SORTING

Sorting:

- The arrangement of data in a preferred order is called sorting.
- By sorting data, it is easier to search through it quickly and easily.
- The simplest example of sorting is a dictionary.

Types of Sorting:

1. Bubble Sort:

- **Bubble Sort** is a simple algorithm which is used to sort a given set of **n** elements provided in form of an array with **n** number of elements.
- Bubble Sort compares all the element one by one and sort them based on their values.
- If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will **swap** both the elements and then move on to compare the second and the third element, and so on.
- If we have total **n** elements, then we need to repeat this process for **n-1** times.
- It is known as **bubble sort**, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index.

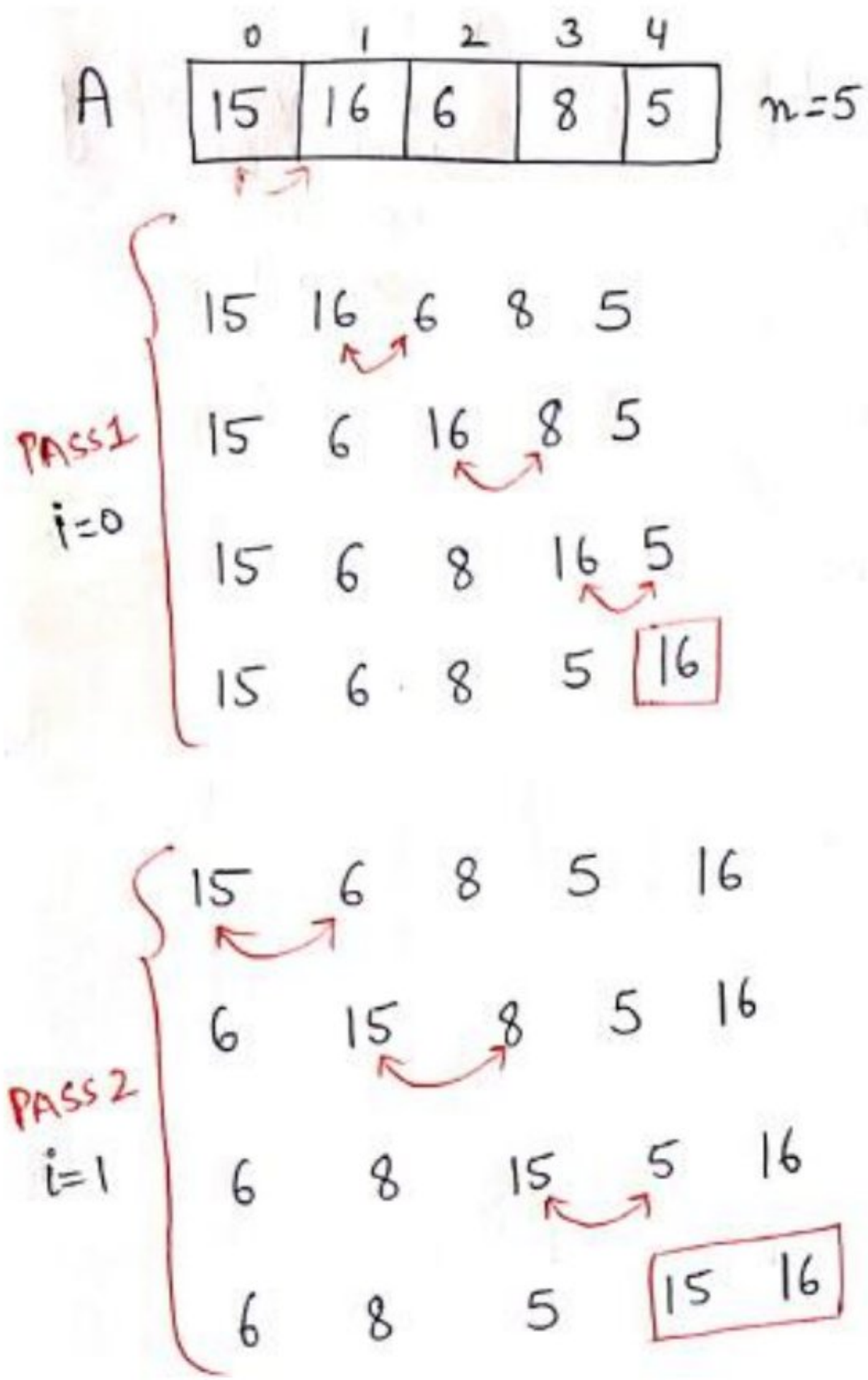
Implementing Bubble Sort Algorithm

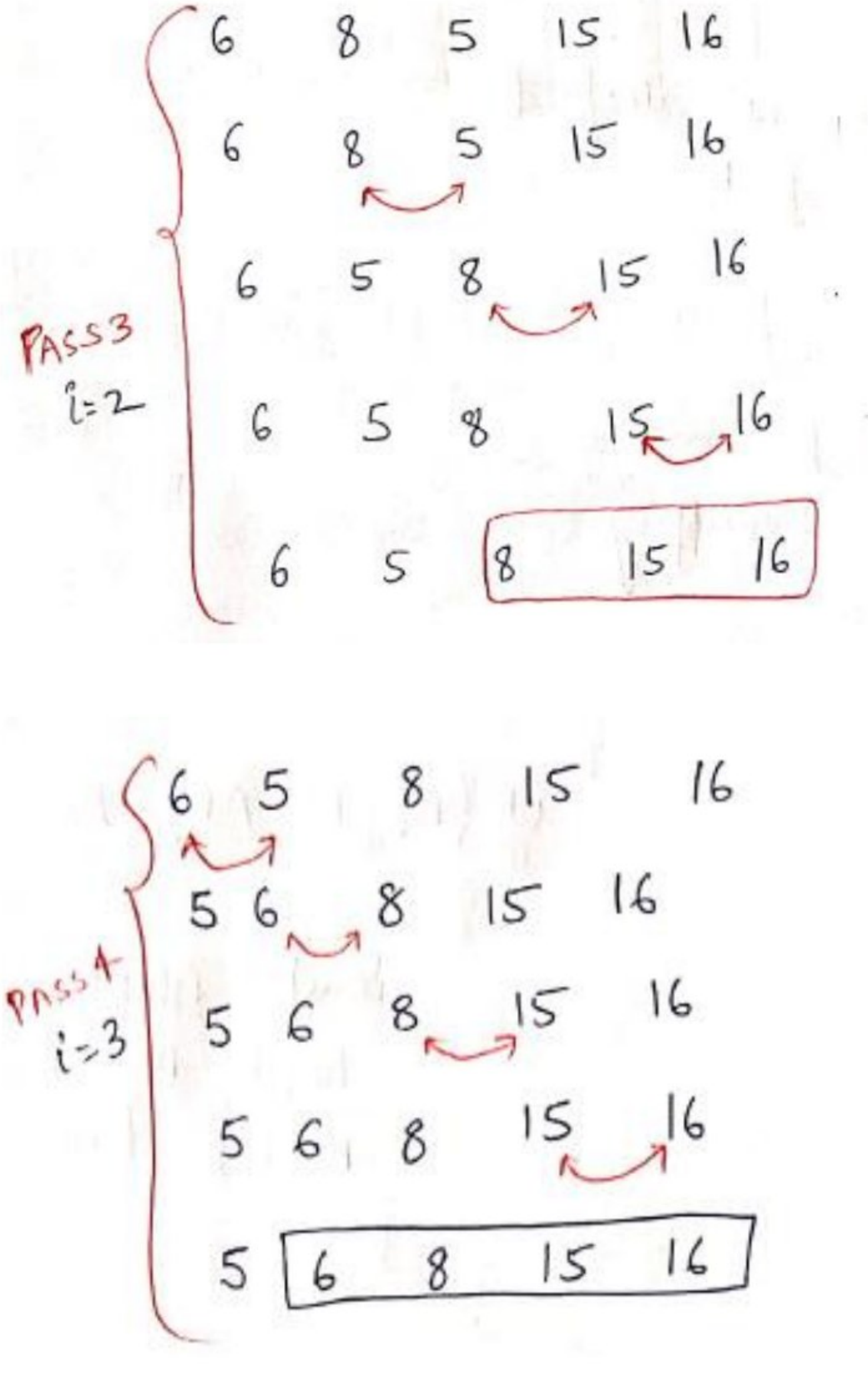
Following are the steps involved in bubble sort (for sorting a given array in ascending order):

1. Starting with the first element (index = 0), compare the current element with the next element of the array.

2. If the current element is greater than the next element of the array, swap them.
3. If the current element is less than the next element, move to the next element. **Repeat Step 1**.

