Insertion sort

Source code:

#insertion\_sort

import time

st = time.time()

def insertionSort(arr):

    # Traverse through 1 to len(arr)

    for i in range(1, len(arr)):

        key = arr[i]

        j = i - 1

        while j >= 0 and key < arr[j]:

            arr[j + 1] = arr[j]

            j -= 1

        arr[j + 1] = key

    #return print1(arr)

def readFile(filename, l):

    with open(filename, "r") as f:

        for line in f:

            l.append(int(line))

    if f.closed == False:

        f.close()

def main():

    txt\_array = []

    readFile("1000\_random.txt", txt\_array)

    size = len(txt\_array)

    insertionSort(txt\_array)

main()

#End time

et = time.time()

# get the execution time

elapsed\_time = (et - st )

print('Execution time:', elapsed\_time, 's')

elapsed\_time\_ms = round(elapsed\_time \*1000,2)

print('Execution time:', elapsed\_time\_ms, 'ms')

Quick sort

Source code:

#insertion\_sort

import time

st = time.time()

def insertionSort(arr):

    # Traverse through 1 to len(arr)

    for i in range(1, len(arr)):

        key = arr[i]

        j = i - 1

        while j >= 0 and key < arr[j]:

            arr[j + 1] = arr[j]

            j -= 1

        arr[j + 1] = key

    #return print1(arr)

def readFile(filename, l):

    with open(filename, "r") as f:

        for line in f:

            l.append(int(line))

    if f.closed == False:

        f.close()

def main():

    txt\_array = []

    readFile("1000\_random.txt", txt\_array)

    size = len(txt\_array)

    insertionSort(txt\_array)

main()

#End time

et = time.time()

# get the execution time

elapsed\_time = (et - st )

print('Execution time:', elapsed\_time, 's')

elapsed\_time\_ms = round(elapsed\_time \*1000,2)

print('Execution time:', elapsed\_time\_ms, 'ms')

Hybrid sort

Source Code

#hybird\_sorting

import time

st = time.time()

def insertion\_sort(arr, low, n):

    for i in range(low + 1, n + 1):

        val = arr[i]

        j = i

        while j > low and arr[j - 1] > val:

            arr[j] = arr[j - 1]

            j -= 1

        arr[j] = val

def partition(arr, low, high):

    pivot = arr[high]

    i = j = low

    for i in range(low, high):

        if arr[i] < pivot:

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

            j += 1

    arr[j], arr[high] = arr[high], arr[j]

    return j

def quick\_sort(arr, low, high):

    if low < high:

        pivot = partition(arr, low, high)

        quick\_sort(arr, low, pivot - 1)

        quick\_sort(arr, pivot + 1, high)

        return arr

# Hybrid function -> Quick + Insertion sort

def hybrid\_quick\_sort(arr, low, high):

    while low < high:

        # If the size of the array is less

        # than threshold apply insertion sort

        # and stop recursion

        if high - low + 1 < 100:

            insertion\_sort(arr, low, high)

            break

        else:

            pivot = partition(arr, low, high)

            # Optimised quicksort which works on

            # the smaller arrays first

            # If the left side of the pivot

            # is less than right, sort left part

            # and move to the right part of the array

            if pivot - low < high - pivot:

                hybrid\_quick\_sort(arr, low, pivot - 1)

                low = pivot + 1

            else:

                # If the right side of pivot is less

                # than left, sort right side and

                # move to the left side

                hybrid\_quick\_sort(arr, pivot + 1, high)

                high = pivot - 1

def readFile(filename, l):

    with open(filename, "r") as f:

        for line in f:

            l.append(int(line))

    if f.closed == False:

        f.close()

def main():

    txt\_array = []

    readFile("1000\_random\_with\_middle.txt", txt\_array)

    size = len(txt\_array)

    hybrid\_quick\_sort(txt\_array, 0, size - 1)

main()

#End time

et = time.time()

# get the execution time

elapsed\_time = (et - st )

print('Execution time:', elapsed\_time, 's')

elapsed\_time\_ms = round(elapsed\_time \*1000,2)

print('Execution time:', elapsed\_time\_ms, 'ms')