DATA STRUCTURES

Unit-I

Sorting Techniques

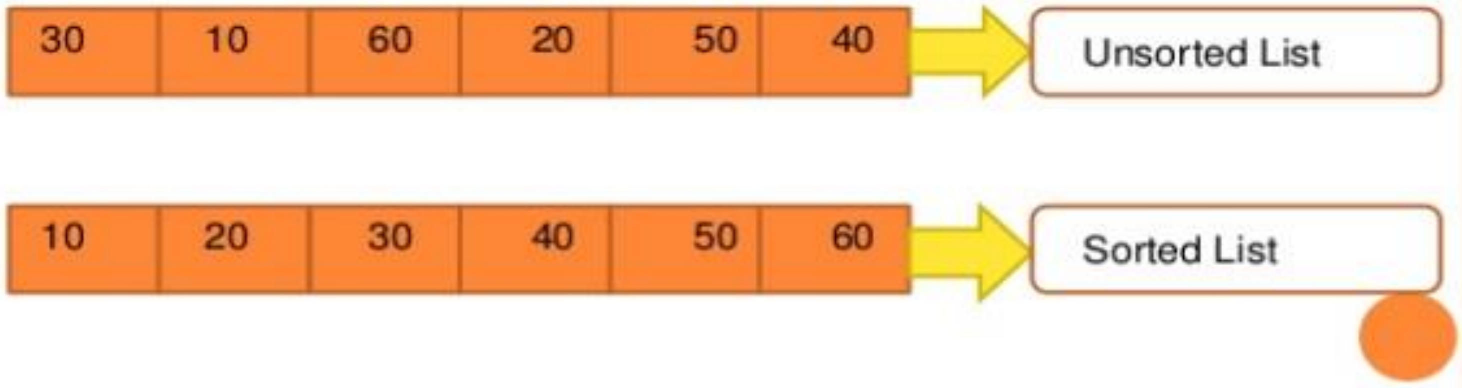
Dr G.Kalyani

Topics

- What is Sorting
- Types of Sorting
- Merge Sort
- Quick Sort
- Selection sort
- Bubble Sort
- Insertion Sort
- Heap Sort
- Comparison of Sorting Techniques

What is Sorting

- Sorting refers to operations of arranging a set of data in a given order.



Types of Sorting

- Internal Sorting:

If all the data to be sorted can be adjusted in main memory then it is called as Internal Sorting.

- External Sorting:

If data to be stored is large and aquires external memory then the type is called as External Sorting.

Merge Sort

Process of Merge sort

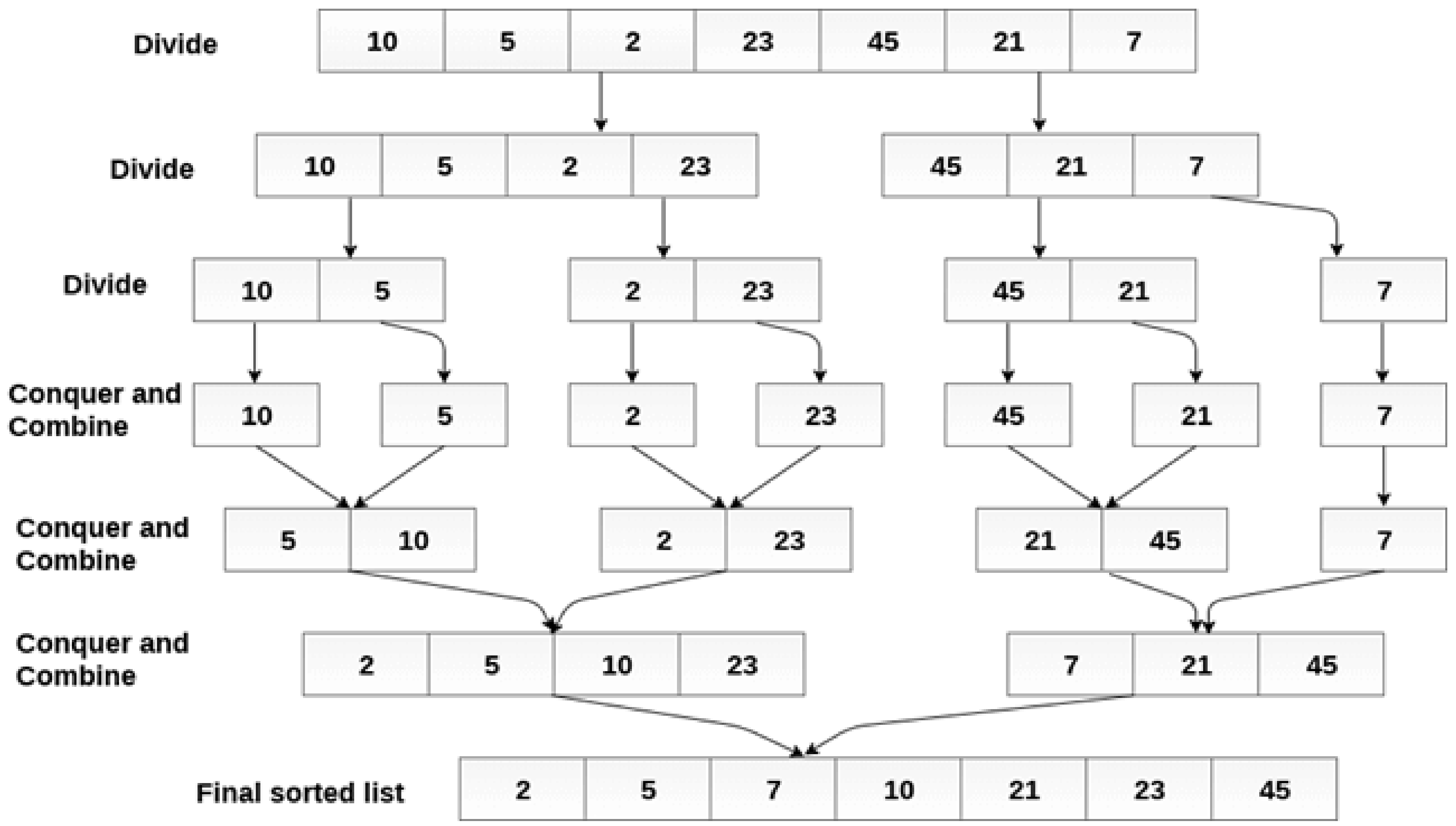
- Merge sort is the algorithm which follows divide and conquer approach. Consider an array A of n number of elements. The algorithm processes the elements in 3 steps.

