1. **Python Program for Recursive Insertion Sort**

# 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],end=" ")

# Driver program to test insertion sort

arr = [12,11,13,5,6]

n = len(arr)

insertionSortRecursive(arr, n)

printArray(arr, n)

**Output:** 5 6 11 12 13

1. **Python Program for QuickSort**

# Python program for implementation of Quicksort Sort

# This implementation utilizes pivot as the last element in the nums list

# It has a pointer to keep track of the elements smaller than the pivot

# At the very end of partition() function, the pointer is swapped with the pivot

# to come up with a "sorted" nums relative to the pivot

# Function to find the partition position

def partition(array, low, high):

  # choose the rightmost element as pivot

  pivot = array[high]

  # pointer for greater element

  i = low - 1

  # traverse through all elements

  # compare each element with pivot

  for j in range(low, high):

    if array[j] <= pivot:

      # If element smaller than pivot is found

      # swap it with the greater element pointed by i

      i = i + 1

      # Swapping element at i with element at j

      (array[i], array[j]) = (array[j], array[i])

  # Swap the pivot element with the greater element specified by i

  (array[i + 1], array[high]) = (array[high], array[i + 1])

  # Return the position from where partition is done

  return i + 1

# function to perform quicksort

def quickSort(array, low, high):

  if low < high:

    # Find pivot element such that

    # element smaller than pivot are on the left

    # element greater than pivot are on the right

    pi = partition(array, low, high)

    # Recursive call on the left of pivot

    quickSort(array, low, pi - 1)

    # Recursive call on the right of pivot

    quickSort(array, pi + 1, high)

data = [1, 7, 4, 1, 10, 9, -2]

print("Unsorted Array")

print(data)

size = len(data)

quickSort(data, 0, size - 1)

print('Sorted Array in Ascending Order:')

print(data)

**Output:** Unsorted Array

[1, 7, 4, 1, 10, 9, -2]

Sorted Array in Ascending Order:

[-2, 1, 1, 4, 7, 9, 10]

1. **Python Program for Iterative Quick Sort**

# Python program for implementation of Quicksort

# This function is same in both iterative and recursive

def partition(arr,l,h):

  i = ( l - 1 )

  x = arr[h]

  for j in range(l , h):

    if arr[j] <= x:

      # increment index of smaller element

      i = i+1

      arr[i],arr[j] = arr[j],arr[i]

  arr[i+1],arr[h] = arr[h],arr[i+1]

  return (i+1)

# Function to do Quick sort

# arr[] --> Array to be sorted,

# l --> Starting index,

# h --> Ending index

def quickSortIterative(arr,l,h):

  # Create an auxiliary stack

  size = h - l + 1

  stack = [0] \* (size)

  # initialize top of stack

  top = -1

  # push initial values of l and h to stack

  top = top + 1

  stack[top] = l

  top = top + 1

  stack[top] = h

  # Keep popping from stack while is not empty

  while top >= 0:

    # Pop h and l

    h = stack[top]

    top = top - 1

    l = stack[top]

    top = top - 1

    # Set pivot element at its correct position in

    # sorted array

    p = partition( arr, l, h )

    # If there are elements on left side of pivot,

    # then push left side to stack

    if p-1 > l:

      top = top + 1

      stack[top] = l

      top = top + 1

      stack[top] = p - 1

    # If there are elements on right side of pivot,

    # then push right side to stack

    if p+1 < h:

      top = top + 1

      stack[top] = p + 1

      top = top + 1

      stack[top] = h

# Driver code to test above

arr = [4, 3, 5, 2, 1, 3, 2, 3]

n = len(arr)

quickSortIterative(arr, 0, n-1)

print ("Sorted array is:")

for i in range(n):

  print ("%d" %arr[i])

**Output:** Sorted array is:

1

2

2

3

3

3

4

5

1. **Python Program for Selection Sort**

# Selection sort in Python

# time complexity O(n\*n)

#sorting by finding min\_index

def selectionSort(array, size):

  for ind in range(size):

    min\_index = ind

    for j in range(ind + 1, size):

      # select the minimum element in every iteration

      if array[j] < array[min\_index]:

        min\_index = j

    # swapping the elements to sort the array

    (array[ind], array[min\_index]) = (array[min\_index], array[ind])

arr = [-2, 45, 0, 11, -9,88,-97,-202,747]

size = len(arr)

selectionSort(arr, size)

print('The array after sorting in Ascending Order by selection sort is:')

print(arr)

