Sorting

Sorting is an operation which means arranging a list of data elements in a particular order. **For example**, suppose a list of values given below:

After applying sorting list of values will be appear as shown below:

Sorting Algorithms

incapp

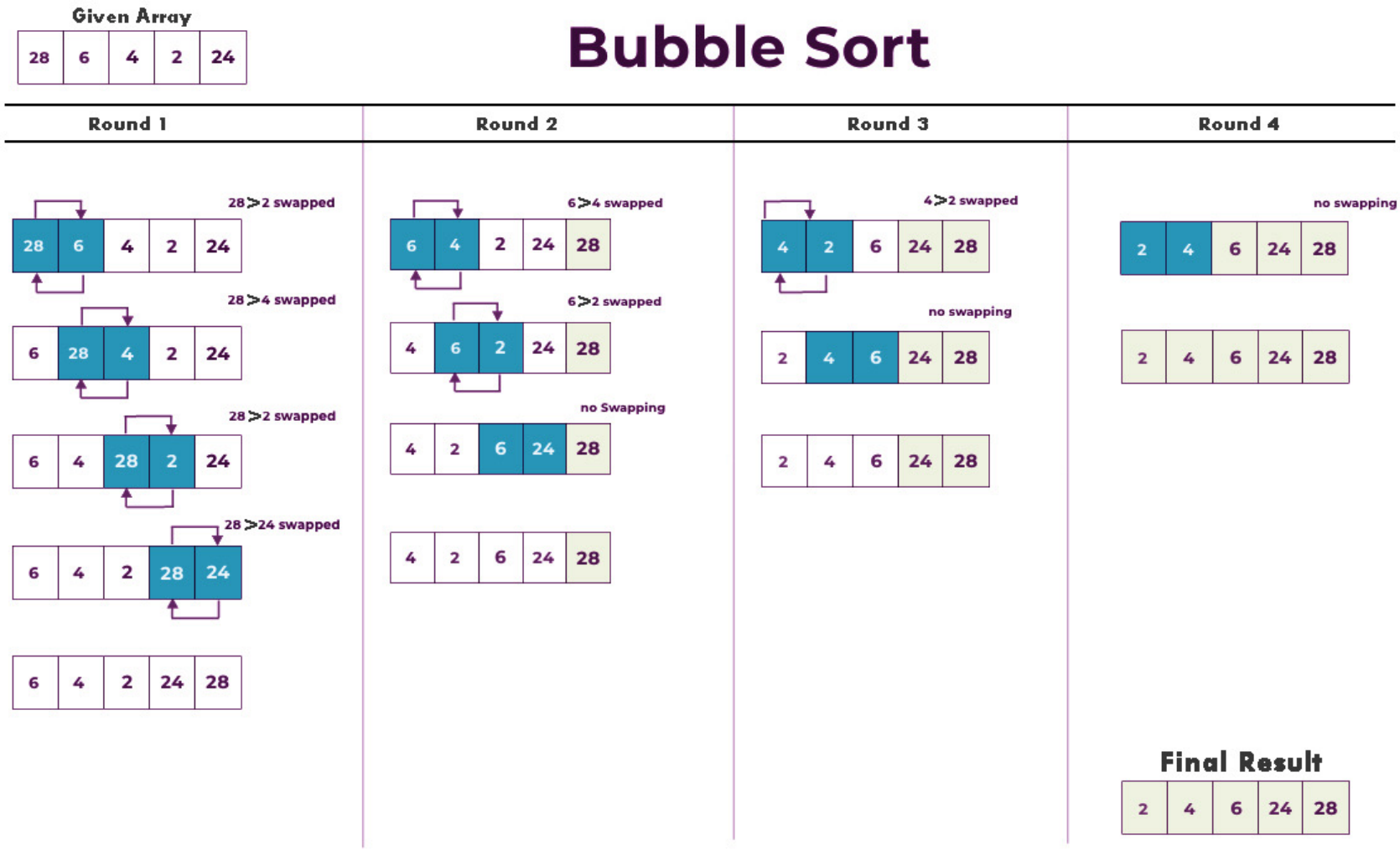
Types of Sorting Algorithms
Bubble Sort
Selection Sort
Insertion Sort
Radix Sort
Counting Sort
Quick Sort
Merge Sort
Heap Sort

Bubble Sort

In this method, each element is compared with its adjacent element. If the first element is large than the second one then the position of the elements are interchanged, otherwise it is not changed. Then next element is compared with its adjacent element and the same process is repeated for all the elements in the array. During the first iteration, the first largest element occupies the last position. During the second iteration, the second largest element occupies the second last position. The same process is repeated until no more elements are left for comparison. Finally the collection is sorted.

