DATA STRUCTURES AND ALGORITHMS

DR SAMABIA TEHSIN

BS (AI)

Sorting

Sorting refers to ordering data in an increasing or decreasing manner according to some linear relationship among the data items.

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order.

Sorting

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Following are some of the examples of sorting in real-life scenarios –

- **Telephone Directory** – The telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily.
- **Dictionary** – The dictionary stores words in an alphabetical order so that searching of any word becomes easy.

Sorting Techniques

Bubble Sort

Insertion sort

Selection sort

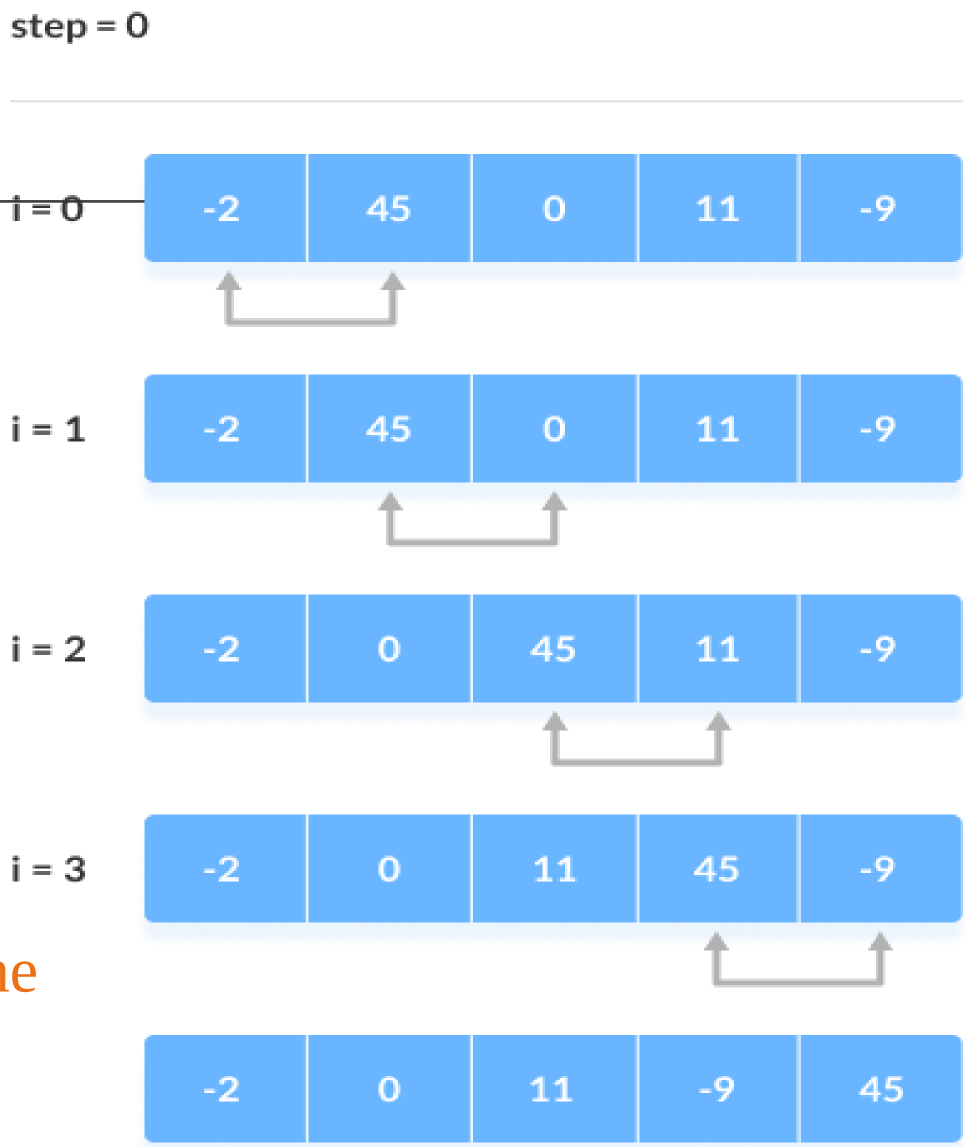
Merge sort

Heap sort

And many more.....

