1. **Python Program for Topological Sorting**

#Python program to print topological sorting of a DAG

from collections import defaultdict

#Class to represent a graph

class Graph:

  def \_\_init\_\_(self,vertices):

    self.graph = defaultdict(list) #dictionary containing adjacency List

    self.V = vertices #No. of vertices

  # function to add an edge to graph

  def addEdge(self,u,v):

    self.graph[u].append(v)

  # A recursive function used by topologicalSort

  def topologicalSortUtil(self,v,visited,stack):

    # Mark the current node as visited.

    visited[v] = True

    # Recur for all the vertices adjacent to this vertex

    for i in self.graph[v]:

      if visited[i] == False:

        self.topologicalSortUtil(i,visited,stack)

    # Push current vertex to stack which stores result

    stack.insert(0,v)

  # The function to do Topological Sort. It uses recursive

  # topologicalSortUtil()

  def topologicalSort(self):

    # Mark all the vertices as not visited

    visited = [False]\*self.V

    stack =[]

    # Call the recursive helper function to store Topological

    # Sort starting from all vertices one by one

    for i in range(self.V):

      if visited[i] == False:

        self.topologicalSortUtil(i,visited,stack)

    # Print contents of stack

    print (stack)

g= Graph(6)

g.addEdge(5, 2);

g.addEdge(5, 0);

g.addEdge(4, 0);

g.addEdge(4, 1);

g.addEdge(2, 3);

g.addEdge(3, 1);

print ("Following is a Topological Sort of the given graph")

g.topologicalSort()

**Output:** Following is a Topological Sort of the given graph

[5, 4, 2, 3, 1, 0]

1. **Python Program for Radix Sort**

# Python program for implementation of Radix Sort

# A function to do counting sort of arr[] according to

# the digit represented by exp.

def countingSort(arr, exp1):

  n = len(arr)

  # The output array elements that will have sorted arr

  output = [0] \* (n)

  # initialize count array as 0

  count = [0] \* (10)

  # Store count of occurrences in count[]

  for i in range(0, n):

    index = (arr[i]/exp1)

    count[int((index)%10)] += 1

  # Change count[i] so that count[i] now contains actual

  # position of this digit in output array

  for i in range(1,10):

    count[i] += count[i-1]

  # Build the output array

  i = n-1

  while i>=0:

    index = (arr[i]/exp1)

    output[ count[ int((index)%10) ] - 1] = arr[i]

    count[int((index)%10)] -= 1

    i -= 1

  # Copying the output array to arr[],

  # so that arr now contains sorted numbers

  i = 0

  for i in range(0,len(arr)):

    arr[i] = output[i]

# Method to do Radix Sort

def radixSort(arr):

  # Find the maximum number to know number of digits

  max1 = max(arr)

  # Do counting sort for every digit. Note that instead

  # of passing digit number, exp is passed. exp is 10^i

  # where i is current digit number

  exp = 1

  while max1/exp > 0:

    countingSort(arr,exp)

    exp \*= 10

# Driver code to test above

arr = [ 170, 45, 75, 90, 802, 24, 2, 66]

radixSort(arr)

for i in range(len(arr)):

  print(arr[i],end=" ")

**Output:** 2 24 45 66 75 90 170 802

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

# Python Program implementation

# of binary insertion sort

def binary\_search(arr, val, start, end):

  # we need to distinguish whether we should insert

  # before or after the left boundary.

  # imagine [0] is the last step of the binary search

  # and we need to decide where to insert -1

  if start == end:

    if arr[start] > val:

      return start

    else:

      return start+1

  # this occurs if we are moving beyond left\'s boundary

  # meaning the left boundary is the least position to

  # find a number greater than val

  if start > end:

    return start

  mid = (start+end)/2

  if arr[mid] < val:

    return binary\_search(arr, val, mid+1, end)

  elif arr[mid] > val:

    return binary\_search(arr, val, start, mid-1)

  else:

    return mid

def insertion\_sort(arr):

  for i in range(1, len(arr)):

    val = arr[i]

    j = binary\_search(arr, val, 0, i-1)

    arr = arr[:j] + [val] + arr[j:i] + arr[i+1:]

    return arr

print("Sorted array:", insertion\_sort([37, 23, 0, 17, 12, 72, 31,46, 100, 88, 54]))