1. If A Contains 0 or 1 elements then it is already sorted, otherwise, Divide A into two sub-array of equal number of elements.
2. Conquer means sort the two sub-arrays recursively using the merge sort.
3. Combine the sub-arrays to form a single final sorted array maintaining the ordering of the array.

Example for Merge Sort

- Consider the following array of 7 elements. Sort the array by using merge sort.
- A = {10, 5, 2, 23, 45, 21, 7}

Example for Merge Sort



Algorithm for Merge Sort

```
Algorithm MergeSort(a[], beg, end)
{
    if (beg<end)
    {
        mid = (beg+end)/2;
        MergeSort(a,beg,mid);
        MergeSort(a,mid+1,end);
        Merge(a,beg,mid,end);
    }
}
```

```
Algorithm merge(a[], beg, mid, end)
{
    i=beg;        / /pointer  for 1st part
    j=mid+1;  //pointer for 2nd part
     k = beg;  // pointer for temp array
    while (i<=mid && j<=end)
    {
        if (a[i]<a[j])
        {
            temp[k] = a[i];
            i = i+1;
        }
        else
        {
            temp[k] = a[j];
            j = j+1;
        }
        k++;
    }
    if (i>mid)
    {
        while (j<=end)
        {
            temp[k] = a[j];
            k++;
            j++;
        }
    }
    else
    {
        while (i<=mid)
        {
            temp[k] = a[i];
            k++;
            i++;
        }
    }
   // to copy the values from temp array to main
  array
    x = beg;
    while (x<k)
    {
        a[x]=temp[x];
        x++;
    }
}
```

Quick Sort

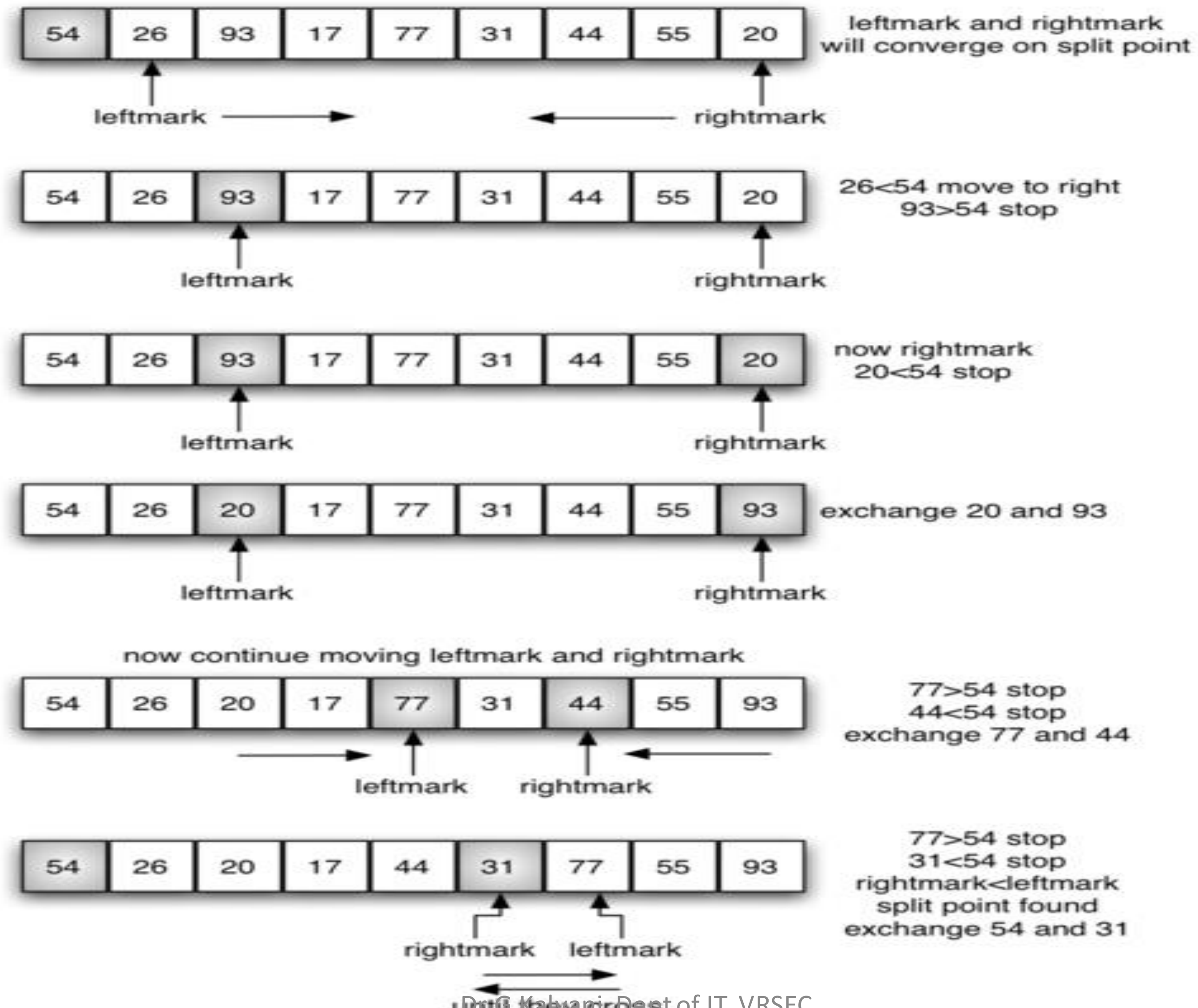
Process of Quick Sort

It finds the element called **pivot** which divides the array into two halves in such a way that elements in the left half are smaller than pivot and elements in the right half are greater than pivot.

