Algorithm Homework 1

b97502049 電機四 蔡昀達

1. Implement

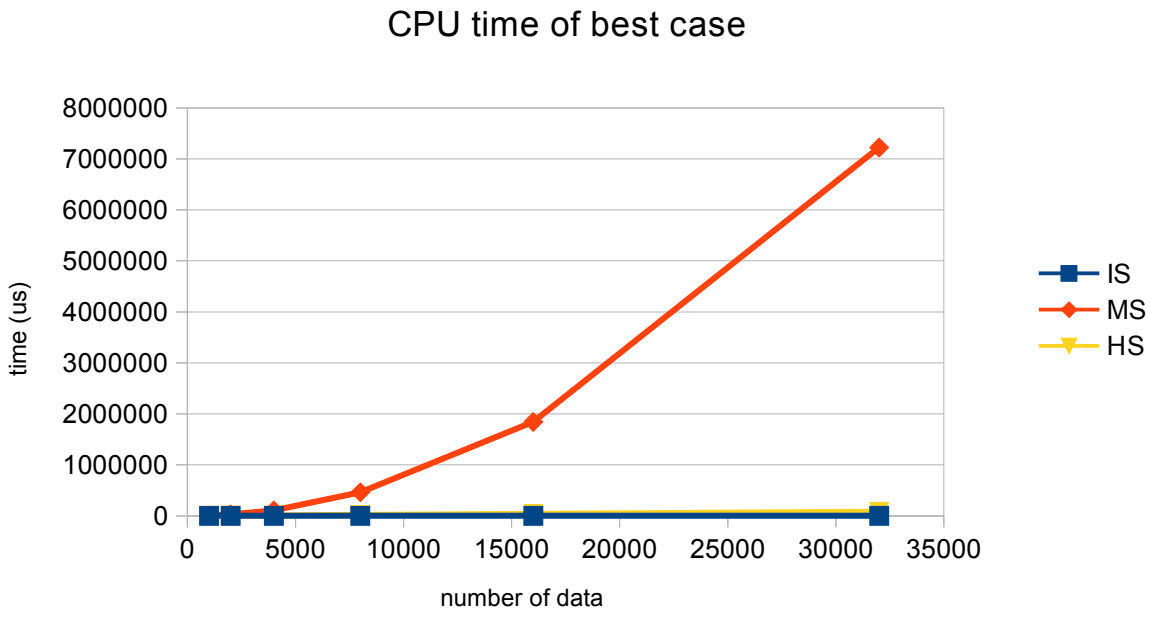
   我的設計方式先創立一個 MySort 的類別來封裝三種 sorting method 分別為 insertion_sort 、mergeSort、heapSort，並且將從檔案讀入的內容原始資料儲存為一個 STL vector 作為 MySort 型別的 data member，Insertion sort, merge sort 基本上就是參考老師所給的範例程式碼，而 heap sort 的部份則是參考課本上的 pseudo 寫成，其中要注意的是課本在 array index 的編排是從 1 開始，但是 C/C++卻是從 0 開始。

   因為這次的作業需要大量重複執行./mysort 這隻程式，為了簡化重複且多餘的手動執行過程，我寫了一個 shell script 來一次產生所有的數據，

2. 實驗數據

Input size	IS		MS		HS	
	CPU time(us)	Memory(MB)	CPU time(us)	Memory(MB)	CPU time(us)	Memory(MB)
1000.bc	95	3.27734	7824	3.27734	2092	3.27734
1000.wc	22283	3.27734	7996	3.27734	1477	3.27734
1000.ac	16429	3.27734	7955	3.27734	1594	3.27734
2000.bc	224	3.27734	28023	3.27734	3675	3.27734
2000.wc	84753	3.27734	29223	3.27734	3221	3.27734
2000.ac	43004	3.27734	28900	3.27734	3471	3.27734
4000.bc	231	3.27734	106754	3.27734	7955	3.27734
4000.wc	351422	3.27734	106916	3.27734	7529	3.27734
4000.ac	170669	3.27734	107178	3.27734	9476	3.27734
8000.bc	457	3.27734	462791	3.27734	18065	3.27734
8000.wc	1376726	3.27734	462832	3.27734	16098	3.27734
8000.ac	683381	3.27734	464774	3.27734	16597	3.27734
16000.bc	960	3.41016	1842032	3.41016	37573	3.41016
16000.wc	5575008	3.41016	1822565	3.41016	34912	3.41016
16000.ac	2782337	3.41016	1854448	3.41016	36930	3.41016
32000.bc	1898	3.65625	7224365	3.65625	80275	3.65625
32000.wc	22341629	3.65625	7426304	3.65625	76468	3.65625
32000.ac	11107584	3.65625	7255184	3.65625	80222	3.65625