Example: Let's consider an array with values $\{15, 16, 6, 8, 5\}$



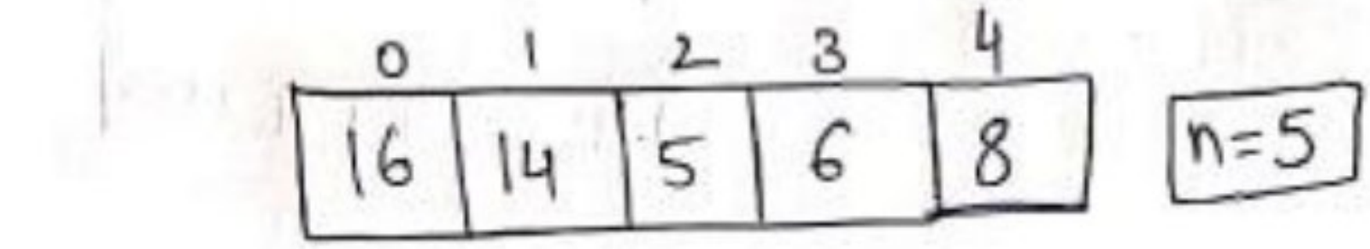


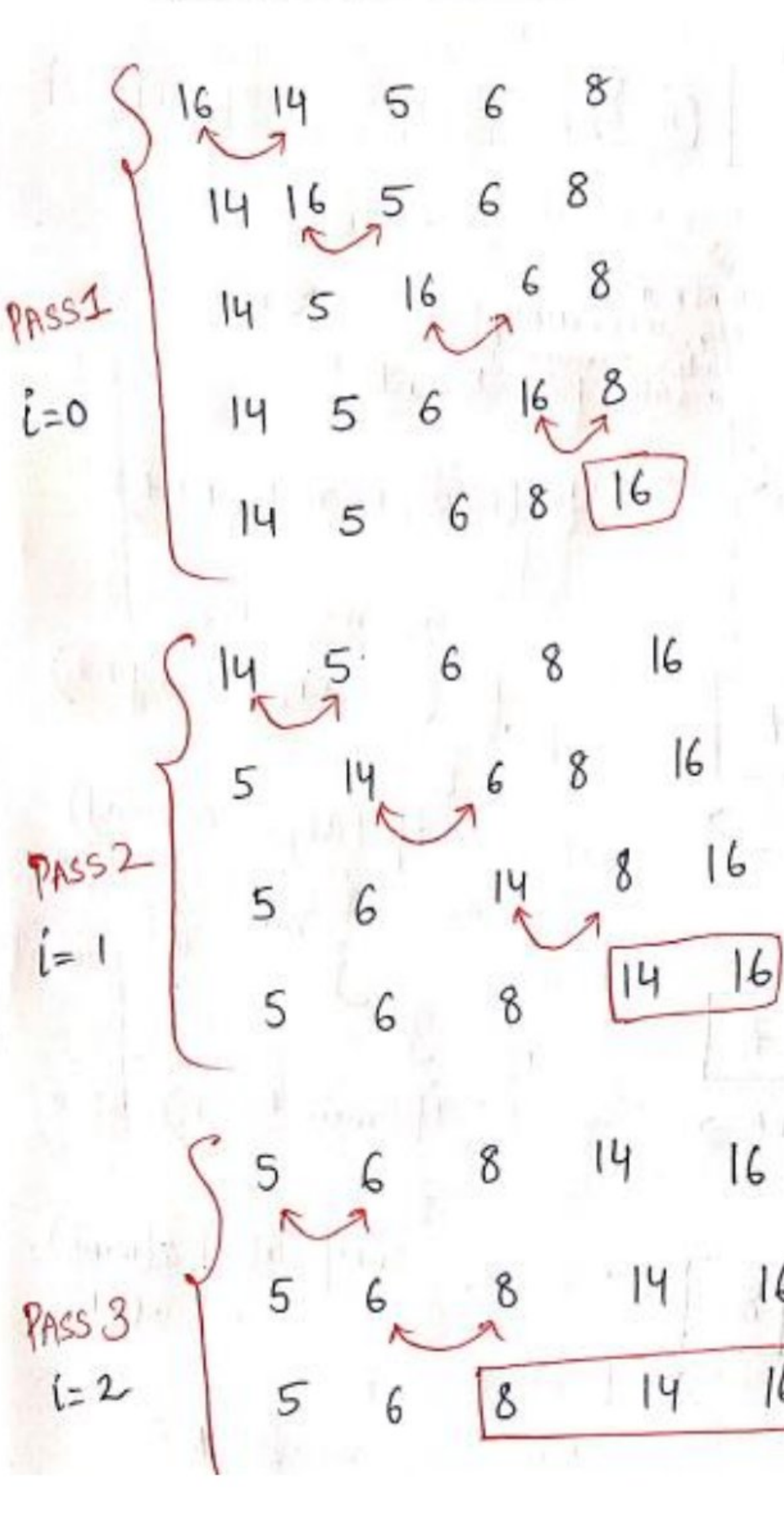
```
for(i=0; i<n-1; i++)
{
  for(j=0; j<n-1; j++)
  {
    if (A[j] > A[j+1])
    {
      temp = A[j];
      A[j] = A[j+1];
      A[j+1] = temp;
    }
  }
}
```

```
for(i=0; i<n-1; i++)
{
  for(j=0; j<n-1-i; j++)
  {
    if (A[j] > A[j+1])
    {
      temp = A[j];
      A[j] = A[j+1];
      A[j+1] = temp;
    }
  }
}
```

n-1-i
↳ unnecessary comparison's can be avoided.

