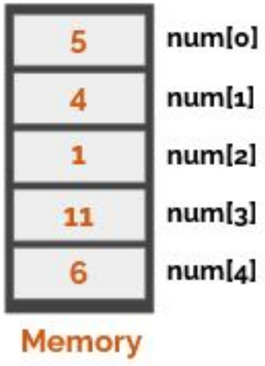
# Sorting in Arrays

# Array

Previously, we **initialized** an integer array for 5 numbers.

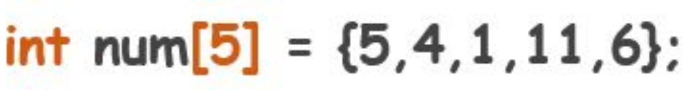


```
int num[5] = {5,4,1,11,6};
```

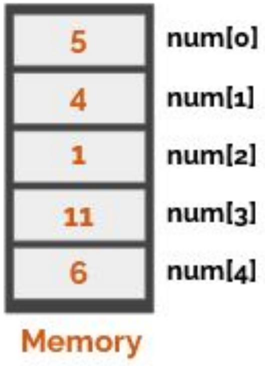


# Unsorted Array

The data in this num array is not in an order.



```
int num[5] = {5,4,1,11,6};
```



| | |
|---|---|
| 5 | num[0] |
| 4 | num[1] |
| 1 | num[2] |
| 11 | num[3] |
| 6 | num[4] |

Memory

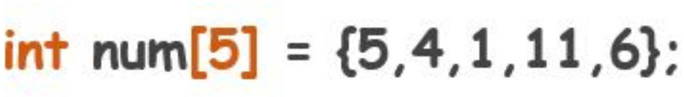
# Sorting

This process of ordering data elements in the array is called Sorting.
There are two types of orders in Sorting.

1. Ascending Order

2. Descending Order

# Sorting

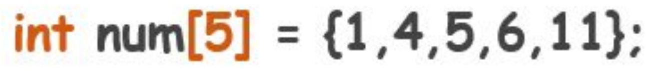


```
int num[5] = {5,4,1,11,6};
```

This process of ordering data elements in the array is called Sorting.

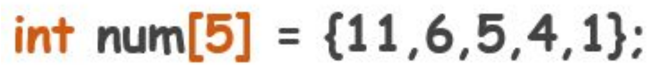
There are two types of orders in Sorting.

1. Ascending Order



```
int num[5] = {1,4,5,6,11};
```

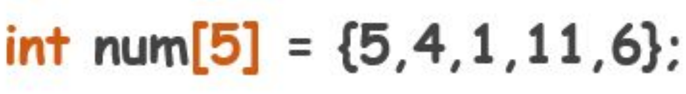
2. Descending Order



```
int num[5] = {11,6,5,4,1};
```

# Working Example

Write a C++ program that Sorts the array in Descending Order.



```
int num[5] = {5,4,1,11,6};
```

# Algorithm for Sorting

Step 1: Make a new array and set the index to 0.

Step 2: Find the Largest Element in the original array

Step 3: Place the largest Element on the available index of the new array.

Step 4: Replace the largest element in the original array with a lowest number (i.e., -1)

Step 5: Update the index of the new array.

