SORTING ALGORITHMS

(1) Bubble Sort

CODE :

```
def bubble_sort(array):

    sorted = False
    counter = 0

    while not sorted :
        sorted = True

        for i in range(len(array)-1-counter):

            if array[i] > array[i+1] :

                array[i], array[i+1] = array[i+1
                                       array[i]
                sorted = False

        counter += 1

    return array
```

*Insertion Sort

- Divide array in 2 parts : sorted

not sorted

take elements one by one and insert

[7, 3, 8, 5, 1]

sorted | not sorted

at first step $i = 3$ $\boxed{3}$

$j = i - 1 = 7$

is array[j] > $\boxed{3}$

↓ if yes, swap

[3, 7, 8, 5, 1]

sorted | not sorted

↓

is array[j] > $\boxed{8}$

No

[3, 7, 8, 5, 1]

sorted | not sorted

↓

is array [j] > 5

↓ Yes

[3, 5, 7, 8, 1]

sorted | not sorted

↓

[1, 3, 5, 7, 8]

Time Complexity = $O(n^2)$

CODE :

```
def insertion_sort(array) :
    for i in range(1, len(array)) :
        j = i - 1
        temp = array[i]
        while j >= 0 and array[j] > temp
            array[j+1] = array[j]
            j -= 1
        array[j+1] = temp
    return array
```

* Selection Sort :

- in each pass we will identify the smallest number and put it towards front.

CODE :

```
def selection_sort(array):
    for i in range(len(array)) :
        smallest = i
```

```
smallest = i
for j in range(i+1, len(nums))
    if nums[j] < nums[i]
        smallest = j
if i != smallest
    nums[i], nums[smallest] = nums[smallest], nums[i]
return nums
```

* Merge Sort

- Divide and conquer algorithm

[7, 3, 8, 5, 1, 9, 5]

[7, 3, 8, 5] , [1, 9, 5]

[7, 3] [8, 5] [1, 9] [5]

[7], [3], [8], [5] [1], [9], [5]

Divide

[3, 7] , [5, 8] [1, 9], [5]

[3, 5, 7, 8] [1, 5, 9]

[1, 3, 5, 5, 7, 8, 9]

Conquer

CODE:

```
def merge_sort(array):
```

```
def merge_sorted_arrays(arr1, arr2):
    #a function to join two sorted array
    i, j = 0, 0
    merged_array = [ ]

    while i < len(arr1) and j < len(arr2):

        if arr1[i] < arr2[j] :

            merged_array.append (arr[i])
            i += 1
        else :
            merged_array. append (arr [j])
            j += 1

    # if any element left in arr1 or arr2
      append that in merged array

    while i < len (arr1) :
        merged_array . append (arr1[i])
        i+ = 1

    while j < len (arr2) :
        merged_array. append (arr2[j])
        j+ = 1

    return merged_array

if len(array) <= 1 :
    return array

mid = len(array) // 2

left_side = merge_sort (array[: mid] )
right_side = merge_sort ( array [mid : ])
```

```
return merge_sorted_arrays(left_side,
                           right_side)
```

* Quick Sort

[14, 22, 12, 18, 19, 11, 8, 13, 9]

pivot i swap j

move j → till greater than pivot

← move j till less than pivot

Now we will swap the both i and j as both have met the required condition

[14, 9, 12, 18, 19, 11, 8, 13, 22]

i swap j

14, 9, 12, 13, 19, 11, 8, 18, 22

i swap j

14, 9, 12, 13, 8, 11, 19, 18, 22

swap j i

Since j > i we stop

(11, 9, 12, 13, 8) 14, (19, 18, 22)
Pivot

Now recursively call the quick sort on both part