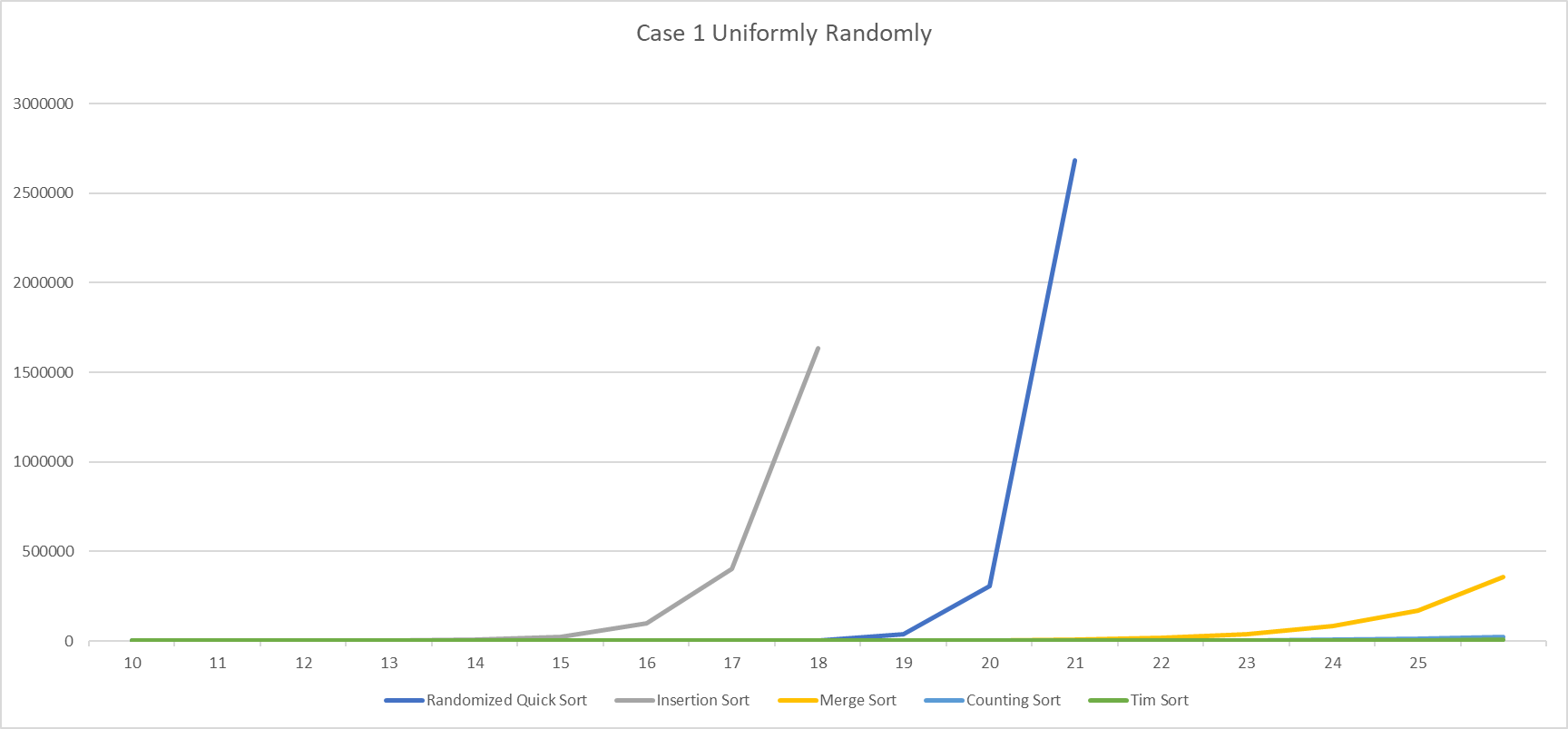
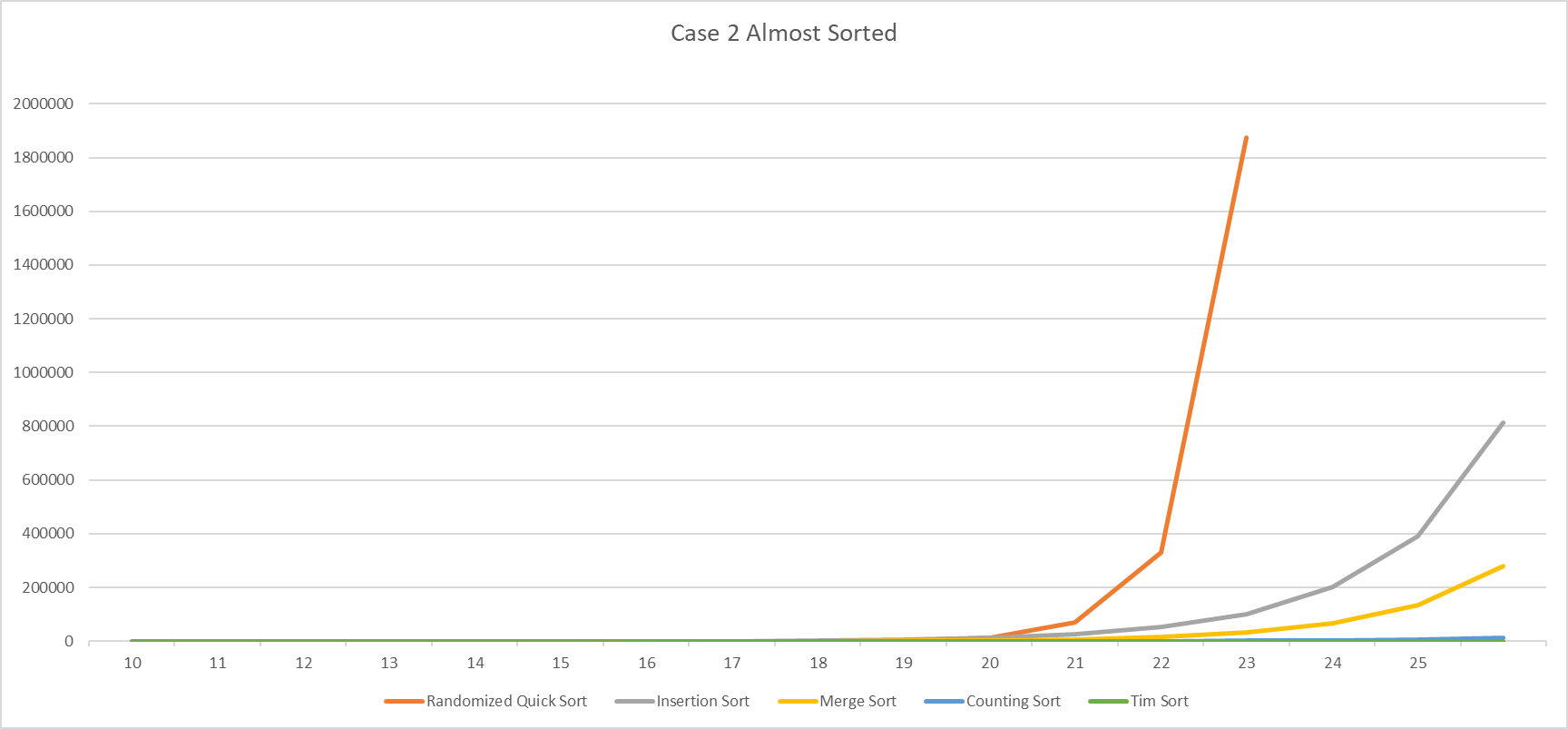
**資料結構 作業一**

