CPU Run Time And Memories Used

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	12500	0	12500	0	12500	1	12500
4000.case3	6.999	12500	1	12500	1	12500	1	12500
4000.case1	4.999	12500	1	12500	1	12500	1	12500
16000.case2	0	12648	0	12648	1	12648	1	12648
16000.case3	84.987	12648	2.999	12648	1	12648	2	12648
16000.case1	40.994	12648	2.999	12648	2	12648	0.999	12648
32000.case2	0	12648	0	12648	2	12648	2.999	12648
32000.case3	303.954	12648	3.999	12648	2	12648	8	12648
32000.case1	144.978	12648	6.998	12648	3.999	12648	3.999	12648
1000000.case2	3	18668	4.999	18668	44.993	18668	93.985	18668
1000000.case3	251860	18668	94.986	20524	46.993	18668	91.986	18668
1000000.case1	125897	18668	157.976	20524	93.986	18668	160.976	18668

GnuPlot of data point before sorting and after sorting

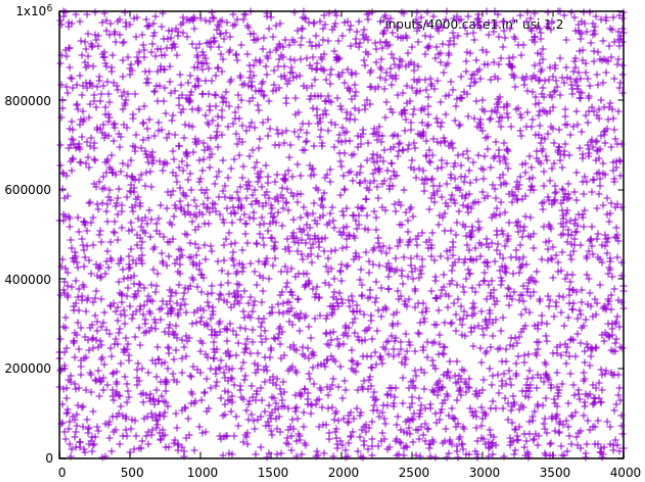

Fig 1. 4000.case1.in (before sorting)

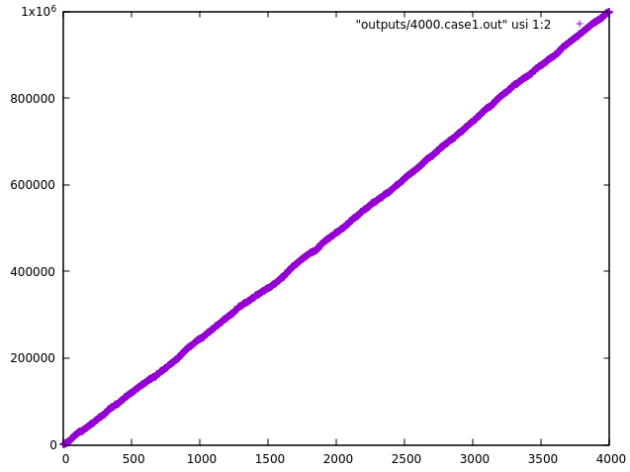

Fig 2. 4000.case1.out (after sorting)

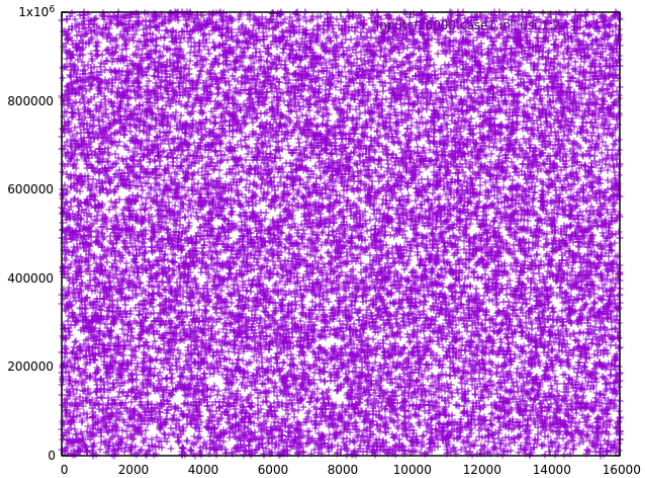

Fig 3. 16000.case1.in (before sorting)