Example: Let's consider an array with values {16, 14, 5, 6, 8}





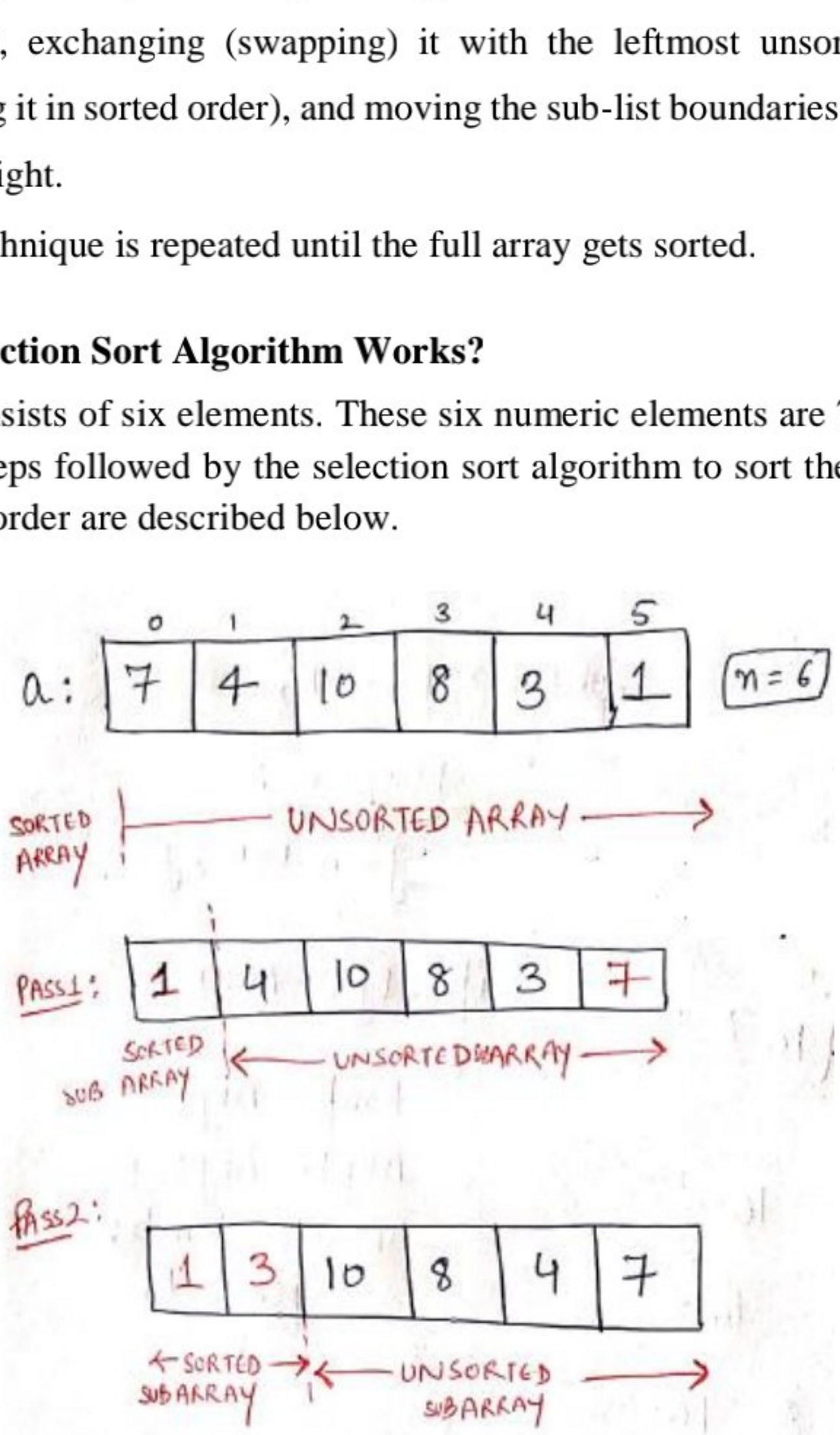
```
for(i=0; i<n-1; i++)
{
    flag=0;
    for(j=0; j<n-1-i; j++)
    {
        if(A[j] > A[j+1])
        {
            temp = A[j];
            A[j] = A[j+1];
            A[j+1] = temp;
            flag=1;
        }
    }
    if(flag==0)
        break;
}
```