**資科二 簡傑**

1. 繪製折線圖





* 註 (一) : 數據的時間單位為毫秒(ms)
* 註 (二) : 本次試驗使用的 Quick Sort 程式碼在 k = 20以後，partition 函式中pivot選擇時頻繁出現問題，故Quick Sort 時間在 k = 20 以後，皆是估計數據。
* 註 (三) : 本次試驗環境在k = 27 以後會出現，Memory Error的錯誤訊息，故本次試驗並無k = 27以後的數據，相當可惜。

1. Sorting 演算法來源
   1. Insertion Sort:

https://www.geeksforgeeks.org/python-program-for-insertion-sort/

* 1. Merge Sort:

https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E6%8E%92%E5%BA%8F%E6%B3%95%E9%80%B2%E9%9A%8E-%E5%90%88%E4%BD%B5%E6%8E%92%E5%BA%8F%E6%B3%95-6252651c6f7e

* 1. Randomized Quick Sort:

https://gist.github.com/mayank11196/3ecfe606763d30e05d23e22a56ab17d3

* 1. Counting Sort: 親自撰寫
  2. Tim Sort: Python 內建 Sort

1. 實驗環境
   1. 系統:

Distributor ID: Ubuntu

Description: Ubuntu 20.04 LTS

Release: 20.04

Codename: focal

* 1. 主機規格:

Cpu model name : AMD Ryzen 3 3300X 4-Core Processor

cpu MHz : 3999.844

cache size : 512 KB

MemTotal : 32862476 kB

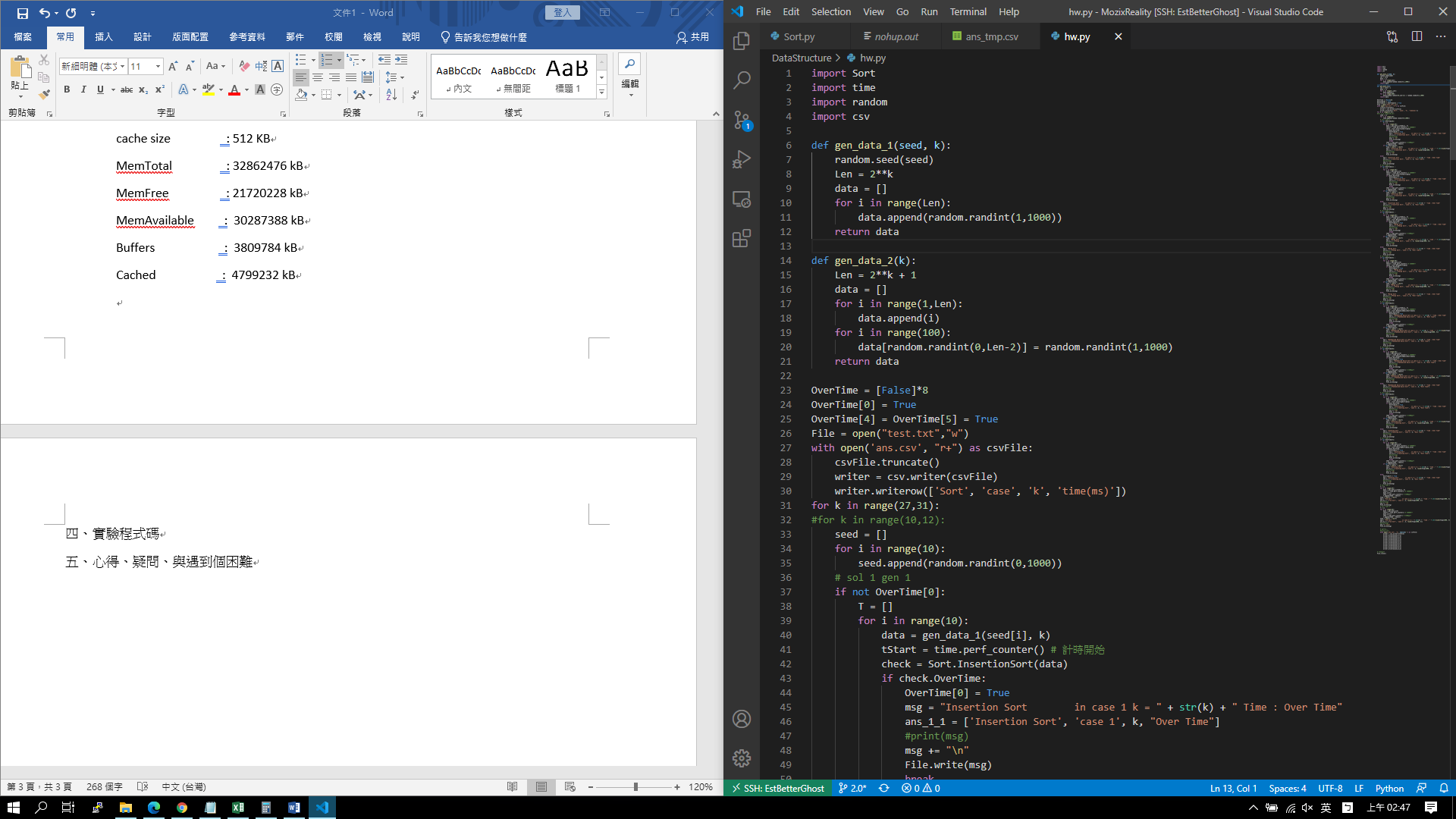
MemFree : 21720228 kB