Three steps

- ➢Find pivot that divides the array into two halves.
- ➢Quick sort the left half.
- ➢Quick sort the right half.

Quick sort example



Quick Sort Algorithm

```
Algorithm Quicksort(A[], Start, End)
{
    if (Start < End)
    {
        p = Partition(A, Start, End);
        Quicksort(A, Start, p - 1);
        Quicksort(A, p + 1, End);
    }
}
```

Quick Sort Algorithm

```
Algorithm Partition(A[], Start, End)
 {
        p = A[Start], i = Start + 1, j = End;
        while (i < j)
         {
            while (i < End && a[i] < p)   i++;
            while (j > Start && a[j] >= p)   j--;
            if (i < j)
            {
                temp = a[i]; a[i] = a[j]; a[j] = temp;
            }
        }
        a[Start] = a[j];
        a[j] = p;
        return j;
}
```

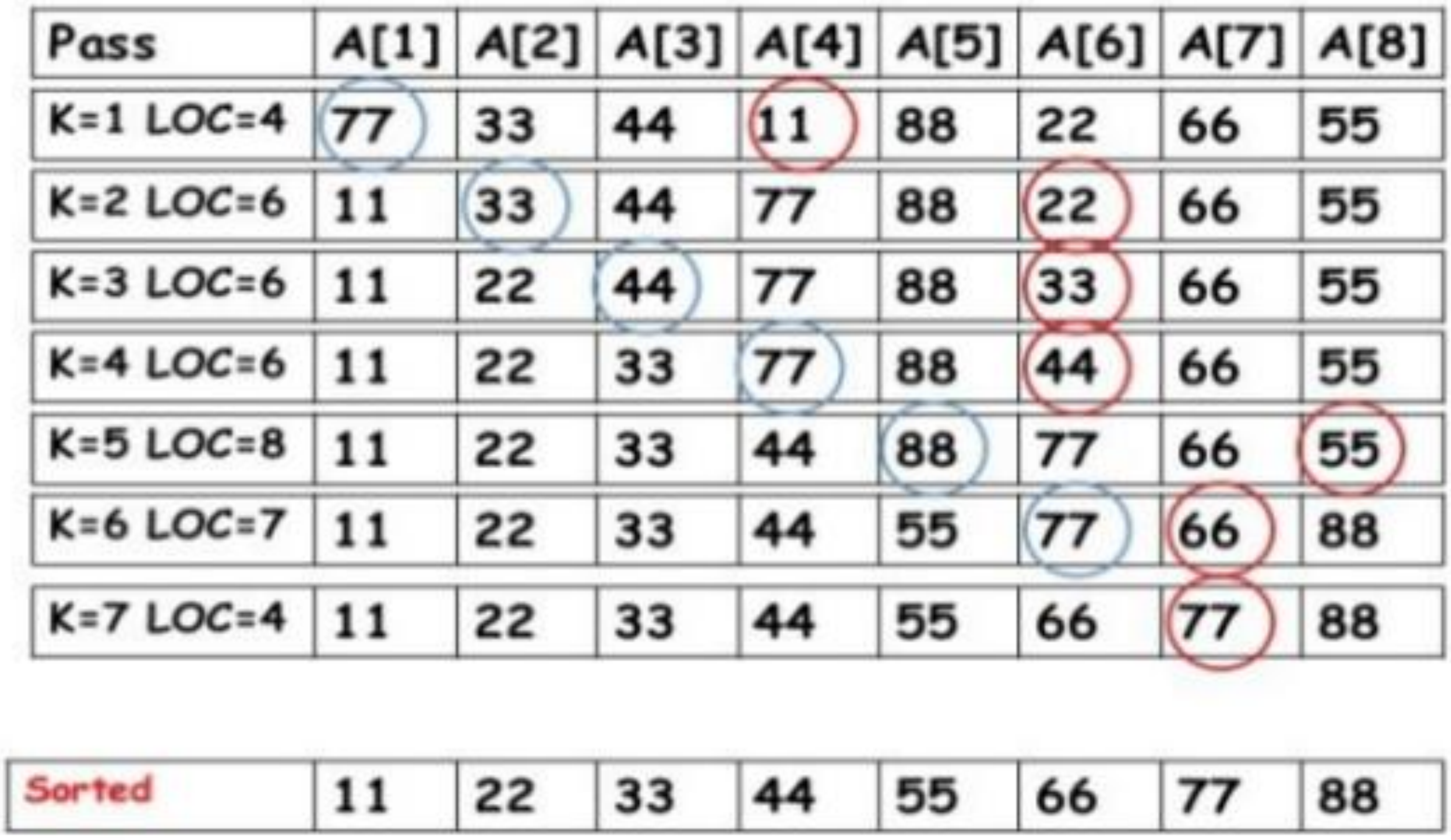
Selection Sort

Process of Selection Sort

Suppose an array A with N elements is in memory. Selection sort works as follows

First find the smallest element in the list and put it in the first position. Then, find the second smallest element in the list and put it in the second position and so on.

Example for Selection Sort



Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1 LOC=4	77	33	44	11	88	22	66	55
K=2 LOC=6	11	33	44	77	88	22	66	55
K=3 LOC=6	11	22	44	77	88	33	66	55
K=4 LOC=6	11	22	33	77	88	44	66	55
K=5 LOC=8	11	22	33	44	88	77	66	55
K=6 LOC=7	11	22	33	44	55	77	66	88
K=7 LOC=4	11	22	33	44	55	66	77	88