2. Selection Sort

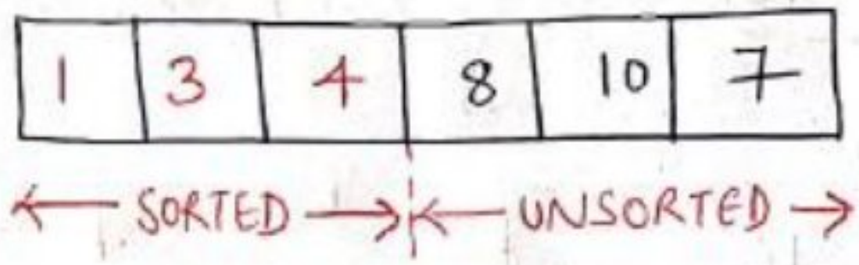
- The selection is a straightforward process of sorting values.
- This **sorting** algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the **sorted** part at the left end and the unsorted part at the right end. Initially, the **sorted** part is empty and the unsorted part is the entire list.
- The algorithm proceeds by finding the smallest element in the unsorted sub-list, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sub-list boundaries one element to the right.
- The technique is repeated until the full array gets sorted.

How the Selection Sort Algorithm Works?

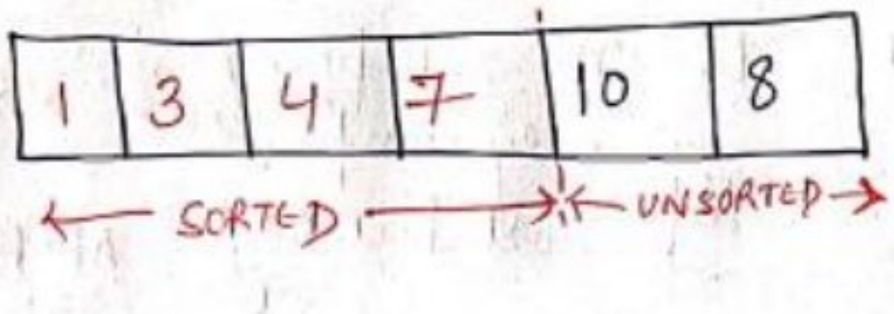
The array consists of six elements. These six numeric elements are 7, 4, 10, 8, 3 and 1. The steps followed by the selection sort algorithm to sort these elements in ascending order are described below.



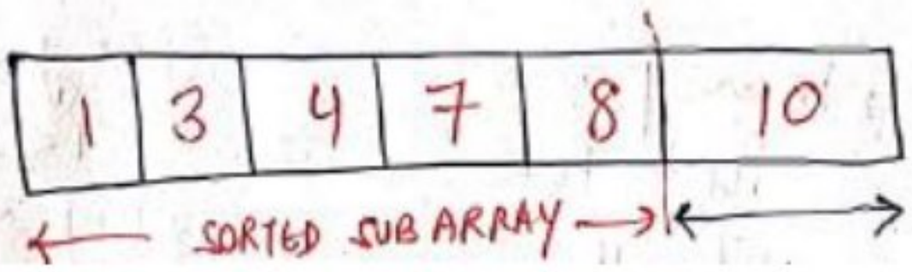
PASS 3:



PASS 4:



PASS 5:




```
for(i=0; i<n-1; i++)
  {
    int min=i;
    for(j=i+1; j<n; j++)
    {
      if (a[j] < a[min])
        { min=j;
        }
    }
    if(min != i)
    {
      swap(a[i], a[min]);
    }
}
```

