

Sorting Algorithms

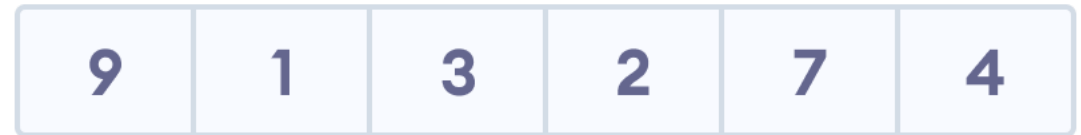
Part 01

2022/2023

Sorting Algorithm

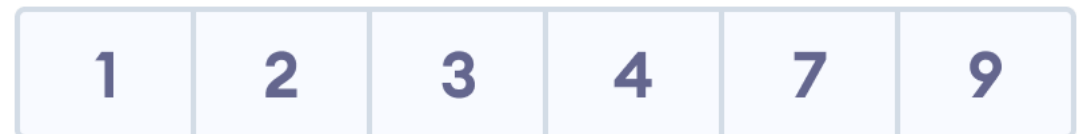
- A sorting algorithm is used to arrange elements of an array/list in a specific order. For example,
- There are various sorting algorithms that can be used to complete this operation. And, we can use any algorithm based on the requirement.

Unsorted Array



↓ sorting algorithm

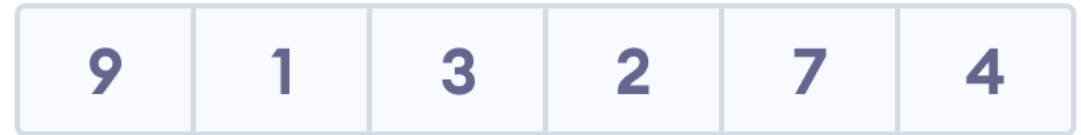
Sorted Array



Sorting Algorithm

- Different Sorting Algorithms.
 - Bubble Sort
 - Selection Sort
 - Insertion Sort
 - Merge Sort
 - Quicksort
 - Counting Sort
 - Radix Sort
 - Bucket Sort
 - Heap Sort
 - Shell Sort

Unsorted Array



↓ sorting algorithm

Sorted Array



Bubble Sort

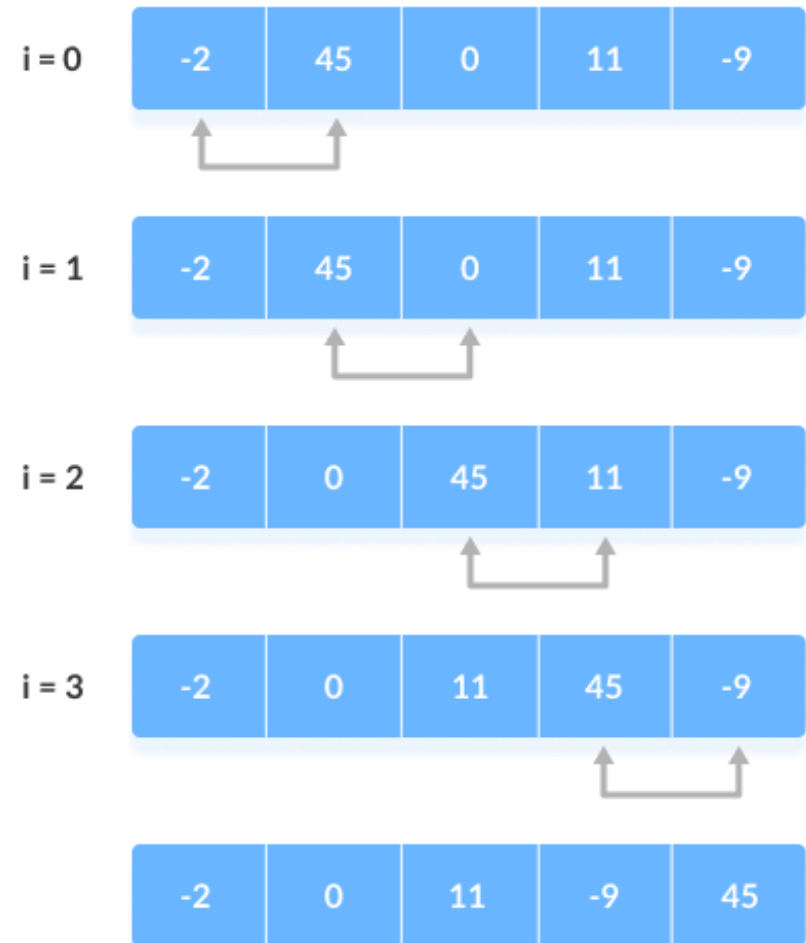
- Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.
- Just like the movement of air bubbles in the water that rise up to the surface, each element of the array move to the end in each iteration. Therefore, it is called a bubble sort.



