Searching and Sorting

Searching

Searching means to find out whether a particular element is present in the given array/list. For instance, when you visit a simple google page and type anything that you want to know/ask about, basically you are searching that topic in the google's vast database for which google is using some technique in order to provide the desired result to you.

There are basically three types of searching techniques:

- Linear search
- Binary search
- Ternary search

We have already discussed the linear searching technique. In this module, we will be discussing binary search. We will not be discussing ternary search over here, for that You may explore the link mentioned below to know more about it.

Link for ternary search: (It is preferred to first-of-all clearly understand binary search and then move onto the ternary search)

https://cp-algorithms.com/num_methods/ternary_search.html

Linear Search

In this, we have to find a particular element in the given array and return the index of the same, in case it is not present the convention is to return -1. This can be simply done by traversing each element index one-by-one and then comparing the value at that index with the desired value.

Suppose you want to find a particular element in the sorted array, then following this technique, you have to traverse all the array elements for searching one element but guess if you only have to search at most half of the array indices for performing the same operation. This can be achieved through binary search.

Advantages of Binary search:

- This searching technique is fast and easier to implement.
- Requires no extra space.
- Reduces time complexity of the program to a greater extent. (The term **time complexity** might be new to you, you will get to understand this when you will be studying algorithmic analysis. For now, just consider it as the time taken by a particular algorithm in its execution and time complexity is determined by the number of operations that are performed by that algorithm i.e., time complexity is directly proportional to the number of operations in the program).

Binary Search

Now, let's look at what binary searching is.

Prerequisite: Binary search has one pre-requisite, the array must be sorted unlike the linear search where elements could be any order.

Let us consider the array:

0	1	2	3	4
1	**2**	**3**	**4**	**5**

You can see it is an array of size 5 with elements inside the boxes and indices above them. Our target element is 2.

Steps:

1. Find the middle index of the array. In case of even length of the array consider the index that appears first out of two middle ones.
2. Now, we will compare the middle element with the target element. In case they are equal then we will simply return the middle index.
3. In case they are not equal, then we will check if the target element is less than or greater than the middle element.
    - In case, the target element is less than the middle element, it means that the target element, if present in the given array, will be on the left side of the middle element as the array is sorted.
    - Otherwise, the target element will be on the right side of the middle element.
4. This helps us discard half of the length of the array and we have reduced the number of operations.
5. Now, since we know that the element is on the left, we will assume that the array's starting and ending indices to be:
    - **For left:** start = 0, end = n/2-1
    - **For right:** start = n/2, end = n-1

      where n are the total elements in the array.
6. We will repeat this process until we find the target element. In case start and end becomes equal or start becomes more than end, and we haven't found the target element till now, we will simply return -1 as that element is not present in the array.

In the example above, the middle element is 3 at index 2, and the target element is 2 which is greater than the middle element, so we will move towards the left part. Now marking start = 0, and end = n/2-1 = 1, now middle = (start + end)/2 = 0. Now comparing the 0-th index element with 2, we find that 2 > 1, hence we will be moving towards the right. Updating the start = 1 and end = 1, middle becomes 1, comparing 1-st index of the array with the target element, we find they are equal, meaning from here we will simply return 1 (the index of the element).

Now try implementing the approach yourself...

Implementation

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

int binarySearch(int arr[], int n, int x) {
        int start = 0, end = n - 1;
        while(start <= end) {
                int mid = (start + end) / 2;
                if(arr[mid] == x) {
                        return mid;
                }
                else if(x < arr[mid]) {
                        end = mid - 1;
                }
                else {
                        start = mid + 1;
                }
        }

        return -1;
}

int main() {
        // Take array input from the user
        int n;
        cin >> n;

        int input[100];

        for(int i = 0; i < n; i++) {
                cin >> input[i];
        }

        int x;
        cin >> x;