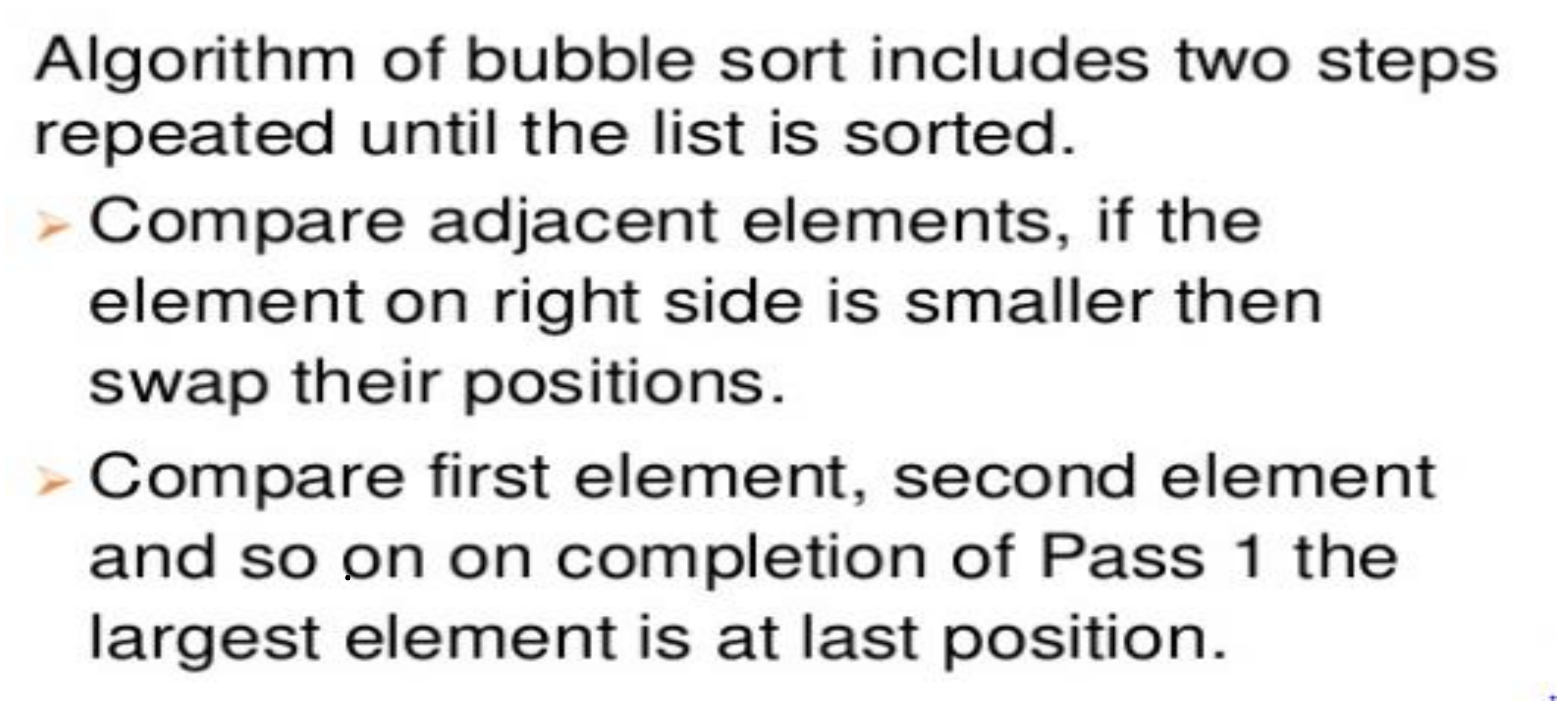
Sorted	11	22	33	44	55	66	77	88

Selection sort Algorithm

```
Algorithm Selection sort(A,n)
{
    for i = 0  to n-1
    {
        minpos = i;
        for j = i+1  to n-1
         {
            if (A[j] < A[minpos])
                minpos = j;
         }
         temp = A[i];
        A[i] = A[minpos];
        A[minpos] = temp;
     }
}
```