Step 6: Repeat Step 2, 3, 4 and 5

# Algorithm for Sorting: Step 1

Original Array

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | 11 | 6 |

New Array

| | | | | |
|---|---|---|---|---|
| | | | | |

# Algorithm for Sorting: Step 2

## Original Array

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | 11 | 6 |

## New Array

| | | | | |
|---|---|---|---|---|
| | | | | |

# Algorithm for Sorting: Step 3

Original Array

|  |  |  |  |  |
|---|---|---|---|---|
| 5 | 4 | 1 | 11 | 6 |

New Array

|  |  |  |  |  |
|---|---|---|---|---|
| 11 |  |  |  |  |

# Algorithm for Sorting: Step 4

## Original Array

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | -1 | 6 |

## New Array

| | | | | |
|---|---|---|---|---|
| 11 | | | | |

# Algorithm for Sorting: Step 5

Original Array

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | -1 | 6 |

New Array

| | | | | |
|---|---|---|---|---|
| 11 | | | | |

# Algorithm for Sorting: 2nd Iteration

Original Array

|  |  |  |  |  |
|---|---|---|---|---|
| 5 | 4 | 1 | -1 | 6 |

New Array

|  |  |  |  |  |
|---|---|---|---|---|
| 11 |  |  |  |  |

# Algorithm for Sorting: 2nd Iteration

## Original Array

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | -1 | -1 |

## New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | | | |

# Algorithm for Sorting: 3rd Iteration

## Original Array

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | -1 | -1 |

## New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | | | |

# Algorithm for Sorting: 3rd Iteration

Original Array

| | | | | |
|---|---|---|---|---|
| -1 | 4 | 1 | -1 | -1 |

New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | | |

# Algorithm for Sorting: 4th Iteration

## Original Array

| | | | | |
|---|---|---|---|---|
| -1 | 4 | 1 | -1 | -1 |

## New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | | |

# Algorithm for Sorting: 4th Iteration

## Original Array

|  |  |  |  |  |
|---|---|---|---|---|
| -1 | -1 | 1 | -1 | -1 |

## New Array

|  |  |  |  |  |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 |  |

# Algorithm for Sorting: 5th Iteration

## Original Array

| | | | | |
|---|---|---|---|---|
| -1 | -1 | 1 | -1 | -1 |

## New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | |

# Algorithm for Sorting: 5th Iteration

## Original Array

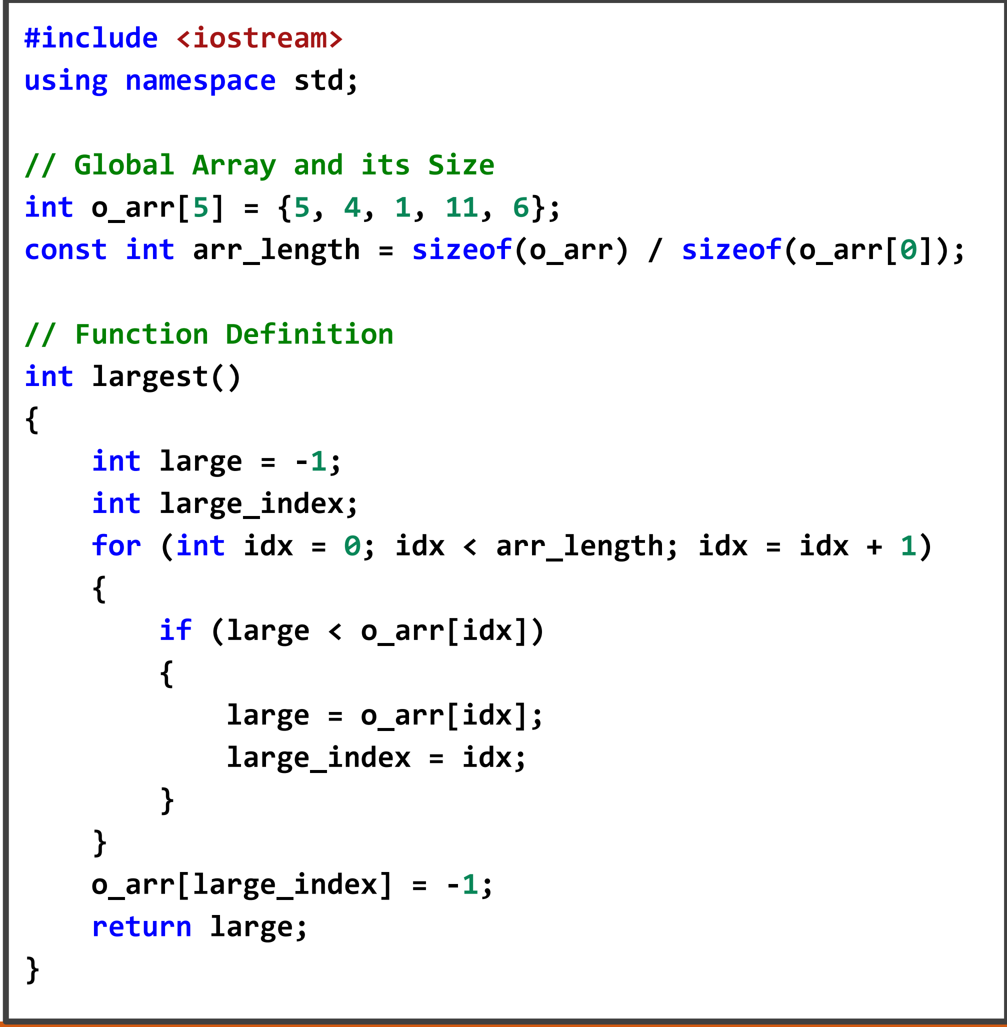
| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |

## New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

# Solution

Let's make a function that search an array and **find the largest element** from the array and then returns it.



```cpp
#include <iostream>
using namespace std;

// Global Array and its Size
int o_arr[5] = {5, 4, 1, 11, 6};
const int arr_length = sizeof(o_arr) / sizeof(o_arr[0]);

// Function Definition
int largest()
{
    int large = -1;
    int large_index;
    for (int idx = 0; idx < arr_length; idx = idx + 1)
    {
        if (large < o_arr[idx])
        {
            large = o_arr[idx];
            large_index = idx;
        }
    }
    o_arr[large_index] = -1;
    return large;
}
```

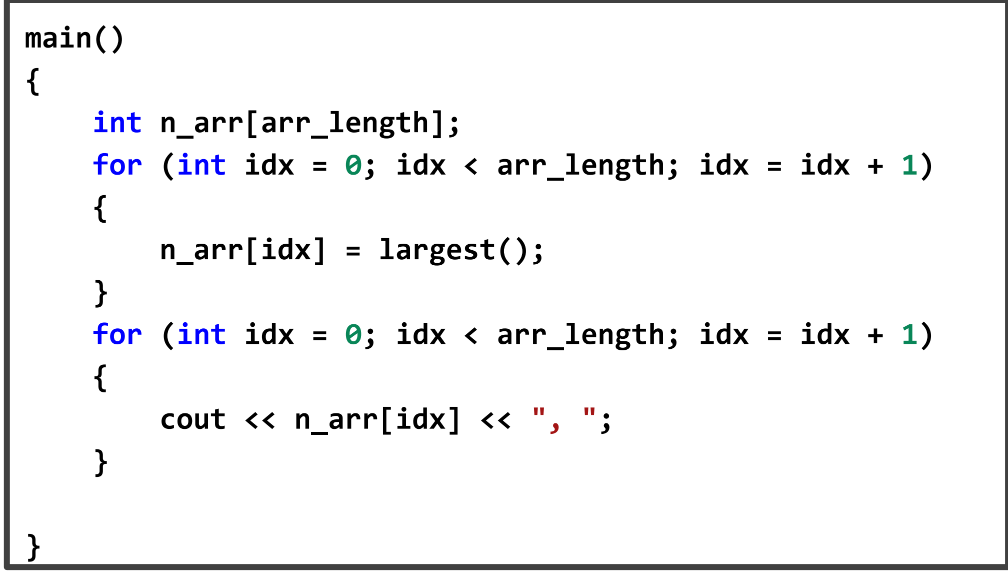
# Solution

Now, use that function and make a new array and populate its elements in descending order.

```
main()
{
    int n_arr[arr_length];
    for (int idx = 0; idx < arr_length; idx = idx + 1)
    {
        n_arr[idx] = largest();
    }
}
```

# Solution

Now, print the new array.



```
main()
{
    int n_arr[arr_length];
    for (int idx = 0; idx < arr_length; idx = idx + 1)
    {
        n_arr[idx] = largest();
    }
    for (int idx = 0; idx < arr_length; idx = idx + 1)
    {
        cout << n_arr[idx] << ", ";
    }

}
```

# Algorithm for Sorting

There are different methods for sorting. Lets see another one which sorts the elements in the same array.