Working of Bubble Sort

- Suppose we are trying to sort the elements in ascending order.
- **I. First Iteration (Compare and Swap)**
 1. Starting from the first index, compare the first and the second elements.
 2. If the first element is greater than the second element, they are swapped.
 3. Now, compare the second and the third elements. Swap them if they are not in order.
 4. The above process goes on until the last element.

step = 0



Working of Bubble Sort

- Suppose we are trying to sort the elements in ascending order.
- **2. Remaining Iteration**
 - The same process goes on for the remaining iterations.
 - After each iteration, the largest element among the unsorted elements is placed at the end.

step = 1

i = 0



i = 1



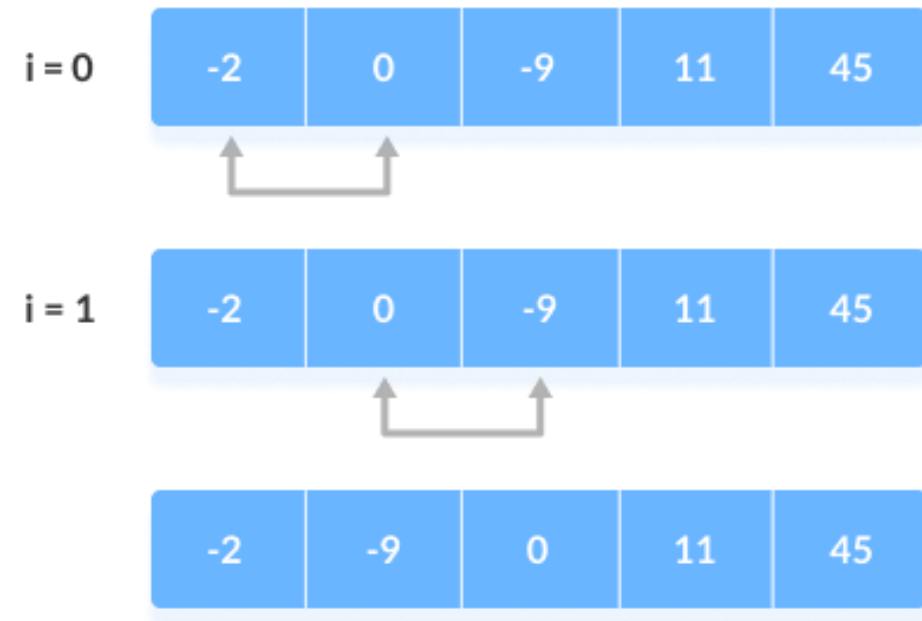
i = 2



Working of Bubble Sort

- In each iteration, the comparison takes place up to the last unsorted element.

step = 2

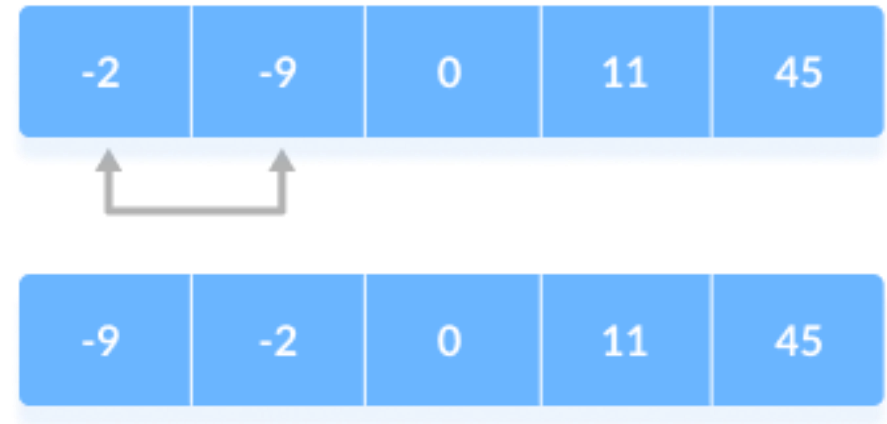


Working of Bubble Sort

- The array is sorted when all the unsorted elements are placed at their correct positions.

step = 3

i = 0



Working of Bubble Sort

```
// Bubble sort in C++

#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;
            }
        }
    }
}
```