MemAvailable : 30287388 kB

Buffers : 3809784 kB

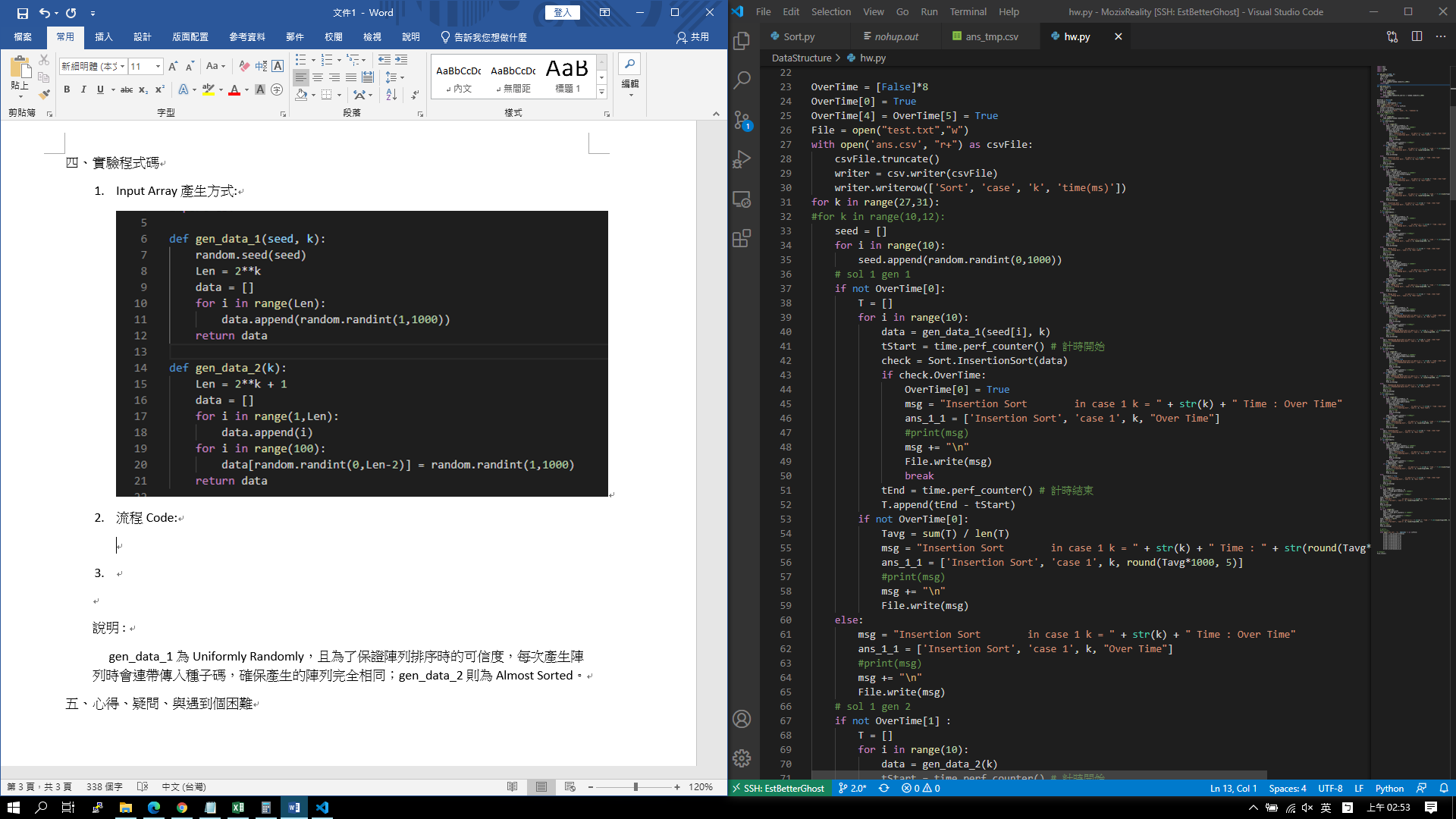
Cached : 4799232 kB

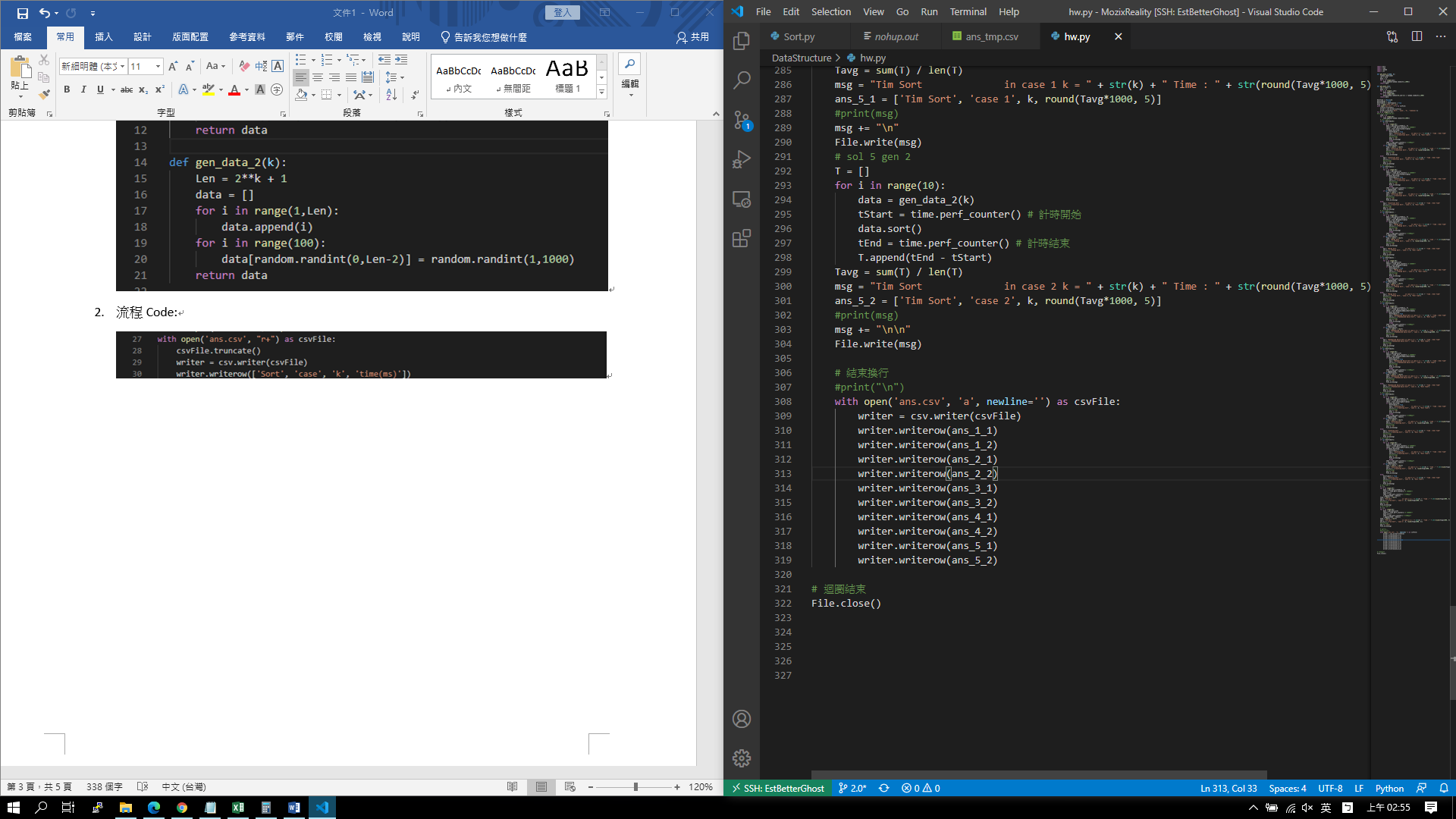
1. 實驗程式碼
   1. Input Array 產生方式:



說明 :