3. 圖表 & 討論
Best case



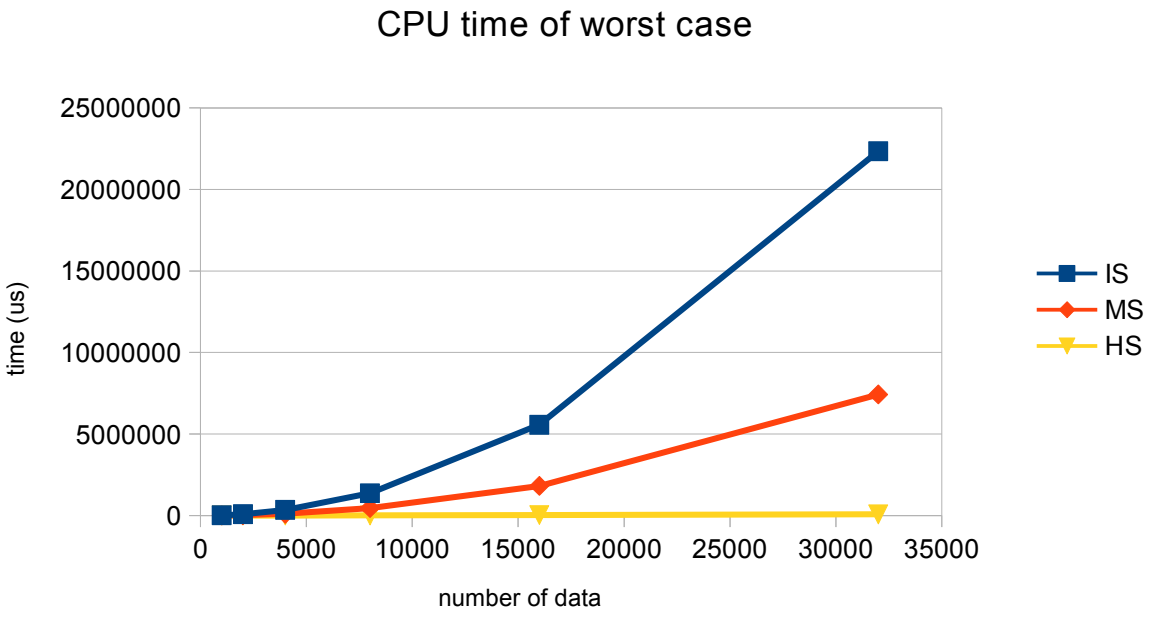
Input size	IS_bc		MS_bc		HS_bc	
	CPU time(us)	Memory(MB)	CPU time(us)	Memory(MB)	CPU time(us)	Memory(MB)
1000.bc	95	3.27734	7824	3.27734	2092	3.27734
2000.bc	224	3.27734	28023	3.27734	3675	3.27734
4000.bc	231	3.27734	106754	3.27734	7955	3.27734
8000.bc	457	3.27734	462791	3.27734	18065	3.27734
16000.bc	960	3.41016	1842032	3.41016	37573	3.41016
32000.bc	1898	3.65625	7224365	3.65625	80275	3.65625

根據課本上所推導出來的結果，三種 sorting algorithm 的 computing complexity 分別如下：

1. insertion sort: Θ(n) (for the best case)
2. merge sort: Θ(nlgn)
3. heap sort: O(nlgn)

由上面的結果可以看出來 insertion sort 的執行所需要的時間基本上就是隨著資料點數倍增而倍數成長，而 merge sort 則是因為無論何種 case 其結果都是 tightly bound 在 nlgn 因此運算的時間是最多的，而再課本的分析當中只能確定 heap sort 的 upper bound 是 nlgn 因此在者裡所觀察到的 heap sort 隨著資料點數成長幅度小於 nlgn 因此符合理論上的預測。