Bubble Sort

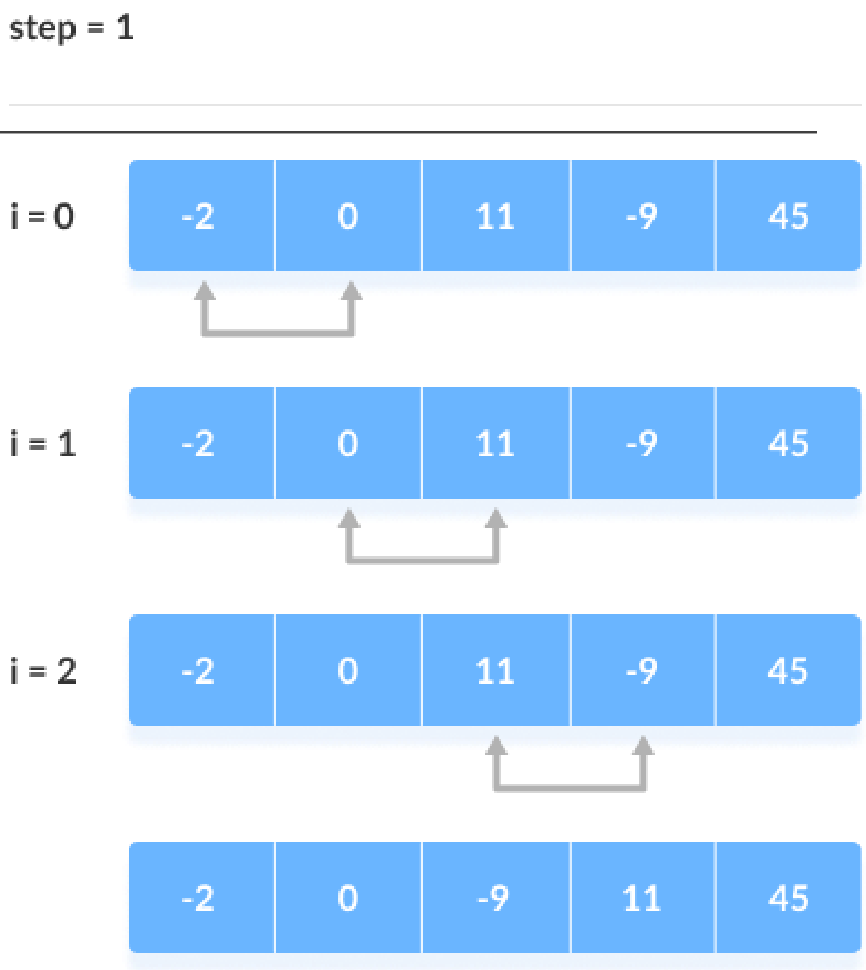
Starting from the first index, compare the first and the second elements.
If the first element is greater than the second element, they are swapped.
Now, compare the second and the third elements. Swap them if they are not in order.
The above process goes on until the last element.



Bubble Sort

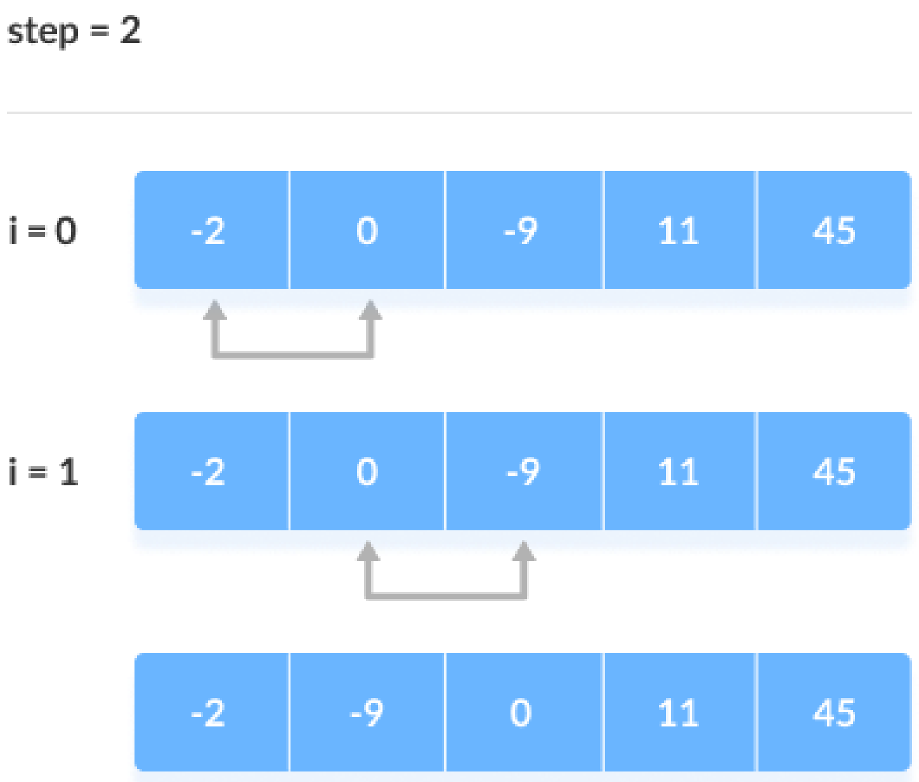
The same process goes on for the remaining iterations.

After each iteration, the largest element among the unsorted elements is placed at the end.



