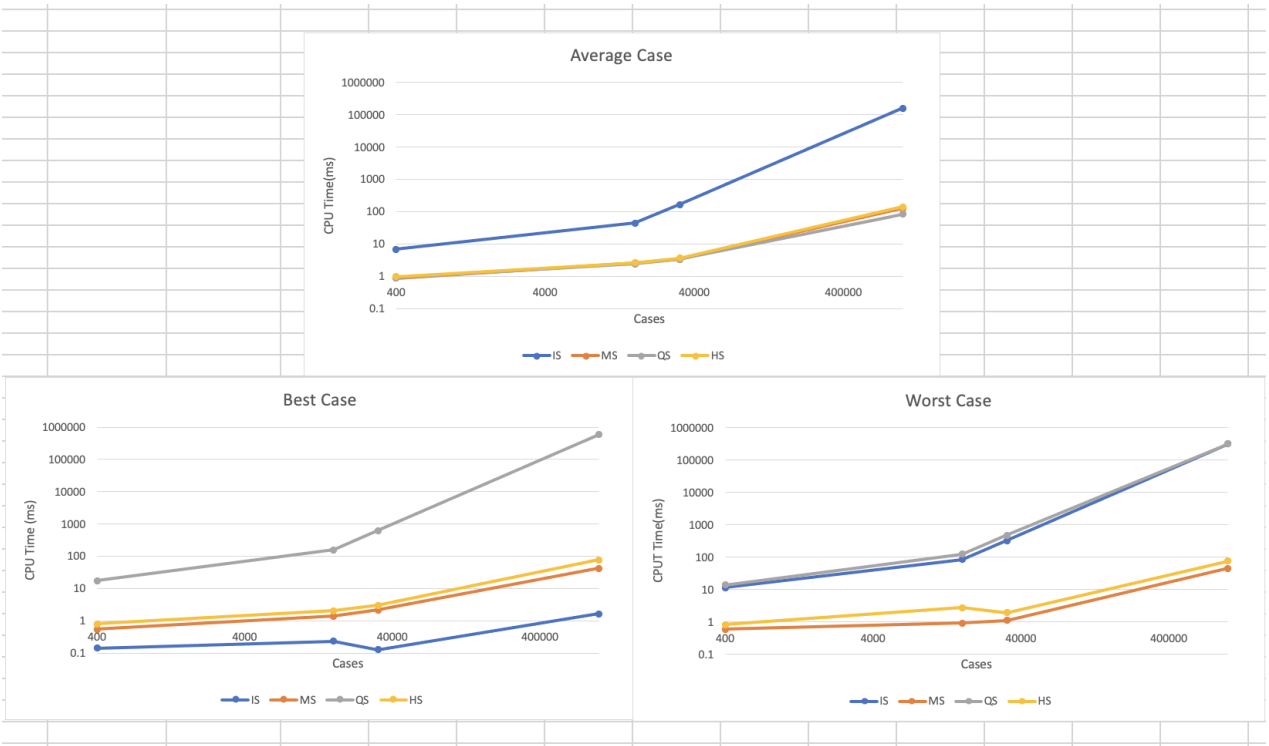
陳守仁
B09901116
EE3

PA1 Report

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	0.14	5904	0.546	5904	17.564	5972	0.796	5904
4000.case3	11.343	5904	0.586	5904	13.702	5904	0.81	5904
4000.case1	6.919	5904	0.874	5904	0.904	5904	0.964	5904
16000.case2	0.229	6056	1.388	6056	158.965	6684	2.012	6056
16000.case3	85.313	6056	0.906	6056	124.162	6304	2.732	6056
16000.case1	43.965	6056	2.522	6056	2.412	6056	2.654	6056
32000.case2	0.125	6188	2.149	6188	616.162	7504	2.976	6188
32000.case3	325.307	6188	1.102	6188	481.033	6740	1.921	6188
32000.case1	165.762	6188	3.352	6188	3.339	6188	3.619	6188
1000000.case2	1.63	12144	42.426	13876	590245	56844	76.75	12144
1000000.case3	318593	12144	45.335	13876	317270	27252	75.236	12144
1000000.case1	161252	12144	124.769	13876	82.602	12144	141.556	12144

Note 1: QS 1000000 case2/case3 using ulimit -s 262144 to set the stack size to 256MB

Note 2: The code was ran on EDA Union Lab Machines.



陳守仁
B09901116
EE3

Best Case Analysis(theoretical):

Insertion Sort : $\Theta(n)$

Merge Sort : $\Theta(n\log(n))$

Quick Sort : $\Theta(n^2)$

Heap Sort : $\Theta(n\log(n))$

In the best case, insertion sort is only required to go through the entire loop without having to exchange or sort anything, thus, it is only a $\Theta(n)$ type function. Merge sort, quick sort, and heap sort all have the time complexity of nlog(n) as we can see from the graph. Merge sort and heap sort are really obvious having the same time tendency. Quick sort will be worst when already sorted or reversely sorted since each partition gives a n-1 : 0 split every single iteration so its time complexity is $\Theta(n^2)$. Merge sort calls recurrence of twice in the sorting function that divides the array into 2 and the Merge function runs n times in the loop which means it's time complexity function will be $2T(n/2)+\Theta(n)$, it will then have a time complexity of $\Theta(n\log(n))$. Similarly, heap sort utilizes the same concept thus giving the same time complexity, the height is log(n) and the trickle down the root to make swaps takes n time complexity.

Average Case Analysis(theoretical):

Insertion Sort : $\Theta(n^2)$

Merge Sort : $\Theta(n\log(n))$

Quick Sort : $\Theta(n\log(n))$

Heap Sort : $\Theta(n\log(n))$

In the average case, the graphs shows clearly that Merge Sort, Quick Sort, and Heap Sort has the same time complexity. Insertion sort's time complexity is higher than them. For insertion sort, all the permutation is equally likely, the loop will execute about n/2 times each iteration(from 1 to n), so it give $\Theta(n^2)$. Merge Sort is the same as best case, the time complexity function will be $2T(n/2)+\Theta(n)$. For quick sort, average case will be its best case, each partition gives a [n/2] : [n/2]-1 split which will give a time complexity function $T(n) = 2T(n/2) + \Theta(n)$. Heap sort utilizes the same concept thus giving the same time complexity, the height is log(n) and the trickle down the root to make swaps takes n time complexity.

Worst Case Analysis(theoretical):

陳守仁
B09901116
EE3

Insertion Sort : $\Theta(n^2)$

Merge Sort : $\Theta(n\log(n))$

Quick Sort : $\Theta(n^2)$

Heap Sort : $\Theta(n\log(n))$

In the worst case, insertion sort and quick sort have the same time tendency, and heap and merge sort have the same time tendency. Insertion sort goes through all cases and swap iterating through all the case, thus it has a a time complexity of $\Theta(n^2)$. Quick sort will be worst when already sorted or reversely sorted since each partition gives a n-1 : 0 split every single iteration so its time complexity is $\Theta(n^2)$. Merge Sort is the same as best case and average case, the time complexity function will be $2T(n/2)+\Theta(n)$. Heap sort utilizes the same concept thus giving the same time complexity, the height is log(n) and the trickle down the root to make swaps takes n time complexity, thus giving a time complexity of $\Theta(n\log(n))$.