        cout << binarySearch(input, n, x) << endl;
}
```

Note: There are other approaches also to perform binary searching like Recursion which you will study in advanced topics.

Sorting

Sorting means an arrangement of elements in an ordered sequence either in increasing(ascending) order or decreasing(descending) order. Sorting is very important and many softwares and programs use this. Binary search also requires sorting. There are many different sorting techniques. The major difference is the amount of space and time they consume while being performed in the program.

For now, we will be discussing the following sorting techniques only:

- Selection sort
- Bubble sort
- Insertion sort

Let's discuss them one-by-one...

Selection sort:

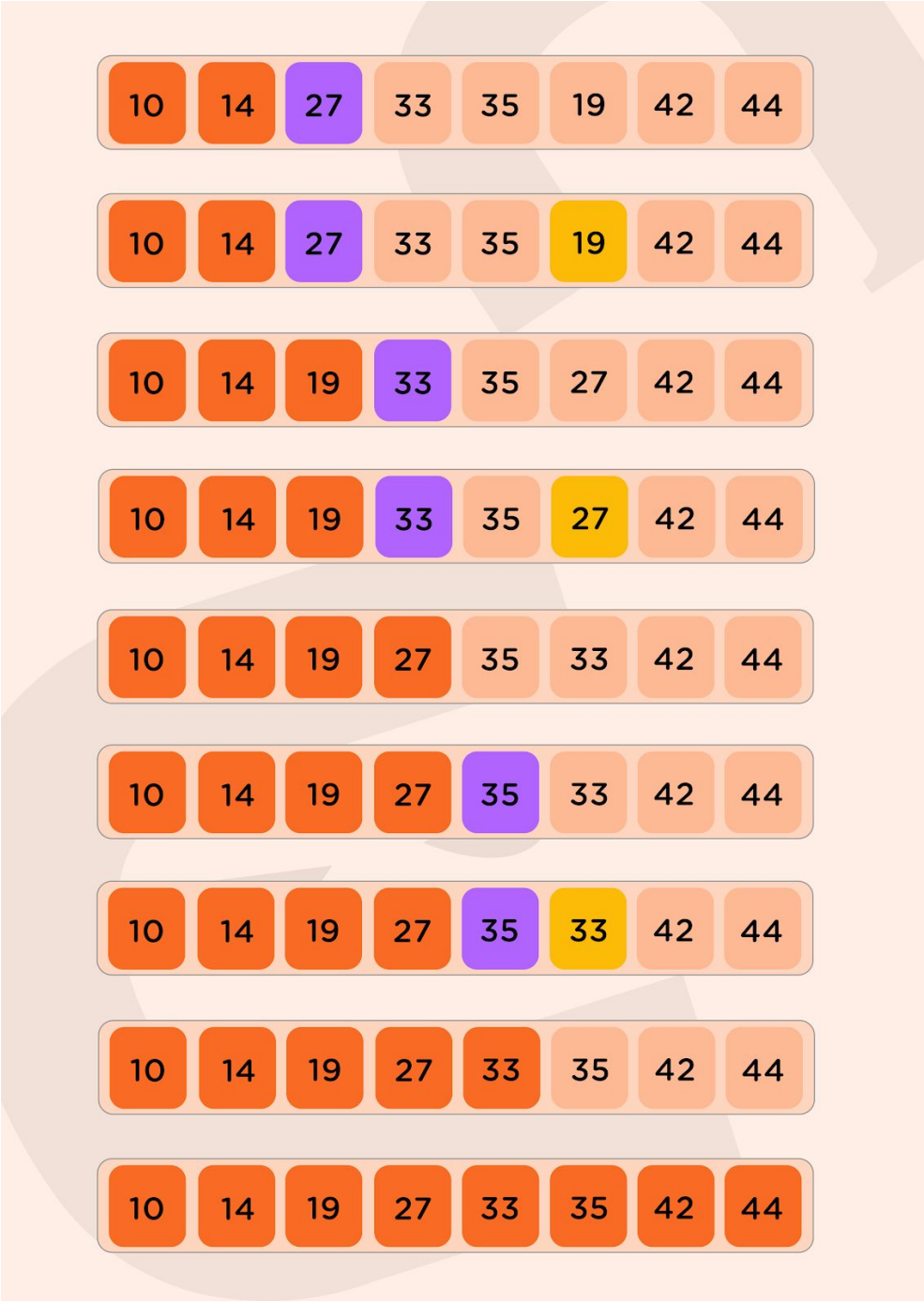
Steps: (sorting in increasing order)

1. First-of-all, we will find the smallest element of the array and swap that with the element at index 0.
2. Similarly, we will find the second smallest and swap that with the element at index 1 and so on...
3. Ultimately, we will be getting a sorted array in increasing order only.

Let us look at the example for better understanding:

Consider the following depicted array as an example. You want to sort this array in increasing order.

Following is the pictorial diagram for better explanation of how it works:



This is how we obtain the sorted array at last.

Now looking at the implementation of Selection Sort:

Implementation of selection sort

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

void selectionSort(int input[], int n) {
        for(int i = 0; i < n-1; i++ ) {
        // Find min element in the array
        int min = input[i], minIndex = i;
        for(int j = i+1; j < n; j++) {
                if(input[j] < min) {
                        min = input[j];
                        minIndex = j;
                }
        }
        // Swap
        int temp = input[i];
        input[i] = input[minIndex];
        input[minIndex] = temp;
        }
}

int main() {
        int input[] = {3, 1, 6, 9, 0, 4};
        selectionSort(input, 6);
        for(int i = 0; i < 6; i++) {
                cout << input[i] << " ";

        }
        cout << endl;
}
```