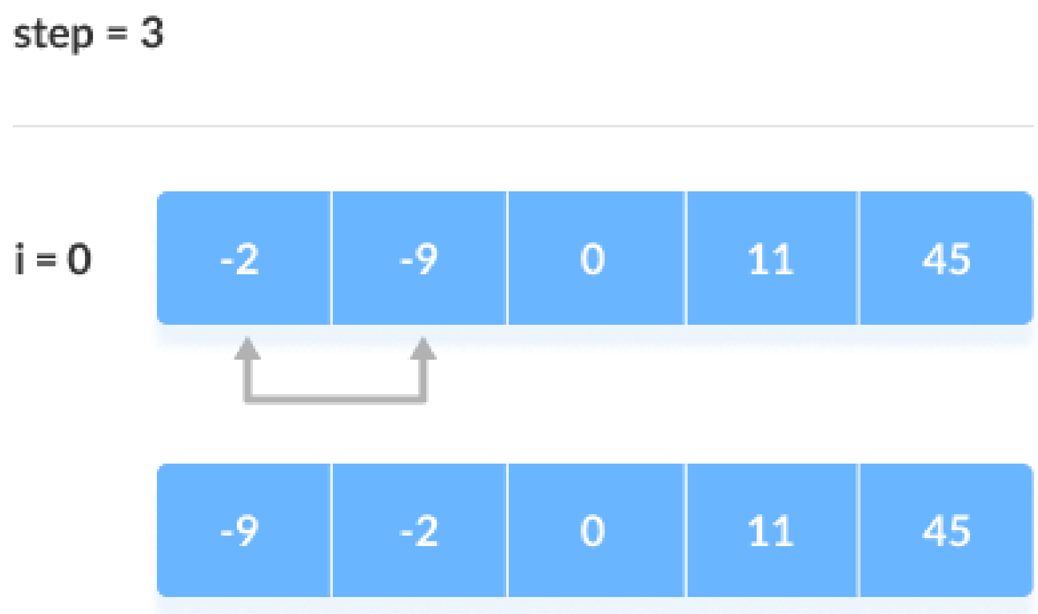
Bubble Sort

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



Bubble Sort

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



Bubble Sort

```
// 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;
      }
    }
  }
}
```

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.

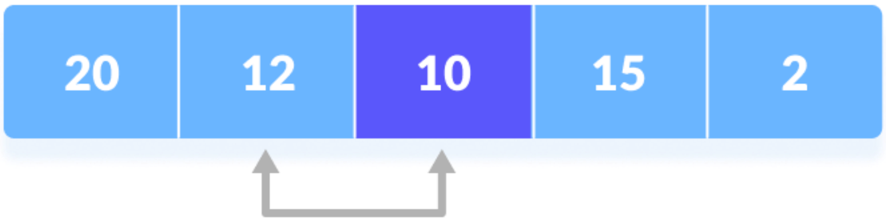
Selection Sort

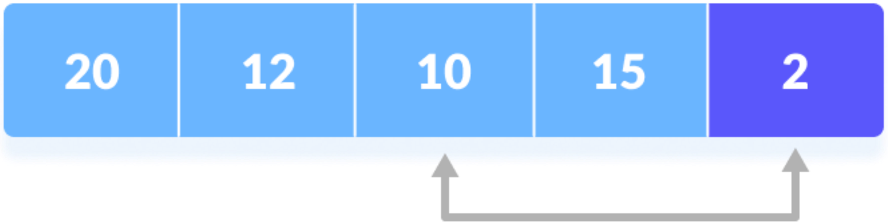
Set the first element as minimum.

Selection Sort

Compare `minimum` with the second element. If the second element is smaller than `minimum`, assign the second element as `minimum`.

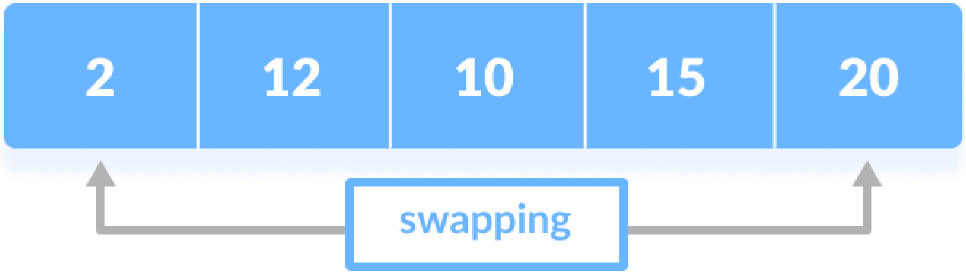
Compare `minimum` with the third element. Again, if the third element is smaller, then assign `minimum` to the third element otherwise do nothing. The process goes on until the last element.





