```python
# Python program for implementation of Selection
# Sort
import sys
A = [64, 25, 12, 22, 11]

# Traverse through all array elements
for i in range(len(A)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    # Swap the found minimum element with
    # the first element
    A[i], A[min_idx] = A[min_idx], A[i]

# Driver code to test above
print ("Sorted array")
for i in range(len(A)):
    print("%d" %A[i]),

#==========================================


# Python program for implementation of Bubble Sort

def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i]),
#==========================================
```

```python
# Python program for implementation of Insertion Sort

# Function to do insertion sort
def insertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):

        key = arr[i]

        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >= 0 and key < arr[j] :
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = key


# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
for i in range(len(arr)):
    print ("% d" % arr[i])

# This code is contributed by Mohit Kumra

#==========================================



# Python3 implementation of QuickSort

# This Function handles sorting part of quick sort
# start and end points to first and last element of
# an array respectively
def partition(start, end, array):

    # Initializing pivot's index to start
    pivot_index = start
    pivot = array[pivot_index]

    # This loop runs till start pointer crosses
    # end pointer, and when it does we swap the
    # pivot with element on end pointer
    while start < end:

        # Increment the start pointer till it finds an
        # element greater than pivot
        while start < len(array) and array[start] <= pivot:
            start += 1
```

```
        # Decrement the end pointer till it finds an
        # element less than pivot
        while array[end] > pivot:
            end -= 1

        # If start and end have not crossed each other,
        # swap the numbers on start and end
        if(start < end):
            array[start], array[end] = array[end], array[start]

    # Swap pivot element with element on end pointer.
    # This puts pivot on its correct sorted place.
    array[end], array[pivot_index] = array[pivot_index], array[end]

    # Returning end pointer to divide the array into 2
    return end

# The main function that implements QuickSort
def quick_sort(start, end, array):

    if (start < end):

        # p is partitioning index, array[p]
        # is at right place
        p = partition(start, end, array)

        # Sort elements before partition
        # and after partition
        quick_sort(start, p - 1, array)
        quick_sort(p + 1, end, array)

# Driver code
array = [ 10, 7, 8, 9, 1, 5 ]
quick_sort(0, len(array) - 1, array)

print(f'Sorted array: {array}')

# This code is contributed by Adnan Aliakbar
#==========================================


# Python program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) > 1:

        # Finding the mid of the array
        mid = len(arr)//2

        # Dividing the array elements
        L = arr[:mid]

        # into 2 halves
        R = arr[mid:]
```

```
        # Sorting the first half
        mergeSort(L)

        # Sorting the second half
        mergeSort(R)

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

# Code to print the list


def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()


# Driver Code
if __name__ == '__main__':
    arr = [12, 11, 13, 5, 6, 7]
    print("Given array is", end="\n")
    printList(arr)
    mergeSort(arr)
    print("Sorted array is: ", end="\n")
    printList(arr)

# This code is contributed by Mayank Khanna
#==========================================



# Python program for implementation of heap Sort
```

```python
# To heapify subtree rooted at index i.
# n is size of heap


def heapify(arr, n, i):
    largest = i  # Initialize largest as root
    l = 2 * i + 1     # left = 2*i + 1
    r = 2 * i + 2     # right = 2*i + 2

    # See if left child of root exists and is
    # greater than root
    if l < n and arr[largest] < arr[l]:
        largest = l

    # See if right child of root exists and is
    # greater than root
    if r < n and arr[largest] < arr[r]:
        largest = r

    # Change root, if needed
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]  # swap

        # Heapify the root.
        heapify(arr, n, largest)

# The main function to sort an array of given size


def heapSort(arr):
    n = len(arr)

    # Build a maxheap.
    for i in range(n//2 - 1, -1, -1):
        heapify(arr, n, i)

    # One by one extract elements
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]  # swap
        heapify(arr, i, 0)


# Driver code
arr = [12, 11, 13, 5, 6, 7]
heapSort(arr)
n = len(arr)
print("Sorted array is")
for i in range(n):
    print("%d" % arr[i]),
# This code is contributed by Mohit Kumra
#=========================================
```

```python
# Python Program for implementation of
# Recursive Bubble sort

class bubbleSort:
    """
    bubbleSort:
        function:
            bubbleSortRecursive : recursive
                function to sort array
            __str__ : format print of array
            __init__ : constructor
                function in python
        variables:
            self.array = contains array
            self.length = length of array
    """

    def __init__(self, array):
        self.array = array
        self.length = len(array)

    def __str__(self):
        return " ".join([str(x)
                         for x in self.array])

    def bubbleSortRecursive(self, n=None):
        if n is None:
            n = self.length

        # Base case
        if n == 1:
            return