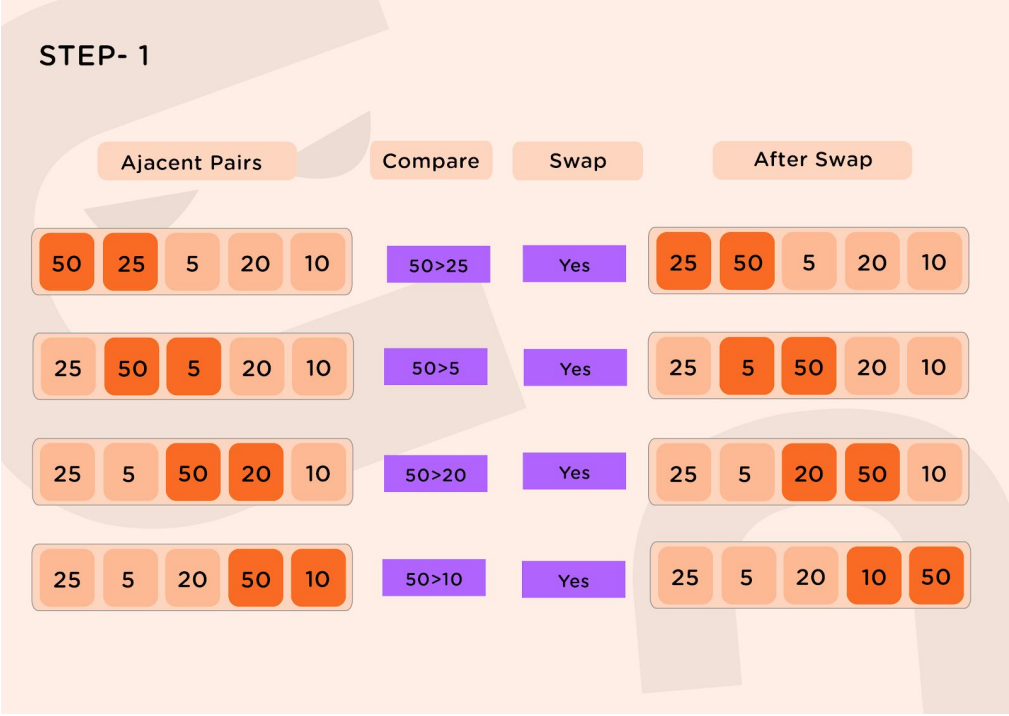
Bubble sort:

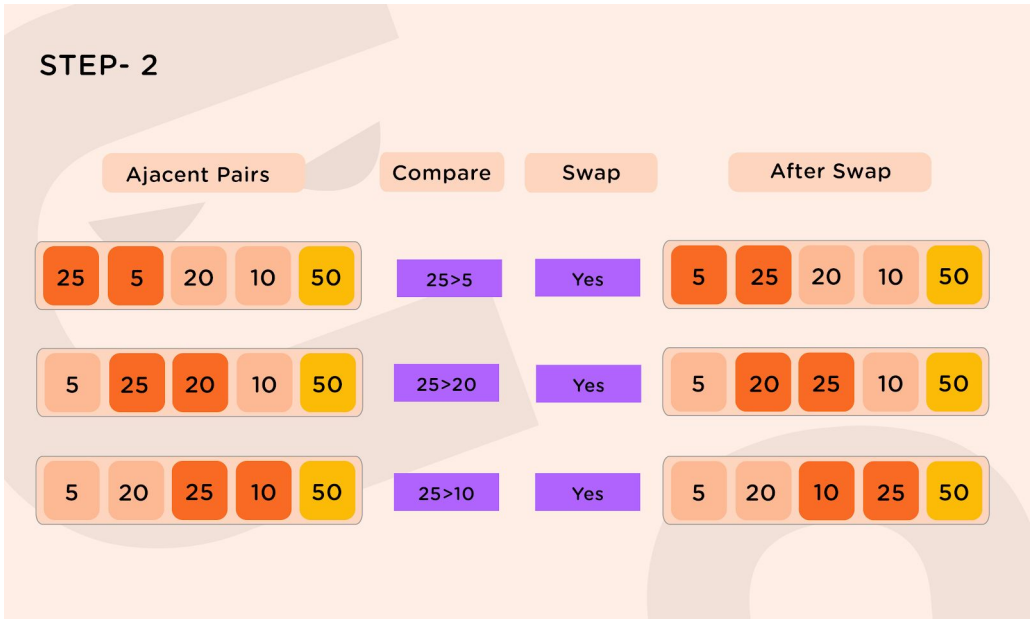
In selection sort the elements from the start get placed at the correct position first and then the further elements, but in the bubble sort, the elements start to place correctly from the end.

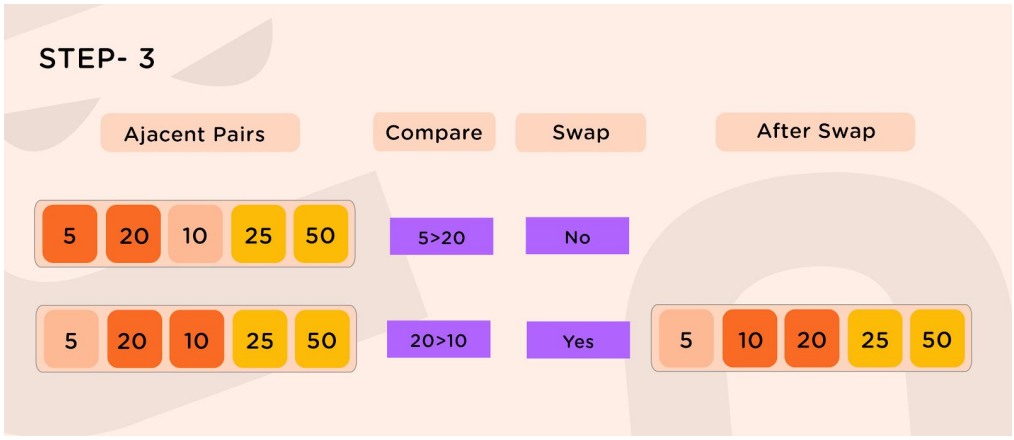
In this technique, we just compare the two adjacent elements of the array and then sort them manually by swapping if not sorted. Similarly, we will compare the next two elements (one from the previous position and the corresponding next) of the array and sort them manually. This way the elements from the last get placed in their correct position. This is the difference between selection sort and bubble sort. Consider the following depicted array as an example. You want to sort this array in increasing order.