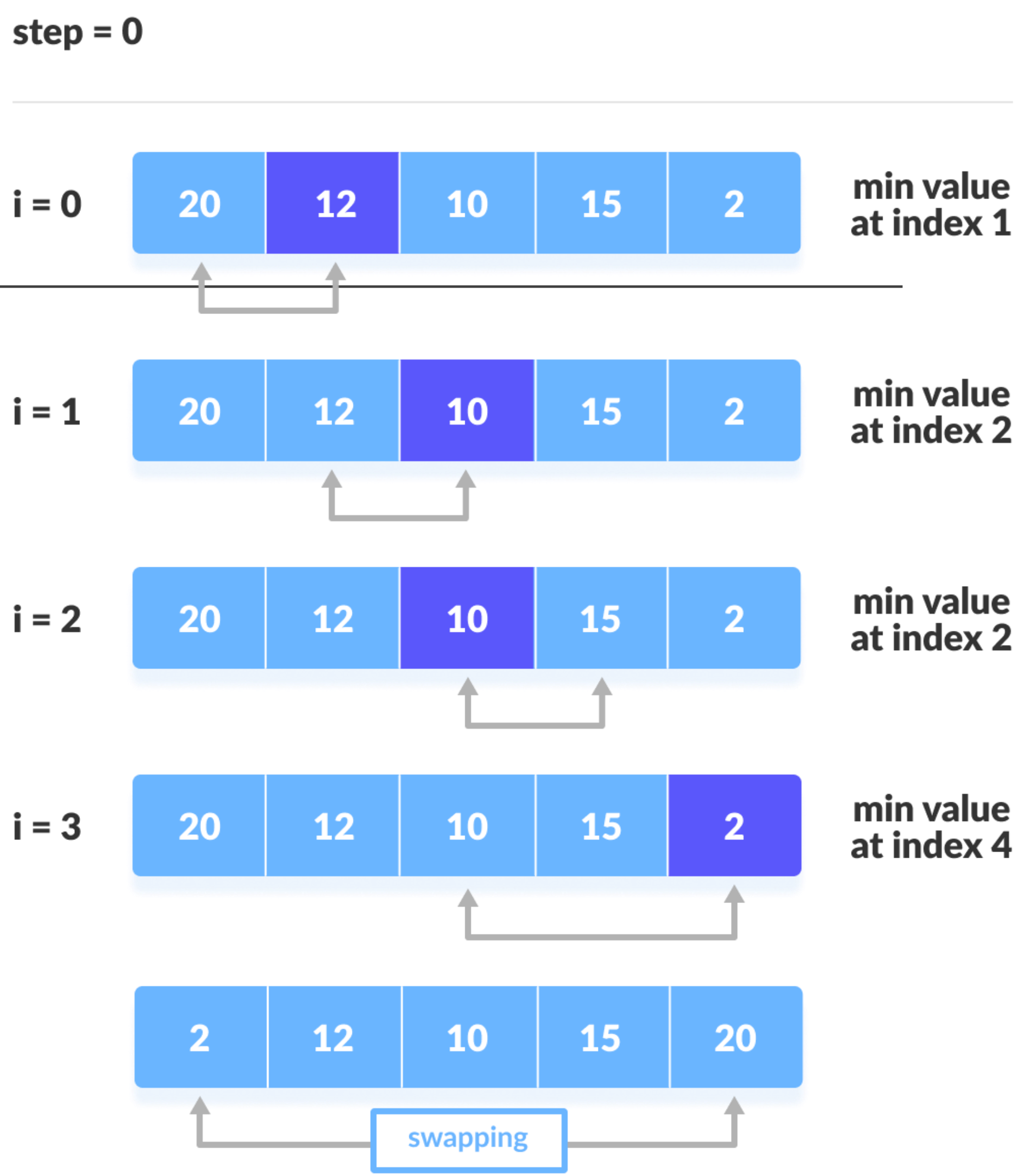
Selection Sort

After each iteration, minimum is placed in the front of the unsorted list.



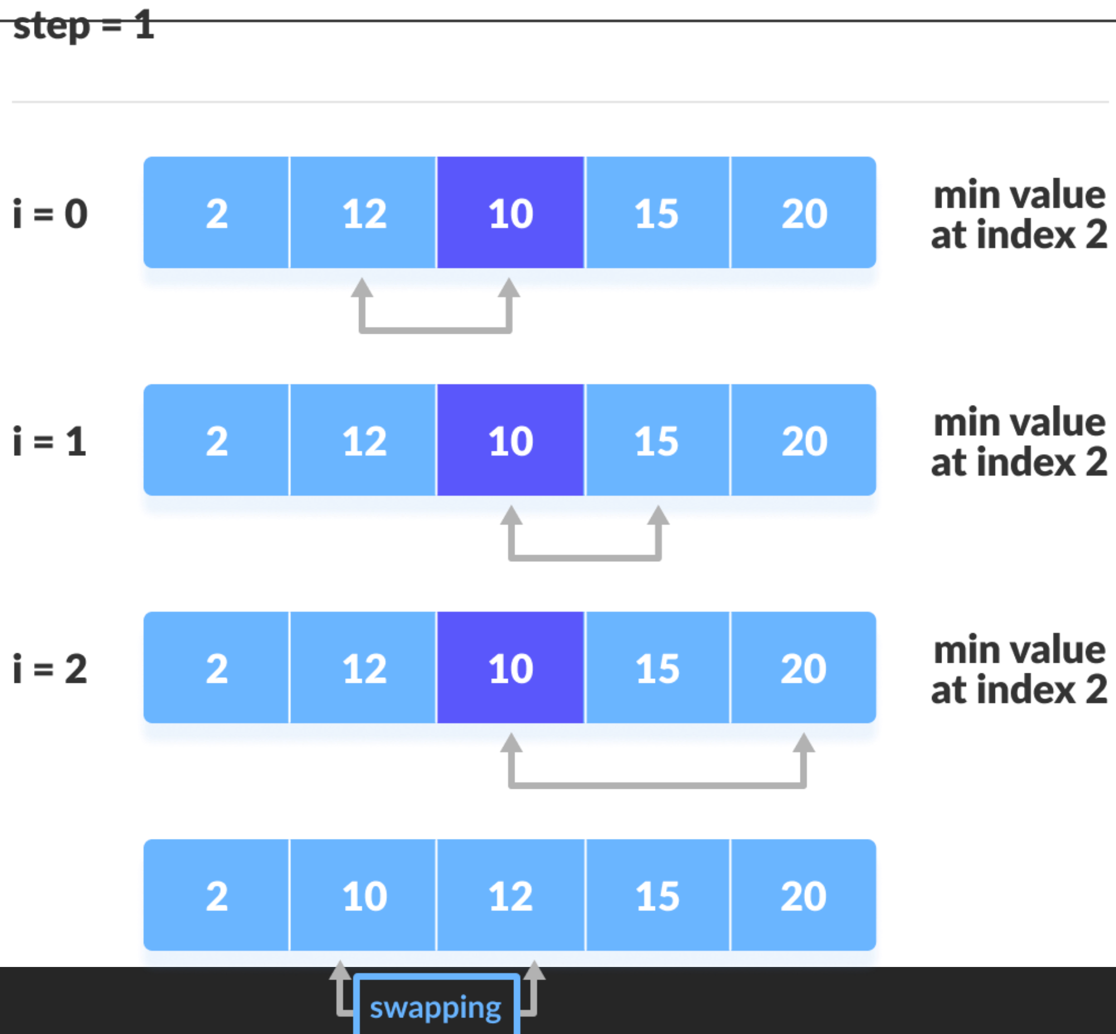
Selection Sort

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.