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | 11 | 6 |

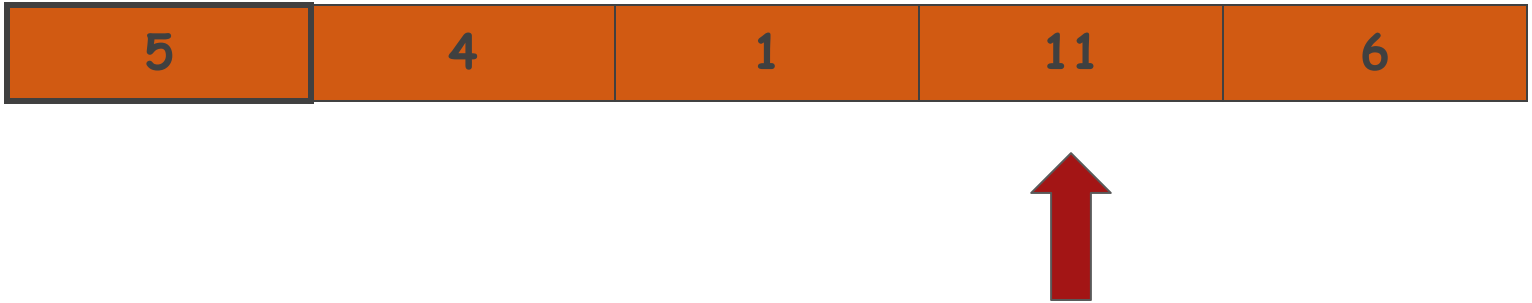
# Algorithm for Sorting

**Step 1:** Sort the 0th index by swapping with the largest element.

| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | 11 | 6 |

# Algorithm for Sorting

**Step 1:** Sort the 0th index by swapping with the largest element.



| | | | | |
|---|---|---|---|---|
| 5 | 4 | 1 | 11 | 6 |

# Algorithm for Sorting

**Step 1:** Sort the 0th index by swapping with the largest element.

| | | | | |
|---|---|---|---|---|
| 11 | 4 | 1 | 5 | 6 |

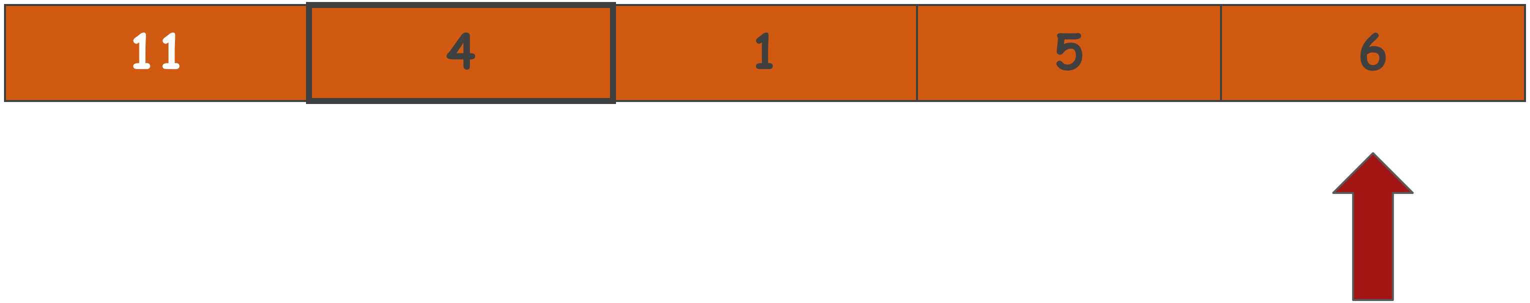
# Algorithm for Sorting

**Step 2:** Sort the 1st index by swapping with the largest element in the rest of unsorted array.

| | | | | |
|---|---|---|---|---|
| 11 | 4 | 1 | 5 | 6 |

# Algorithm for Sorting

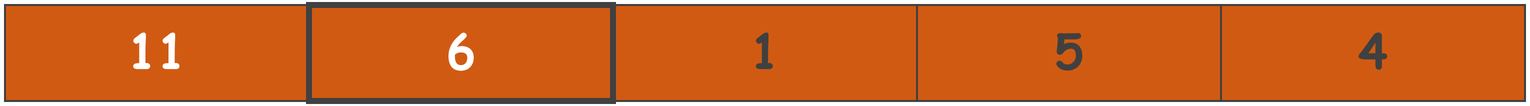
**Step 2:** Sort the 1st index by swapping with the largest element in the rest of unsorted array.



