Stevens Institute of Technology
CS 515
Homework 5

Due: May 17

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

First Name: Zimu **Last Name: Jiao**
Stevens ID: 10458119

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 1,00,000 random numbers by filling out the table. Briefly describe the cases that quick_sort_random would perform better than quick_sort_first and why?

```
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))
```

	Generate List	Insertion Sort	Quick Sort First Pivot	Quick Sort Random Pivot
Time				

Stevens Institute of Technology
CS 515
Homework 5

```
# Test:
if __name__ == '__main__':
    Start = time.time()
    Random_list = random_numbers_generator(num=100000, max=100000*10)
    print("Generating random list, time:{}".format(time.time() - Start))

    print("Next: sort part")
    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 as pivot, time:{}".format(time.time()-Start))
```

```
Generating random list, time:0.13597488403320312
Next: sort part
Sort random list by insertion, time:431.6716630458832
Sort random list by Quick sort using the first as pivot, time:0.3769071102142334
Sort random list by Quick sort using a random as pivot, time:0.3822612762451172
```

```
# Test:
if __name__ == '__main__':
    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 as pivot, time:{}".format(time.time() - Start))
```

```
if __name__ == '__main__'
ZimuJiao_hw5
"C:\Users\ZIMU JIAO\Desktop\venv\Scripts\python.exe" "C:/Users/ZIMU JIAO/Desktop/CS515-Python-Lab.Homework/ZimuJiao_hw5.py"
Generating random list, time:0.19068074226379395
Sort random list by insertion, time:733.3139848709106
Sort random list by Quick sort using the first as pivot, time:0.525615930557251
Sort random list by Quick sort using a random as pivot, time:0.5514678955078125
```

```
Generating random list, time:0.16112685203552246
Sort random list by Quick sort using the first as pivot, time:0.6483862400054932
Sort random list by Quick sort using a random as pivot, time:0.5574860572814941

Process finished with exit code 0
```

	Generate List	Insertion	Quick_sort_first	Quick_sort_random
Time _Jupter	0.13597488403320312	431.6716630458832	0.3769071102142334	0.3822612762451172
Time_ PyCharm	0.19068074226379395	733.3139848709106	0.525615930557251	0.5514678955078125
100,000			0.5340020656585693	0.542048454284668
			0.6483862400054932	0.5574860572814941
			0.5462384223937988	0.5231273174285889
			0.5574753284454346	0.5994505882263184
1,000,000			7.880513668060303	7.542190074920654
			7.617148876190155	7.721079587936401
			7.900965929031372	8.367348909378052

I run it two times first and the result is above.
It's clear that Insertion_sort take much more time than Quick_sort.
As showed the runtime of random_quick_sort are close to that of first_quick_sort.
I am not sure the differences in implementations have an impact on runtime, since random have one more condition_check.
So I run more times after, without insertion_sort because it's too slow, and they are still close but not always the same one win.
Quick_sort_first will have the worst_case runtime when the sequence is.total reversed. The random one can reduce the runtime in some way.
But since the list is random, the advantage of random_quick_sort are not work well, the result is not so clear show that random_sort is better. If the list is in logical order, I think the random one will be better.