Iterative Merge Sort

import time  
import random  
import matplotlib.pyplot as plt  
  
import psutil  
import tracemalloc  
import os  
  
  
  
def mergeSort(a):  
 width = 1  
 n = len(a)  
  
 while (width < n):  
  
 l = 0  
 while (l < n):  
 r = min(l + (width \* 2 - 1), n - 1)  
 m = (l + r) // 2  
 if width > n // 2:  
 m = r - (n % width)  
 merge(a, l, m, r)  
 l += width \* 2  
 width \*= 2  
 return a  
  
  
def merge(a, l, m, r):  
 n1 = m - l + 1  
 n2 = r - m  
 L = [0] \* n1  
 R = [0] \* n2  
 for i in range(0, n1):  
 L[i] = a[l + i]  
 for i in range(0, n2):  
 R[i] = a[m + i + 1]  
  
 i, j, k = 0, 0, l  
 while i < n1 and j < n2:  
 if L[i] > R[j]:  
 a[k] = R[j]  
 j += 1  
 else:  
 a[k] = L[i]  
 i += 1  
 k += 1  
  
 while i < n1:  
 a[k] = L[i]  
 i += 1  
 k += 1  
  
 while j < n2:  
 a[k] = R[j]  
 j += 1  
 k += 1  
  
x\_coordinate = []  
y\_coordinate = []  
  
  
def my\_complex\_analysis\_method():  
 pass  
  
  
for k in range(1, 6):  
 array = [random.randint(1, 1000) for i in range(10 \*\* k)]  
 start = time.time()  
 mergeSort(array)  
 print(array)  
 print("\nThe size of the array is : ", len(array))  
 print("Time taken for executing :", round(time.time() - start, 6), "sec")  
  
 print("Memory taken:\t", psutil.Process(os.getpid()).memory\_info().rss / 1024 \*\* 2, "MB")  
 tracemalloc.stop()  
 x\_coordinate.append(100\* k)  
 y\_coordinate.append(round(time.time() - start, 6))  
  
  
print('\nRAM memory % used:', psutil.virtual\_memory()[2])  
plt.plot(x\_coordinate, y\_coordinate, marker="o")  
plt.xlabel("Size of n (The number of inputs)")  
plt.ylabel("The Time taken to execute ")  
plt.show()

**Recursive Merge Sort**

import time  
import random  
import matplotlib.pyplot as plt  
  
import psutil  
import os  
  
  
def merge(a,b):  
 c = []  
 while len(a) != 0 and len(b) != 0:  
 if a[0] < b[0]:  
 c.append(a[0])  
 a.remove(a[0])  
 else:  
 c.append(b[0])  
 b.remove(b[0])  
 if len(a) == 0:  
 c += b  
 else:  
 c += a  
 return c  
  
  
  
def mergeSort(x):  
 if len(x) == 0 or len(x) == 1:  
 return x  
 else:  
 middle = len(x)//2  
 a = mergeSort(x[:middle])  
 b = mergeSort(x[middle:])  
 return merge(a,b)  
  
x\_coordinate = []  
y\_coordinate = []  
  
  
def my\_complex\_analysis\_method():  
 pass  
  
  
for k in range(1, 6):  
 array1 = [random.randint(1, 1000) for i in range(10 \*\* k)]  
 start = time.time()  
 print(mergeSort(array1))  
  
 print("\nThe size of the array is : ", len(array1))  
 print("Time taken for executing :", round(time.time() - start, 6), "sec")  
  
 print("Memory taken:\t", psutil.Process(os.getpid()).memory\_info().rss / 1024 \*\* 2, "MB")  
 x\_coordinate.append(100 \* k)  
 y\_coordinate.append(round(time.time() - start, 6))  
  
print('\nRAM memory % used:', psutil.virtual\_memory()[2])  
plt.plot(x\_coordinate, y\_coordinate, marker="o")  
plt.xlabel("Size of n (The number of inputs)")  
plt.ylabel("The Time taken to execute ")  
plt.show()