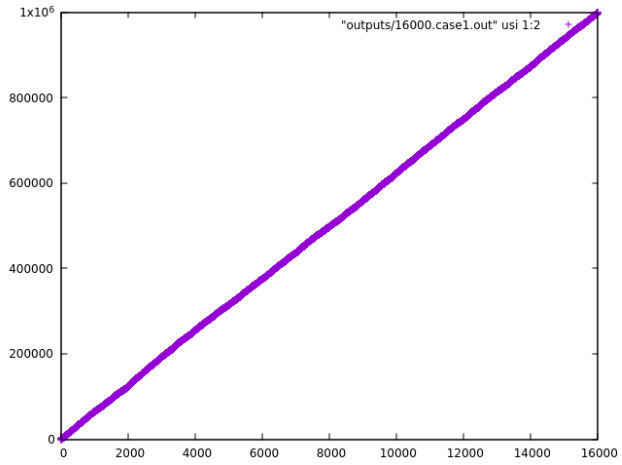

Fig 4. 16000.case1.out (after sorting)

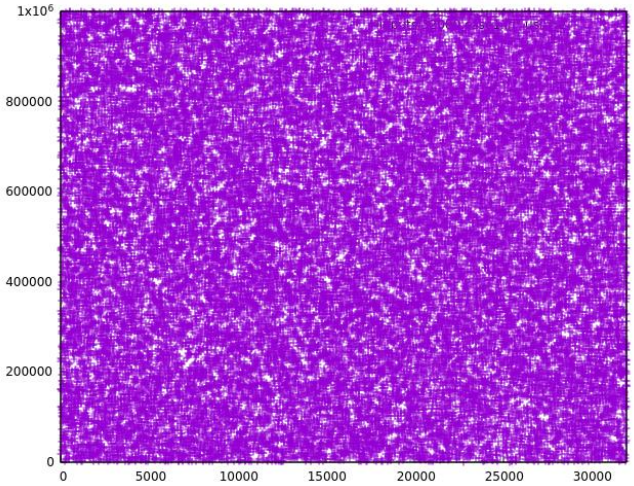

Fig 5. 32000.case1.in (before sorting)

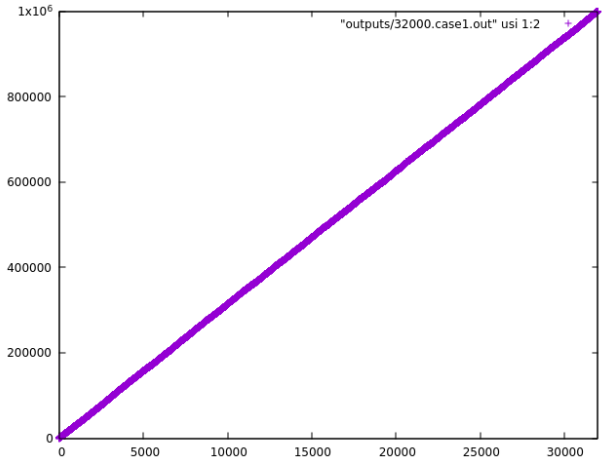

Fig 6. 32000.case1.out (after sorting)