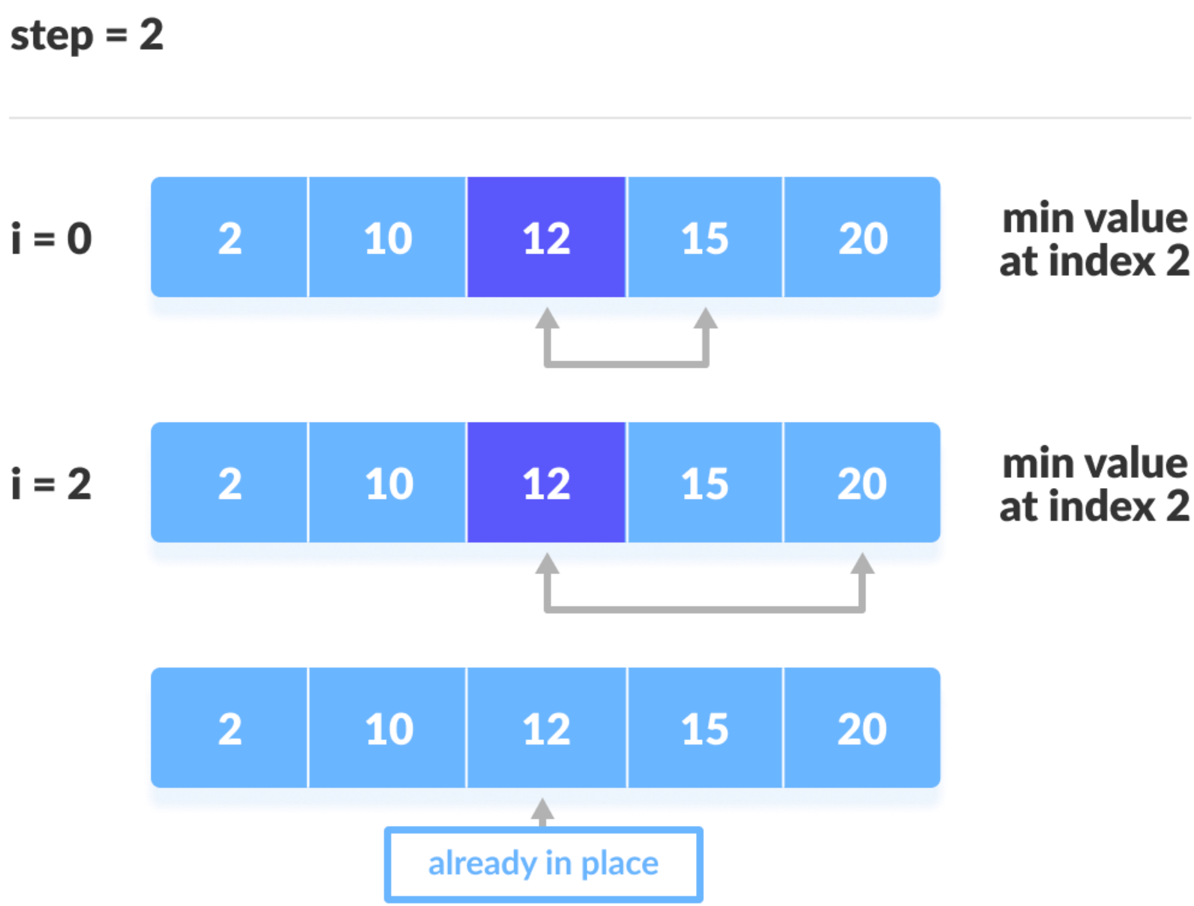
Selection Sort

The second iteration



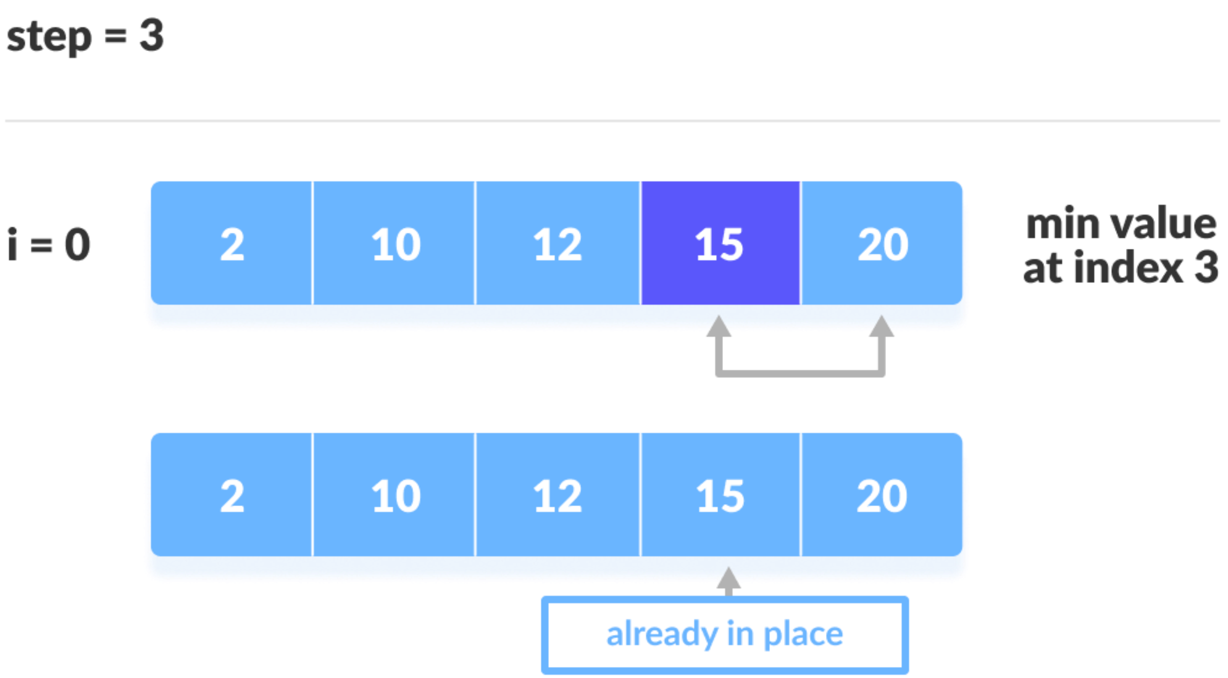
Selection Sort

The third iteration



Selection Sort

The fourth iteration



Selection Sort

Algorithm

```
selectionSort(array, size)
repeat (size - 1) times
        set the first unsorted element as the minimum
        for each of the unsorted elements
                if element < currentMinimum
                        set element as new minimum
        swap minimum with first unsorted position
end selectionSort
```

Selection Sort

```
// Selection sort in C++
// function to swap the position
//of two elements
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}
```

```
void selectionSort(int array[], int size) {
  for (int step = 0; step < size - 1; step++) {
    int min_idx = step;
