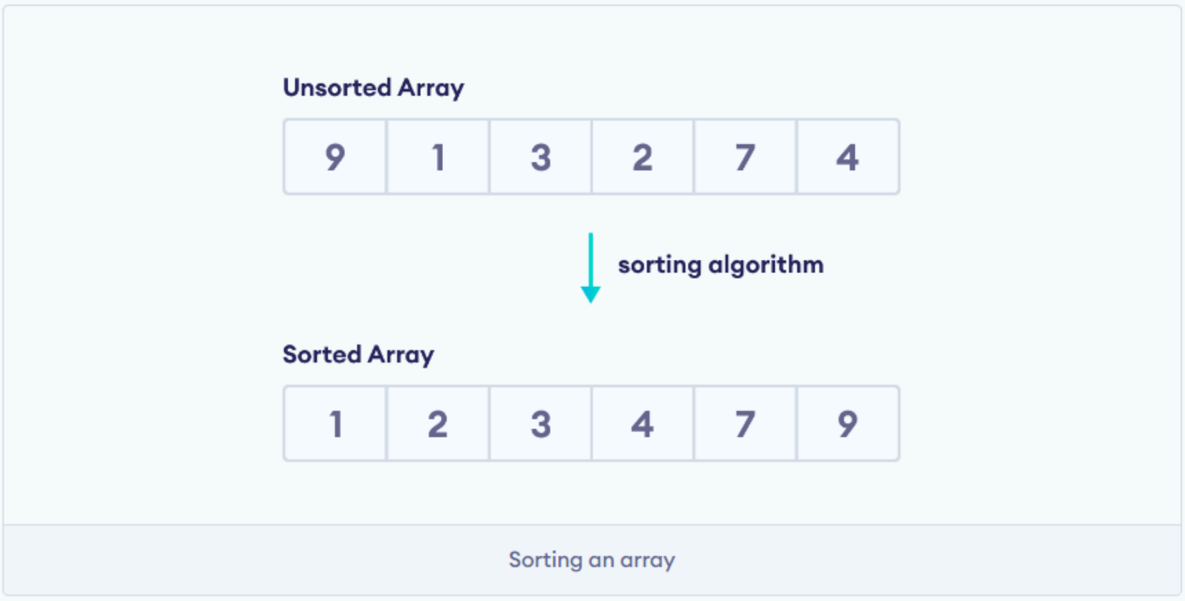
Reference

Sorting

A sorting algorithm is used to arrange elements of an array/list in a specific order. For example,



Sorting an array

- Stability of Sorting Algorithm

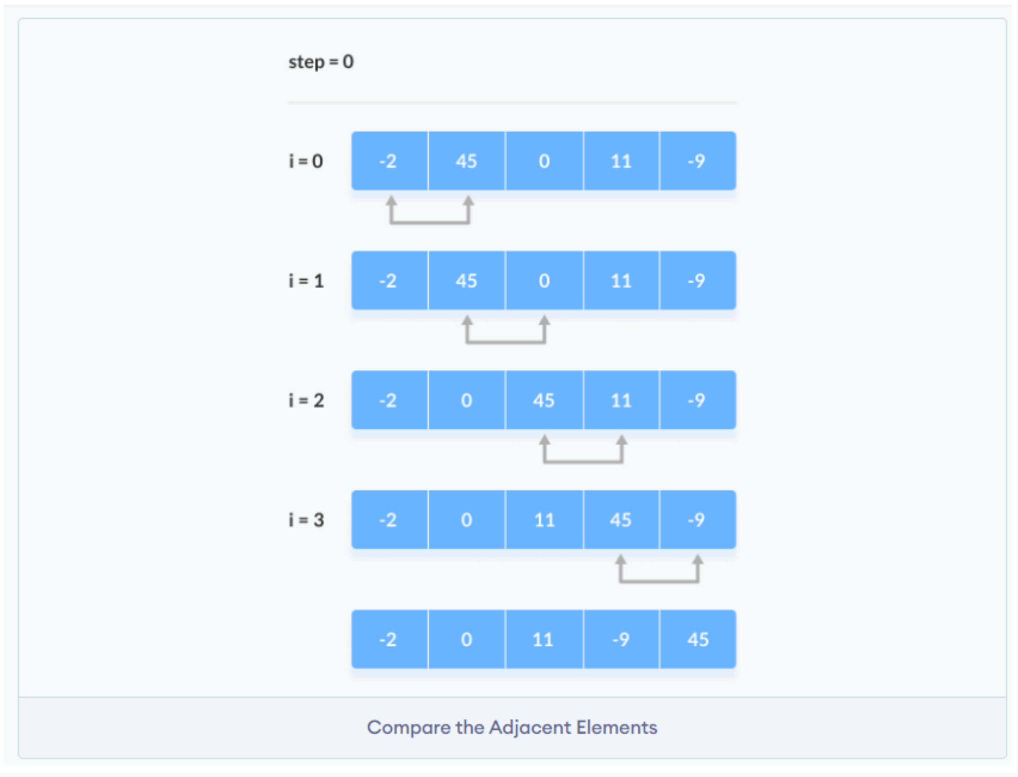
  A sorting algorithm is considered stable if the two or more items with the same value maintain the same relative positions even after sorting.