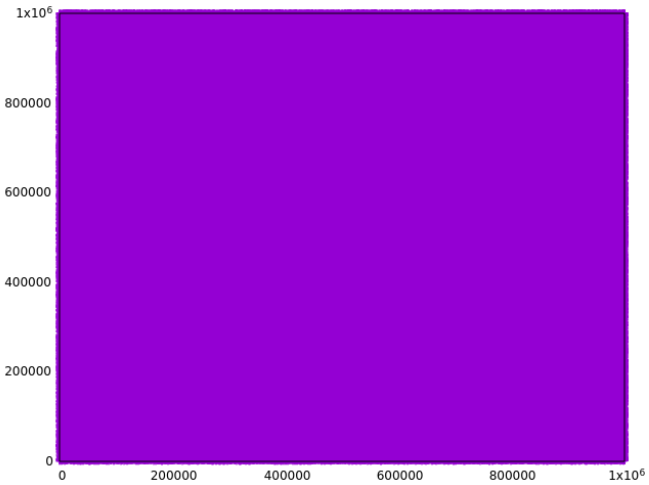

Fig 7. 1000000.case1.in (before sorting)

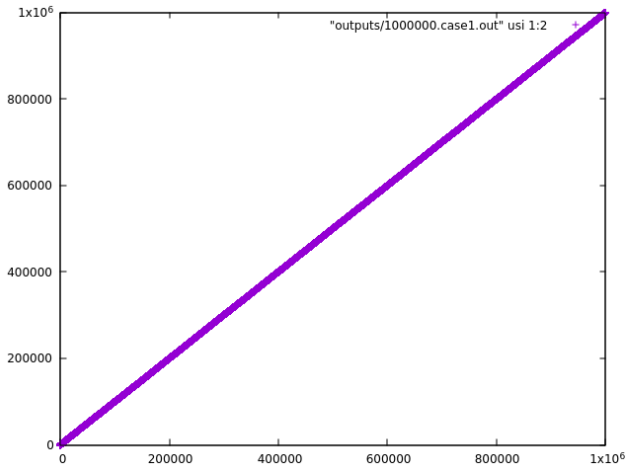

Fig 8. 1000000.case1.out (after sorting)

Implementation of each sort

Insertion sort:

我的 Insertion sort 的做法為，將第 i 個數加入「前 i−1 張排序過」的數字組合，得到 i 張排序過的數字組合，細節與教授講義中的 pseudo code 相同。

Merge sort:

Merge sort 我的實作方法大致上也同於上課所學到的 pseudo code，唯一比較不一樣的是我在進入 Merge 這個 function 之前加了一行判斷式，if(data[middle1] > data[middle2])。有了這個判斷式之後，當左邊 subarray 的最大值小於右邊 subarray 的最小值，就不用進行合併的動作。如此一來就能夠省去很多步驟數，加快運算速度。

Heap sort:

Heap sort 我的方法也是跟講義上一模一樣，先透過 Maxheapify 將 Max Heap 建起來，然後在一個一個按順序把 root 拉到 array 後面，依照 pseudo code 的方法。

Quick sort:

Quick sort 我的實作方法與講義上 randomly 的方法相同，有先加入亂數種子去隨機選取 pivot，這樣就可以很大機率避免碰到像 case2、case3 那種 worst case 的狀況，其餘部分與講義上相同。

