Due: April 27th at 11:59 pm

*[Note: Please use it for your own reference. Do not upload it online or share it with people outside of the class. ]*

***Please submit your homework in a file called hw8.py***

You need to implement and use the sorting algorithms you learn from the class to sort 1000 random elements. And for the testing step, you need to implement a random number generator function and test your algorithms by following the steps listed below.

1. Call the random number generator function and generate 1000 numbers.
2. Pass random numbers array into the sorting algorithms you implemented.
3. Check how the algorithms work.

Note: the generated random numbers should be stored in a List ( Random List). And your sorting algorithms should not alter the Random List. You could create a new List to store the numbers of the Random List or pass random\_list.copy() into the function.

Q1 (10 pts): Implement random\_numbers\_generator

(Hint: You might use the random package to generate a random number (import random). And one helpful function random.randint(min, max) will return a number within the range of min and max. For further details or other functions, you could check python document on its webpage)

def random\_numbers\_generator(num=1000, min=0, max=10000):

"""

Returns num of random elements

Precondition: num, min, max should be int, max should bigger than num

Example:

random\_numbers\_generator(num=3), return [645, 23, 7512]

random\_numbers\_generator(num=3, min=0, max=10), return [5, 3, 1]

random\_numbers\_generator(num=3, min=5, max=10), return [6, 9, 7]

"""

Q2 (20 pts): Implement Insertion Sort

def insertion\_sort(random\_list):

"""

Returns random\_list and sorted\_list

Example:

insertion\_sort([645, 23, 7512]), return [645,23,7512], [23,645,7512]

insertion\_sort([5, 3, 1]), return [5, 3, 1], [1, 3, 5]

insertion\_sort([6, 9, 7]), return [6, 9, 7], [6, 7, 9]

"""

Q3 (20 pts): Implement Quick Sort (Use the first element as pivot)

def Quick\_sort\_first(random\_list):

"""

Returns random\_list and sorted\_list

Example:

Quick\_sort\_first([645, 23, 7512]), return [645,23,7512], [23,645,7512]

Quick\_sort\_first([5, 3, 1]), return [5, 3, 1], [1, 3, 5]

Quick\_sort\_first([6, 9, 7]), return [6, 9, 7], [6, 7, 9]

"""

Q4 (30 pts): Implement Quick Sort (Use a random element in the list as pivot)

def Quick\_sort\_random(random\_list):

"""

Returns random\_list and sorted\_list

Example:

Quick\_sort\_random([645, 23, 7512]), return [645,23,7512], [23,645,7512]

Quick\_sort\_random([5, 3, 1]), return [5, 3, 1], [1, 3, 5]

Quick\_sort\_random([6, 9, 7]), return [6, 9, 7], [6, 7, 9]

"""

Q5 (20 pts): Compare these three sorting algorithms by sorting an increasing amount of numbers. For each number, run the algorithm three times and average the results. For this setting, please say which algorithm is better, Quick\_sort\_random or Quick\_sort\_first, and say why. When might this be different?

import time

Start = time.time()

Random\_list = random\_numbers\_generator(num=100000, max=100000\*10)

print(‘Generating random list, time{}’.format(time.time()- Start))

Start = time.time()

\_, sorted\_list = insertion\_sort(Random\_list)

print(‘Sort random list by insertion, time{}’.format(time.time()- Start))

Start = time.time()

\_, sorted\_list = Quick\_sort\_first(Random\_list)

print(‘Sort random list by Quick Sort using the first as pivot, time{}’.format(time.time()- Start))

Start = time.time()

\_, sorted\_list = Quick\_sort\_random(Random\_list)

print(‘Sort random list by Quick Sort using a random num as pivot, time{}’.format(time.time()- Start))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Amount of Elements | Generate  List | Insertion Sort | Quick Sort First Pivot | Quick Sort  Random Pivot |
| 10 |  |  |  |  |
| 100 |  |  |  |  |
| 1,000 |  |  |  |  |
| 10,000 |  |  |  |  |
| 50,000 |  |  |  |  |
| 100,000 |  |  |  |  |
| 1,000,000 |  |  |  |  |