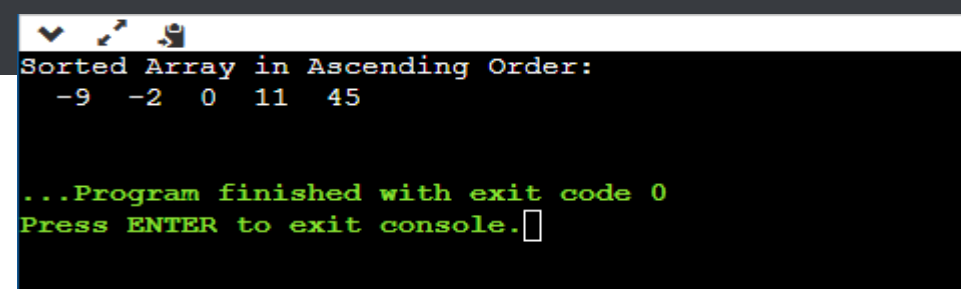
```
// print array
void printArray(int array[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << " " << array[i];
    }
    cout << "\n";
}

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);
}
```



```
Sorted Array in Ascending Order:
-9 -2 0 11 45

...Program finished with exit code 0
Press ENTER to exit console. □
```

Optimized Bubble Sort

```
// Optimized bubble sort in C++

#include
using namespace std;

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

    // loop to access each array element
    for (int step = 0; step < (size-1); ++step) {

        // check if swapping occurs
        int swapped = 0;

        // loop to compare two elements