| | | | | |
|---|---|---|---|---|
| 11 | 4 | 1 | 5 | 6 |

# Algorithm for Sorting

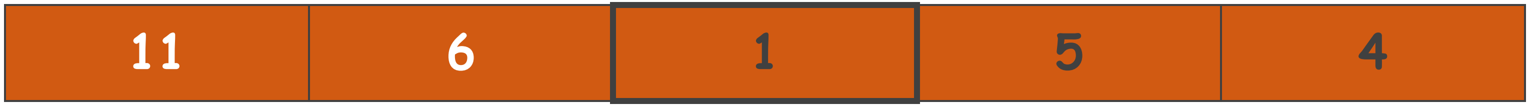
**Step 2:** Sort the 1st index by swapping with the largest element in the rest of unsorted array.



| | | | | |
|---|---|---|---|---|
| 11 | 6 | 1 | 5 | 4 |

# Algorithm for Sorting

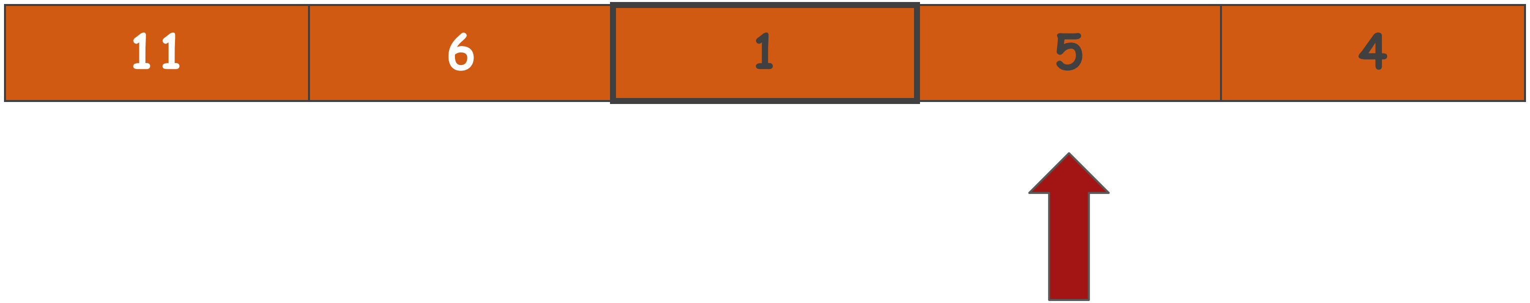
**Step 3:** Sort the 2nd index by swapping with the largest element in the rest of unsorted array.



| | | | | |
|---|---|---|---|---|
| 11 | 6 | 1 | 5 | 4 |

# Algorithm for Sorting

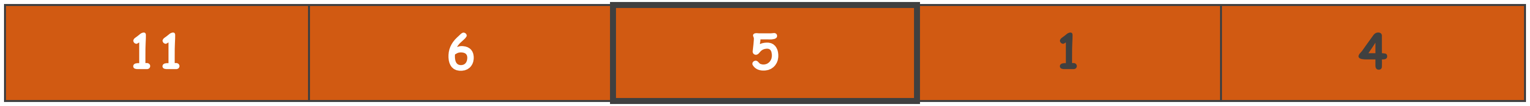
**Step 3:** Sort the 2nd index by swapping with the largest element in the rest of unsorted array.