  For example, in the image below, there are two items with the same value 3. An unstable sorting algorithm allows two possibilities where the two positions of 3 may or may not be maintained.

- ***Bubble Sort:***

  Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.

  Just like the movement of air bubbles in the water that rise up to the surface, each element of the array move to the end in each iteration. Therefore, it is called a bubble sort.



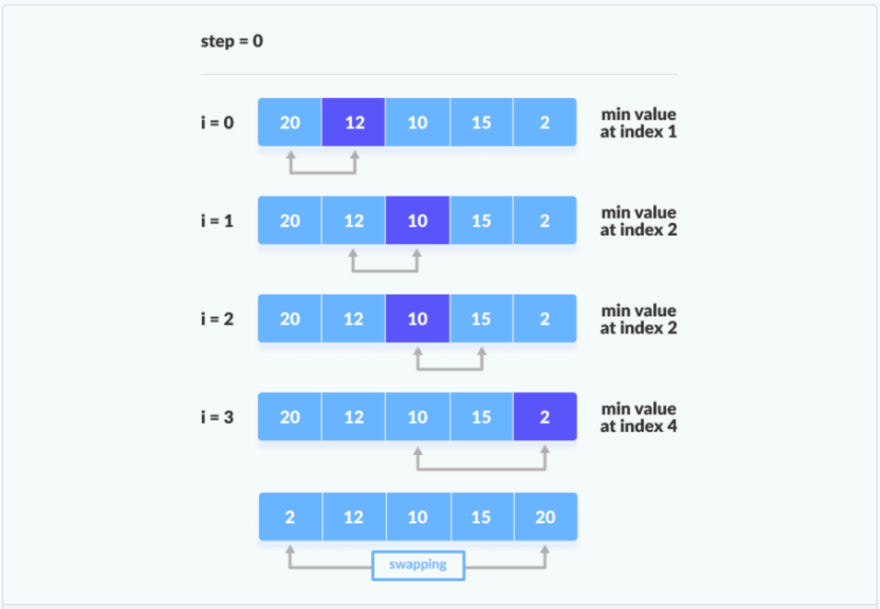
Compare the Adjacent Elements

Example Code:

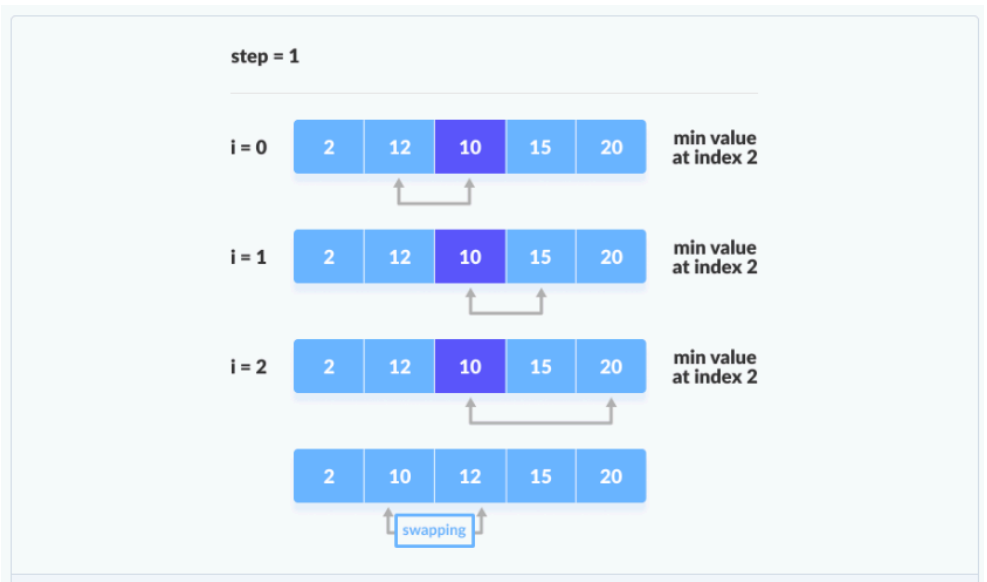
```
#include <iostream>
using namespace std;

// perform bubble sort
void bubbleSort(int array[], int size) {

  // loop to access each array element
  for (int step = 0; step < size; ++step) {

    // loop to compare array elements
    for (int i = 0; i < size - step; ++i) {

// compare two adjacent elements change > to < to sort in descending order
      if (array[i] > array[i + 1]) {

        // swapping elements if elements are not in the intended order
        int temp = array[i];
        array[i] = array[i + 1];
        array[i + 1] = temp;
      }
    }
  }
}
```

```
// print array
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    cout << "  " << array[i];
  }
  cout << "\n";
}

int main() {
  int data[] = {-2, 45, 0, 11, -9};

 // find array's length
  int size = sizeof(data) / sizeof(data[0]);

  bubbleSort(data, size);

  cout << "Sorted Array in Ascending Order:\n";
  printArray(data, size);
}
```

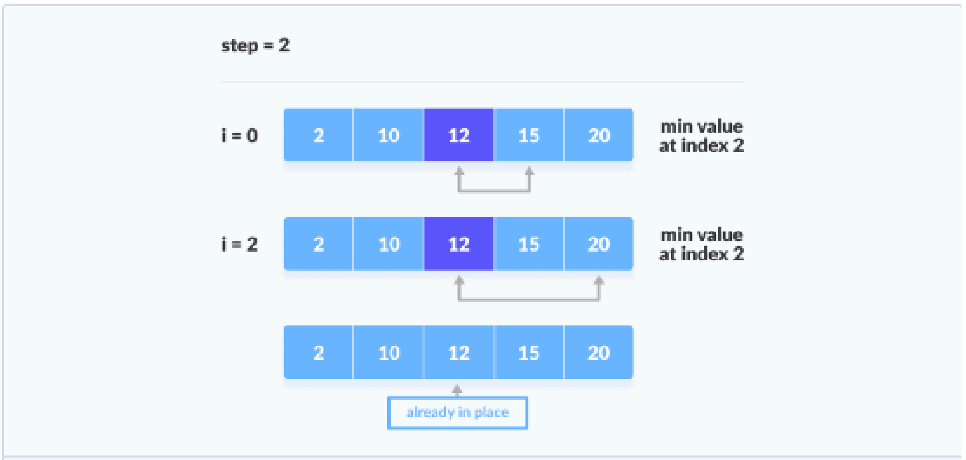
- ***Selection Sort***
  Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.