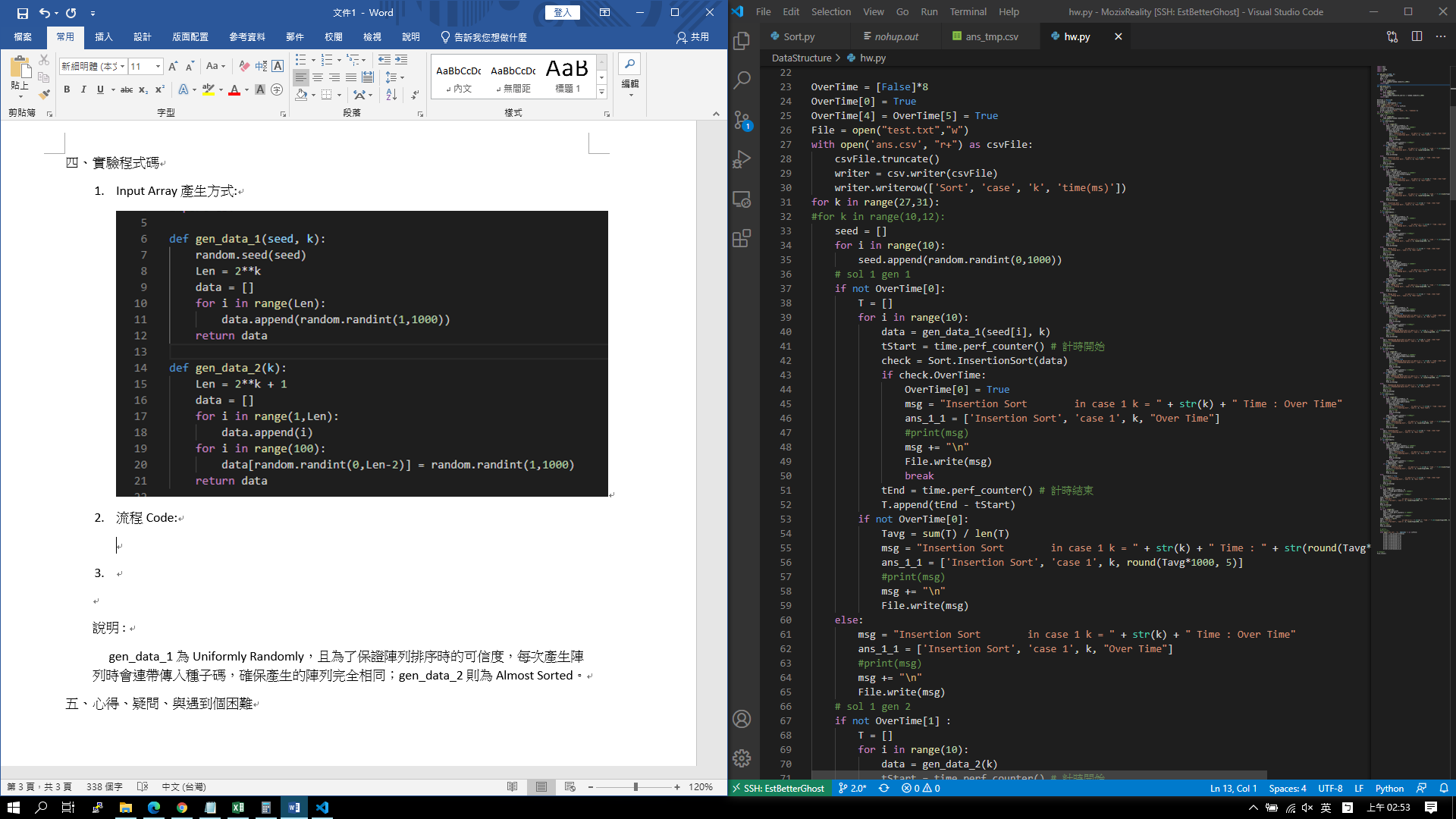
gen\_data\_1為Uniformly Randomly，且為了保證陣列排序時的可信度，每次產生陣列時會連帶傳入種子碼，確保產生的陣列完全相同；gen\_data\_2則為Almost Sorted。

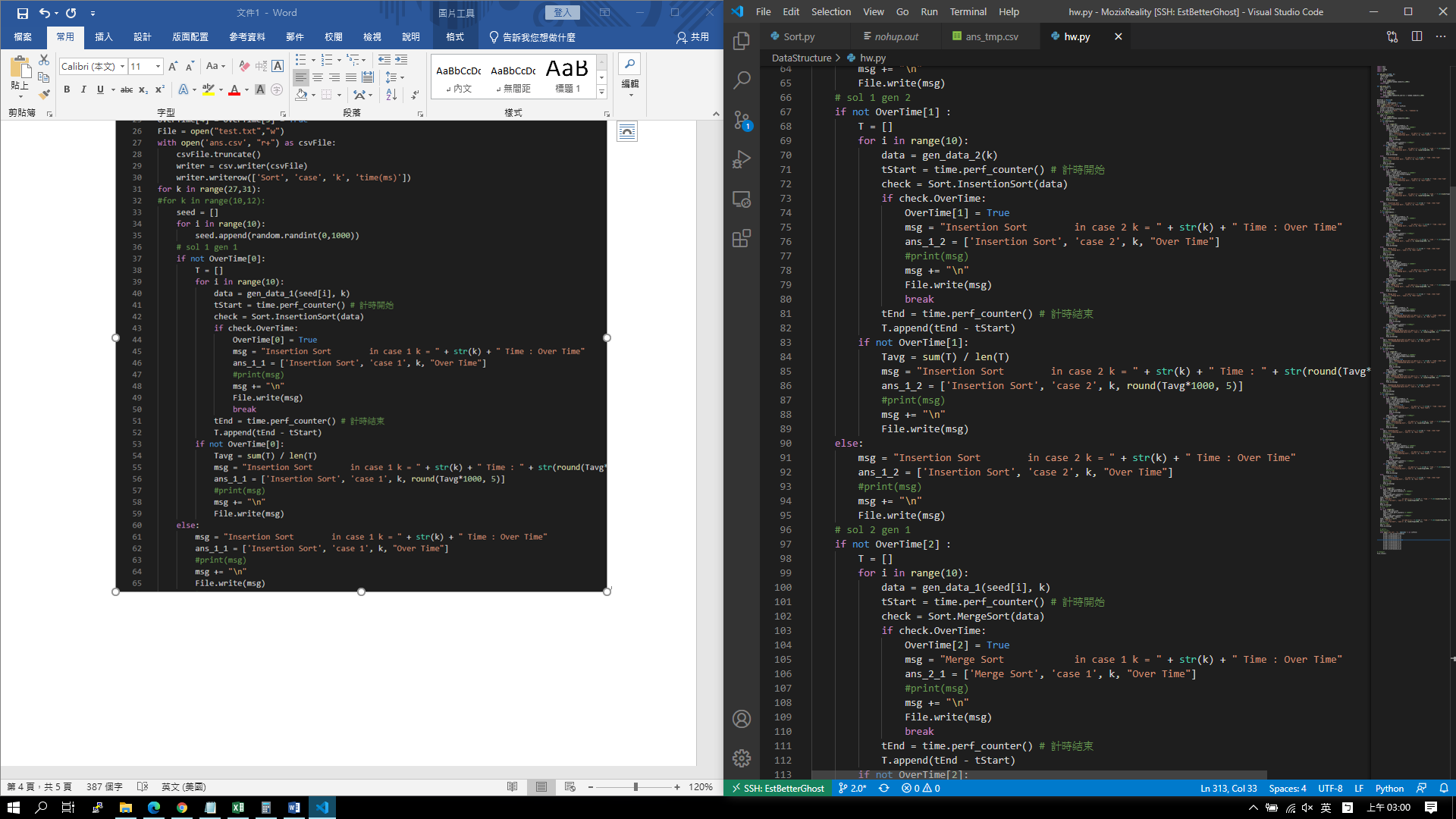
* 1. 資料儲存: 