Asymptotic analysis

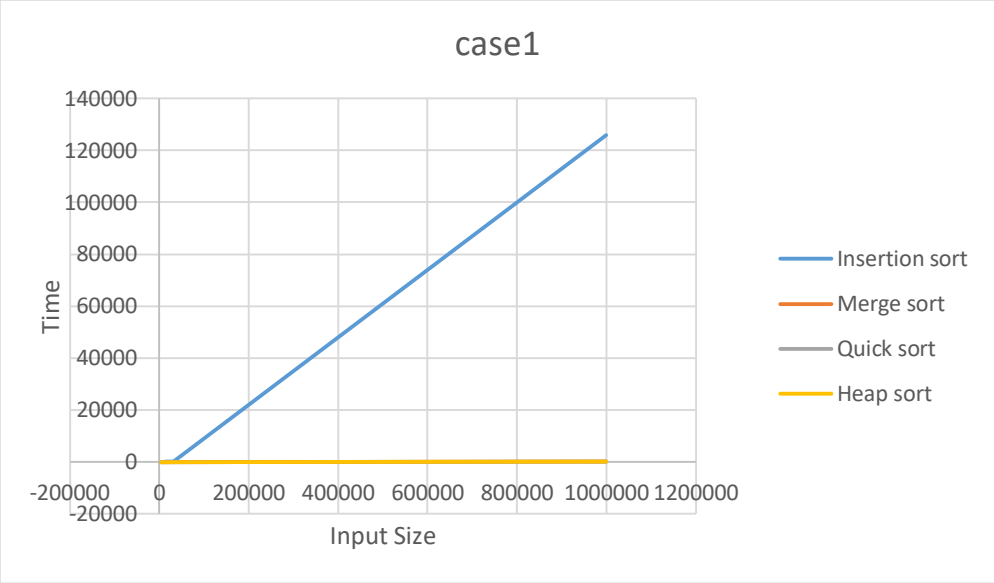

Fig 9. Case1 (include IS)

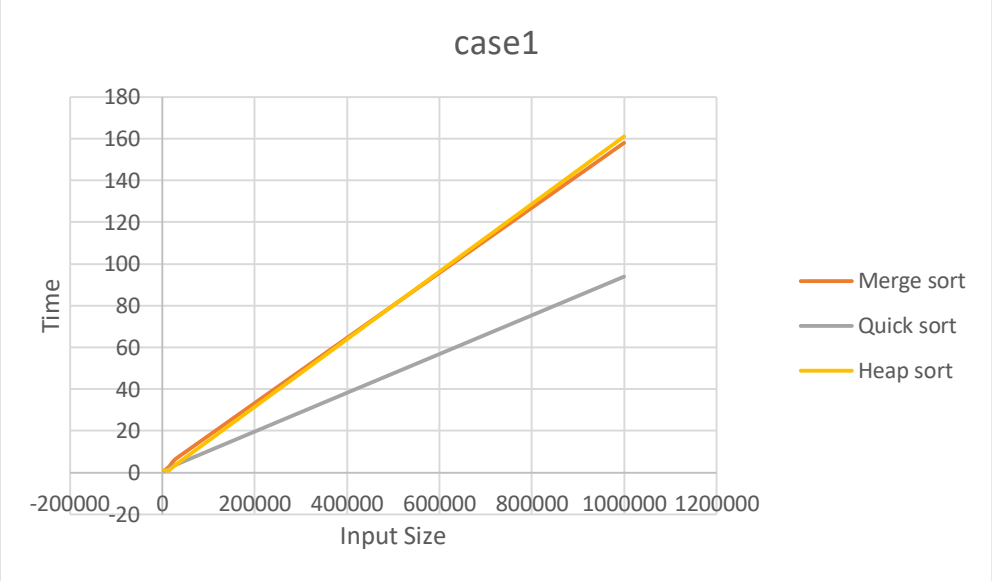

Fig 10. Case1 (without IS)

在 case1 中，很明顯 Insertion sort 運行時間遠大於其他三者，時間複雜度為 O(n^2)。而就其他三種 sort 的表現來說，在這個 case 中 Quick sort 的表現最為突出，因為在程式碼中 Quick sort 的總行數相較於其他兩者運行的步驟數是比較少的，而如上面 implementation 所說的，雖然在這種 random case 中，我用的實作方法顯然幫助不大，因為不會選到 worse case 的 pivot，但可以看到其表現仍然很優秀。最後，就結果十分接近的 Merge sort 與 Heap sort 來說，Merge sort 略勝 Heap sort 一點點，可能是因為像上面 implementation 寫的，我在 Merge sort 的 Merge 這個動作前加了一個判斷式，去決定要不要做合併的動作。如此一來，就能讓 merge sort 變成 O($nlgn$)，使其 Best case 運行速度更快。而對於 Heap sort 來說，在這次的 code 中我並沒有加任何判斷式去增進 Best case 的速度，所以全部的動作都會做過一次，每把最大的 element 取出到 array 的最後，就要再 Maxheapify 一次。