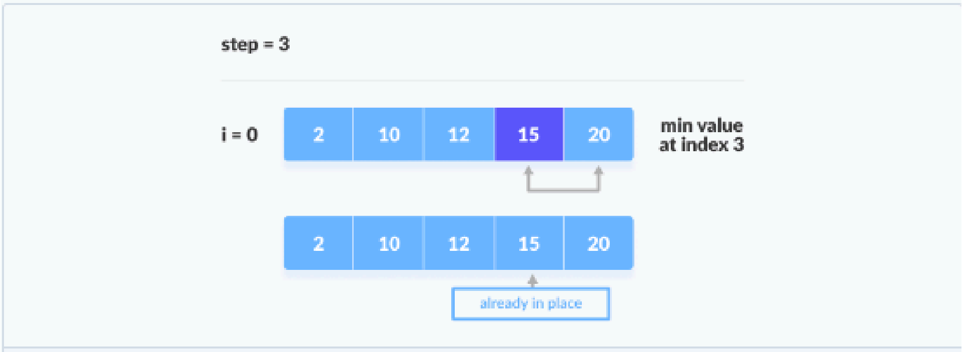
The first iteration



The second iteration



The third iteration

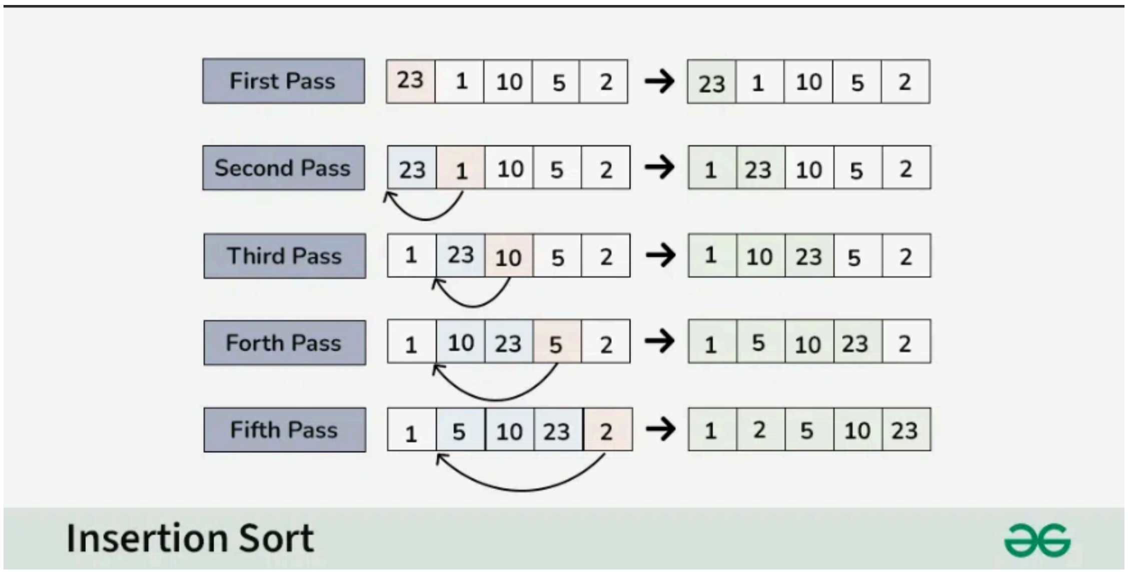


The fourth iteration

- ***Insertion Sort:***
  Insertion sort is a simple sorting algorithm that works by building a sorted array one element at a time. It is considered an "in-place" sorting algorithm, meaning it doesn't require any additional memory space beyond the original array.

To achieve insertion sort, follow these steps:

➢ We have to start with second element of the array as first element in the array is assumed to be sorted.
➢ Compare second element with the first element and check if the second element is smaller then swap them.
➢ Move to the third element and compare it with the second element, then the first element and swap as necessary to put it in the correct position among the first three elements.
➢ Continue this process, comparing each element with the ones before it and swapping as needed to place it in the correct position among the sorted elements.
➢ Repeat until the entire array is sorted.