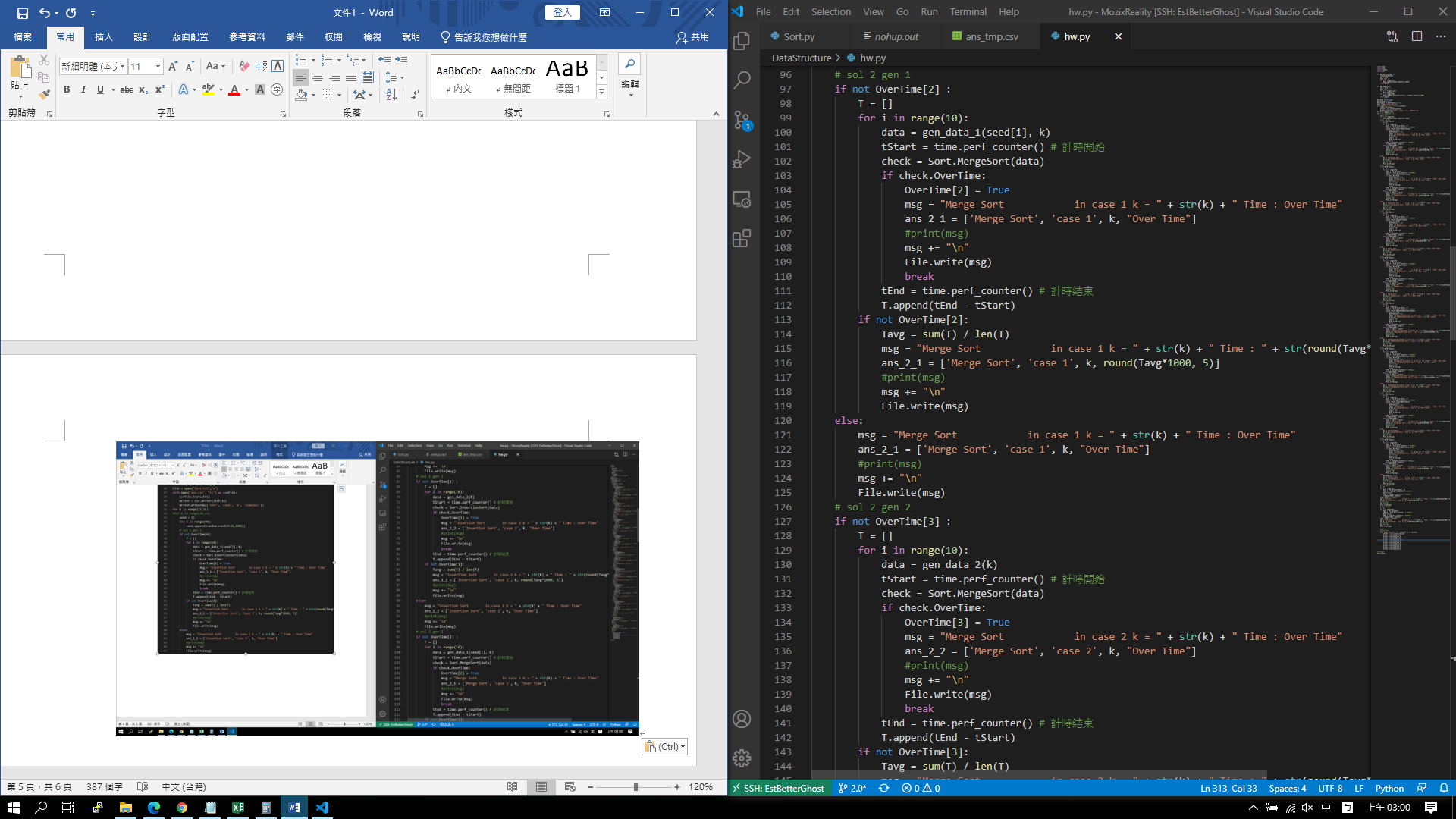


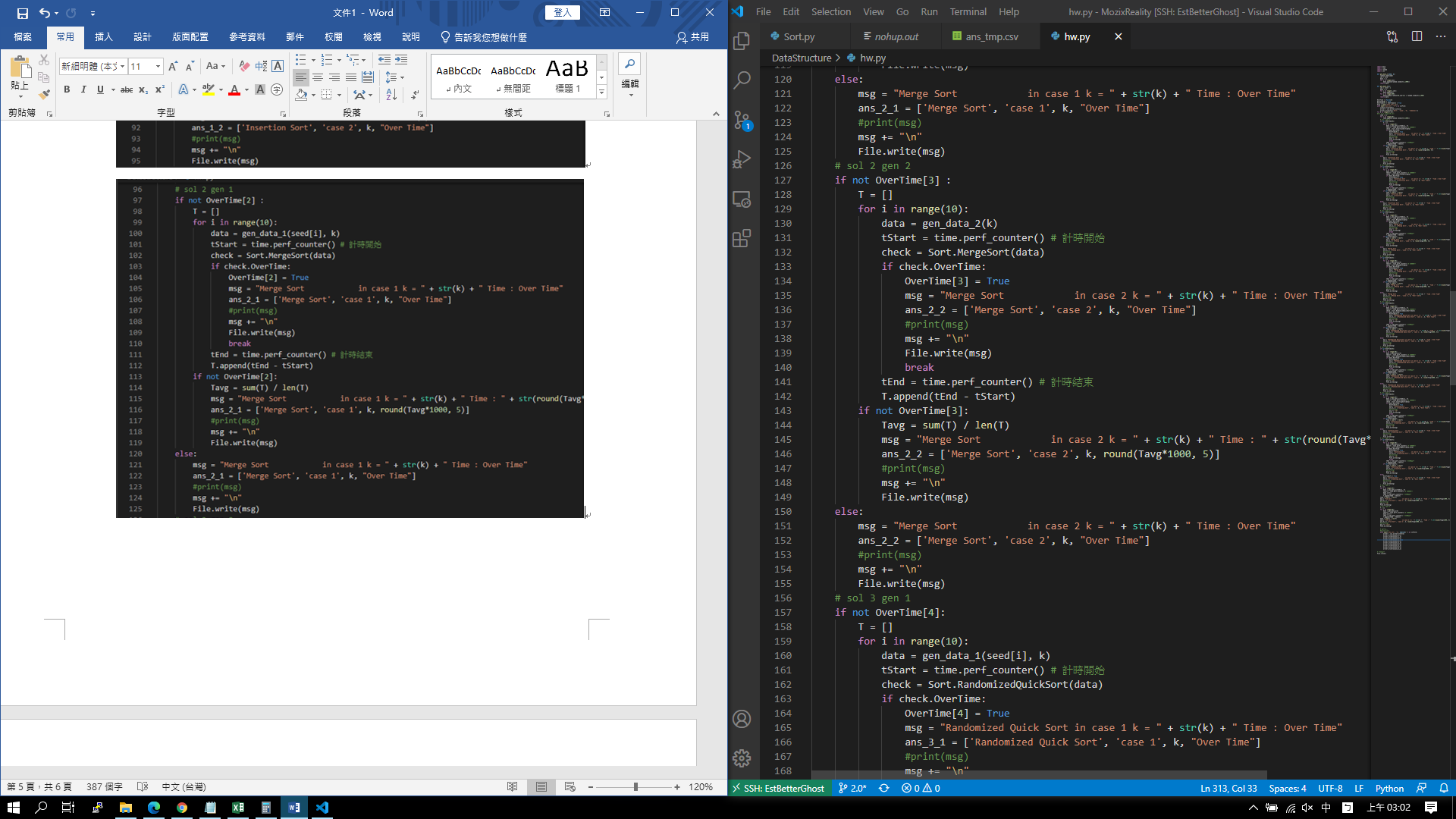