Fig 11. Case2 (include IS)

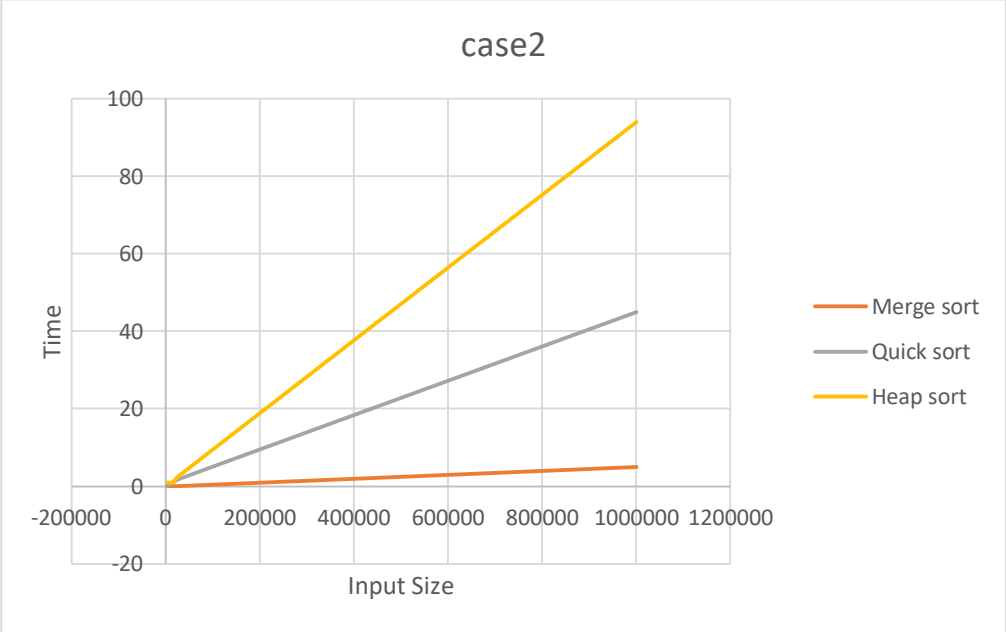

Fig 12. Case2 (without IS)

在 case2 中，裡面的 data points 是以 ascending 的方式排列，可視為是已經 sorted 過的 data。因此，對於 Insertion sort 來說只是 linear time case，複雜度為θ(n)，是 Insertion sort 的 best case，表現勝過其餘的 sort。而對於第二快的 Merge sort 來說，如我在 implementation 中提到，加了一個判斷需不需要 merge 的判斷式，因此是一個 recursive call 的 linear time case，複雜度為θ(n)。然而，對於必較慢的 Heap sort 跟 Quick sort 而言，它們複雜度還是θ($nlgn$)下，但是 Quick sort 會比 Heap sort 快的原因與 case 1 一樣，因為程式碼運行的步驟數比較少，所以才會有這樣的結果。