**Iterative Quick Sort**

import time  
import random  
import matplotlib.pyplot as plt  
  
import psutil  
import os  
  
  
def partition(arr, l, h):  
 i = (l - 1)  
 x = arr[h]  
  
 for j in range(l, h):  
 if arr[j] <= x:  
 i = i + 1  
 arr[i], arr[j] = arr[j], arr[i]  
  
 arr[i + 1], arr[h] = arr[h], arr[i + 1]  
 return (i + 1)  
  
  
def quickSortIterative(arr, l, h):  
 size = h - l + 1  
 stack = [0] \* (size)  
 top = -1  
 top = top + 1  
 stack[top] = l  
 top = top + 1  
 stack[top] = h  
  
 while top >= 0:  
  
 # Pop h and l  
 h = stack[top]  
 top = top - 1  
 l = stack[top]  
 top = top - 1  
  
 p = partition(arr, l, h)  
  
 if p - 1 > l:  
 top = top + 1  
 stack[top] = l  
 top = top + 1  
 stack[top] = p - 1  
  
 if p + 1 < h:  
 top = top + 1  
 stack[top] = p + 1  
 top = top + 1  
 stack[top] = h  
  
  
x\_coordinate = []  
y\_coordinate = []  
  
  
def my\_complex\_analysis\_method():  
 pass  
  
  
for k in range(1, 6):  
 array1 = [random.randint(1, 1000) for i in range(10 \*\* k)]  
 start = time.time()  
 quickSortIterative(array1, 0, len(array1) - 1)  
 print(array1)  
  
 print("\nThe size of the array is : ", len(array1))  
 print("Time taken for executing :", round(time.time() - start, 6), "sec")  
  
 print("Memory taken:\t", psutil.Process(os.getpid()).memory\_info().rss / 1024 \*\* 2, "MB")  
 x\_coordinate.append(100 \* k)  
 y\_coordinate.append(round(time.time() - start, 6))  
  
print('\nRAM memory % used:', psutil.virtual\_memory()[2])  
plt.plot(x\_coordinate, y\_coordinate, marker="o")  
plt.xlabel("Size of n (The number of inputs)")  
plt.ylabel("The Time taken to execute ")  
plt.show()

**QuickSort Recursive**

import time  
import random  
import matplotlib.pyplot as plt  
  
import psutil  
import os  
  
  
def partition(arr, low, high):  
 i = (low - 1)  
 pivot = arr[high]  
  
 for j in range(low, high):  
  
 if arr[j] <= pivot:  
 i += 1  
 arr[i], arr[j] = arr[j], arr[i]  
  
 arr[i + 1], arr[high] = arr[high], arr[i + 1]  
 return (i+1)  
  
  
  
def quickSort(arr, low, high):  
 if low < high:  
 pi = partition(arr, low, high)  
 quickSort(arr, low, pi - 1)  
 quickSort(arr, pi + 1, high)  
  
x\_coordinate = []  
y\_coordinate = []  
  
  
def my\_complex\_analysis\_method():  
 pass  
  
  
for k in range(1, 6):  
 array1 = [random.randint(1, 1000) for i in range(10 \*\* k)]  
 start = time.time()  
 quickSort(array1, 0, len(array1) - 1)  
 print(array1)  
  
 print("\nThe size of the array is : ", len(array1))  
 print("Time taken for executing :", round(time.time() - start, 6), "sec")  
  
 print("Memory taken:\t", psutil.Process(os.getpid()).memory\_info().rss / 1024 \*\* 2, "MB")  
 x\_coordinate.append(100 \* k)  
 y\_coordinate.append(round(time.time() - start, 6))  
  
print('\nRAM memory % used:', psutil.virtual\_memory()[2])  
plt.plot(x\_coordinate, y\_coordinate, marker="o")  
plt.xlabel("Size of n (The number of inputs)")  
plt.ylabel("The Time taken to execute ")  
plt.show()