3. Insertion Sort:

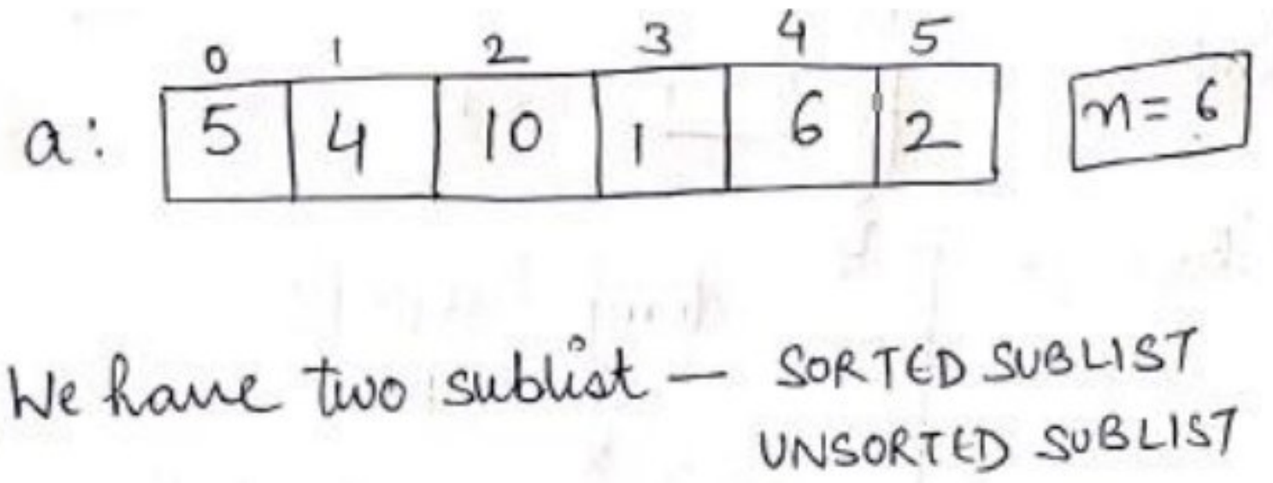
- Consider you have 10 cards out of a deck of cards in your hand. And they are sorted, or arranged in the ascending order of their numbers.
- If I give you another card, and ask you to **insert** the card in just the right position, so that the cards in your hand are still sorted. What will you do?
- Well, you will have to go through each card from the starting or the back and find the right position for the new card, comparing its value with each card.

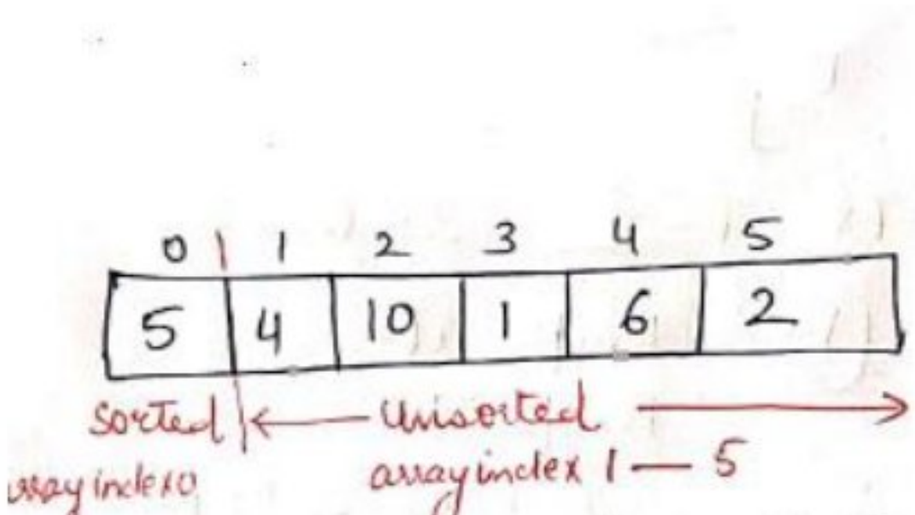
  Once you find the right position, you will **insert** the card there.

How Insertion Sort Works?

1. The first step involves the comparison of the element with its adjacent element.
2. And if at every comparison reveals that the element can be inserted at a particular position, then space is created for it by shifting the other elements one position to the right and inserting the element at the suitable position.
3. The above procedure is repeated until all the element in the array is at their apt position.

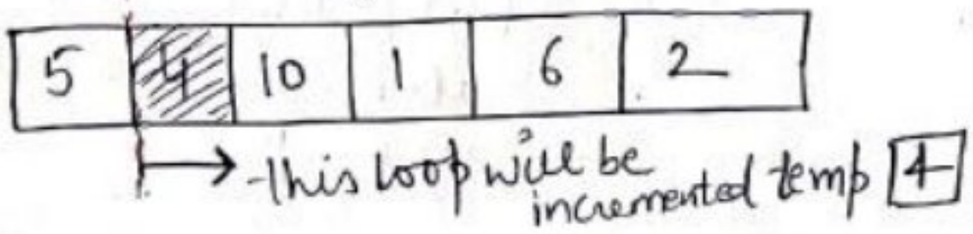
Example: Consider an array with values **[5, 4, 10, 1, 6, 2]**

