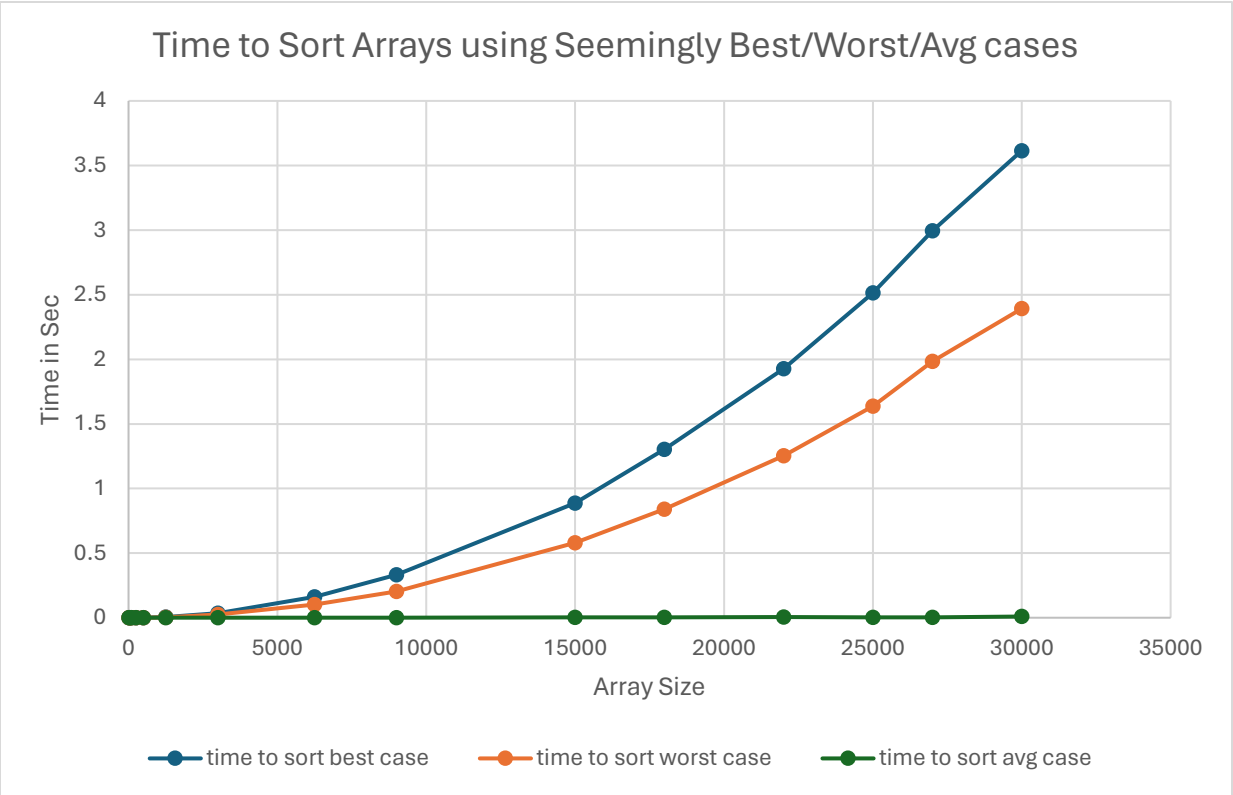

		time to sort best case	time to sort worst case	time to sort avg case
Array size	10	0.000002	0.000001	0.000001
Array size	50	0.000009	0.000007	0.000003
Array size	100	0.000034	0.000023	0.000008
Array size	250	0.000223	0.000171	0.000025
Array size	500	0.000936	0.000826	0.000046
Array size	1250	0.005698	0.003711	0.000131
Array size	3000	0.035787	0.024913	0.000359
Array size	6250	0.160714	0.102168	0.000732
Array size	9000	0.333532	0.204334	0.001121
Array size	15000	0.887953	0.581137	0.001955
Array size	18000	1.302789	0.840036	0.002346
Array size	22000	1.927637	1.253794	0.005255
Array size	25000	2.513437	1.636548	0.003566
Array size	27000	2.993606	1.983222	0.003924
Array size	30000	3.614742	2.391986	0.009337

Why was best-case and worst-case so much slower than avg case?

That's because best/worst case consistently chose pivots that basically removed the whole point of quicksort. They chose pivots that were the largest and smallest elements in the arrays. This resulted in many more swaps than necessary. Avg case was quickest because the element chosen was actually random because the elements in the array were randomly distributed. So when it came to swapping, the arrays made were partitioned much nicer, resulting in much faster sorting. I think best-case could be much faster if I added checks that checked if the array happened to be sorted after each swap.

3. Mathematical derivation:

The average case analysis is the average behavior of the algorithm over all possibilities. We select the last element as the pivot and partition the array into 2 groups around said pivot.

Let's guess the time complexity is $T(n) = n*\log(n)$

We'll verify this later.

We can define the recurrence relation based on the behavior of the quicksort. In the average case, we would have 2 splits where $T(n) = T(n/2) + T(n/2) + n$.

If I then substitute nlogn for T(n) we get:

$n*\log(n) = n/2*\log(n/2) + n/2*\log(n/2) + n$

$n*\log(n) = 2(n/2)\log(n/2) + n$

$= n\log(n/2) + n$

$= n(\log(n) – \log(2)) + n$

$=n\log(n) – n + n$

$n*\log(n) = n\log(n) – n + n$

so it holds

it would work if we had super unequal splits too:

Let’s say we had 90% of elements on one part:

$T(n) = 0.9T(n) + 0.1T(n/10) + n$

plug in $n*\log(n)$ for $T(n)$

$n*\log(n) = 0.9n\log(n) + 0.1(n/10*\log(n/10)) + n$

$0.1*n*\log(n) = 0.1*n/10\log(n/10) + n$

$0.1*n*\log(n) = 0.1n*\log n \quad – n + n$

so it is verified