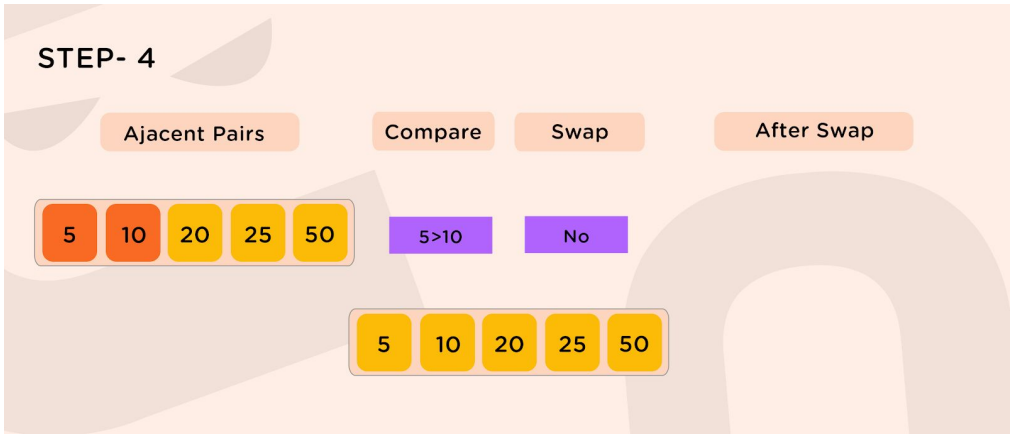
50	25	5	20	10

Following is the pictorial diagram for better explanation of how it works:




STEP- 2
Ajacent Pairs
Compare
Swap
After Swap
25 5 20 10 50
25>5
Yes
5 25 20 10 50
5 25 20 10 50
25>20
Yes
5 20 25 10 50
5 20 25 10 50
25>10
Yes
5 20 10 25 50


STEP- 3
Ajacent Pairs
Compare
Swap
After Swap
5 20 10 25 50
5>20
No
5 20 10 25 50
20>10
Yes
5 10 20 25 50


STEP- 4
Ajacent Pairs
Compare
Swap
After Swap
5 10 20 25 50
5>10
No
5 10 20 25 50

Implementation of bubble sort

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

void bubbleSort(int array[], int size) {

  // run loops two times: one for walking throught the array
  // and the other for comparison
  for (int step = 0; step < size - 1; ++step) {
    for (int i = 0; i < size - step - 1; ++i) {

      // To sort in descending order, change > to < in this line.
      if (array[i] > array[i + 1]) {

        // swap if greater is at the rear position
        int temp = array[i];
        array[i] = array[i + 1];
        array[i + 1] = temp;
      }
    }
  }
}

// function to print the array
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    cout << "  " << array[i];
  }
  cout << endl;
}