| | | | | |
|---|---|---|---|---|
| 11 | 6 | 1 | 5 | 4 |

# Algorithm for Sorting

**Step 3:** Sort the 2nd index by swapping with the largest element in the rest of unsorted array.



| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 1 | 4 |

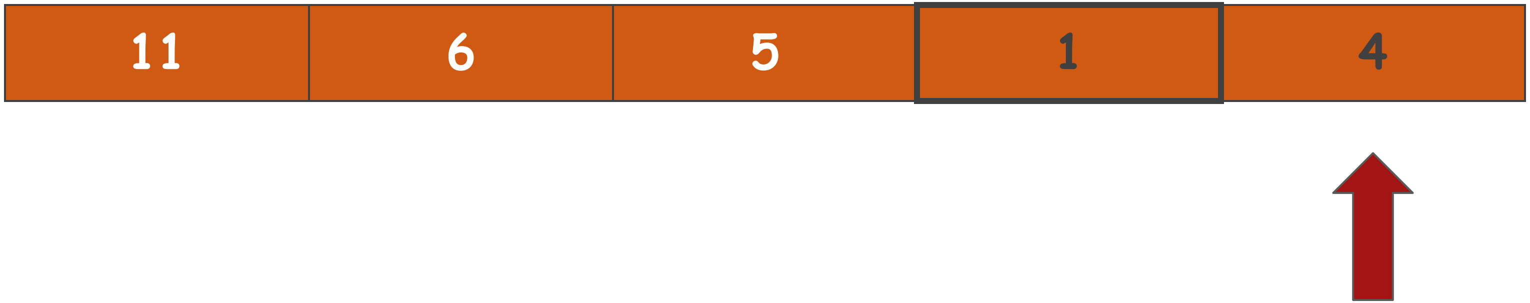
# Algorithm for Sorting

**Step 4:** Sort the 3rd index by swapping with the largest element in the rest of unsorted array.

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 1 | 4 |

# Algorithm for Sorting

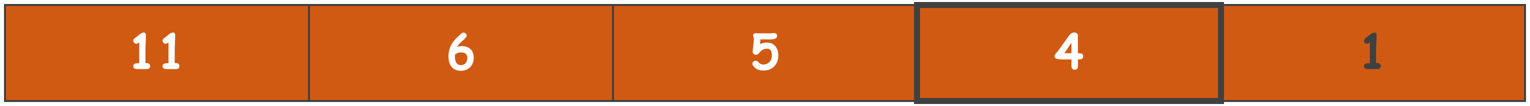
**Step 4:** Sort the 3rd index by swapping with the largest element in the rest of unsorted array.



| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 1 | 4 |

# Algorithm for Sorting

**Step 4:** Sort the 3rd index by swapping with the largest element in the rest of unsorted array.



| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

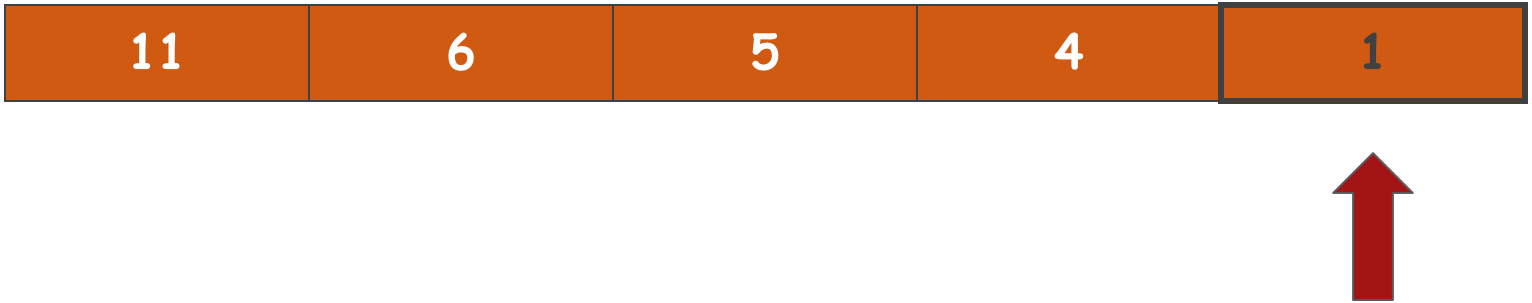
# Algorithm for Sorting

**Step 5:** Sort the 4th index by swapping with the largest element in the rest of unsorted array.

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

# Algorithm for Sorting

**Step 5:** Sort the 4th index by swapping with the largest element in the rest of unsorted array.



| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

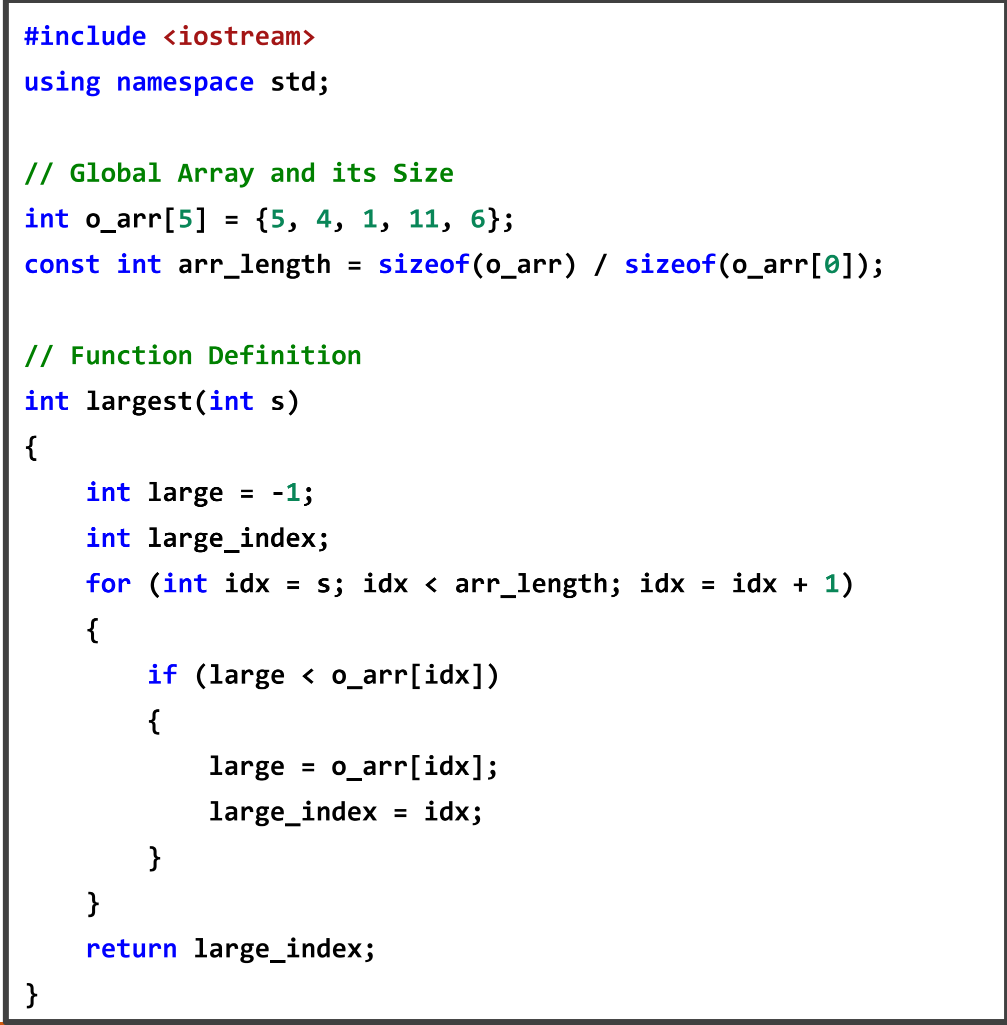
# Algorithm for Sorting

**Step 5:** Sort the 4th index by swapping with the largest element in the rest of unsorted array.

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

# Solution

Let's make a function that search an array from a **specific index** and **find the largest index** from the array.



```cpp
#include <iostream>
using namespace std;

// Global Array and its Size
int o_arr[5] = {5, 4, 1, 11, 6};
const int arr_length = sizeof(o_arr) / sizeof(o_arr[0]);

// Function Definition
int largest(int s)
{
    int large = -1;
    int large_index;
    for (int idx = s; idx < arr_length; idx = idx + 1)
    {
        if (large < o_arr[idx])
        {
            large = o_arr[idx];
            large_index = idx;
        }
    }
    return large_index;
}
```

# Solution

Now, use that function to get the index of the largest element.

```
main()
{
    int largest_idx;
    int temp;
    for (int idx = 0; idx < arr_length; idx = idx + 1)
    {
        largest_idx = largest(idx);

    }
}
```