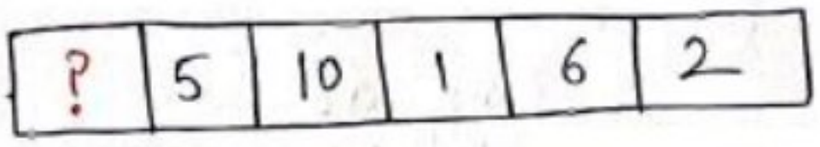


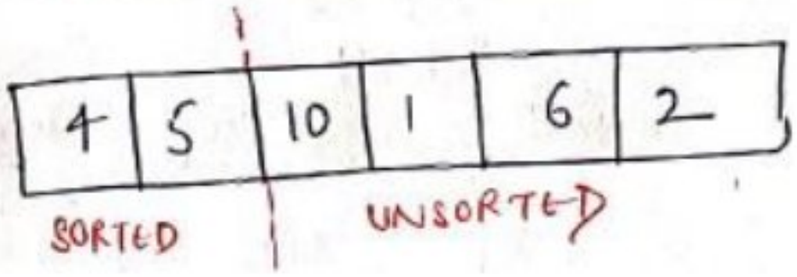


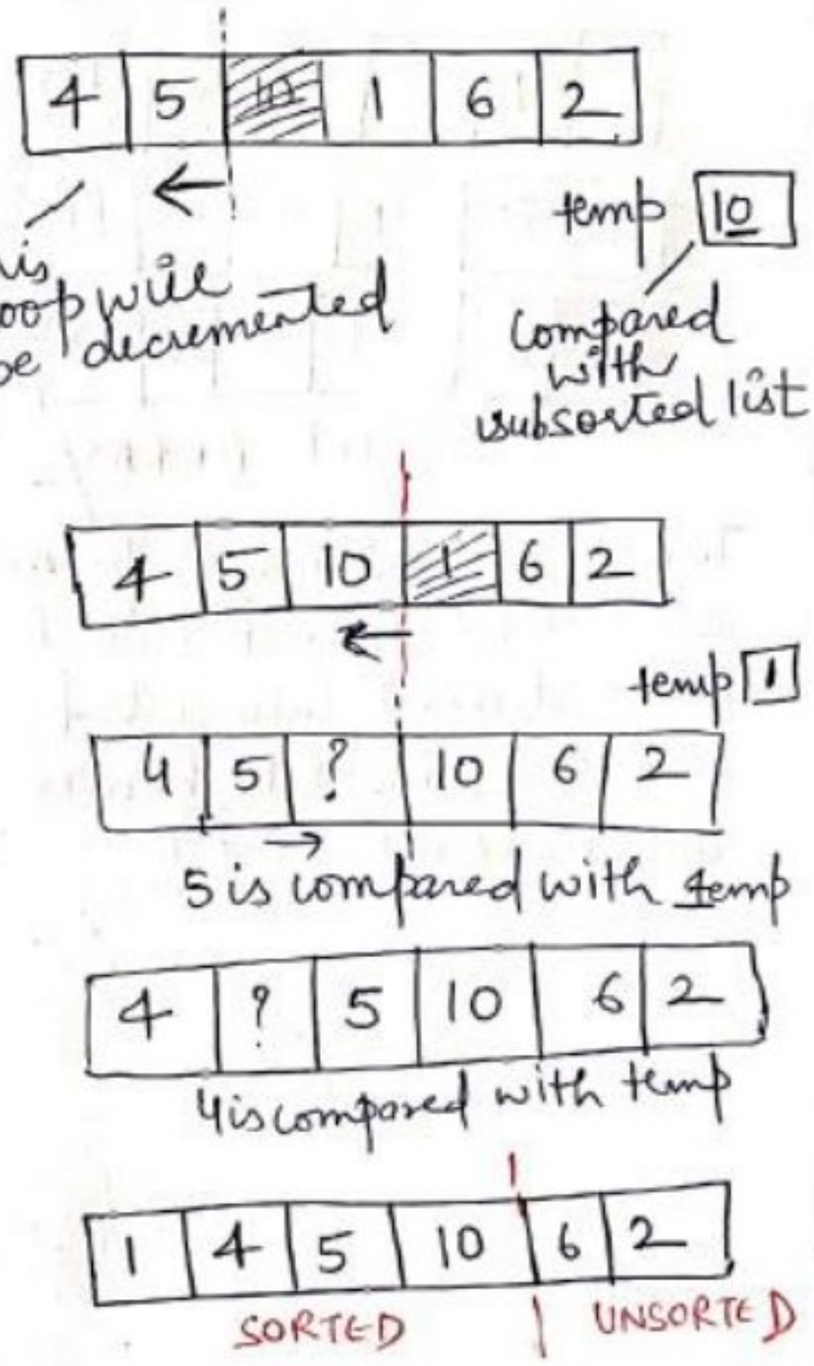
We take single element in sorted list as one element is always sorted.