// driver code
int main() {
  int data[] = {-2, 45, 0, 11, -9};
  int size = sizeof(data) / sizeof(data[0]);
  bubbleSort(data, size);
  cout << "Sorted Array in Ascending Order:\n";
  printArray(data, size);
}
```

Insertion sort:

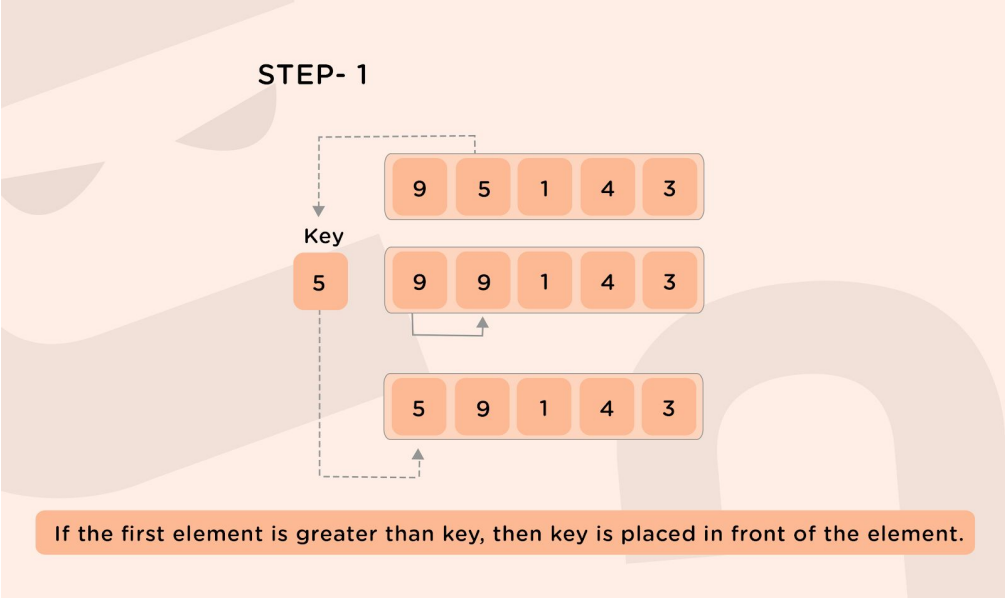
Insertion Sort works similar to how we sort a hand of playing cards.
Imagine that you are playing a card game. You're holding the cards in your hand, and these cards are sorted. The dealer hands you exactly one new card. You have to put it into the correct place so that the cards you're holding are still sorted. In selection sort, each element that you add to the sorted subarray is no smaller than the elements already in the sorted subarray. But in our card example, the new card could be smaller than some of the cards you're already holding, and so you go down the line, comparing the new card against each card in your hand, until you find the place to put it. You insert the new card in the right place, and once again, your hand holds fully sorted cards. Then the dealer gives you another card, and you repeat the same procedure. Then another card, and another card, and so on, until the dealer stops giving you cards. This is the idea behind **insertion sort**.
