Sorting Algorithms

Basic info

- All the runtimes are in microseconds
- Number of iterations for smaller n values is 25000. For higher n values, 2500
- Approximations were used for n^2 runtime for higher n values based on previously observed values for runtime
- For random number generation, we use rand() and to generate different sequences srand(time(0)) is used to have a different seed every time.
- '~' is used for some entries, showing that they are not practically observed due to time or space issues

1. **Randomised Quicksort vs Quicksort**

n (# of iterations)	**1e2 (25000)**	**1e3 (25000)**	**1e4 (25000)**	**1e5 (2500)**	**1e6 (2500)**
2nlog_e(n)	921	13815	184206	2302585	27631021
Average Running time of Quicksort	5.09	74.79	995.426	12452.7	176636
Average Running time of Randomised Quicksort	7.37	109.29	1426.87	16796.7	193532
Average number of comparisons in Quicksort	647	10986	155812	2015735	24782346
Average number of comparisons in Randomised Quicksort	837	12971	175807	2218853	26785783
Average value of double sort time in Quicksort	26.616	2136.6	206422	~2x10^7	~2x10^9
Average value of double sort time in Randomised Quicksort	15.99	200.54	2408.27	27444.1	311133

Inferences

- A overhead is observed for randomised quicksort because of selection of random pivot in single sort
- We can also see the average number of comparisons is almost the same in both cases
- Normal quicksort runtime goes n^2 when double sort is done(worst case comes in when sorting a already sorted array), whereas randomised quicksort average runtime lies on nlogn complexity

2. **Randomised Quicksort vs Mergesort vs Optimised Mergesort**

n (# of iterations)	**1e2 (25000)**	**1e3 (25000)**	**1e4 (25000)**	**1e5 (2500)**	**1e6 (2500)**
2nlog_e(n)	921.34	13815.5	184206.8	2302585	27631021
nlog_2(n)	664.38	9965.78	132877	1660964	19931568
Average Running time of Quicksort	5.09	74.79	995.426	12452.7	176636
Average Running time of Mergesort	10.277	131.75	1593.6	18611.7	216801
Average Running time of Optimised-Mergesort	4.85	71.43	940.116	11485.8	~150000
Average number of comparisons in Quicksort	647	10986	155812	2015735	24782346
Average number of comparisons in Mergesort/ Optimised-mergesort	541	8707	120451	1536378	18674254
Number of times mergesort outperforms quicksort	0.176%	0.028%	0.064%	0.04%	0.012%
Number of times optimised-mergesort outperforms quicksort	92.19%	99.5%	99.95%	99.974%	~99.99%

Inferences
- Mergesort has overhead compared to randomised quicksort, due to copying of elements to new arrays.
- The average number of comparisons made in mergesort is comparatively less than in quicksort.
- Due to the overhead in mergesort, in very few cases it outperforms quicksort.
- In optimised mergesort we avoid copying to the same array, we get better performance than quicksort, where 99% of the time it performs better than quicksort

3. **Reliability of Randomised Quicksort**

n **(# of iterations)**	**1e2** **(25000)**	**1e3** **(25000)**	**1e4** **(25000)**	**1e5** **(2500)**	**1e6** **(2500)**
Average runtime of randomised quicksort	7.37	109.29	1426.87	16796.7	193532
2nlog_e(n)	921.34	13815.5	184206.8	2302585	27631021
Average number of comparisons in Randomised Quicksort	837	12971	175807	2218853	26785783
Number of cases where runtime exceeds the average by 5%	17.83%	8.384%	9.308%	9.16%	2.24%
Number of cases where runtime exceeds the average by 10%	17.83%	2.94%	1.904%	1.4%	0.56%
Number of cases where runtime exceeds the average by 20%	8.22%	1.032%	0.276%	0.44%	0.08%
Number of cases where runtime exceeds the average by 30%	5.1%	0.432%	0.16%	0.16%	0.04%
Number of cases where runtime exceeds the average by 50%	0.99%	0.06%	0.088%	0.12%	0.007%
Number of cases where runtime exceeds the average by 100%	0.076%	0.001%	0.016%	0.04%	0%

Inferences

- The average number of comparisons made in randomised quicksort follows 2nlog_e(n) function.
- Assuming a uniform distribution is obtained over large number of iterations while calculating runtime, we can say that about 65%, 83%, 81%, 80%, 95% of runtimes lies in ±5% of average runtime.
- It is almost negligible to find runtimes twice the average for higher input sizes.

4. **Effect of Fraction of Repetition of Elements on Runtime**

Testing factors

- Fixed Array Size = 100000
- Randomly assigned values from fixed ranges for each test to set an upperbound on number of unique elements
- Runtime is in microseconds

% of Unique elements(upperbound)	2-way Partition Quicksort	3-way Partition Quicksort
100%	13352.7	17834.2
90%	13439.4	16685.5
50%	13545.6	15936.7
20%	13677.3	14400.1
10%	14004.1	13190.2
1%	30230.3	9431.09
0.1%	211368	5935.36
0.01%	1894637	2897.95

Inferences

- As number of unique elements percent decreases 3-way partition quicksort becomes fast as less number of recursive calls are made.
- In case of 2-way partition, the runtime goes quadratic as the recursive tree become unbalanced.