Bubble sort gets its name from the fact that data "Bubbles" to the top of the dataset. Bubble sort is alternatively called "**Sinking sort**" for the opposite reason, which is that some elements of data sink to the bottom of the dataset.

Bubble Sort


Given Array
28 6 4 2 24
Bubble Sort
Round 1
28 > 2 swapped
28 > 4 swapped
28 > 2 swapped
28 > 24 swapped
6 4 2 24 28
Round 2
6 > 4 swapped
6 > 2 swapped
no Swapping
4 2 6 24 28
Round 3
4 > 2 swapped
no swapping
2 4 6 24 28
Round 4
no swapping
2 4 6 24 28
Final Result
2 4 6 24 28

Algo for Bubble Sort

Here **arr** is the array, **length** is the number of elements stored in array, **temp** will be used in interchanging values and **i** and **j** are loop counters, then the algorithm is:

1. Repeat for i=0 to length-2
    a. Repeat for j=0 to length-2-i
        i. Check if(arr[j]>arr[j+1], then
            - Set temp=arr[j]
            - Set arr[j]=arr[j+1]
            - Set arr[j+1]=temp
2. Exit

Selection Sort

In this method, the 0^{th} element is compared with all other elements. If 0^{th} element is found to be greater than the compared element then they are interchanged. Thus after the first iteration the smallest element is placed at the 0^{th} position. Similarly in the second iteration second smallest element is placed at 1^{st} position. The same procedure is repeated until the whole array is sorted.

Selection Sort

incapp

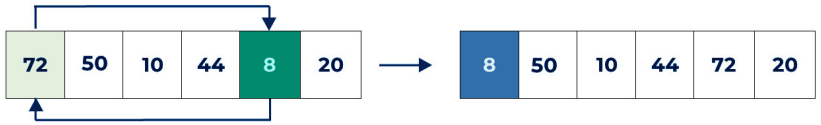
Selection Sort

72	50	10	44	8	20

Initial Array

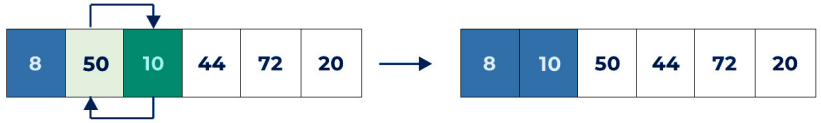
8 is min. was swapped with element at 0th index

- Sorted elements [] = 8
- UnSorted elements [] = 50 10 44 72 20



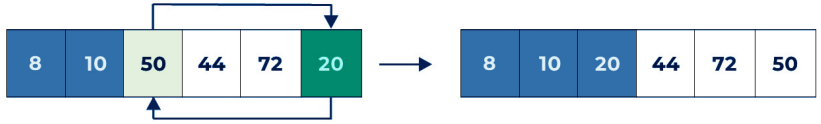
10 is min. was swapped with element at 1st index

- Sorted elements [] = 8 10
- UnSorted elements [] = 50 44 72 20



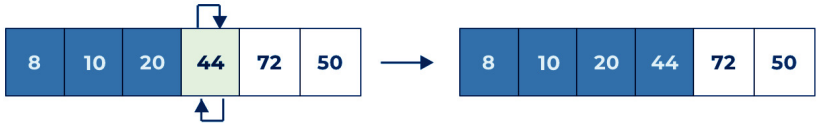
20 is min. was swapped with element at 2nd index

- Sorted elements [] = 8 10 20
- UnSorted elements [] = 44 72 50



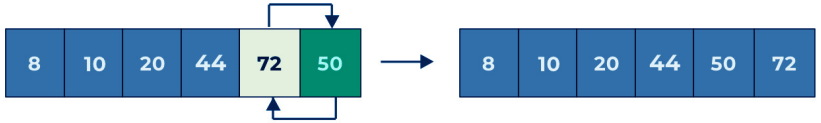
44 is min. was swapped with itself at 3rd index

