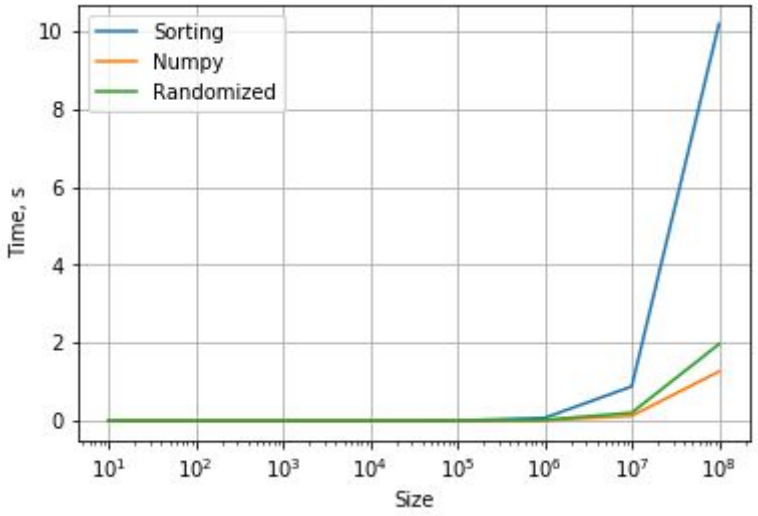
1. Complexity:

	Sorting	Randomised
Average	$n \cdot logn$	
Worst	n^2	$4 \cdot n^{3/4} log(4n^{3/4}) + n^{3/4} log\, n^{3/4}$ + n

   The complexity of finding median by sorting matches that of the algorithm used. For randomised Quicksort, the asymptotic complexity is as mentioned above.
2. The result of mupy and sorting algorithms always match, whilst for small arrays (n<100), randomised algorithm fails frequently, which should be counted as an error. The amount of errors for n<10 is about 10%, and about 2% for n<100. In case of success the result of randomised algorithm always matches other methods, as it is a Las Vegas algorithm.
3. According to the results of the experiments (7% of fails for 10-element arrays, 1% for 100-element ones, 0% for larger arrays), the amount of fails reduces drastically with the increase of array size. This corresponds to the theoretical estimate of $P_{fail}=n^{-1/4}$.
4. The dependence of the time of running of different algorithms is depicted below. As the blue curve illustrates the sorting algorithm, it is expectedly the highest one, its asymptotic complexity being the highest. Judging by the location of 'Numpy' curve, a conjecture that numpy uses a highly efficient sorting tool in the randomised median algorithm can be made. The difference between the curves is supported by the complexity estimates from question 1.



5. The way of handling a failure of the algorithm depends on the size of the input array. In case of small arrays (n<10), it's reasonable to use a sorting version instead, whereas for n>10, the probability of failing twice in a row is insignificantly small, and it is recommended to restart the algorithm
6. The only way to fail the algorithm without knowing the seed is providing an array of equal elements as an input. For known seed, however, it could be done by arranging the input in such an order that the sample R includes either the lowest or the highest elements only.