        # One pass of bubble sort. After
        # this pass, the largest element
        # is moved (or bubbled) to end.
        for i in range(n - 1):
            if self.array[i] > self.array[i + 1]:
                self.array[i], self.array[i +
                1] = self.array[i + 1], self.array[i]

        # Largest element is fixed,
        # recur for remaining array
        self.bubbleSortRecursive(n - 1)

# Driver Code
def main():
    array = [64, 34, 25, 12, 22, 11, 90]

    # Creating object for class
    sort = bubbleSort(array)

    # Sorting array
    sort.bubbleSortRecursive()
```

```
    print("Sorted array :\n", sort)


if __name__ == "__main__":
    main()

# Code contributed by Mohit Gupta_OMG,
# improved by itsvinayak
#============================================


# Recursive Python program for insertion sort
# Recursive function to sort an array using insertion sort

def insertionSortRecursive(arr,n):
    # base case
    if n<=1:
        return

    # Sort first n-1 elements
    insertionSortRecursive(arr,n-1)
    '''Insert last element at its correct position
        in sorted array.'''
    last = arr[n-1]
    j = n-2

    # Move elements of arr[0..i-1], that are
    # greater than key, to one position ahead
    # of their current position
    while (j>=0 and arr[j]>last):
        arr[j+1] = arr[j]
        j = j-1

    arr[j+1]=last

# A utility function to print an array of size n
def printArray(arr,n):
    for i in range(n):
        print (arr[i]),

# Driver program to test insertion sort
arr = [12,11,13,5,6]
n = len(arr)
insertionSortRecursive(arr, n)
printArray(arr, n)

# Contributed by Harsh Valecha


#==========================================================

# Recursive Python Program for merge sort

def merge(left, right):
```

```
    if not len(left) or not len(right):
        return left or right

    result = []
    i, j = 0, 0
    while (len(result) < len(left) + len(right)):
        if left[i] < right[j]:
            result.append(left[i])
            i+= 1
        else:
            result.append(right[j])
            j+= 1
        if i == len(left) or j == len(right):
            result.extend(left[i:] or right[j:])
            break

    return result

def mergesort(list):
    if len(list) < 2:
        return list

    middle = int(len(list)/2)
    left = mergesort(list[:middle])
    right = mergesort(list[middle:])

    return merge(left, right)

seq = [12, 11, 13, 5, 6, 7]
print("Given array is")
print(seq);
print("\n")
print("Sorted array is")
print(mergesort(seq))

# Code Contributed by Mohit Gupta_OMG
#=============================================================




# A typical recursive Python
# implementation of QuickSort

# Function takes last element as pivot,
# places the pivot element at its correct
# position in sorted array, and places all
# smaller (smaller than pivot) to left of
# pivot and all greater elements to right
# of pivot
def partition(arr, low, high):
    i = (low - 1)        # index of smaller element
    pivot = arr[high]    # pivot
```

```
    for j in range(low, high):

        # If current element is smaller
        # than or equal to pivot
        if arr[j] <= pivot:

            # increment index of
            # smaller element
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return (i + 1)

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index

# Function to do Quick sort
def quickSort(arr, low, high):
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr, low, high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi + 1, high)

# Driver Code
if __name__ == '__main__' :

    arr = [4, 2, 6, 9, 2]
    n = len(arr)

    # Calling quickSort function
    quickSort(arr, 0, n - 1)

    for i in range(n):
        print(arr[i], end = " ")

#============================================================




# Python program for counting sort

# The main function that sort the given string arr[] in
# alphabetical order
def countSort(arr):
```

```
    # The output character array that will have sorted arr
    output = [0 for i in range(len(arr))]

    # Create a count array to store count of individual
    # characters and initialize count array as 0
    count = [0 for i in range(256)]

    # For storing the resulting answer since the
    # string is immutable
    ans = ["" for _ in arr]

    # Store count of each character
    for i in arr:
        count[ord(i)] += 1

    # Change count[i] so that count[i] now contains actual
    # position of this character in output array
    for i in range(256):
        count[i] += count[i-1]

    # Build the output character array
    for i in range(len(arr)):
        output[count[ord(arr[i])]-1] = arr[i]
        count[ord(arr[i])] -= 1

    # Copy the output array to arr, so that arr now
    # contains sorted characters
    for i in range(len(arr)):
        ans[i] = output[i]
    return ans

# Driver program to test above function
arr = "geeksforgeeks"
ans = countSort(arr)
print("Sorted character array is % s" %("".join(ans)))

# This code is contributed by Nikhil Kumar Singh
```