hw1

Jingtao Scott Hong jh4ctf

September 14, 2021

1 Experiment 1

The sample size I chose for the experiment is

X = 500
2X = 1000
3X = 1500

To examine the obvious run time difference between two sorting method is by eyes. I can tell the latency of answers returned by the terminal by adding up the size of the array to be sorted. Alternatively, I used time package in python to better time the run time difference between two methods. Eventually, the size rounded up to X = 2000 to show a significant difference. Then I tested with size of 2X = 4000 and 3X = 6000.

The run time difference showed up to be:

Time Difference of 500: 0.008185386657714844,
Time Difference of 1000: 0.0331859588623046,
Time Difference of 1500: 0.06977963447570801

It can explained by the average run time of insertSort to be n^2 and that of quickSort to be n*log(n). It is close to my expectation because run time of nlog(n) is way faster than that of n^2.

2 Experiment 2

The difference is because we are sorting in the best cases -1, which makes insertSort as (n-1+1) = n and quickSort's best case as n*log(n). When I ran with the size from previous experiment, the console couldn't take such large size. Therefore, I changed to 500 and make a significant time difference as :

```
Sorting using insertion sort...DONE
Checking if sorted correctly...DONE
0.0003457069396972656
Sorting using quicksort...DONE
Checking if sorted correctly...DONE
0.02081775665283203
```

The result is close to my expectation that insertionSort is faster than quickSort. Because nlogn (best runtime of quickSort) is bigger than n(Best runtime for insertSort. So when it is the best case, the insertionSort is faster than quickSort.

3 Experiment 3

When I ran with the list of size 500 in reverse order, the output shows up to be:

```
Sorting using insertion sort...DONE
```

```
Checking if sorted correctly...DONE
0.0180208683013916
Sorting using quicksort...DONE
Checking if sorted correctly...DONE
0.014802932739257812
```

The result is close to my expectation because the reversed order list put two sorting methods into the worst case.Therefore, the ru ntime difference isn't significant because both worst case of two sorting methods are n^2.

4 Experiment 4

The output is:

```
Sorting using insertion sort...DONE
Checking if sorted correctly...DONE
Sorting using quick sort...DONE
Checking if sorted correctly...DONE
insertion: 0.003454353450045
quick: 0.010234410339355547
```

InsertSort in this case is much faster than QuickSort because in a almost sorted list because insertSort sort a almost sorted list in almost perfect case as there are not many elements that has to be adjusted, most of elements are almost in place. QuickSort however still sort the list in its average way and run-time which is n*log(n) because its way of sorting is impacted by the how much the list is sorted.

5 Experiment 5

There is the output:

```
Minisize of 5: 0.00483393669128418
Minisize of 10: 0.004209280014038086
Minisize of 15: 0.003949165344238281
Minisize of 20: 0.00380706787109375
Minisize of 30: 0.0040340423583984375
Minisize of 45: 0.0047149658203125
```

And the the runtime is the fast when the minisize is aournd 20. and it is slightly faster than when the minisize is 1 when we did in the Experiment 1.