    for (int i = step + 1; i < size; i++) {

// To sort in descending order, change > to < in
//this line.
      // Select the minimum element in each loop.
      if (array[i] < array[min_idx])
        min_idx = i;
    }

    // put min at the correct position
    swap(&array[min_idx], &array[step]);
  }
}
```

Insertion Sort

Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.

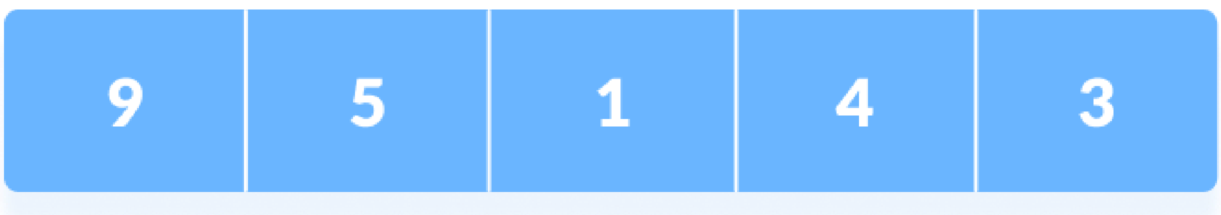
Insertion sort works similarly as we sort cards in our hand in a card game.

We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put in their right place.

A similar approach is used by insertion sort.

Insertion sort

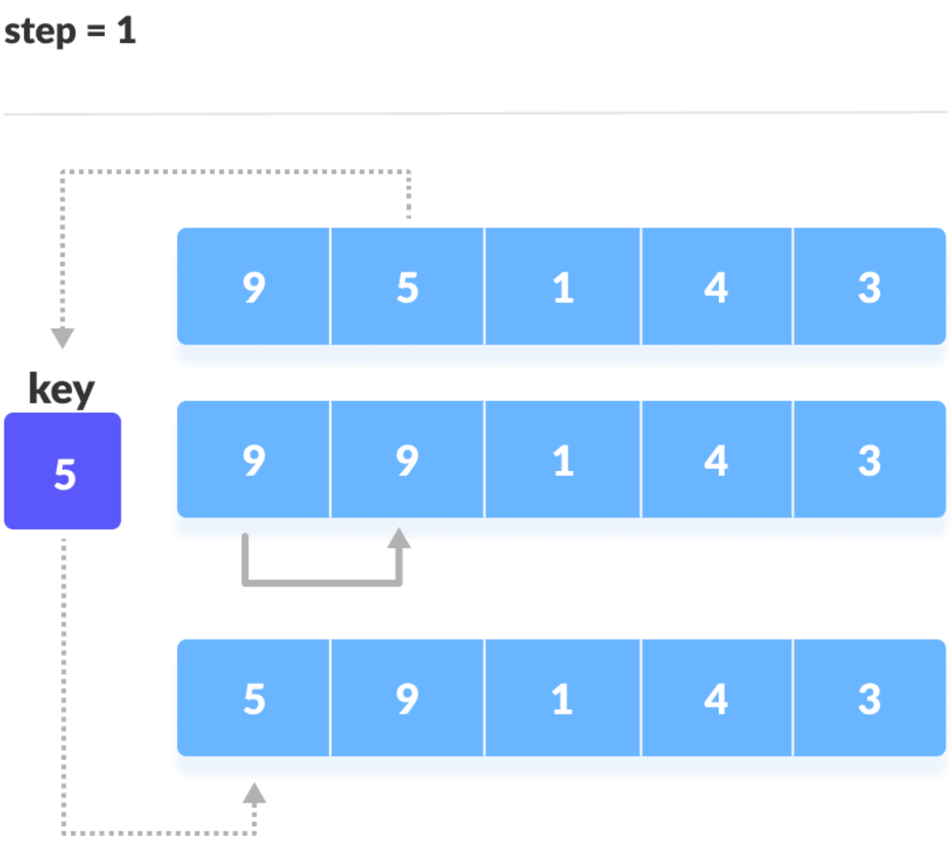
Initial array



Insertion sort

The first element in the array is assumed to be sorted. Take the second element and store it separately in key.

