

a. pseudo code:

● 主程式：

step 1. 定義一個 structure，可以包含公司或是學生的資訊：

- isCom(是否為 company)
- id(編號)
- integrity(品行)
- score(學期成績)
- salary(公司可提供薪資)
- desire_salary(學生可獲得的薪資)
- decision(最後學生抉擇的公司)
- side(後續 Divide and Conquer 標記用)

step 2. 建立 overall 的 structure 陣列並匯入 student.txt 跟 company.txt 的資料

step 3. 針對 integrity(品行成績)作 **merge sort()**

step 4. Divide and Conquer 來得出學生們的 decision --> 使用 **DC()** function

step 5. 針對編號(id)作 **merge sort()**

step 6. 將學生的編號(id)及最終選擇(decision)循序寫入 output.txt

● Divide and Conquer 副程式 DC():

step 1. 建立二維座標系，將 integrity 設為 X 軸座標，而 score 設為 Y 軸座標

step 2. 取出目前按照 integrity 排序好的陣列之 midpoint，以之為垂直線劃分出左、右區塊

step 3. DC(左區塊)

step 4. DC(右區塊)

step 5. 陣列按照 score 大小進行 **merge sort()**

step 6. 若「左區塊」有公司資料且薪水比「右區塊」的學生的 desire_salary 更高，則更新此位學生的 desire_salary 和 decision

● 排序副程式 **merge sort()**:

step 1. 取出陣列 midpoint，根據此 midpoint 取出左右兩條 sub array

step 2. merge sort(起點, midpoint)

step 3. merge sort(midpoint + 1, 終點)

step 4. merge 兩條 sub array，確保排序正確

b. Experimental results and analysis (Time complexity)

● Time Complexity 預測：

(ps) student number: n_1 , company number: n_2 , $n = n_1 + n_2$

➤ Divide and Conquer 副程式 DC()

根據上頁的步驟分析：

step. 3 含有 merge sort --> $(c_1 * \lg n)$

step. 4 含有 merge sort --> $(c_2 * \lg n)$

$$T(n) = 2T(n/2) + c_1 * \lg n + c_2 * \lg n = 2T(n/2) + c * \lg n$$

by Master theorem,

$$T(n) = O(n \lg^2 n)$$

➤ 主程式：

step 1. 定義一個 structure，可以包含公司或是學生的資訊

step 2. 匯入 student.txt 跟 company.txt 的資料 --> $O(n)$

step 3. integrity merge sort() --> $O(\lg n)$

step 4. Divide & Conquer function --> $O(n \lg^2 n)$

step 5. id merge sort() --> $O(\lg n)$

step 6. write in output.txt --> $O(n)$

$$\text{Time Complexity} = O(n) + O(\lg n) + O(n \lg^2 n) + O(\lg n) + O(n) = O(n \lg^2 n)$$

● Time Complexity 驗證

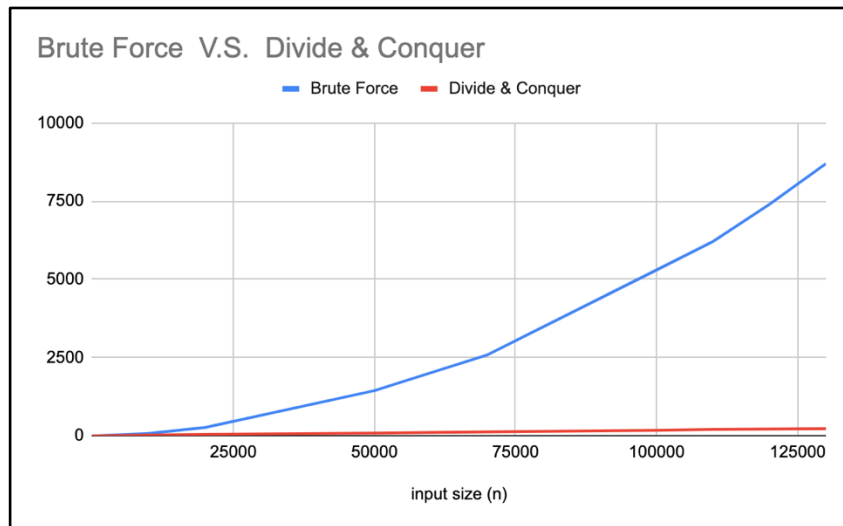
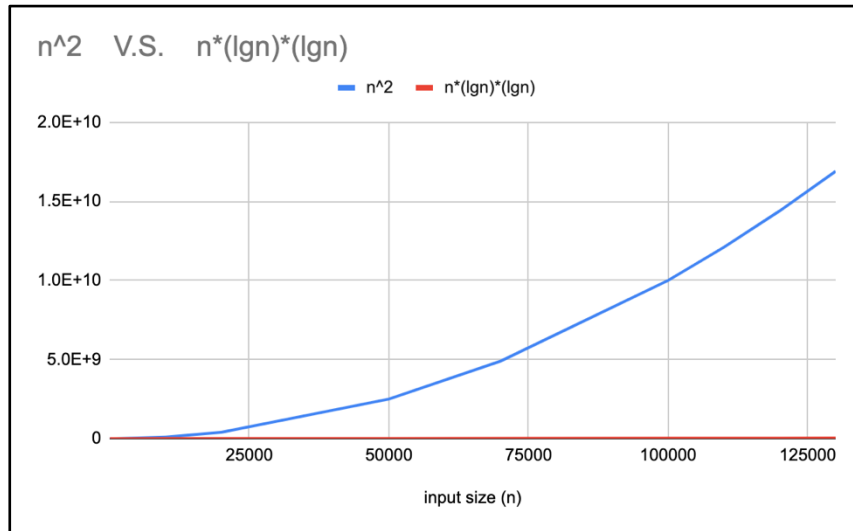
➤ 此次設計的 Divide-and-Conquer Algorithm 主要的 bottleneck 是在 DC() 副程式中針對學期成績(score)作 merge sort 的步驟，因此我設計了一個測試程式來自動生成學生和公司的資料共 n 筆，其中會將 score 設計成 merge sort 的 worst case 排序方式。

➤ 另外我也寫了一份 Brute Force 方法的程式來比對驗證此 Algo 的正確性。
Brute Force $O(n^2)$: 先將公司資料按照薪資大小排序，將著每位學生都檢查所有的公司一遍，看是否符合公司標準。

➤ 由下圖可以觀察兩種方法對應 input size 呈現的趨勢，驗證優化後的時間複雜度為 $O(n \lg^2 n)$

■ n^2 V.S. $n \lg^2 n$ 對應 input size 的產出值

■ Brute Force V.S. Divide-and-Conquer 對應 input size 的程式執行時間



input size (n)	n^2	$n \cdot (\lg n) \cdot (\lg n)$	Brute (second)	D&C (second)
10	100	110.3520627	0.000000	0.000000
100	10000	4414.082507	0.000000	0.000000
1000	1000000	99316.85641	0.000000	0.000000
10000	100000000	1765633.003	70.000000	20.000000
20000	400000000	4082774.501	260.000000	40.000000
50000	2500000000	12183043.79	1440.000000	80.000000
70000	4900000000	18133583.4	2580.000000	120.000000
100000	10000000000	27588015.67	5300.000000	170.000000
110000	12100000000	30851351.53	6210.000000	200.000000
120000	14400000000	34162458.91	7400.000000	210.000000
130000	16900000000	37517650.52	8700.000000	220.000000