        for (int i = 0; i < (size-step-1); ++i) {

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

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

                swapped = 1;
            }
        }
    }
}
```

```
        // no swapping means the array is already sorted
        // so no need of further comparison
        if (swapped == 0)
            break;
    }
}

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

    bubbleSort(data, size);

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

Bubble Sort Complexity

- nearly equals to n^2
- Hence, Complexity: $O(n^2)$

Cycle	Number of Comparisons
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
.....
last	1

Selection Sort Algorithm

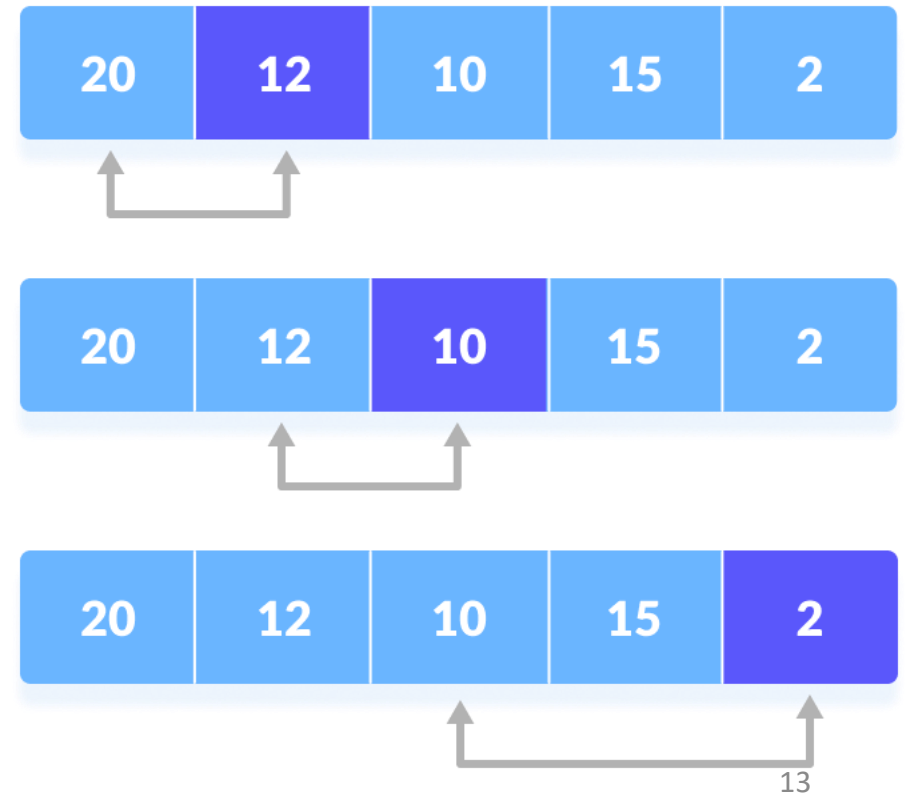
- Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.
- Working of Selection Sort
 1. Set the first element as **minimum**.
 2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.



Selection Sort Algorithm

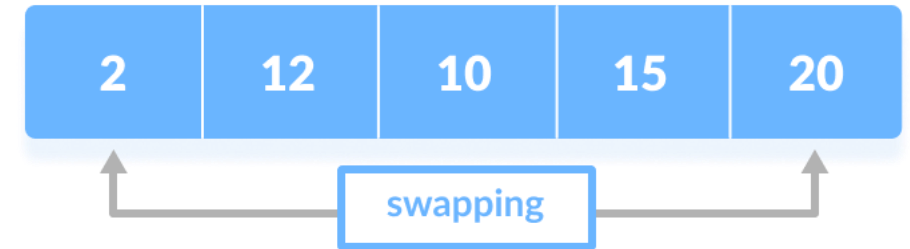
- Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.
- Working of Selection Sort
 1. Set the first element as **minimum**.
 2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.

Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.



Selection Sort Algorithm

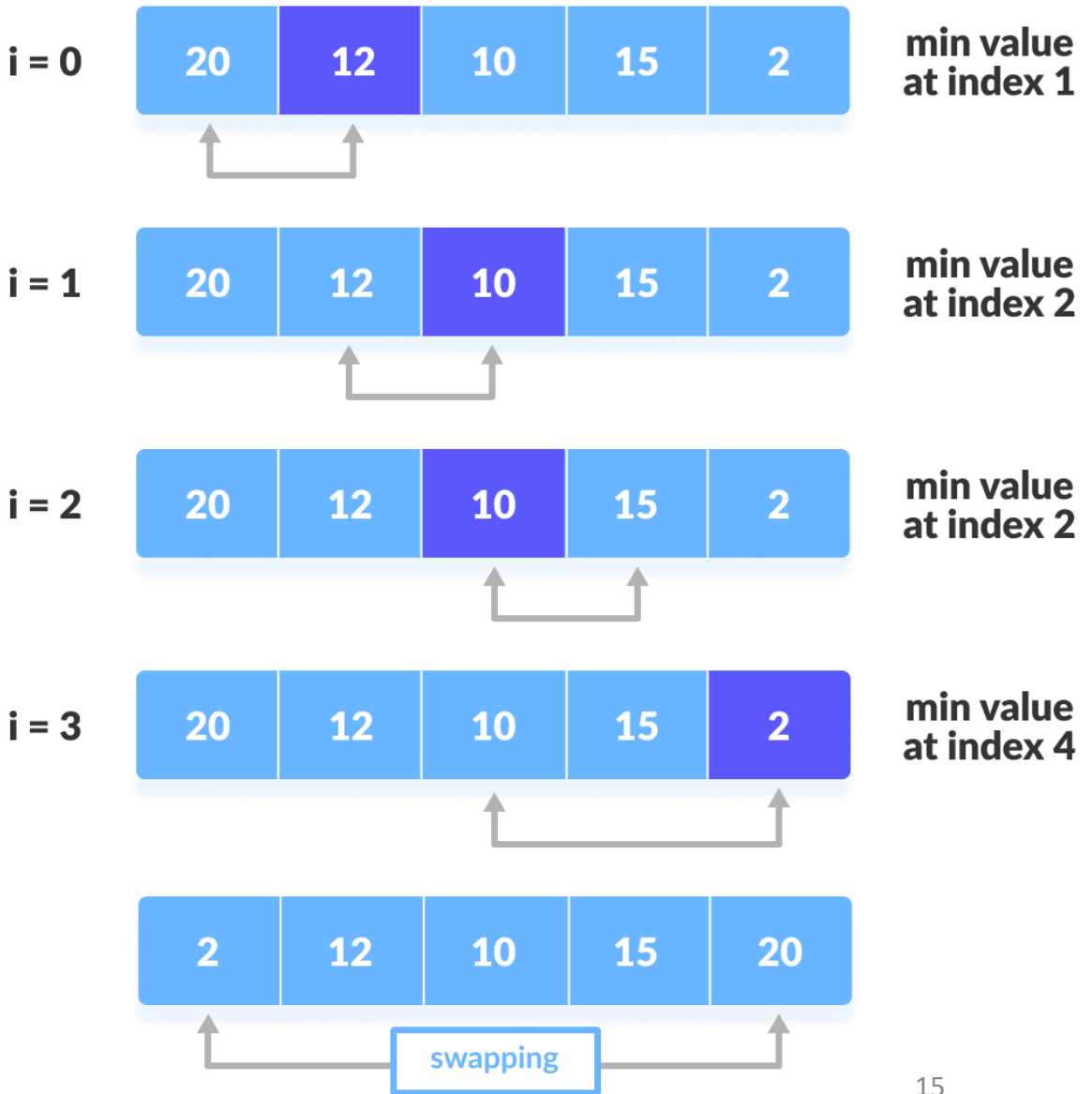
- Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.