BUBBLE SORT

Process for Bubble Sort

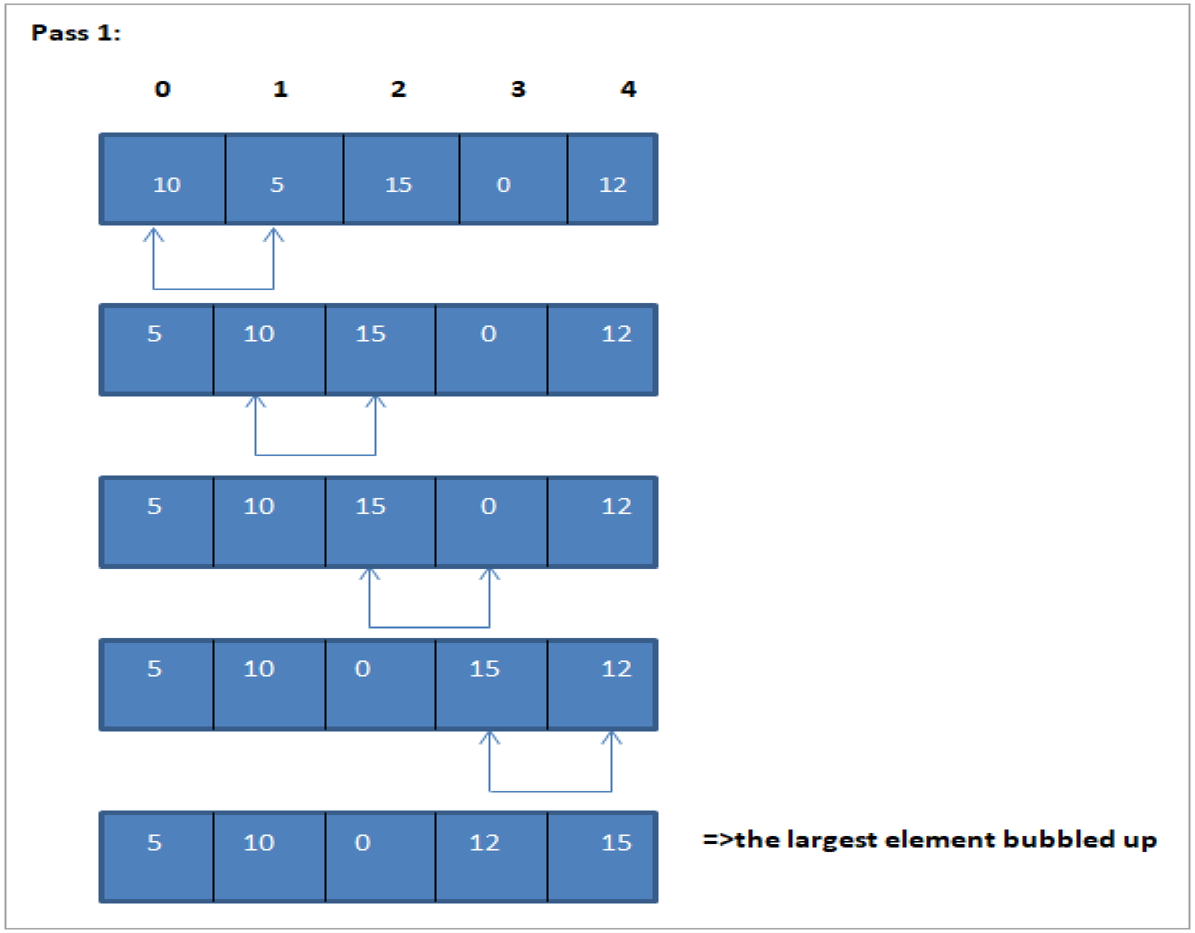


Algorithm of bubble sort includes two steps repeated until the list is sorted.

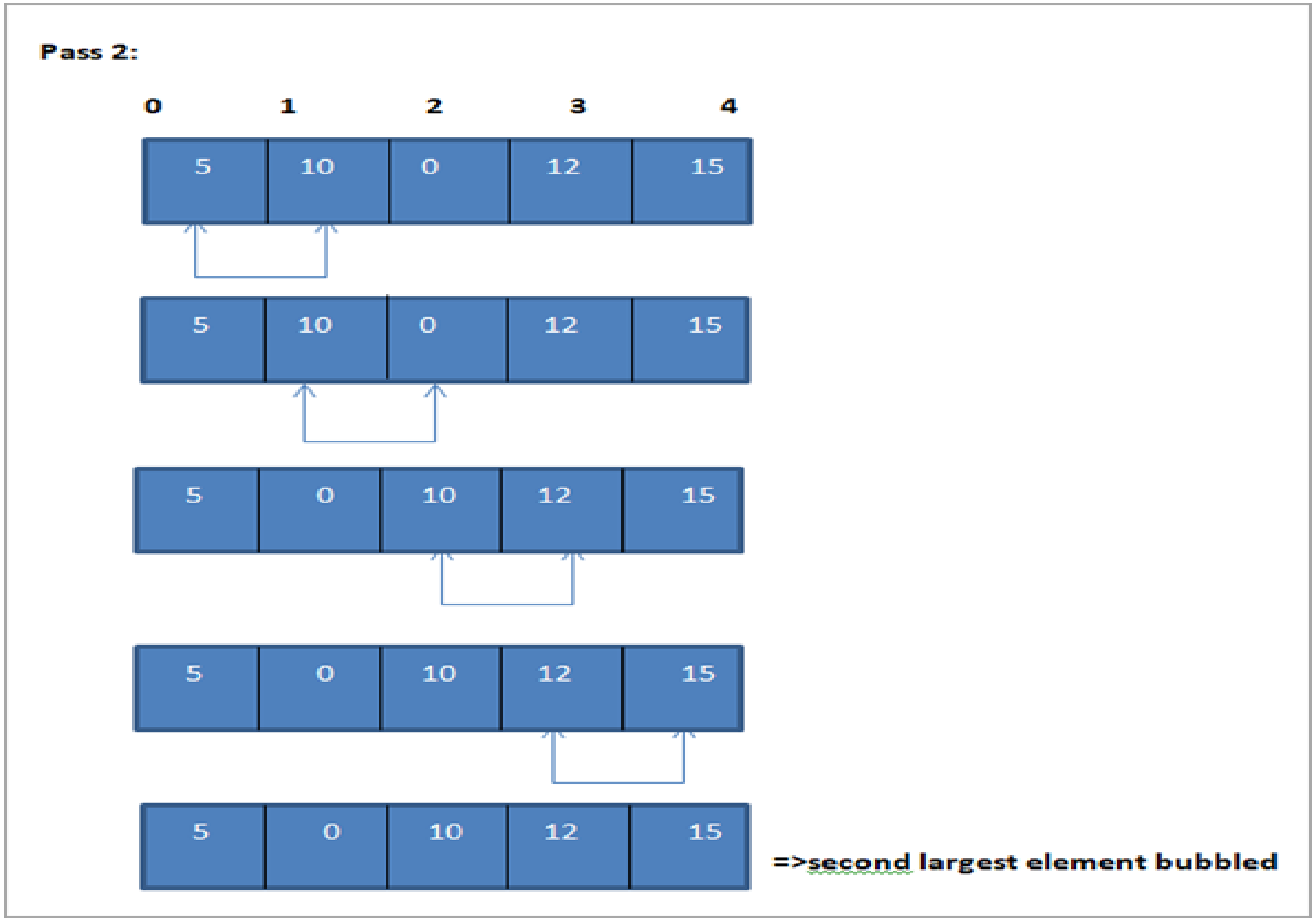
- Compare adjacent elements, if the element on right side is smaller then swap their positions.
- Compare first element, second element and so on on completion of Pass 1 the largest element is at last position.

