[Algorithms] PA1 Report

B08901158 電機三 吳詩昀

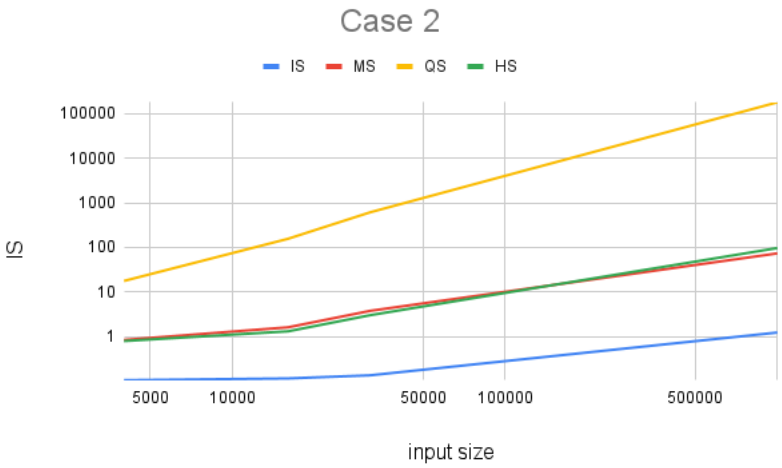
Running Time and Memory Usage Table

Input Size	IS		MS		QS		HS	
	CPU Time (ms)	Memory (kB)	CPU Time (ms)	Memory (kB)	CPU Time (ms)	Memory (kB)	CPU Time (ms)	Memory (kB)
4000.case2.in	0.105	5904	0.839	5904	17.696	5972	0.794	5904
4000.case3.in	12.097	5904	1.054	5904	9.93	5904	0.807	5904
4000.case1.in	6.426	5904	1.625	5904	0.974	5904	0.922	5904
16000.case2.in	0.115	6056	1.612	6056	156.223	6688	1.309	6056
16000.case3.in	85.996	6056	1.726	6056	121.582	6304	1.505	6056
16000.case1.in	45.697	6056	3.598	6056	1.647	6056	2.536	6056
32000.case2.in	0.135	6188	2.017	6188	611.98	7500	3.013	6188
32000.case3.in	331.835	6188	2.116	6188	466.309	6740	2.977	6188
32000.case1.in	165.135	6188	5.49	6188	2.29	6188	3.803	6188
1000000.case2.in	1.231	12144	70.57	14004	N/A	N/A	75.576	12144
1000000.case3.in	N/A	N/A	76.749	14004	N/A	N/A	76.06	12144
1000000.case1.in	N/A	N/A	149.559	14004	82.33	12144	142.523	12144

Growth of Run Time

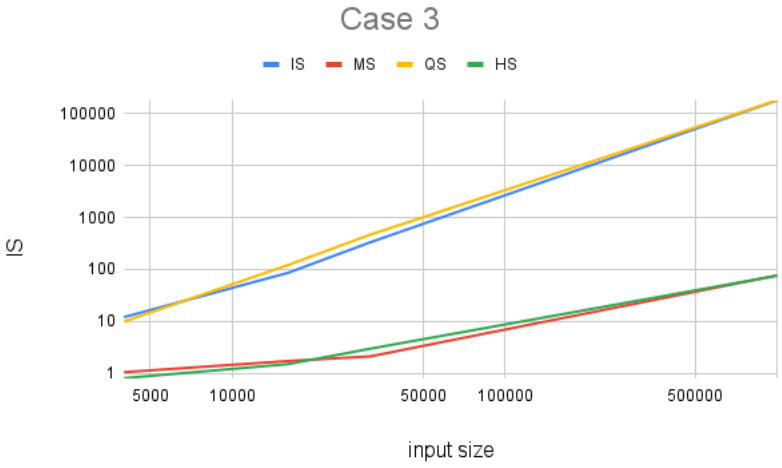
Case 1

Case 1 is the average case. Insertion sort has time complexity $\Theta(n^2)$, while the others are $\Theta(nlogn)$, ending up running faster than insertion sort.



Case 2

The input for case 2 is a sorted array. We can see that insertion sort uses the least amount of time, since it has linear complexity for best case. Quick sort is the slowest sorting method because of its $\Theta(n^2)$ complexity for worst case, which occurs when the array is already sorted.



Case 3

The input for case 3 is a reversely-sorted array, which is a worst case for quick sort and insertion sort. Both of them has $\Theta(n^2)$ time complexity. As for merge sort and heap sort, the time complexity is $\Theta(nlogn)$. Therefore, merge sort and heap sort is running faster than the other two.