Worst case



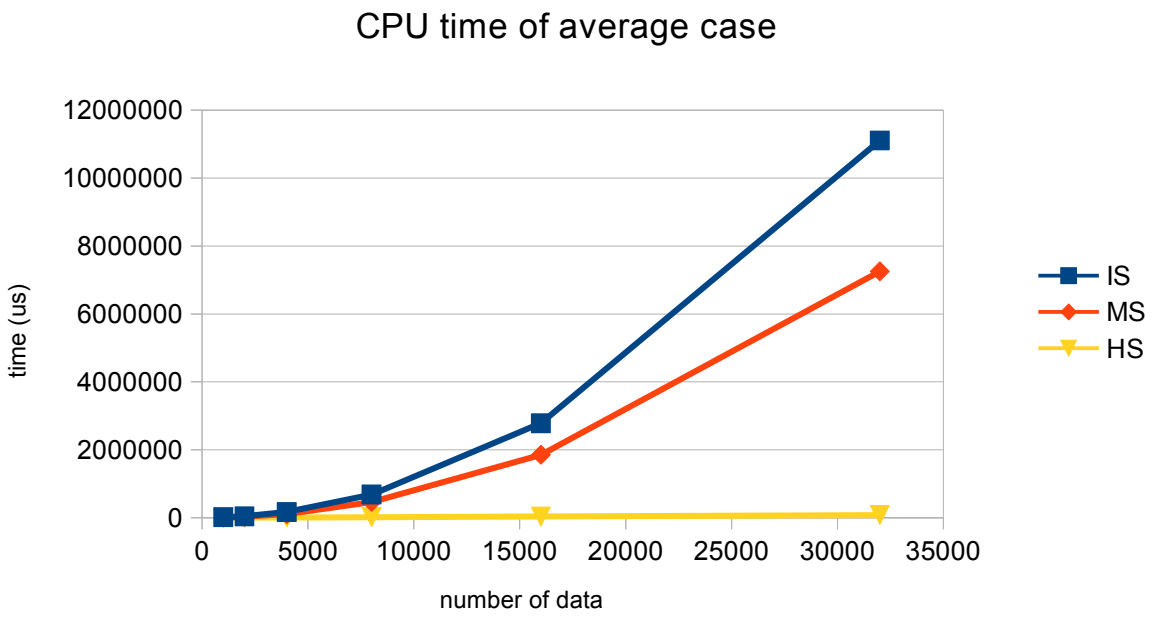
Input size	IS_wc		MS_wc		HS_wc	
	CPU time(us)	Memory(MB)	CPU time(us)	Memory(MB)	CPU time(us)	Memory(MB)
1000.wc	22283	3.27734	7996	3.27734	1477	3.27734
2000.wc	84753	3.27734	29223	3.27734	3221	3.27734
4000.wc	351422	3.27734	106916	3.27734	7529	3.27734
8000.wc	1376726	3.27734	462832	3.27734	16098	3.27734
16000.wc	5575008	3.41016	1822565	3.41016	34912	3.41016
32000.wc	22341629	3.65625	7426304	3.65625	76468	3.65625

根據課本的計算複雜度分析，三種 sorting algorithm 的 computing complexity 分別如下：

1. insertion sort: $\Theta(n^2)$
2. merge sort: $\Theta(nlgn)$
3. heap sort: $O(nlgn)$

而從實驗數據的結果觀察，insertion sort 當資料點數從 1000 成長為 2000 時 CPU 執行的時間成長了 4 倍左右，而後每當資料點數成長為兩倍時資料點數幾乎都成現 4 倍的成長，因此符合 $\Theta(n^2)$。在 merge sort 的部份，可以看出其運算時間成長的趨勢和 best case 並沒有不同，皆為 nlgn。而 heap sort 也可以觀察出每當資料點數倍增的時候程式執行時間大約都在兩倍到多，大約倍 bound 在 nlgn 左右。