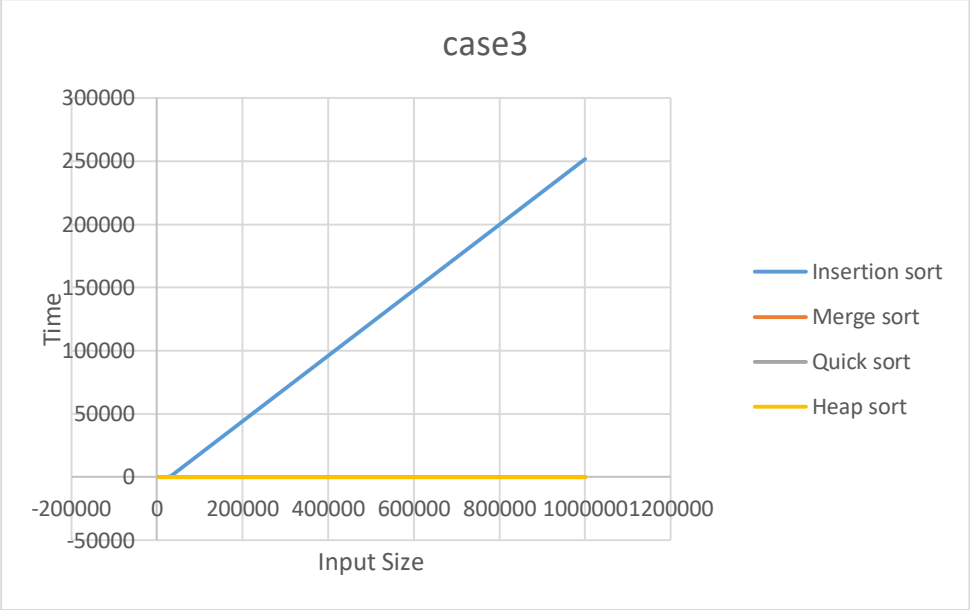

Fig 13. Case3 (include IS)

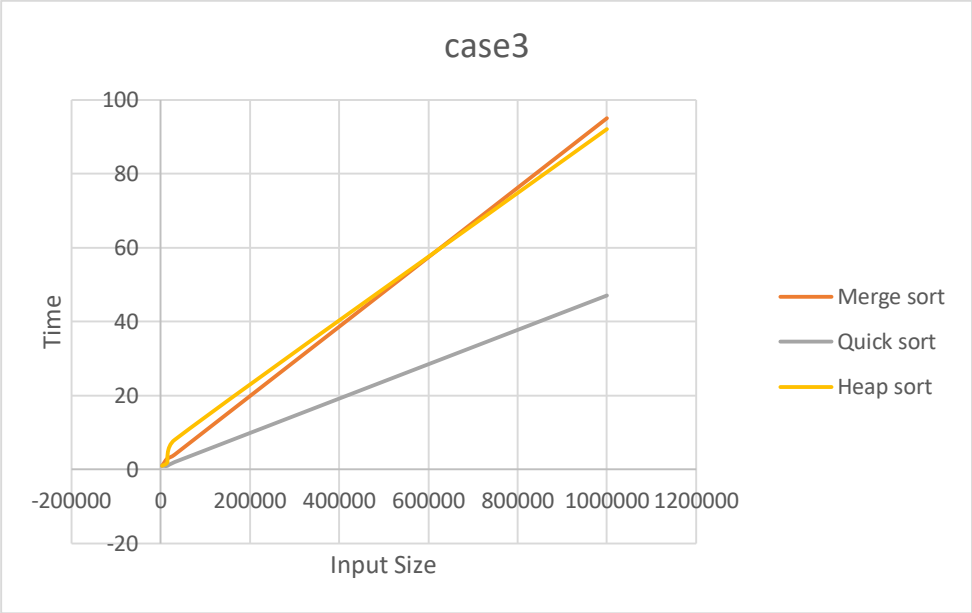

Fig 14. Case3 (without IS)

Case3 對於 insertion sort 來說是 Worst case，比 Case 1 還慢很多，因為 Case 3 的 data points 是以 descending 順序排列，時間複雜度為$\theta(n^2)$。對於 merge sort 而言，case 3 因為 data points 是 descending 排列，所以在 Merge 時會產生最少的比較次數，因此看似會是 Worst case，但卻擁有比 case 1 的 random data points 還優秀的運算速度。而在 case 3 Heap sort 與 Merge sort 運算時間差不多，也與自己的 case2 表現差不多，由此可知 Heap sort 不太會受 ascending order 與 descending order 的影響，但可以確定的是有順序的 data points 比沒順序的 sort 還快。就 Quick sort 來說，會比其他快的原因還是一樣，因為程式碼運行的步驟數少，所以 Quick sort 平均來看都會表現得比 Heap sort 跟 Merge sort 還優秀。然而，如果各個 sort 的 Best case 就不一定了，就像 case2 Insertion sort 與 Merge sort 就表現得比 Quick sort 還好，因此還是得視情況而定。