- Working of Selection Sort
 3. After each iteration, minimum is placed in the front of the unsorted list.
 4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.



Selection Sort Algorithm

- Working of Selection Sort
 - For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

step = 0

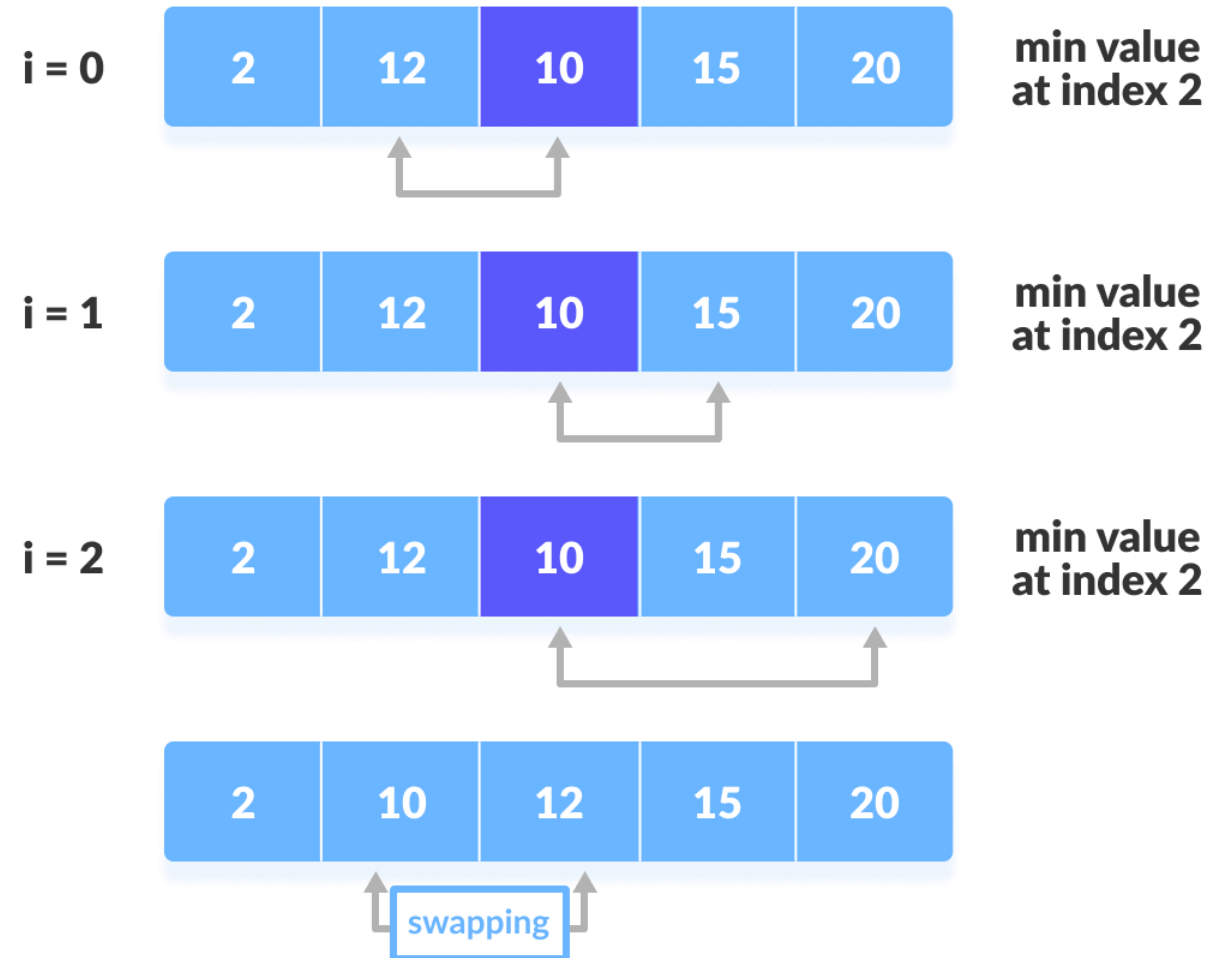


Selection Sort Algorithm

- Working of Selection Sort

4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

step = 1

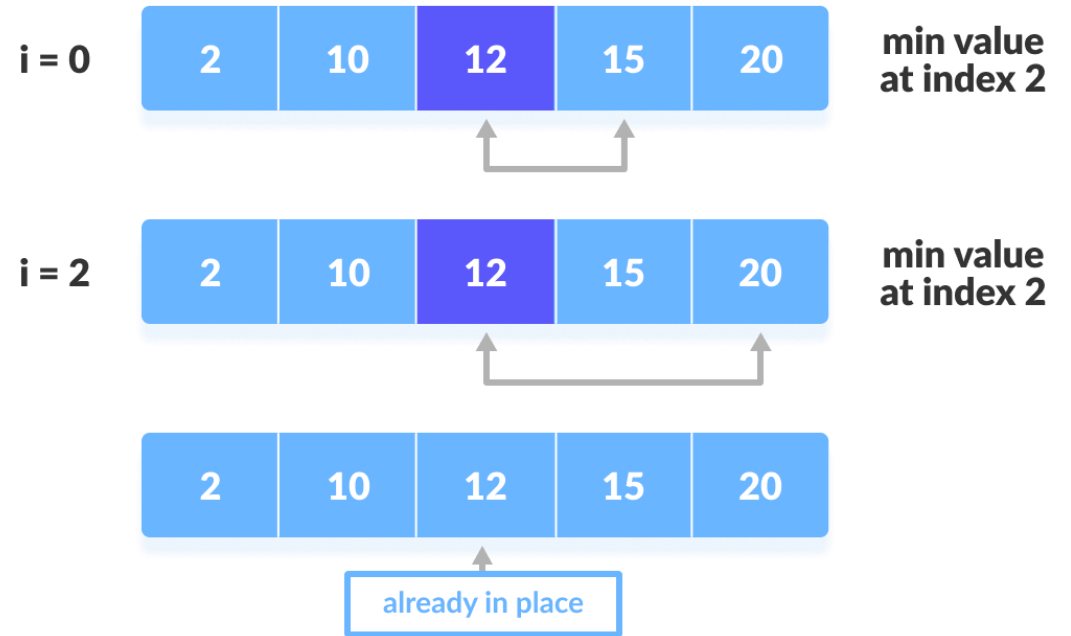


Selection Sort Algorithm

- Working of Selection Sort

4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

step = 2

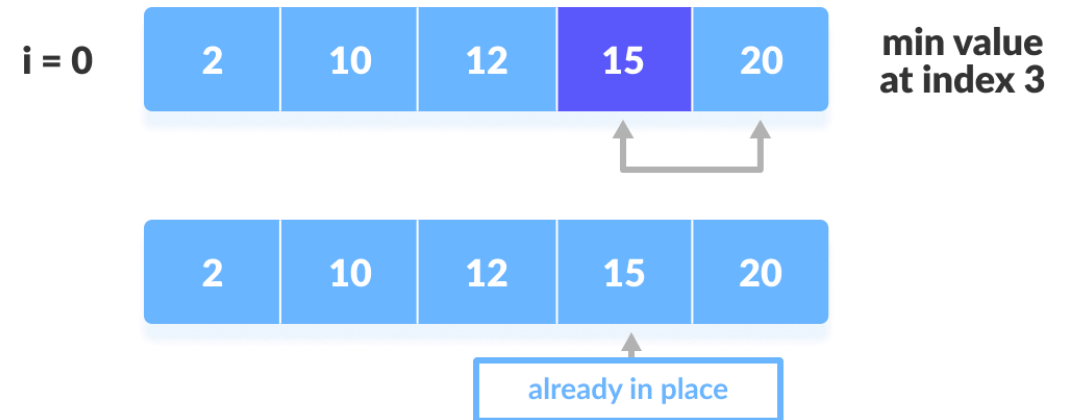


Selection Sort Algorithm