Example for Bubble Sort

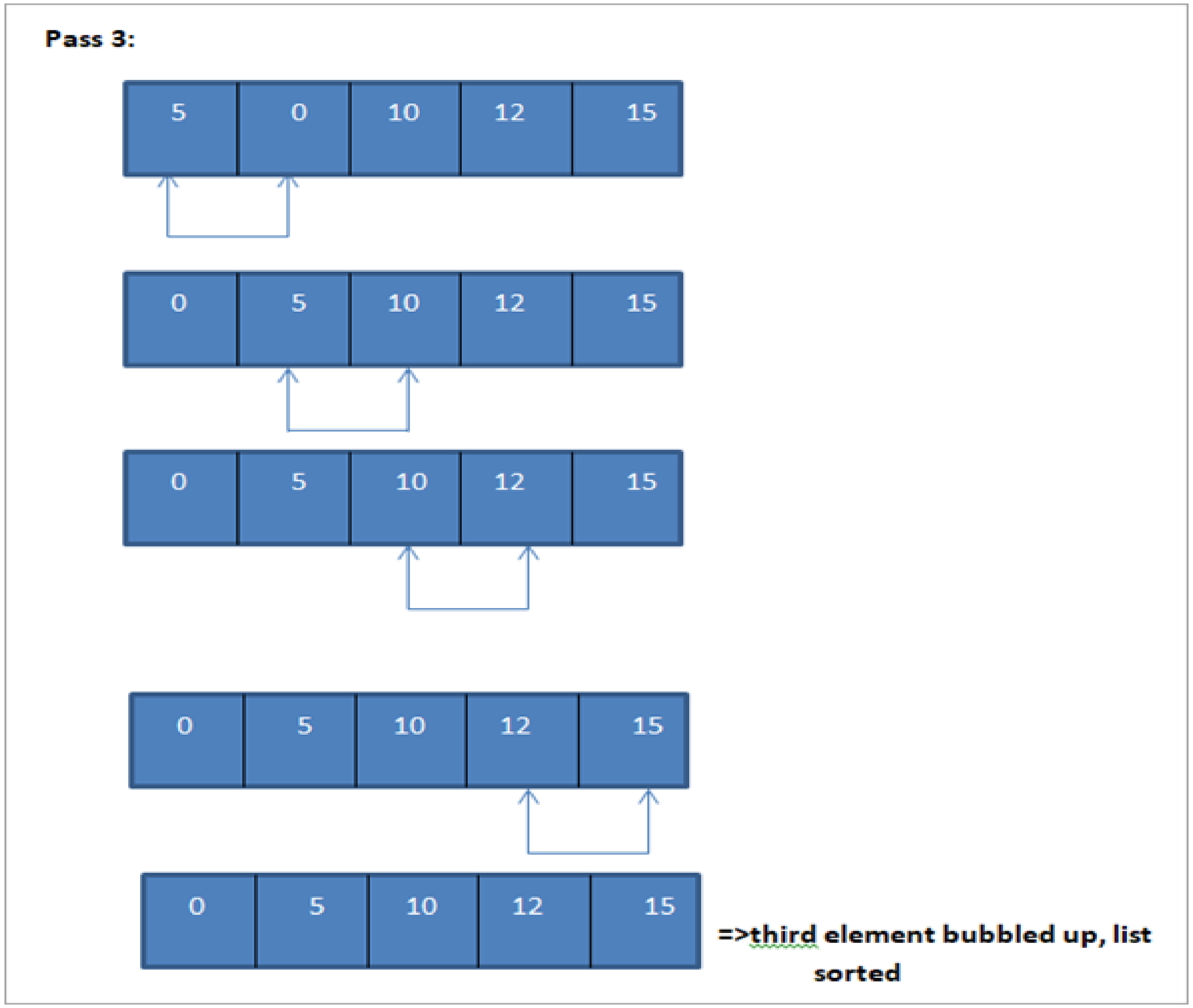
Consider the array elements as: 10,5,15,0,12