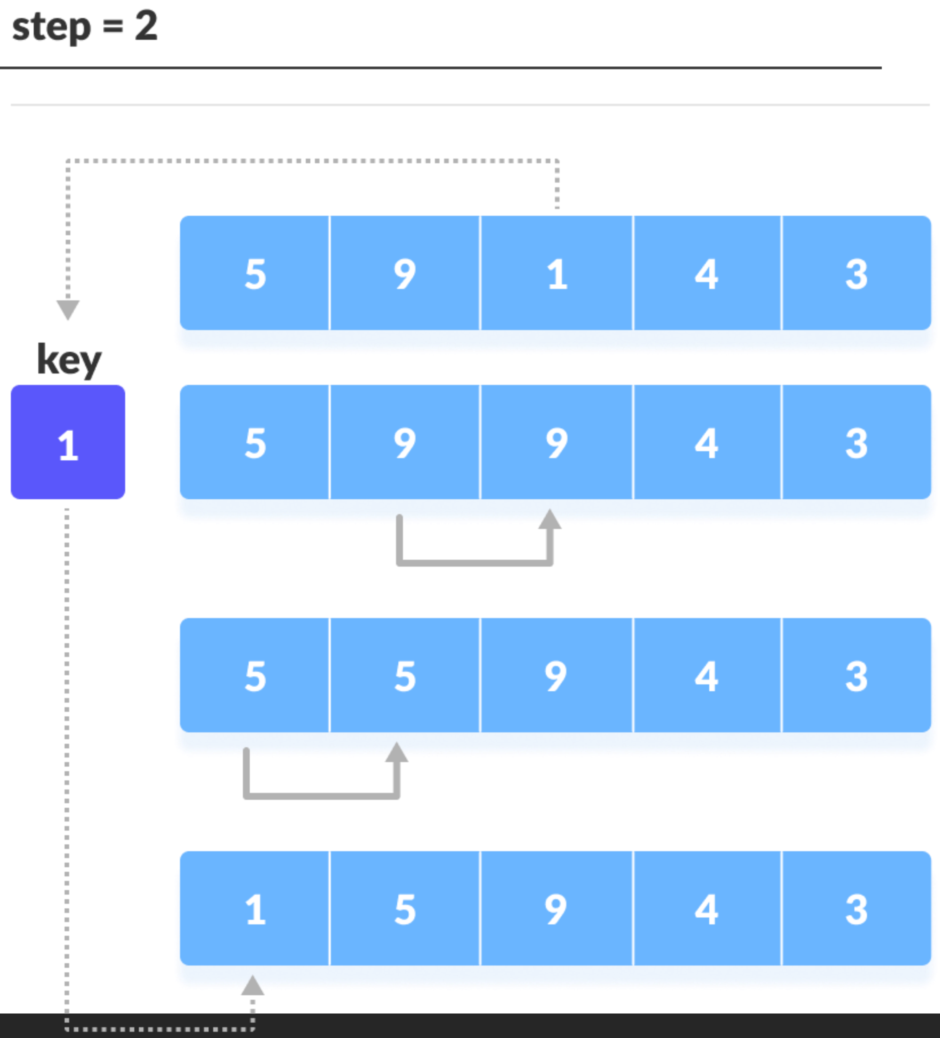
Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.



Insertion sort

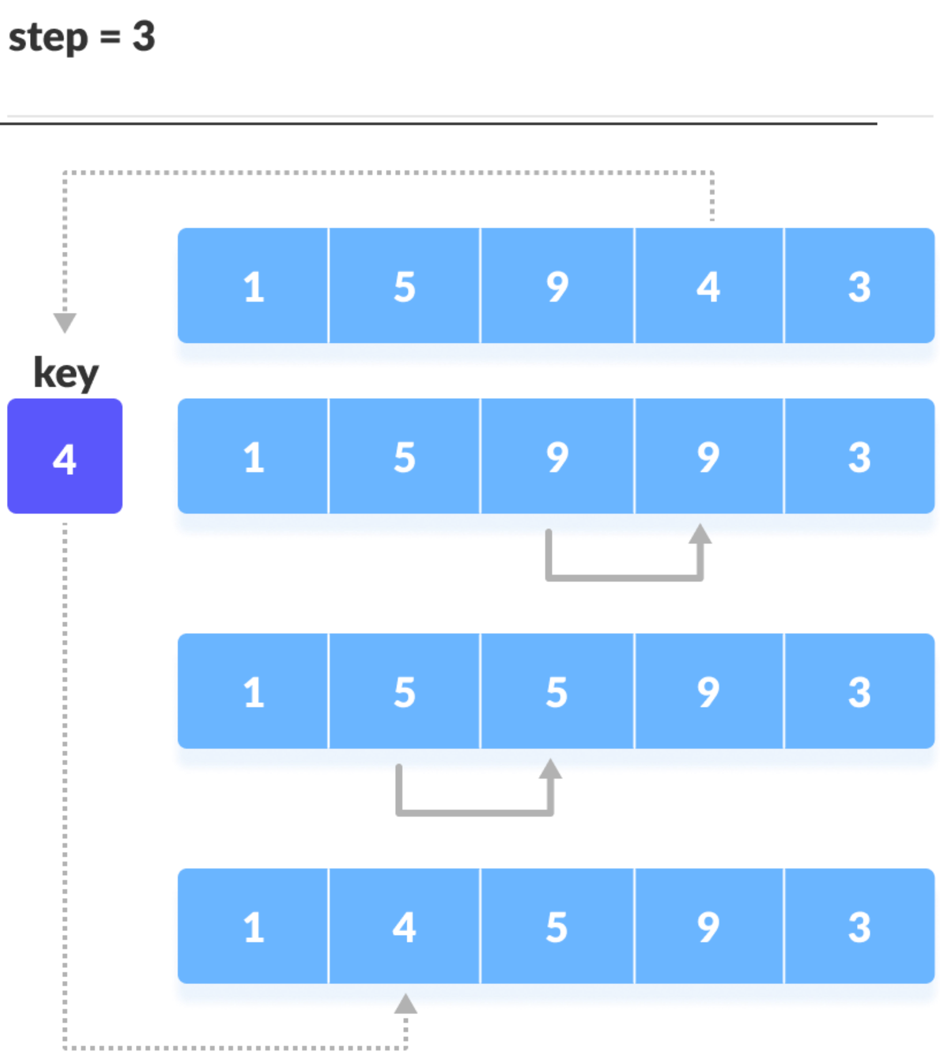
Now, the first two elements are sorted.

Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.

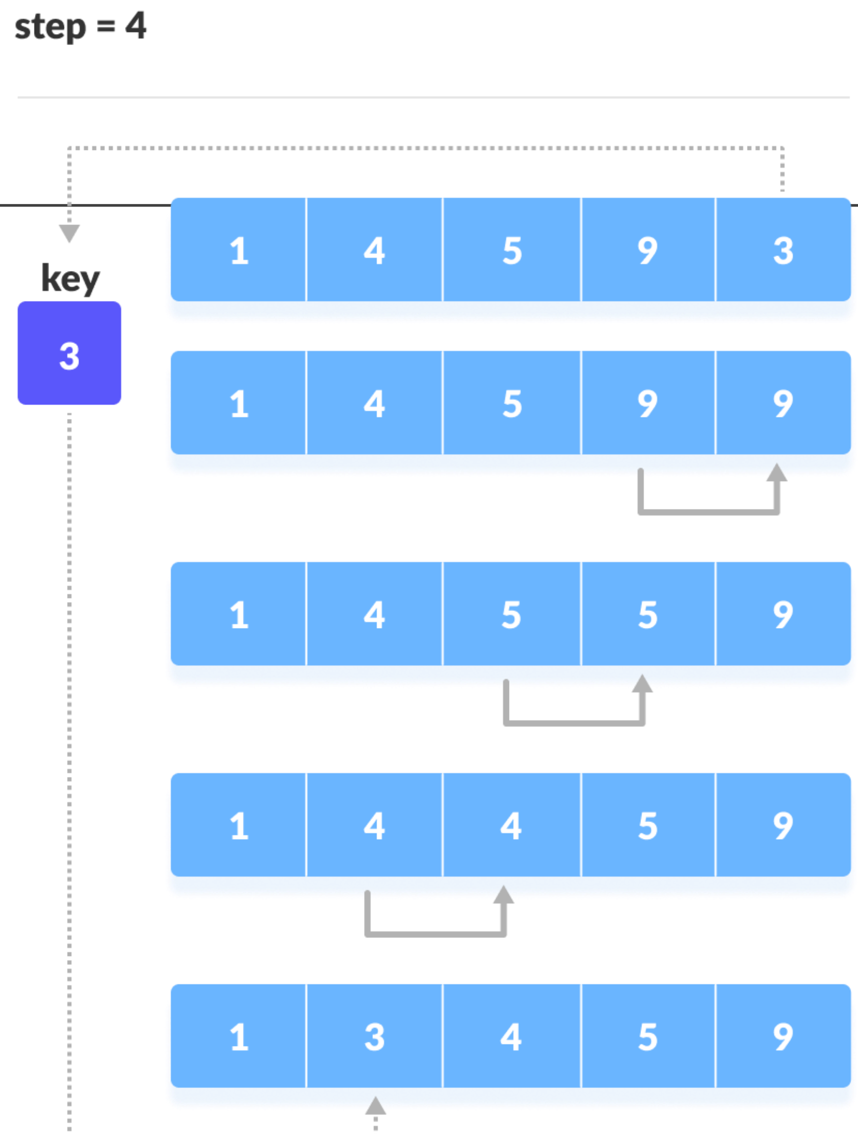


Insertion sort

Similarly, place every unsorted element at its correct position.



Insertion sort



Insertion sort

Insertion Sort Algorithm

```
insertionSort(array)
  mark first element as sorted
  for each unsorted element X
      'extract' the element X
      for j <- lastSortedIndex down to 0
             if current element j > X
                      move sorted element to the right by 1
      break loop and insert X here
end insertionSort
```

TIMING AND OTHER ISSUES

- Bubble, Selectionsort and Insertionsort have a worst-case time of $O(n^2)$, making them impractical for large arrays.
- But they are easy to program, easy to debug.
- Insertionsort also has good performance when the array is nearly sorted to begin with.
- But more sophisticated sorting algorithms are needed when good performance is needed in all cases for large arrays.