# Solution

Now, we have to swap the largest element with the index on which we are present. For that we have to use a temporary variable. So our data doesn't get lost.

```
main()
{
    int largest_idx;
    int temp;
    for (int idx = 0; idx < arr_length; idx = idx + 1)
    {
        largest_idx = largest(idx);
        temp = o_arr[largest_idx];
        o_arr[largest_idx] = o_arr[idx];
        o_arr[idx] = temp;
    }
}
```

# Solution

Now, print the new array.

```
main()
{
    int largest_idx;
    int temp;
    for (int idx = 0; idx < arr_length; idx = idx + 1)
    {
        largest_idx = largest(idx);
        temp = o_arr[largest_idx];
        o_arr[largest_idx] = o_arr[idx];
        o_arr[idx] = temp;
    }
    for (int idx = 0; idx < arr_length; idx = idx + 1)
    {
        cout << o_arr[idx] << ", ";
    }
}
```

# Algorithm for Sorting

## 1st Method

Original Array

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |

New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

## 2nd Method

Original Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

# Algorithm for Sorting

1st Method

Original Array

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |

New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

Out of Place Sorting

2nd Method

Original Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

In Place Sorting

# Algorithm for Sorting

Which one is better?

1st Method

Original Array

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |

New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

Out of Place Sorting

2nd Method

Original Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

In Place Sorting

# Algorithm for Sorting

## Which one is better?

### 1st Method

Original Array

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |

New Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

Out of Place Sorting

### 2nd Method

Original Array

| | | | | |
|---|---|---|---|---|
| 11 | 6 | 5 | 4 | 1 |

✓

In Place Sorting

# Learning Objective

In this lecture, we learnt how to use arrays to solve real world problems of sorting the data elements of the array in an order

# Conclusion

- The process of ordering data elements in the array is called Sorting. There are two types of orders in Sorting.
  - Ascending Order
  - Descending Order
- If one arranges the data elements in order of increasing number then it is called the array is Sorted in Ascending order. If one arranges the data elements in order of decreasing number then it is called the array is Sorted in Descending order.

# Self Assessment

1. Write a program that takes nine numbers between 1 and 10 (excluding one number) and returns the missing number. Note: The array of numbers will be unsorted (not in order). Only one number will be missing.

| Input | Output |
|---|---|
| [1, 2, 3, 4, 6, 7, 8, 9, 10] | 5 |
| [7, 2, 3, 6, 5, 9, 1, 4, 8] | 10 |
| [10, 5, 1, 2, 4, 6, 8, 3, 9] | 7 |

# Self Assessment

2. Write a C++ program that takes an unsorted array and returns the nth smallest integer entered by the user. (the smallest integer is the first smallest, the second smallest integer is the second smallest, etc).

Note:

- n will always be >= 1.
- Each number in the array will be distinct (no duplicates will be there).
- Given an out of bounds parameter (e.g. an array is of size k), and you are asked to find the m > k smallest integer, return -1.

# Self Assessment

## Test Cases:

| Input | Output |
|---|---|
| [1, 3, 5, 7]<br>1 | 1 |
| [1, 3, 5, 7]<br>3 | 5 |
| [7, 3, 5, 1]<br>2 | 3 |

# Self Assessment

3. Given a sequence of integers as an array, determine whether it is possible to obtain a strictly increasing sequence by removing no more than one element from the array.

Note: sequence $a_0, a_1, ..., a_n$ is considered to be a strictly increasing if $a_0 < a_1 < ... < a_n$. Sequence containing only one element is also considered to be strictly increasing.

# Self Assessment

Example:

- For sequence = [1, 3, 2, 1], the output should be solution(sequence) = false.

There is no one element in this array that can be removed in order to get a strictly increasing sequence.

- For sequence = [1, 3, 2], the output should be solution(sequence) = true.

As you can remove 3 from the array to get the strictly increasing sequence [1, 2]. Alternately, you can remove 2 to get the strictly increasing sequence [1, 3].

# Self Assessment

## Test Cases:

| Input | Output |
|---|---|
| sequence: [3, 5, 67, 98, 3] | 1 |
| sequence: [123, -17, -5, 1, 2, 3, 12, 43, 45] | 1 |
| sequence: [0, -2, 5, 6] | 1 |
| sequence: [1, 2, 1, 2] | 0 |