**Output:** The array after sorting in Ascending Order by selection sort is:

[-202, -97, -9, -2, 0, 11, 45, 88, 747]

1. **Python Program for Bubble Sort**

# Python program for implementation of Bubble Sort

def bubbleSort(arr):

  n = len(arr)

  # optimize code, so if the array is already sorted, it doesn't need

  # to go through the entire process

  swapped = False

  # Traverse through all array elements

  for i in range(n-1):

    # range(n) also work but outer loop will

    # repeat one time more than needed.

    # 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]:

        swapped = True

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

    if not swapped:

      # if we haven't needed to make a single swap, we

      # can just exit the main loop.

      return

# 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], end=" ")

**Output:** Sorted array is:

11 12 22 25 34 64 90

1. **Python Program for Merge Sort**

# Python program for implementation of MergeSort

# Merges two subarrays of arr[].

# First subarray is arr[l..m]

# Second subarray is arr[m+1..r]

def merge(arr, l, m, r):

  n1 = m - l + 1

  n2 = r - m

  # create temp arrays

  L = [0] \* (n1)

  R = [0] \* (n2)

  # Copy data to temp arrays L[] and R[]

  for i in range(0, n1):

    L[i] = arr[l + i]

  for j in range(0, n2):

    R[j] = arr[m + 1 + j]

  # Merge the temp arrays back into arr[l..r]

  i = 0  # Initial index of first subarray

  j = 0  # Initial index of second subarray

  k = l  # Initial index of merged subarray

  while i < n1 and j < n2:

    if L[i] <= R[j]:

      arr[k] = L[i]

      i += 1

    else:

      arr[k] = R[j]

      j += 1

    k += 1

  # Copy the remaining elements of L[], if there

  # are any

  while i < n1:

    arr[k] = L[i]

    i += 1

    k += 1

  # Copy the remaining elements of R[], if there

  # are any

  while j < n2:

    arr[k] = R[j]

    j += 1

    k += 1

# l is for left index and r is right index of the

# sub-array of arr to be sorted

def mergeSort(arr, l, r):

  if l < r:

    # Same as (l+r)//2, but avoids overflow for

    # large l and h

    m = l+(r-l)//2

    # Sort first and second halves

    mergeSort(arr, l, m)

    mergeSort(arr, m+1, r)

    merge(arr, l, m, r)

# Driver code to test above

arr = [12, 11, 13, 5, 6, 7]

n = len(arr)

print("Given array is")

for i in range(n):

  print("%d" % arr[i],end=" ")

mergeSort(arr, 0, n-1)

print("\n\nSorted array is")

for i in range(n):

  print("%d" % arr[i],end=" ")

**Output:** Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

1. **Python Program for Iterative Merge Sort**

# 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))

**Output:** Given array is

[12, 11, 13, 5, 6, 7]

Sorted array is

[5, 6, 7, 11, 12, 13]

1. **Python Program for Heap Sort**

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

# Python program for implementation of heap Sort

# 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[i] < 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.

# Since last parent will be at ((n//2)-1) we can start at that location.

  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 to test above

arr = [12, 11, 13, 5, 6, 7, ]

heapSort(arr)

n = len(arr)

print('Sorted array is')

for i in range(n):

  print(arr[i])

**Output:** Sorted array is

5

6

7

11

12

13

1. **Python Program for Counting Sort**

# 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(256)]

  # 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)))

**Output:** Sorted character array is eeeefggkkorss

1. **Python Program for ShellSort**

# Python program for implementation of Shell Sort

def shellSort(arr):

  # Start with a big gap, then reduce the gap

  n = len(arr)

  gap = n/2

  # Do a gapped insertion sort for this gap size.

  # The first gap elements a[0..gap-1] are already in gapped

  # order keep adding one more element until the entire array

  # is gap sorted

  while gap > 0:

    for i in range(gap,n):

      # add a[i] to the elements that have been gap sorted

      # save a[i] in temp and make a hole at position i

      temp = arr[i]

      # shift earlier gap-sorted elements up until the correct

      # location for a[i] is found

      j = i

      while j >= gap and arr[j-gap] >temp:

        arr[j] = arr[j-gap]

        j -= gap

      # put temp (the original a[i]) in its correct location

      arr[j] = temp

    gap /= 2

# Driver code to test above

arr = [ 12, 34, 54, 2, 3]

n = len(arr)

print ("Array before sorting:")

for i in range(n):

  print(arr[i]),

shellSort(arr)

print ("\nArray after sorting:")

for i in range(n):

  print(arr[i])

**Output:** Array before sorting:

12

34

54

2

3