- Sorted elements [] = 8 10 20 44
- UnSorted elements [] = 72 50



50 is min. was swapped at with element at 4th index

- Sorted elements [] = 8 10 20 44 50 72
- UnSorted elements [] =



Final Result

8	10	20	44	50	72

Algo for Selection Sort

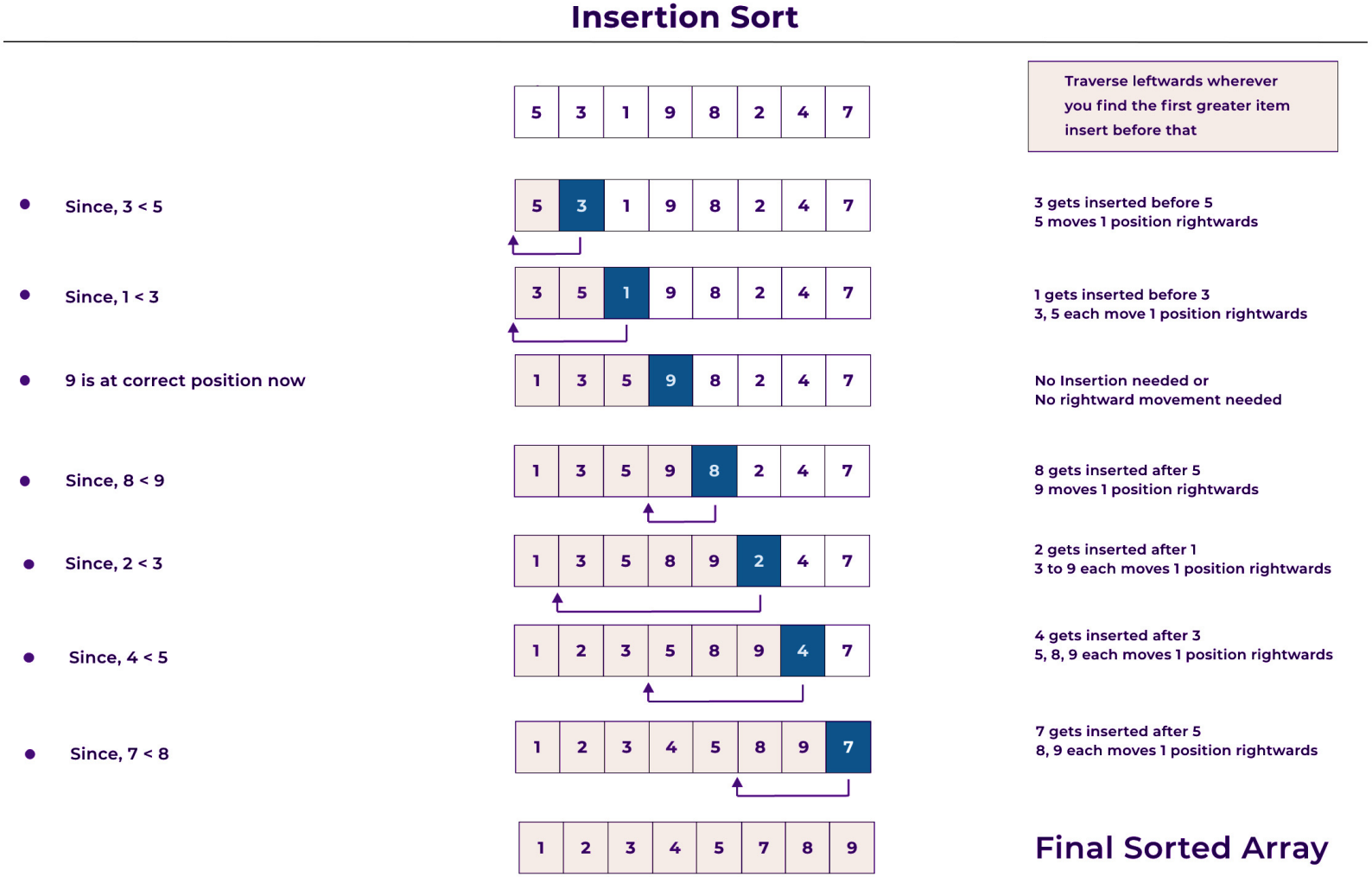
Here **arr** is the array, **length** is the number of elements stored in array, **temp** will be used in interchanging values, **pos** will store the index of minimum value and **i** & **j** are loop counters, then the algorithm is:

1. Repeat for i=0 to length-2
    a. Set temp=arr[i]
    b. Set pos=i;
    c. Repeat for j=i+1 to length-1
        i. Check if arr[j] < temp, then
            - temp=arr[j]
            - pos=j
    d. arr[pos]=arr[i];
    e. arr[i]=temp;
2. Exit

