from random import randint

def swap(arr, i, j):

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

def partition(arr, lo, hi):

pivot = randint(lo, hi)

# swap last element in pivot element

swap(arr, pivot, hi)

i = lo

# set j pointer to the last end of the scan

j = hi - 1

# when i pointer has greater index it has crossed the border of the j pointer

while i <= j:

# if i and j ptrs are < and >= to pivot swap then inc & dec

if arr[i] >= arr[hi] > arr[j]:

swap(arr, i, j)

i += 1

j -= 1

# check if either of the two needs to be incremented or decremented or both

elif arr[i] >= arr[hi] <= arr[j]:

j -= 1

elif arr[j] < arr[hi] > arr[i]:

i += 1

else:

i += 1

j -= 1

# swap pivot element located in [hi] and swap with i

swap(arr, i, hi)

# return new position of fixed pivot element

return i

def quicksort(arr,lo,hi):

# if lo is greater than or equal hi, means array length reached 1

if lo < hi:

fixed = partition(arr, lo, hi)

# process sorted left side of pivot

quicksort(arr, lo, fixed - 1)

# process sorted right side of pivot

quicksort(arr, fixed + 1, hi)

def mergesort(arr, lo, hi):

# lo and hi indeces at starting would be 0 to 9

# if we had an array of 9 elements mid index would be

# (0 + 9) / 2 = 4 then plus 1, which is 5

# this means we want to recursively pass the left most

# elements of the array until the array becomes a single value

# left most indeces in this first recursive iteration would be [0] to [4]

# since 0 < 5 is exclusive we only get indeces from 0 to 4

mid = int((lo + hi) / 2) + 1

# range value or base case is when the array length reaches 1

# because lo will be 0 and hi will be 0

if lo >= hi:

return

# assign left most elements in mid in temp array, 0 < mid

t\_left = arr[:mid]

# assign right most elements in mid in temp array, mid < length

t\_right = arr[mid:]

# process the left side first, pass in the arr that is already halved such that in recursive calls it gets smaller everytime

# in first recursive iteration arr would have length 9 and

# lo and hi indeces will be 0 and 9, because of mid index 5

# calculated from int((0 + 9) / 2) + 1, t\_left will be of length

# 5, since 0 to 5 exclusively will be the only elements it will take

# and will only have indeces 0, 1, 2, 3, 4

# in second recursive iteration arr would have length 5 and

# lo and hi indeces will be 0 and 4, because of mid index 3

# calculated from int((0 + 4) / 2) + 1, t\_left will be of length

# 3, since 0 to 3 exclusively will be the only elements it will take

# and will only have indeces 0, 1, 2

# in third recursive iteration arr would have length 3 and

# lo and hi indeces will be 0 and 2, because of mid index 2

# calculated from int((0 + 2) / 2) + 1, t\_left will be of length

# 1, since 0 to 2 exclusively will be the only elements it will take

# and will only have indeces 0, 1

# in fourth recursive iteration arr would have length 2 and

# lo and hi indeces will be 0 and 1, because of mid index 1

# calculated from int((0 + 1) / 2) + 1, t\_left will be of length

# 1, since 0 to 1 exclusively will be the only elements it will take

# and will only have indeces 0

mergesort(t\_left, lo, mid - 1)

# process right side second, pass the new lo and hi of each array

mergesort(t\_right, lo, hi - mid)

i = j = k = 0

while i != mid and j != (hi - mid + 1):

# arr is only modified for the specific temp array passed

if t\_left[i] > t\_right[j]:

# note that real arr will not be modified until last recursive call has been executed

arr[k] = t\_right[j]

j += 1

else:

arr[k] = t\_left[i]

i += 1

k += 1

# if j or i pointers have still elements left append those remaining elements

if j != hi - mid + 1:

for j in range(j, hi - mid + 1):

arr[k] = t\_right[j]

k += 1

else:

for i in range(i, mid):

arr[k] = t\_left[i]

k += 1

def bubblesort(arr):

for i in range(len(arr) - 1, 0, -1):

for j in range(i):

if(arr[j] > arr[j + 1]):

temp = arr[j]

arr[j] = arr[j + 1]

arr[j + 1] = temp

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

arr = [randint(0, 100) for \_ in range(100)]

mergesort(arr, 0, len(arr) - 1)

print(arr)