Loop over positions in the array, starting with index 1. Each new position is like the new card handed to you by the dealer, and you need to insert it into the correct place in the sorted subarray to the left of that position.

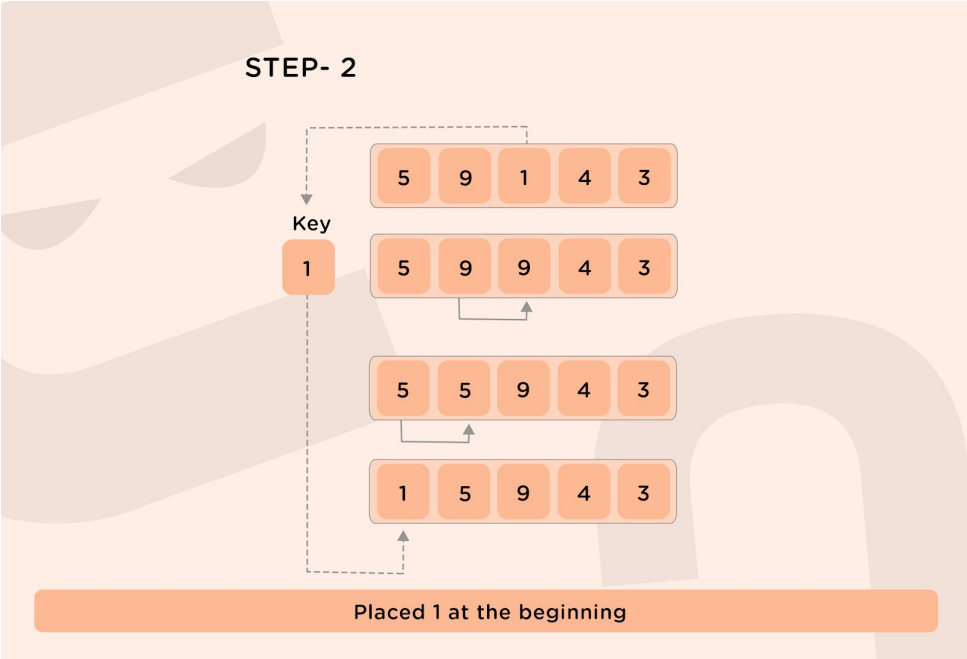
Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.

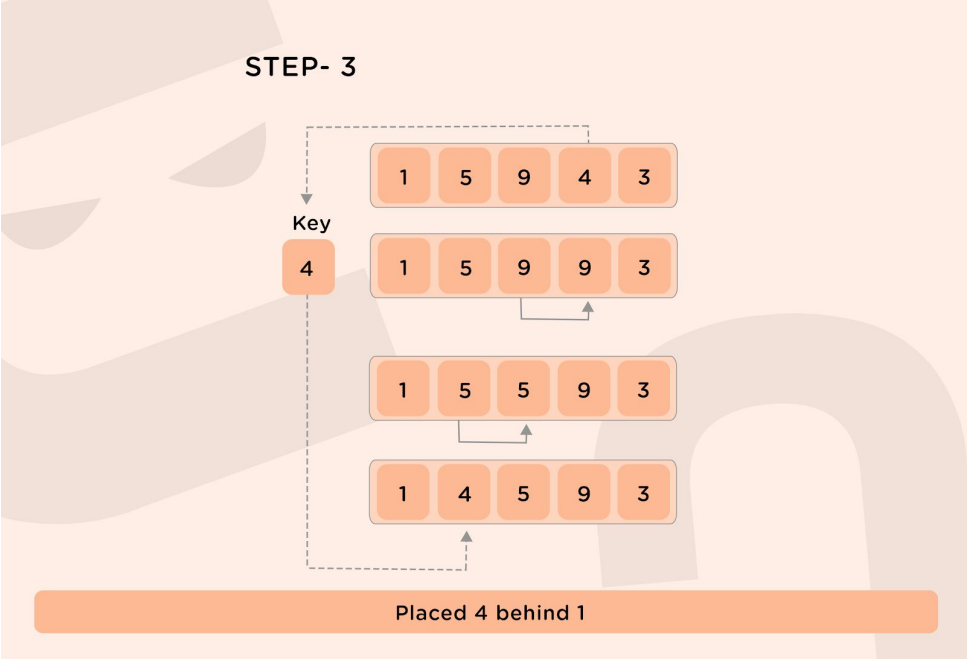
Consider the following depicted array as an example. You want to sort this array in increasing order.

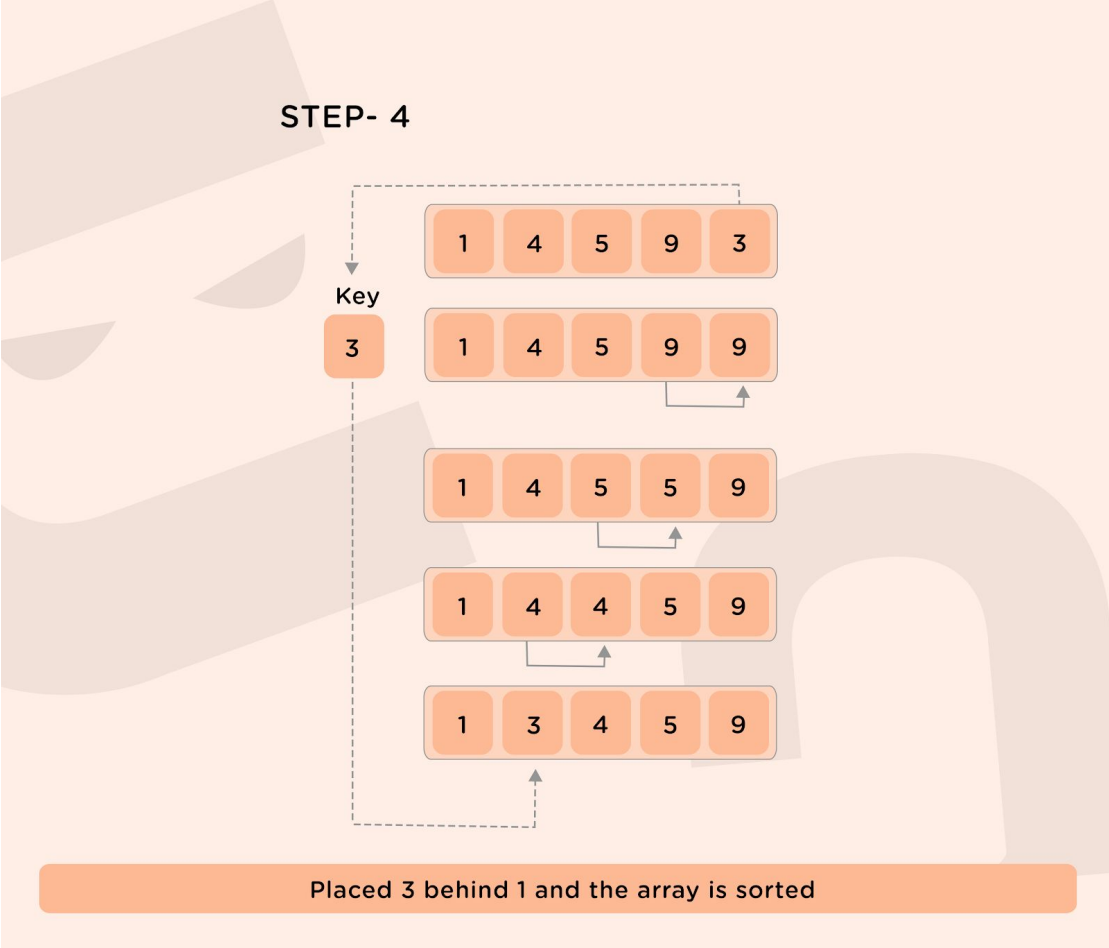
9	5	4	1	3

Following is the pictorial diagram for better explanation of how it works:




STEP- 2
Key
1
5 9 1 4 3
5 9 9 4 3
5 5 9 4 3
1 5 9 4 3
Placed 1 at the beginning


STEP- 3
Key
4
1 5 9 4 3
1 5 9 9 3
1 5 5 9 3
1 4 5 9 3
Placed 4 behind 1



Implementation of insertion sort

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

void display(int array[], int size) {
  for(int i = 0; i<size; i++)
    cout << array[i] << " ";
  cout << endl;
}
void insertionSort(int array[], int size) {
  int key, j;
  for(int i = 1; i<size; i++) {
    key = array[i];                    //take value
    j = i;
    while(j > 0 && array[j-1]>key) {
```

```
      array[j] = array[j-1];
      j--;
    }
    array[j] = key;                    //insert in right place
  }
}
int main() {
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;
  int arr[n];                          //create an array with given number of elements
  cout << "Enter elements:" << endl;
  for(int i = 0; i<n; i++) {
    cin >> arr[i];
  }

  cout << "Array before Sorting: ";
  display(arr, n);

  insertionSort(arr, n);
  cout << "Array after Sorting: ";
  display(arr, n);
}
```

Now, practice different questions to get more familiar with the concepts. In the advanced course, you will study more types of sorting techniques.

Practice

- For binary search:

  https://www.hackerearth.com/practice/algorithms/searching/binary-search/practice-problems/

- For sorting:

  https://www.hackerearth.com/practice/algorithms/sorting/bubble-sort/tutorial/

  (In this link, you will find the questions for bubble sort, checkout the left tab, from there you can switch to other sorting techniques.)