average case

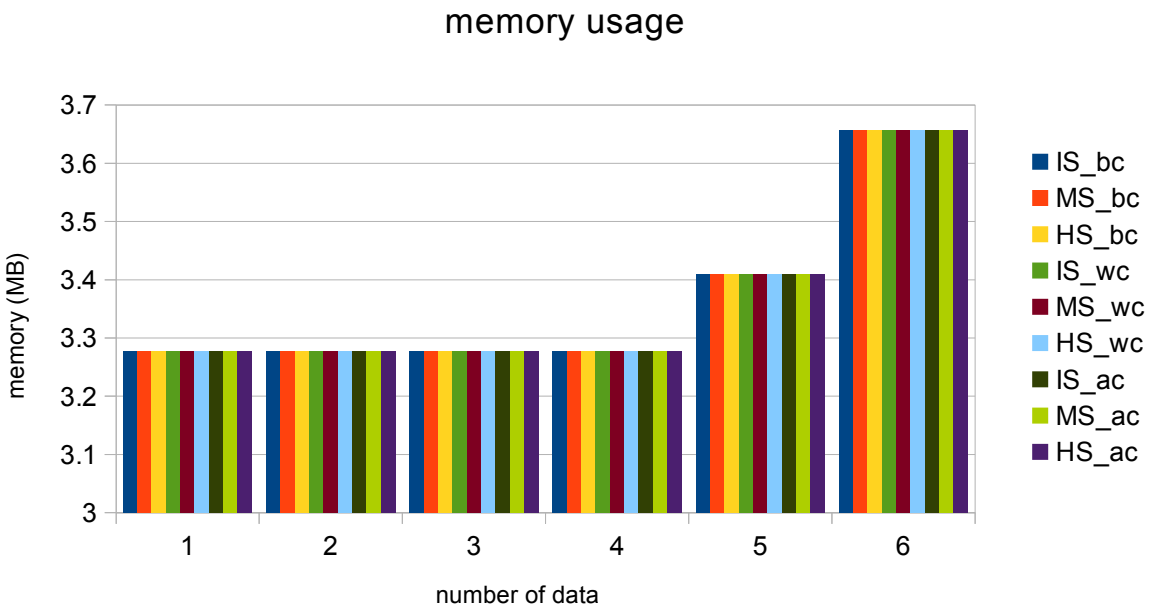


Input size	IS_ac		MS_ac		HS_ac	
	CPU time(us)	Memory(MB)	CPU time(us)	Memory(MB)	CPU time(us)	Memory(MB)
1000.ac	16429	3.27734	7955	3.27734	1594	3.27734
2000.ac	43004	3.27734	28900	3.27734	3471	3.27734
4000.ac	170669	3.27734	107178	3.27734	9476	3.27734
8000.ac	683381	3.27734	464774	3.27734	16597	3.27734
16000.ac	2782337	3.41016	1854448	3.41016	36930	3.41016
32000.ac	11107584	3.65625	7255184	3.65625	80222	3.65625

綜合 best case 以及 worst case 的結果，我們可以知道三種 sorting algorithm 一定會被 bound 在

1. insertion sort: $\Theta(n)$~$\Theta(n^2)$
2. merge sort: $\Theta(nlgn)$
3. heap sort: $O(nlgn)$

的區間，而實驗數據也都符合這樣的預測，因此也能夠對於這次的程式做一個初步的驗證。



在 memory usage 的部份，我是選用老師所給的範例 code 中 tm_usage 的 vmPeak data member 來觀察記憶體的最大使用量。很明顯的可以看出記憶體的使用在不同的 sorting algorithm 以及不同的 sample case 中並沒有任何不同。而再老師所給的 library 中可以看到這個 object 取得記憶體使用狀況的方式是去/proc/self/status 中印出當前的記憶體使用狀況，因此是以作業系統的角度去觀察整之程式的記憶體使用狀況，而在作業系統記憶體管理是直接 allocate 一整塊的記憶體分頁給 user space 的程式，因此除非程式所產生的 stack 或者 heap 大到記憶體分頁的上限否則作業系統不會 allocate 新的記憶體分頁給應用程式。因此我們所觀察到的 pattern 才會是一個一個的 step，這也表示三種演算法再一開始宣告的變數空間相差再一定程度的範圍(記憶體分頁)之內。