- Working of Selection Sort
 - For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

$$\text{Complexity} = O(n^2)$$

step = 3

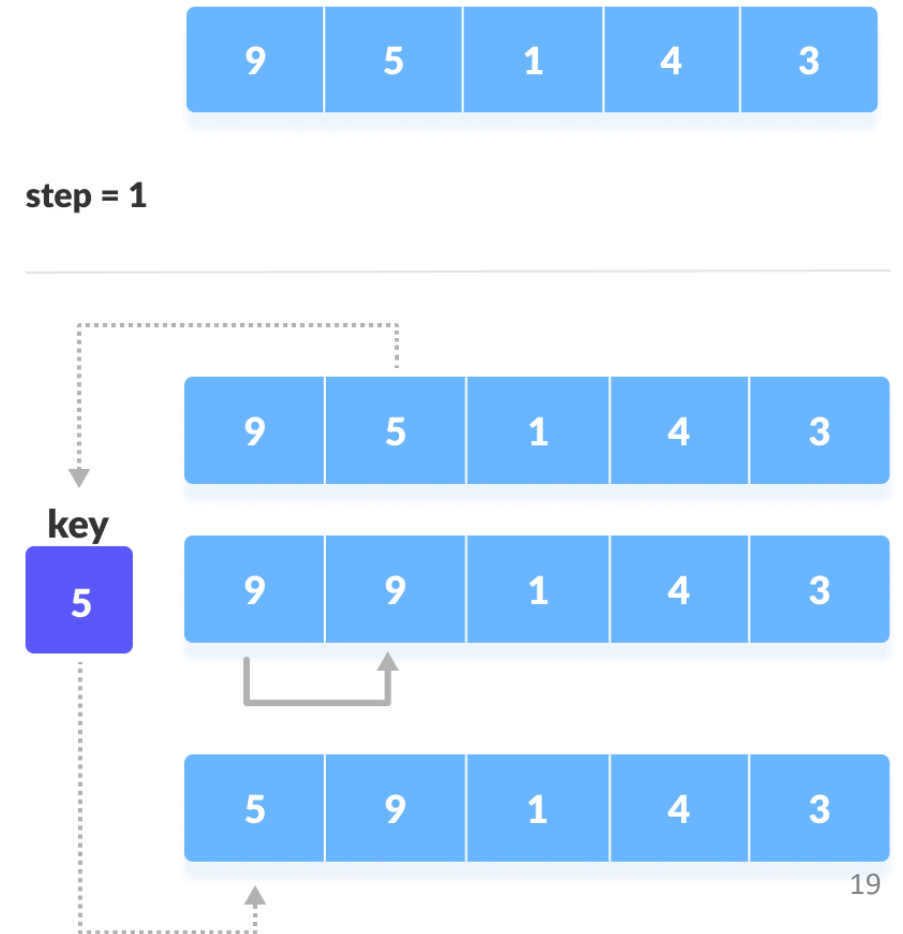


Cycle	Number of Comparison
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
...	...
last	1

Insertion Sort Algorithm

- Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.
- Working of Insertion Sort
- Suppose we need to sort the following array.

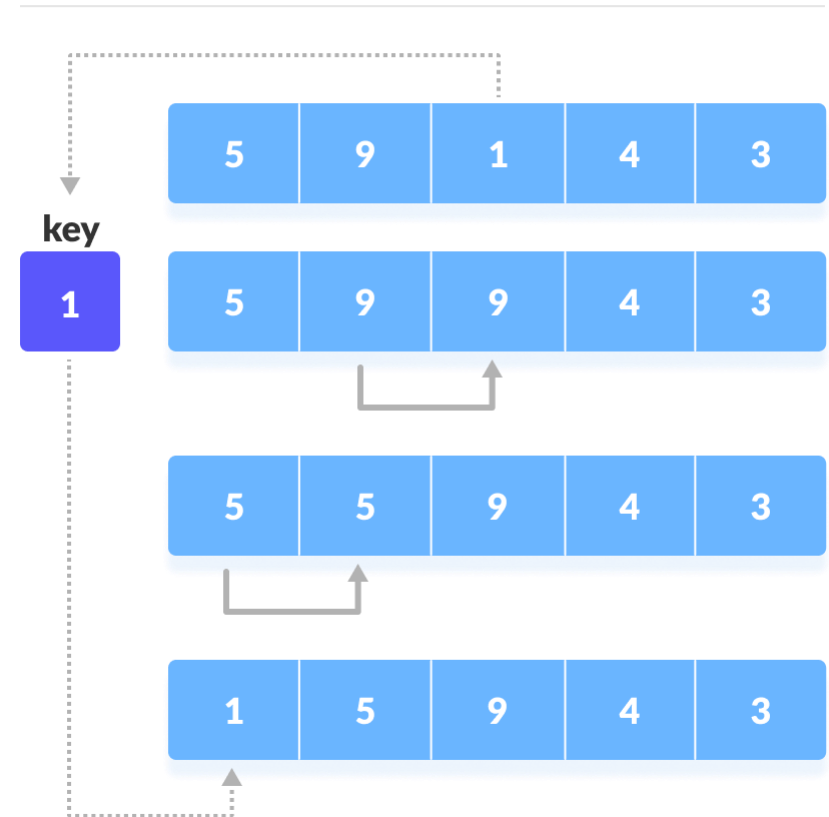
1. The first element in the array is assumed to be sorted. Take the second element and store it separately in **key**.
2. Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.



Insertion Sort Algorithm

- Working of Insertion Sort