Insertion Sort

- In this method, First, we consider the 0^{th} index as sorted part and the rest is unsorted.
- Then we compare the next element of the array with the first element of the array, if the first element is smaller then the second element then we swap the elements, otherwise we place the element after the first element
- Thus each time we insert element the sorted array becomes larger and unsorted becomes smaller, and the new element is to be compare with all the previous elements of the sorted array and adjust the element just next to its smaller element.
- We repeat this process till all the elements in array become part of the sorted array

Insertion Sort



Algo for Insertion Sort

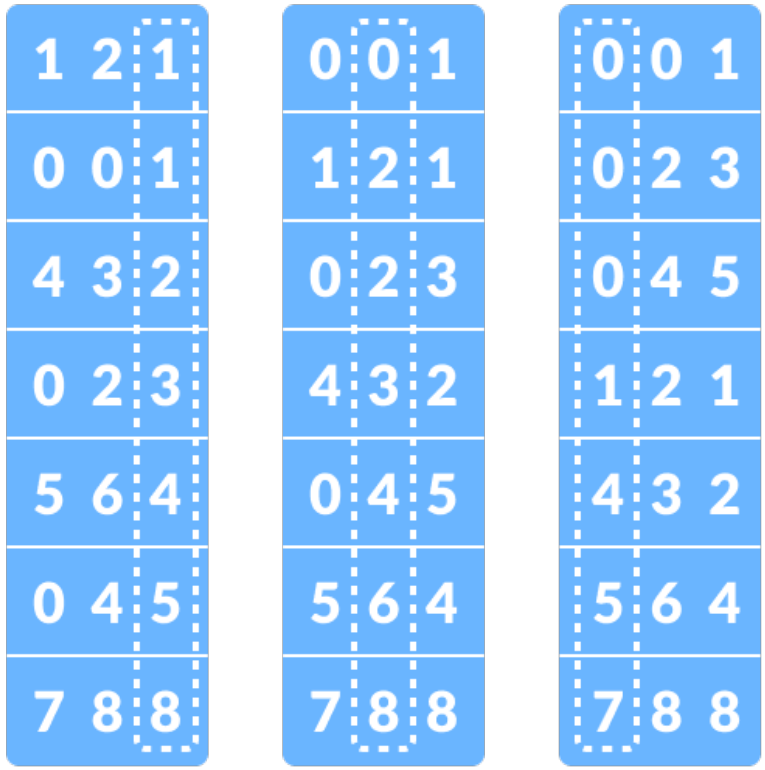
Here **arr** is the array, **length** is the number of elements stored in array, **key** will store each element of array one by one and **i** and **j** are loop counters, then the algorithm is:

1. Repeat for i=1 to length-1
    a. Set key=arr[i]
    b. Set j=i-1
    c. Repeat while j>=0 and arr[j]>key
        - arr[j+1]=arr[j]
        - j=j-1
    d. arr[j+1]=key
2. Exit

Radix Sort

Radix Sort is similar to a register algorithm which we use in day to day work while arranging a list of names, in alphabetical order, in that similar way rad ix sort arranges the given numbers by arranging them in the order of each digit sequentially starting from one's place and moving to ten's or hundred's place depending upon the given data.

Radix Sort



sorting the integers according to units, tens and hundreds place digits

Algo for Radix Sort

Here **arr** and **bucket_count** are the array, **bucket** is 2D array, **length** is the number of elements stored in **arr** array, **div** will store division value, **radix_count** will store the number of digits of largest value, pos will store index of array arr, and **i**, **j** & **k** are loop counters, then the algorithm is:

1. Set div=1
2. Find Largest number
3. Set radix_count =number of digits of Largest number
4. Repeat for i=1 to radix_count
    a. Repeat for j=0 to 9
        i. Set bucket_count[j]=0
    b. Repeat for j=0 to length-1
        i. Set k=(arr[j] / div) % 10
        ii. Set bucket[k][bucket_count[k]]=arr[j]
        iii. Set bucket_count[k]= bucket_count[k] + 1
    c. Set pos=0;
    d. Repeat for j=0 to 9
        - Repeat for k=0 to bucket_count[j]-1
            - arr[pos]=bucket[j] [k]
            - pos=pos+1
    e. div=div*10;
5. Exit