**Output:** Sorted array: [23, 37, 0, 17, 12, 72, 31, 46, 100, 88, 54]

1. **Python Program for Bitonic Sort**

# Python program for Bitonic Sort. Note that this program

# works only when size of input is a power of 2.

# The parameter dir indicates the sorting direction, ASCENDING

# or DESCENDING; if (a[i] > a[j]) agrees with the direction,

# then a[i] and a[j] are interchanged.\*/

def compAndSwap(a, i, j, dire):

  if (dire == 1 and a[i] > a[j]) or (dire == 0 and a[i] > a[j]):

    a[i], a[j] = a[j], a[i]

# It recursively sorts a bitonic sequence in ascending order,

# if dir = 1, and in descending order otherwise (means dir=0).

# The sequence to be sorted starts at index position low,

# the parameter cnt is the number of elements to be sorted.

def bitonicMerge(a, low, cnt, dire):

  if cnt > 1:

    k = cnt//2

    for i in range(low, low+k):

      compAndSwap(a, i, i+k, dire)

    bitonicMerge(a, low, k, dire)

    bitonicMerge(a, low+k, k, dire)

# This function first produces a bitonic sequence by recursively

# sorting its two halves in opposite sorting orders, and then

# calls bitonicMerge to make them in the same order

def bitonicSort(a, low, cnt, dire):

  if cnt > 1:

    k = cnt//2

    bitonicSort(a, low, k, 1)

    bitonicSort(a, low+k, k, 0)

    bitonicMerge(a, low, cnt, dire)

# Caller of bitonicSort for sorting the entire array of length N

# in ASCENDING order

def sort(a, N, up):

  bitonicSort(a, 0, N, up)

# Driver code to test above

a = [3, 7, 4, 8, 6, 2, 1, 5]

n = len(a)

up = 1

sort(a, n, up)

print("\n\nSorted array is")

for i in range(n):

  print("%d" % a[i], end=" ")

**Output:** Sorted array is

1 5 2 6 3 7 4 8

1. **Python Program for Comb Sort**

# Python program for implementation of CombSort

# To find next gap from current

def getNextGap(gap):

  # Shrink gap by Shrink factor

  gap = (gap \* 10)/13

  if gap & lt == 1:

    return 1

  return gap

# Function to sort arr[] using Comb Sort

def combSort(arr):

  n = len(arr)

  # Initialize gap

  gap = n

  # Initialize swapped as true to make sure that

  # loop runs

  swapped = True

  # Keep running while gap is more than 1 and last

  # iteration caused a swap

  while gap != 1 or swapped == 1:

    # Find next gap

    gap = getNextGap(gap)

    # Initialize swapped as false so that we can

    # check if swap happened or not

    swapped = False

    # Compare all elements with current gap

    for i in range(0, n-gap):

      if arr[i] & gt

      arr[i + gap]:

        arr[i], arr[i + gap] = arr[i + gap], arr[i]

        swapped = True

# Driver code to test above

arr = [8, 4, 1, 3, -44, 23, -6, 28, 0]

combSort(arr)

print ("& quot Sorted array: & quot")

for i in range(len(arr)):

  print(arr[i])

1. **Python Program for Pigeonhole Sort**

# Python program to implement Pigeonhole Sort \*/

# source code : "https://en.wikibooks.org/wiki/

# Algorithm\_Implementation/Sorting/Pigeonhole\_sort"

def pigeonhole\_sort(a):

  # size of range of values in the list

  # (ie, number of pigeonholes we need)

  my\_min = min(a)

  my\_max = max(a)

  size = my\_max - my\_min + 1

  # our list of pigeonholes

  holes = [0] \* size

  # Populate the pigeonholes.

  for x in a:

    assert type(x) is int, "integers only please"

    holes[x - my\_min] += 1

  # Put the elements back into the array in order.

  i = 0

  for count in range(size):

    while holes[count] > 0:

      holes[count] -= 1

      a[i] = count + my\_min

      i += 1

a = [8, 3, 2, 7, 4, 6, 8]

print("Sorted order is : ", end =" ")

pigeonhole\_sort(a)

for i in range(0, len(a)):

  print(a[i], end =" ")

