**Week -1 :**

**Selection sort:**

Check the minimum in the array and swap it with the first element in the array and continue until last but one element.  
logic :

Start i at the first position and j in the second loop to check the minimum i.e ( arr[j] < arr[min]) and swap the elements.

Code:

from typing import List

def selection(arr : List[int]):

        for i in range(len(arr)-1):

            min = i

            for j in range(i, len(arr)):

                if(arr[j]<arr[min]):

                    min = j

            temp = arr[min]

            arr[min]= arr[i]

            arr[i]= temp

        return arr

brr = [4,7,3,0,1]

print(selection(brr))

**Time complexity : O(n2)**

**Bubble Sort:**

In this sorting, we compare the adjacent elements and check if the second element is greater than the first element, which is if (arr[j] > arr[j+1]) swap. By doing this sorting we would be pushing the highest element at the last.

Logic:

The first loop starts from the last and continue all the way to the first i.e; i: n-1 to 0

The second loop would start from I and check the elements to get swapped i.e; j : 0 to i.

If (arr[j]> arr[j+1) swap.

Code:

def bubblesort(arr):

    for i in range(len(arr)-1,0,-1):

        for j in range(0, i):

            if arr[j]>arr[j+1]:

                arr[j],arr[j+1]=arr[j+1],arr[j]

    return arr

x = [1,5,8,0,3]

print(bubblesort(x))

**Time Complexity: O(n2) : worst; best : O(n) : if the array is already sorted**

**Insertion sort:**

Here in this sorting technique, we check the element and place it in the correct position by swapping with the elements that are greater than this element.

Logic:

For the first loop we continue to the end of the list i.e; i: 0 to n-1

For the second loop, here we take j from j=i; and we check what are the elements which are greater than arr[j] and swap it until we find out the there are no more swaps possible i.e;

j: loop until j>0 and arr[j-1] > arr[j] , inside the loop swap arr[j-1] and arr[j].

once the swap is done, decrement the j by 1 as you compare for the preceding elements.

Code :

def insertion(arr):

    for i in range(0,len(arr)):

        j=i

        while j > 0 and arr[j-1]>arr[j]:

            arr[j-1],arr[j] = arr[j], arr[j-1]

            j=j-1

    return arr

x = [0,3,1,4,5]

print(insertion(x))

**Time Complexity: O(n2) and the best case is O(n) : if the array is sorted.**

**Merge Sort:**

In this sorting technique, we follow divide and Conquer technique.

In the first step, we divide the whole array in to single arrays and then merge these arrays in the sorted order by comparing the left-right ( the lower element will be placed in the left and the higher element is placed on the right) and finally it would give you a sorted array.

Logic:

Follow a recursive call to divide the arrays i.e; merge(arr,i,j)

The next step we conquer the array which is nothing but placing the values by checking the low and high values based on conditions.

Code:

def merge\_sort(arr:[], low : int, high : int)-> None:

    mid = (low+high)//2

    if low>=high:

        return

    merge\_sort(arr,low,mid)

    merge\_sort(arr,mid+1,high)

    merge(arr,low,mid,high)

def merge(arr:[], low:int, mid: int, high:int)->None:

    left=low

    right=mid+1

    temp=[]

    while left<=mid and right<=high:

        if arr[left]<=arr[right]:

            temp.append(arr[left])

            left+=1

        else:

            temp.append(arr[right])

            right+=1

    while left<=mid:

        temp.append(arr[left])

        left+=1

    while right<=high:

        temp.append(arr[right])

        right+=1

    print(temp)

    for i,val in enumerate(temp):

        arr[low + i]=val

if \_\_name\_\_ == "\_\_main\_\_":

    lst =[3,5,1,9,0]

    merge\_sort(lst,0,len(lst)-1)

    print(lst)

Time complexity : O(nlogn)

Space complexit: O(n)

Quick sort:

Here in this algorithm, we use the pivot element as the first element(it can be any element) and by the first iteration we would be placing this element in the correct position in the resultant sorted array.

And for the next iterations, we perform the same for the left half of the array and right half of the array.

Logic:

Start with a pivot element as the first element,

Check for the element which is higher than the pivot and also in the reverse order check for the element which is lower than the pivot.  
once found out these two elements, swap the elements.

After this, swap the pivot element with j (as j would be crossing i)

Code :

def quick\_sort(arr:[], low:int,high:int)->None:

    if low<high:

        index = partition(arr,low,high)

        quick\_sort(arr,low,index-1)

        quick\_sort(arr,index+1,high)

def partition(arr:[], low:int,high:int)-> int:

    pivot= arr[low]

    i,j=low,high

    while i<j:

        while arr[i]<=pivot and i<=high-1: # here i<=high-1 because as we are incrementing i and it should not exceed the upper limit

            i+=1

        while arr[j]>pivot and j>=low+1: # here j>=low+1 because as we are decrementing j and it should not exceed the lower limit

            j-=1

        if i<j:

            arr[i],arr[j]=arr[j],arr[i]

    arr[j],arr[low]=arr[low],arr[j]

    return j

if \_\_name\_\_ == "\_\_main\_\_":

    lst =[4,3,2,1]

    quick\_sort(lst,0,len(lst)-1)

    print(lst)