Problem 7-2 :

Recreating the quicksort algorithms in python 3.7 I have found that the random algorithm performs worse then the non-random partitioning method. The random sampling of the elements from p to r creates a reasonably well-balanced tree, however the addition of this computational time to select and switch a random element seems to make the algorithm run slightly slower then the non-random quicksort algorithm on randomly created data. If the data was not created randomly the random quicksort method could have performed better than the non-random method. The sources of error that could have occurred such as; the use of a windows system that doesn’t take cpu time rather total running time, and the creation of the random array, have been mitigated by running the program through several repeated runs using the timeit function and averaging the results.

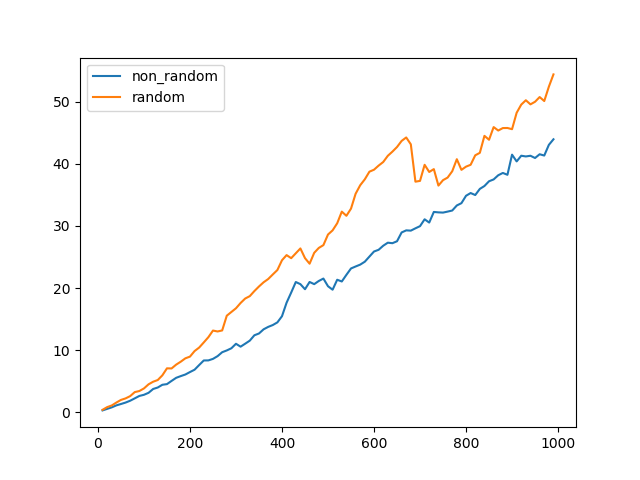
Figure 1 : Ninety nine runs of the quicksort and random-quicksort methods, repeating each run of n number of elements five times. Each run is a multiple of 10 ranging from 10 to 1000

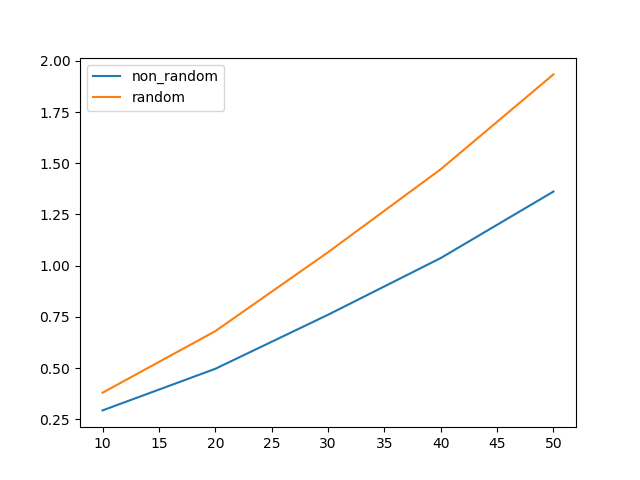
Figure 2 : Forty runs of the quicksort and random-quicksort methods, repeating each run of n number of elements five times.

Figure 3 : Ten runs of the quicksort and random-quicksort methods, repeating each run of n number of elements five times. Each run is 2^k elements where k is the run number. 