Example for Bubble Sort

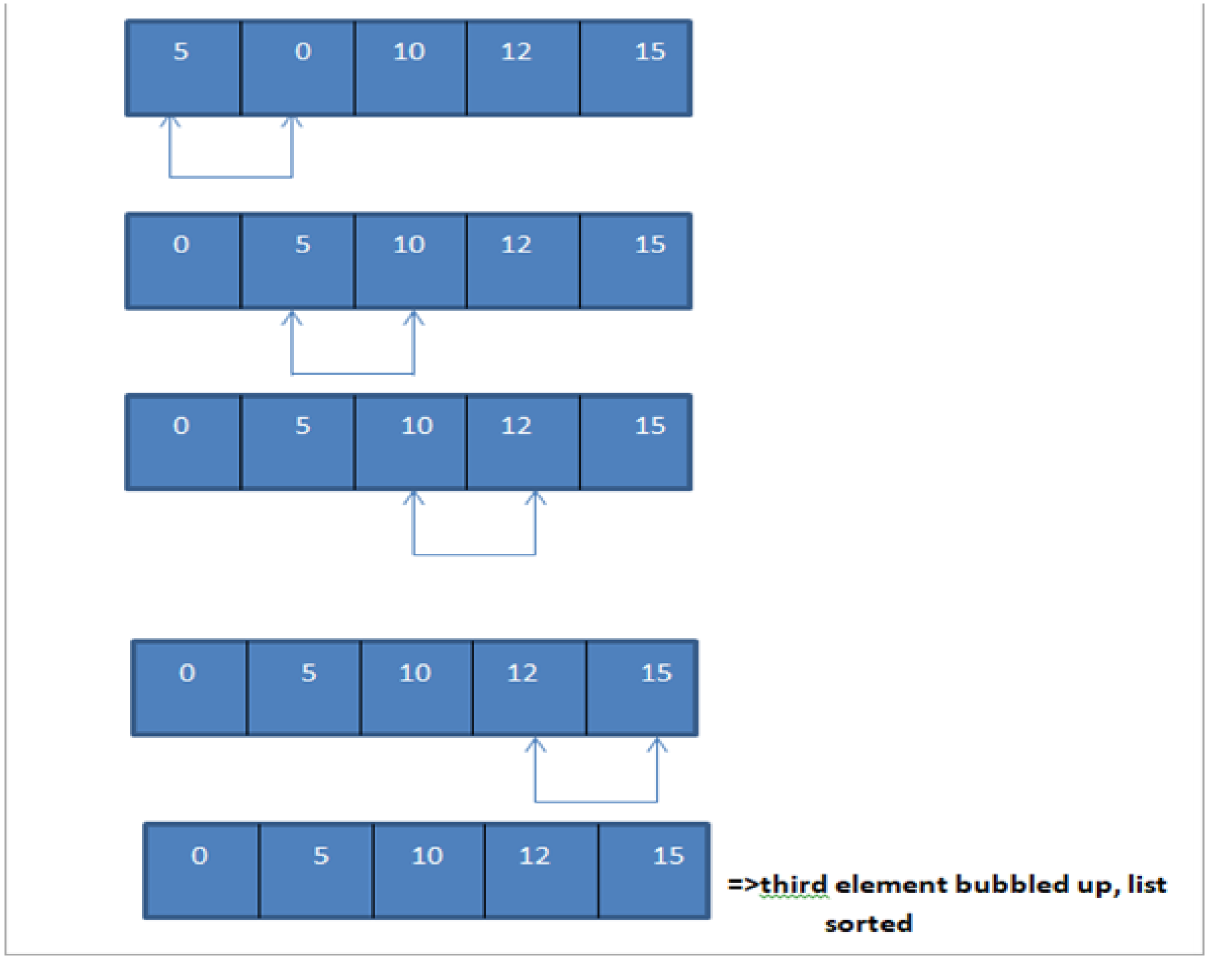


Example for Bubble Sort



Example for Bubble Sort

Pass 4:



Algorithm for Bubble Sort

```
Algorithm Bubble Sort(A, n)
{
    for Pass = 0 to n-1
    {
        for i = 0  to n - step - 1
        {
            if (array[i] > array[i + 1])
            {
                temp = array[i];
                array[i] = array[i + 1];
                array[i + 1] = temp;
            }
        }
    }
}
```

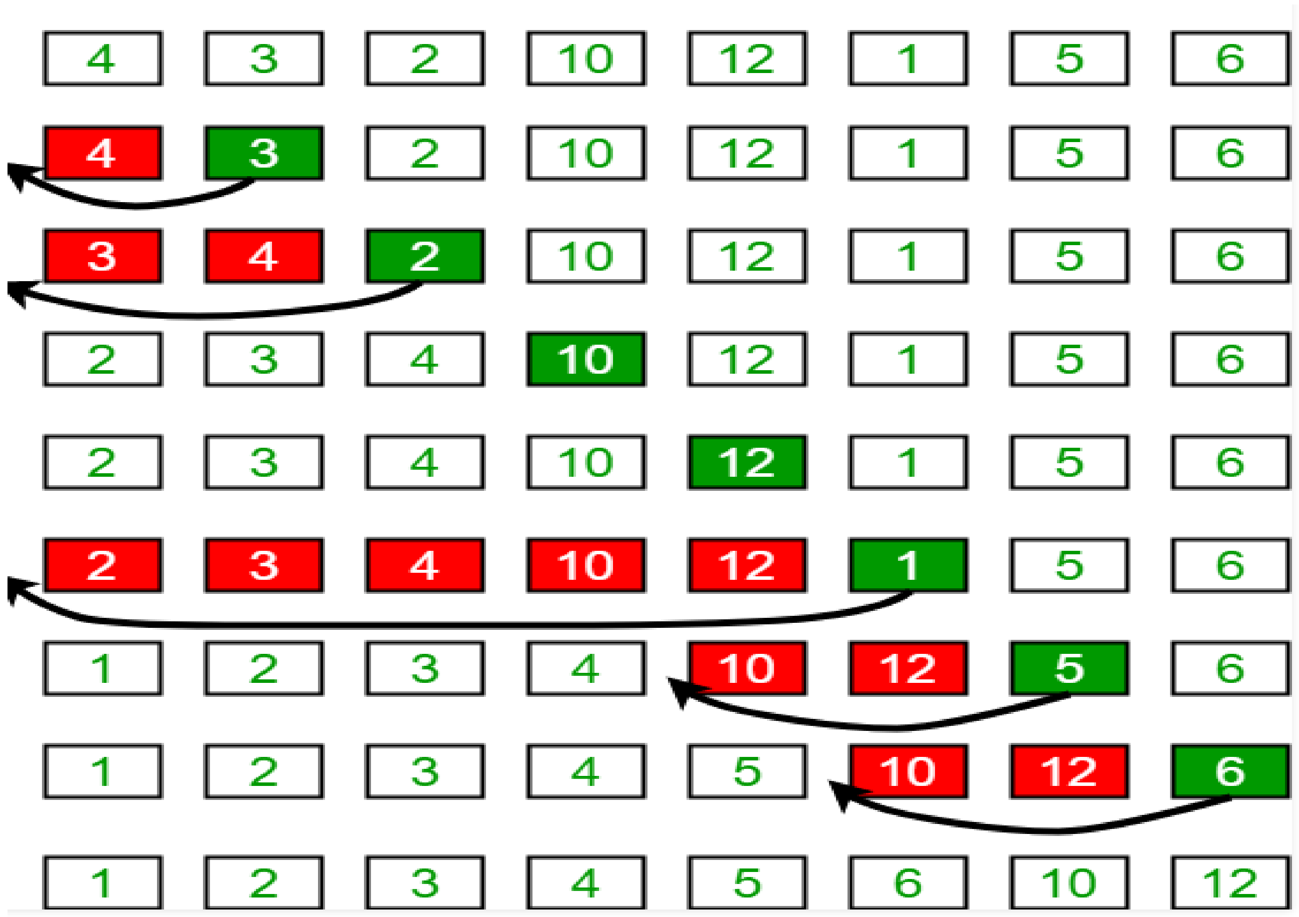
Insertion Sort

Process of Insertion Sort

In insertion sort the elements are compared and inserted to respective index place.

- It starts with comparision of 1st and 0th element in pass 1.
- In pass 2 the second element is compared with the 1st and 0th element.
- Doing so with all the elements in the list appropriate element is inserted by shifting elements on right.