**Output:** Sorted order is : 2 3 4 6 7 8 8

1. **Python Program for Cocktail Sort**

# Python program for implementation of Cocktail Sort

def cocktailSort(a):

  n = len(a)

  swapped = True

  start = 0

  end = n-1

  while (swapped==True):

    # reset the swapped flag on entering the loop,

    # because it might be true from a previous

    # iteration.

    swapped = False

    # loop from left to right same as the bubble

    # sort

    for i in range (start, end):

      if (a[i] > a[i+1]) :

        a[i], a[i+1]= a[i+1], a[i]

        swapped=True

    # if nothing moved, then array is sorted.

    if (swapped==False):

      break

    # otherwise, reset the swapped flag so that it

    # can be used in the next stage

    swapped = False

    # move the end point back by one, because

    # item at the end is in its rightful spot

    end = end-1

    # from right to left, doing the same

    # comparison as in the previous stage

    for i in range(end-1, start-1,-1):

      if (a[i] > a[i+1]):

        a[i], a[i+1] = a[i+1], a[i]

        swapped = True

    # increase the starting point, because

    # the last stage would have moved the next

    # smallest number to its rightful spot.

    start = start+1

# Driver code to test above

a = [5, 1, 4, 2, 8, 0, 2]

cocktailSort(a)

print("Sorted array is:")

for i in range(len(a)):

  print ("%d" %a[i])

**output:** Sorted array is:

0

1

2

2

4

5

8

1. **Python Program for Gnome Sort**

# Python program to implement Gnome Sort

# A function to sort the given list using Gnome sort

def gnomeSort( arr, n):

  index = 0

  while index < n:

    if index == 0:

      index = index + 1

    if arr[index] >= arr[index - 1]:

      index = index + 1

    else:

      arr[index], arr[index-1] = arr[index-1], arr[index]

      index = index - 1

  return arr

# Driver Code

arr = [ 34, 2, 10, -9]

n = len(arr)

arr = gnomeSort(arr, n)

print ("Sorted sequence after applying Gnome Sort :", )

for i in arr:

  print (i)

**Output:** Sorted sequence after applying Gnome Sort :

-9

2

10

34

1. **Python Program for Odd-Even Sort / Brick Sort**

# Python Program to implement

# Odd-Even / Brick Sort

def oddEvenSort(arr, n):

  # Initially array is unsorted

  isSorted = 0

  while isSorted == 0:

    isSorted = 1

    temp = 0

    for i in range(1, n-1, 2):

      if arr[i] > arr[i+1]:

        arr[i], arr[i+1] = arr[i+1], arr[i]

        isSorted = 0

    for i in range(0, n-1, 2):

      if arr[i] > arr[i+1]:

        arr[i], arr[i+1] = arr[i+1], arr[i]

        isSorted = 0

  return

arr = [34, 2, 10, -9]

n = len(arr)

oddEvenSort(arr, n);

for i in range(0, n):

  print(arr[i], end =" ")

**Output:** -9 2 10 34

1. **Python Program for BogoSort or Permutation Sort**

# Python program for implementation of Bogo Sort

import random

# Sorts array a[0..n-1] using Bogo sort

def bogoSort(a):

  n = len(a)

  while (is\_sorted(a)== False):

    shuffle(a)

# To check if array is sorted or not

def is\_sorted(a):

  n = len(a)

  for i in range(0, n-1):

    if (a[i] > a[i+1] ):

      return False

  return True

# To generate permutation of the array

def shuffle(a):

  n = len(a)

  for i in range (0,n):

    r = random.randint(0,n-1)

    a[i], a[r] = a[r], a[i]

# Driver code to test above

a = [3, 2, 4, 1, 0, 5]

bogoSort(a)

print("Sorted array :")

for i in range(len(a)):

  print ("%d" %a[i])

**output:** Sorted array:

0

1

2

3

4

5