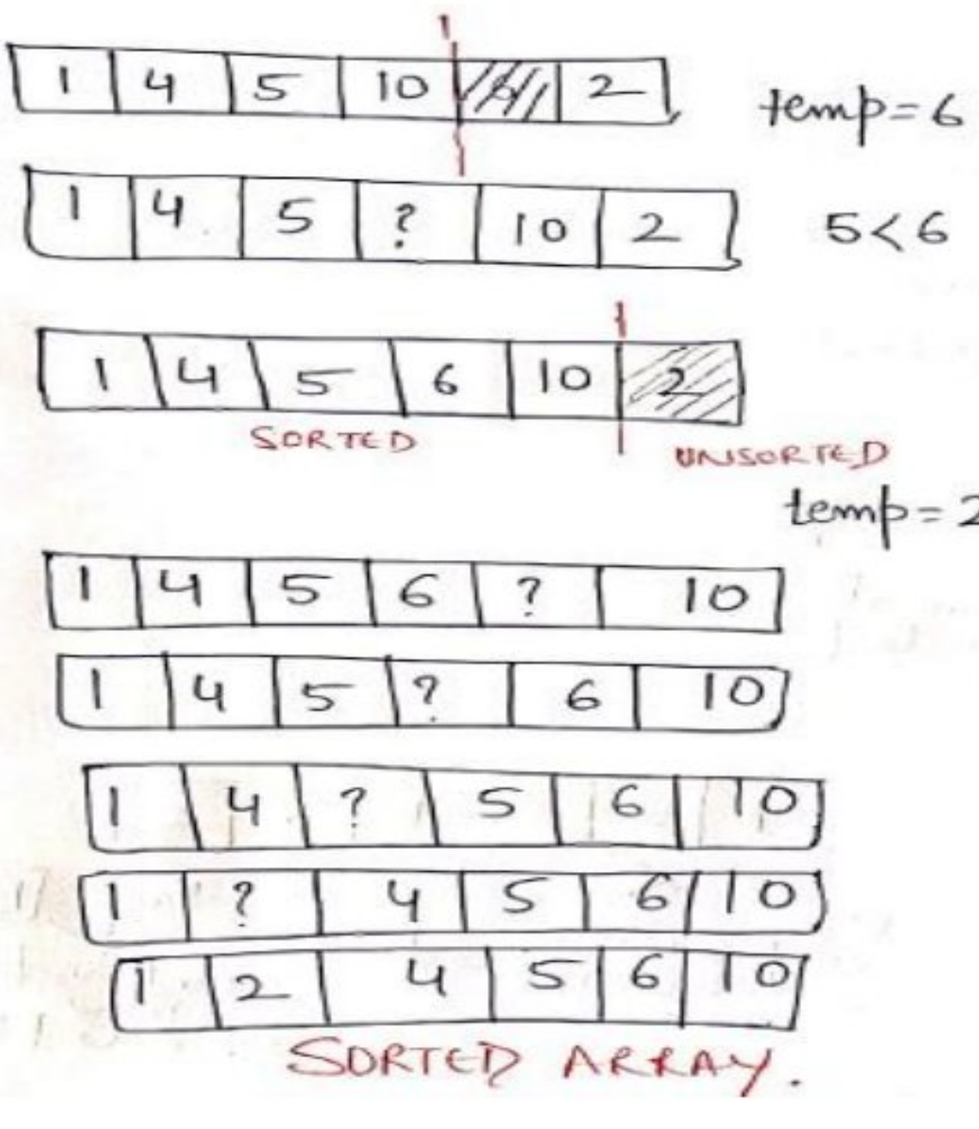
We are starting from index = 1

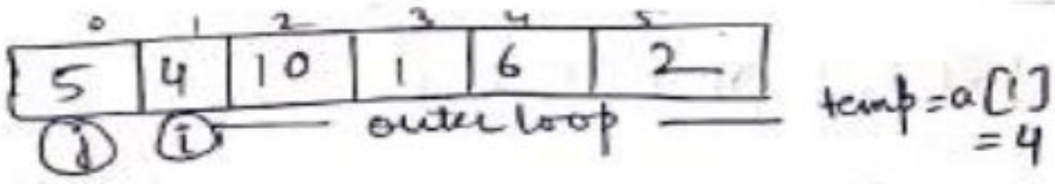













```
for (i=1; i<n; i++)
{
    temp = a[i];
    j = i-1;
  while (j>=0 && a[j]>temp)
  {
      a[j+1] = a[j];
      j--;
  }
  a[j+1] = temp;
}
```