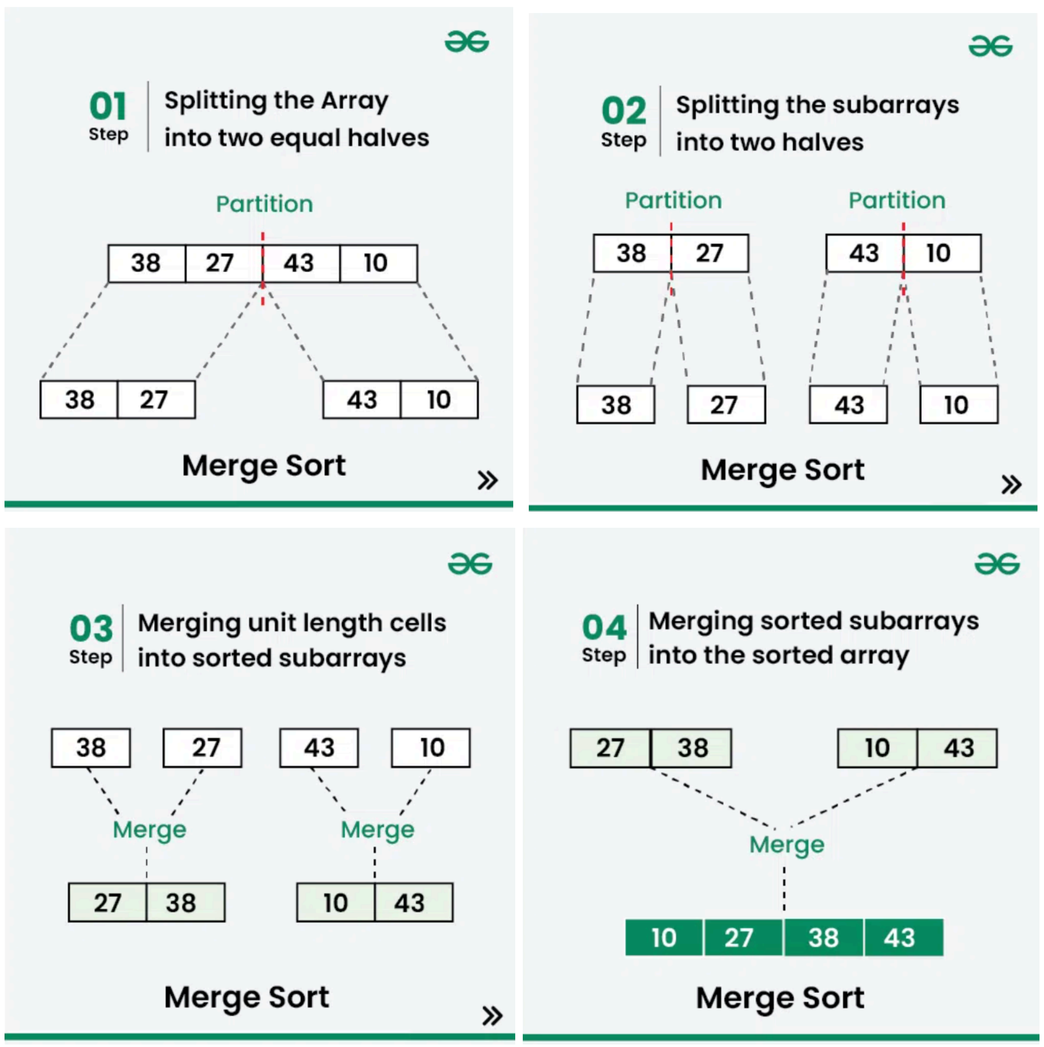


Example:

```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
// Move elements of arr[0..i-1],that are greater than key, to one position ahead of
//their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

- ***Merge Sort:***
  Merge sort is a sorting algorithm that follows the divide-and-conquer approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.

  To achieve merge sort, follow these steps:
  - ➢ *Divide*: Divide the list or array recursively into two halves until it can no more be divided.
  - ➢ *Conquer*: Each subarray is sorted individually using the merge sort algorithm.
  - ➢ *Merge*: The sorted subarrays are merged back together in sorted order. The process continues until all elements from both subarrays have been merged.


01 Step
Splitting the Array into two equal halves
Partition
38 27 43 10
38 27
43 10
Merge Sort
02 Step
Splitting the subarrays into two halves
Partition
Partition
38 27
43 10
38
27
43
10
Merge Sort
03 Step
Merging unit length cells into sorted subarrays
38
27
43
10
Merge
Merge
27 38
10 43
Merge Sort
04 Step
Merging sorted subarrays into the sorted array
27 38
10 43
Merge
10 27 38 43
Merge Sort