 2. Now, the first two elements are sorted.

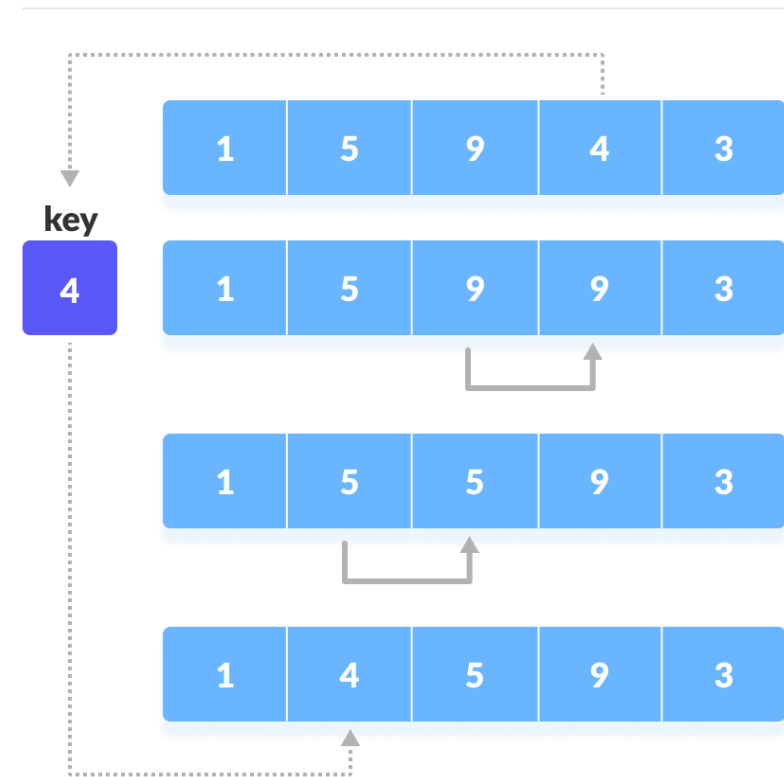
step = 2



Insertion Sort Algorithm

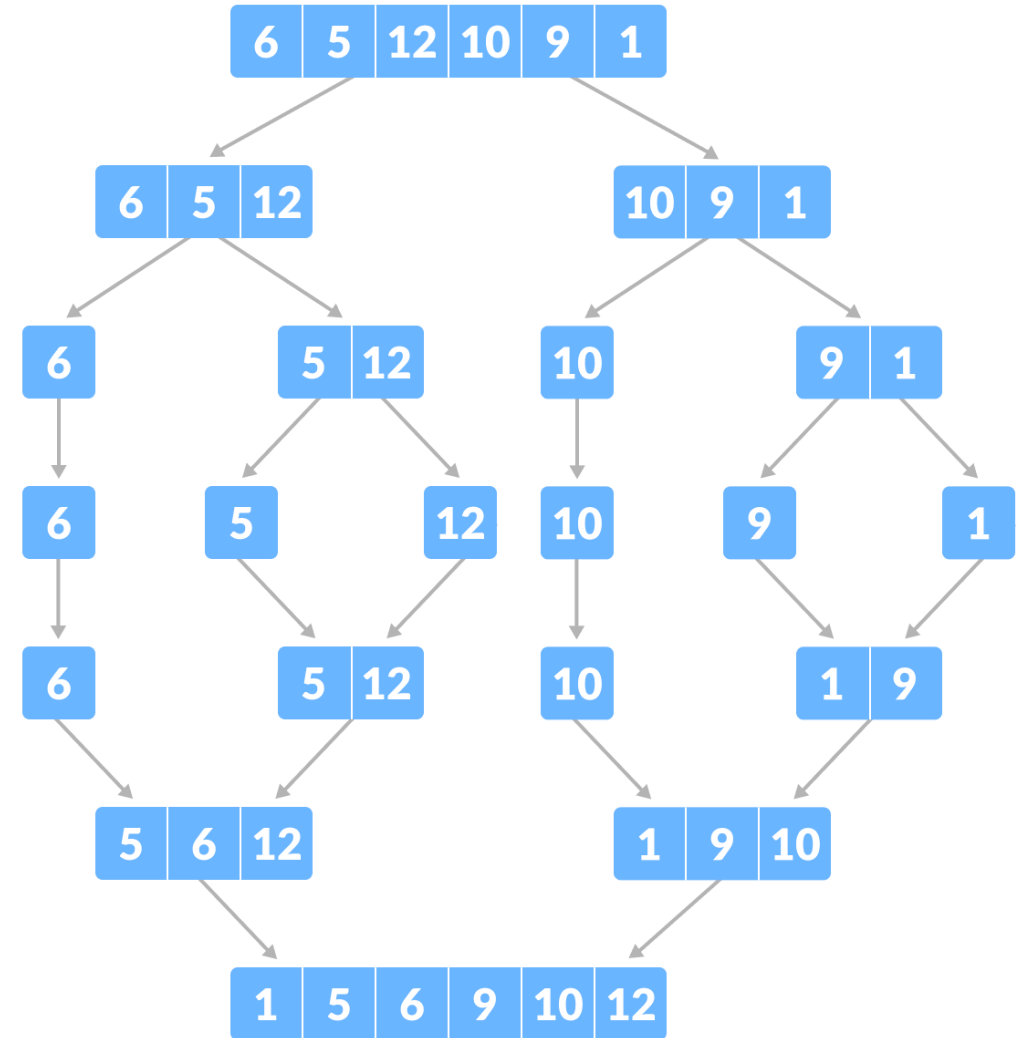
- Working of Insertion Sort
 2. Similarly, place every unsorted element at its correct position.

step = 3



Merge Sort Algorithm

- Merge Sort is one of the most popular sorting algorithms that is based on the principle of **Divide and Conquer Algorithm**.
- Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually. Finally, sub-problems are combined to form the final solution.



References:

- <https://www.programiz.com/dsa/bubble-sort>
- <https://www.programiz.com/dsa/selection-sort>
- <https://www.programiz.com/dsa/insertion-sort>