Example for Insertion Sort



Algorithm for Insertion Sort

```
Algorithm Insertion-sort(A, n)
{
    for j=1 to n-1
    {
        key = A[j]
        i=j-1
        while (i>=0 and A[i]>key)
        {
            A[i+1]=A[i]
            i=i-1
        }
        A[i+1]=key
    }
}
```

Categorization of Sorting Algorithms

- **Comparison Vs Non-comparison based sorting**
  - In comparison based sorting, elements of an array are compared with each other to find the sorted array.
    - Bubble sort, Insertion sort , Selection sort , Merge sort , Heap sort , Quick sort
  - In non-comparison based sorting, elements of array are not compared with each other to find the sorted array.
    - Radix sort, Count sort, Bucket sort

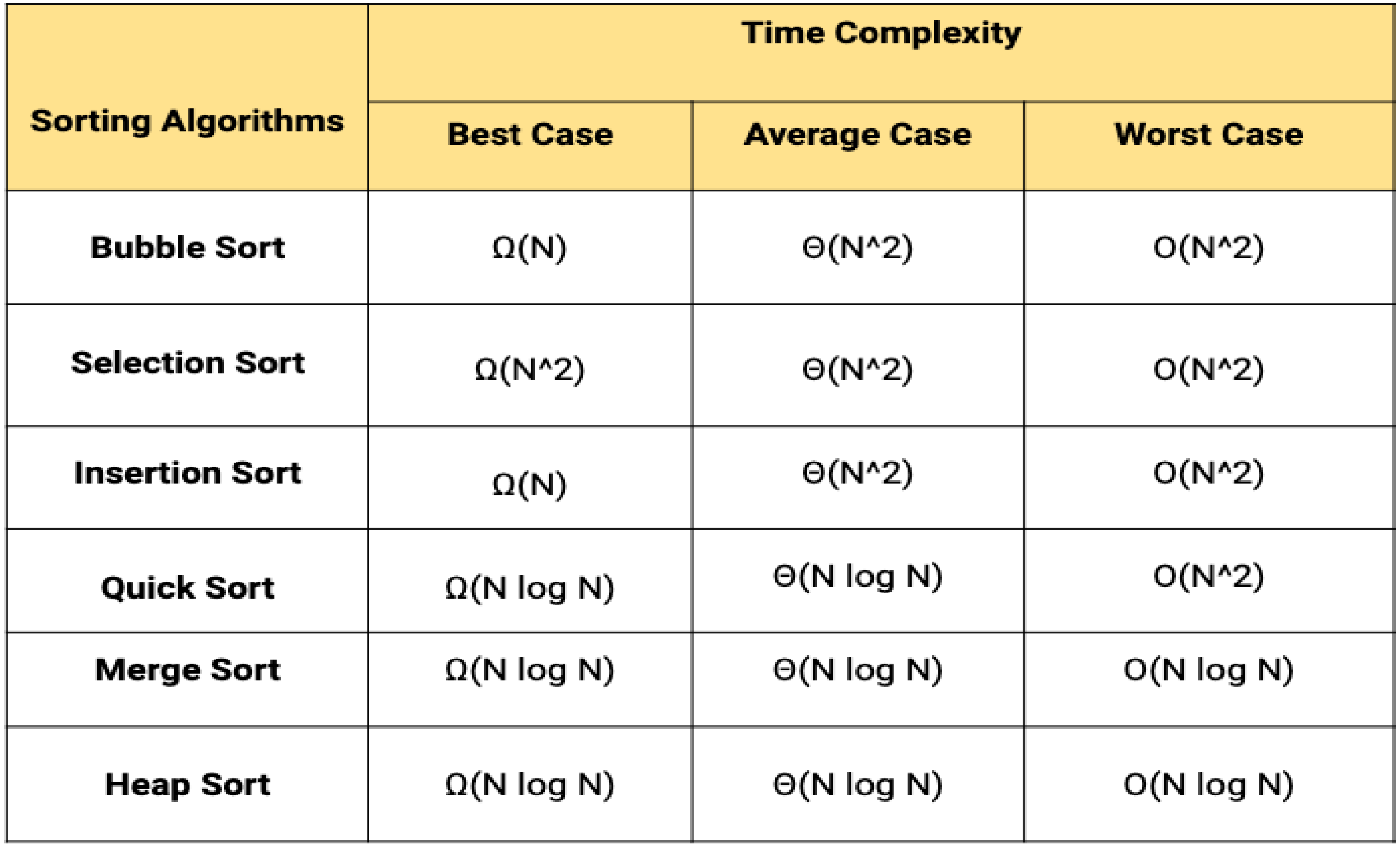
- **In-place Vs Outplace technique**
  - A sorting technique is in-place if it does not use any extra memory to sort the array.
  - Among the comparison based techniques discussed, only merge sort is outplaced technique as it requires an extra array to merge the sorted subarrays.
  - Among the non-comparison based techniques discussed, all are outplaced techniques.

Comparison of Sorting Algorithms

- **Online Vs Offline technique**
  - A sorting technique is considered Online if it can accept new data while the procedure is ongoing i.e. complete data is not required to start the sorting operation.
  - Among the comparison based techniques discussed, only Insertion Sort qualifies for this because of the underlying algorithm it uses i.e. it processes the array (not just elements) from left to right and if new elements are added to the right, it doesn't impact the ongoing operation.

- **Stable Vs Unstable technique**
  - A sorting technique is stable if it does not change the order of elements with the same value.
  - Out of comparison based techniques, bubble sort, insertion sort and merge sort are stable techniques.
  - Selection sort is unstable as it may change the order of elements with the same value. Similarly, quick sort and heap sort are also unstable.

Time Complexity of Sorting Algorithms



Sorting Algorithms	Time Complexity		
	Best Case	Average Case	Worst Case
Bubble Sort	Ω(N)	Θ(N^2)	O(N^2)
Selection Sort	Ω(N^2)	Θ(N^2)	O(N^2)
Insertion Sort	Ω(N)	Θ(N^2)	O(N^2)
Quick Sort	Ω(N log N)	Θ(N log N)	O(N^2)
Merge Sort	Ω(N log N)	Θ(N log N)	O(N log N)
Heap Sort	Ω(N log N)	Θ(N log N)	O(N log N)