說明 : 每執行k大小的陣列，在5個排序演算法都跑完後，將所記算的排序時間紀錄在CSV檔案裡。

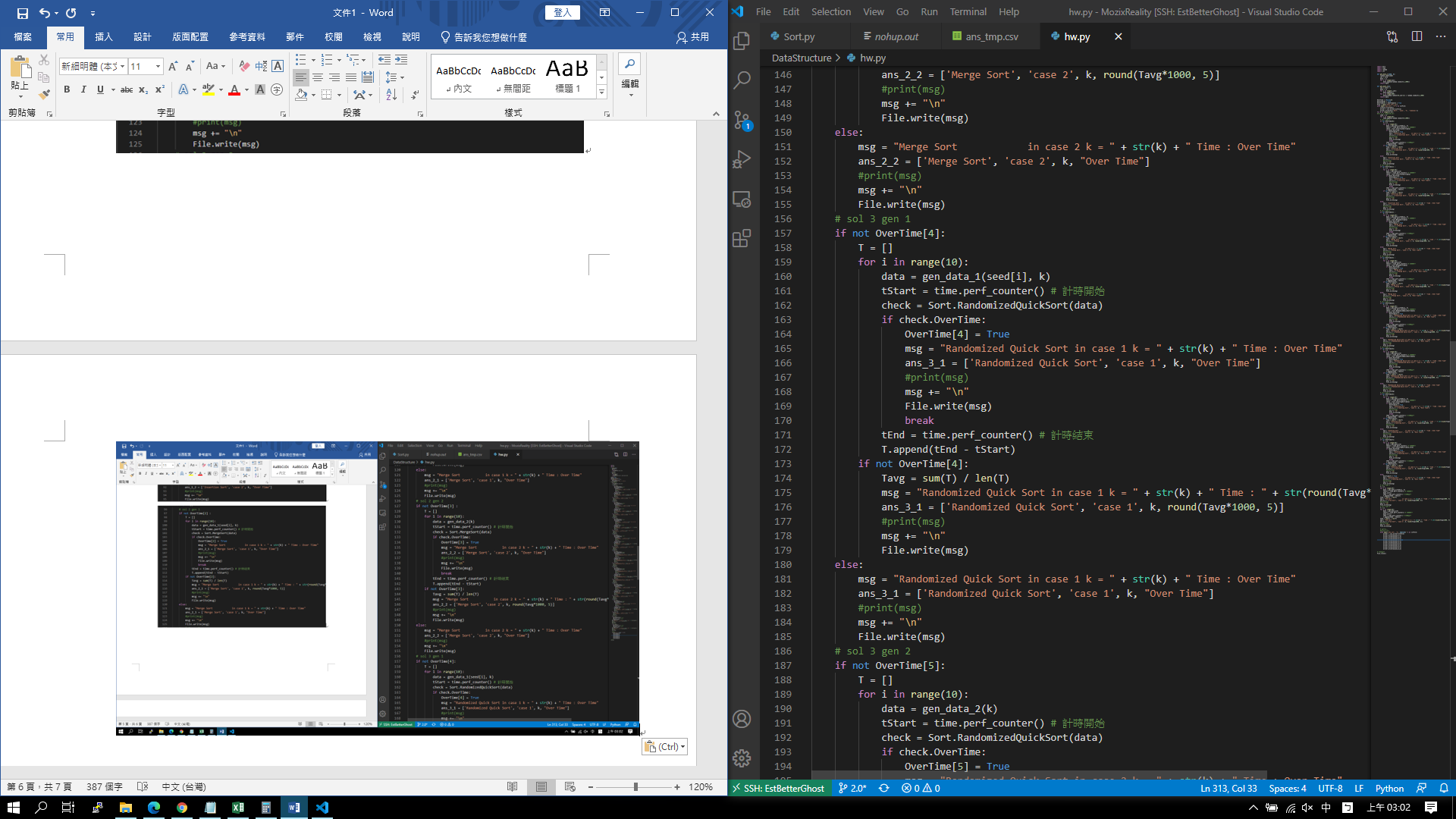
* 1. 流程code:

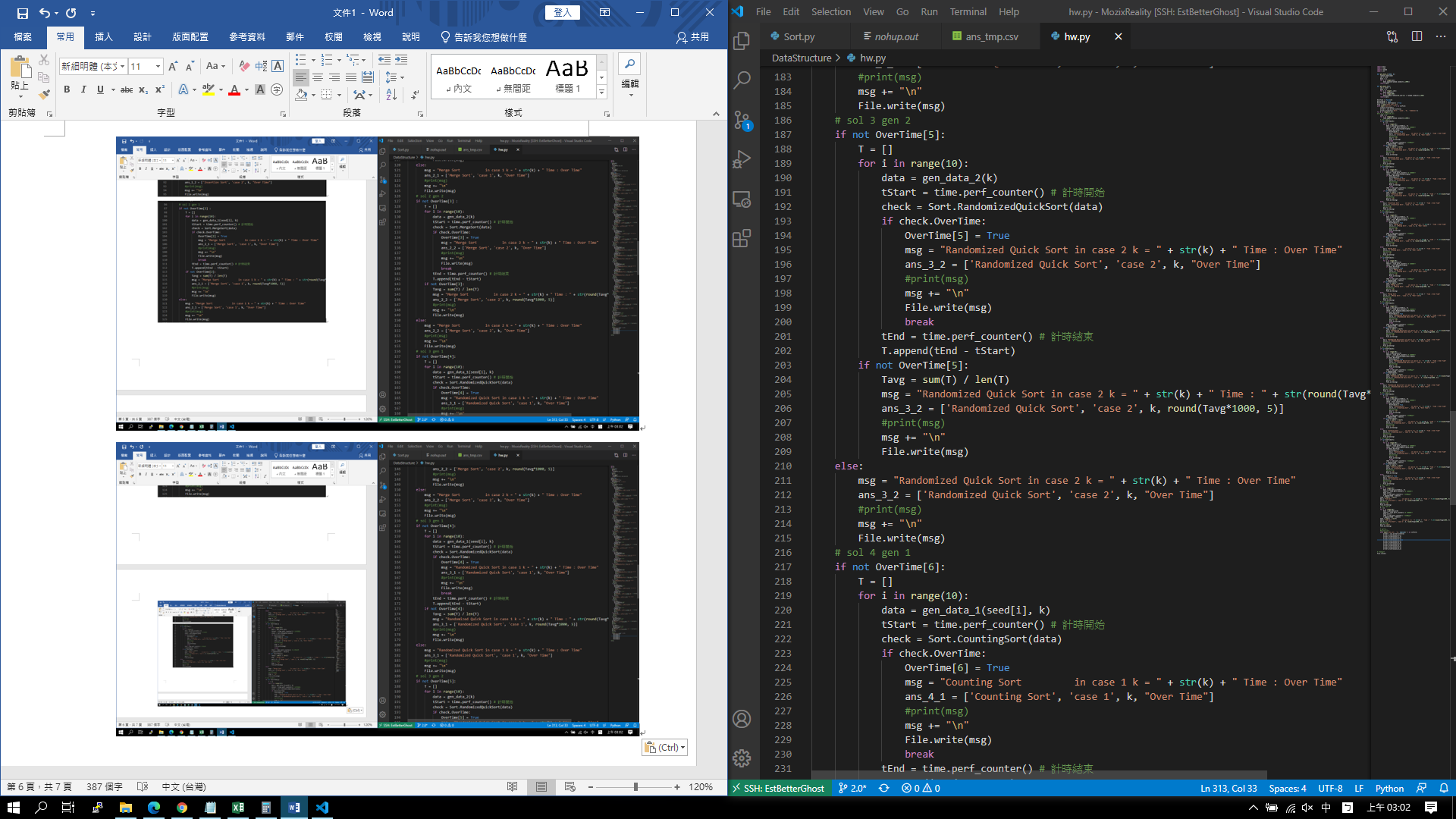


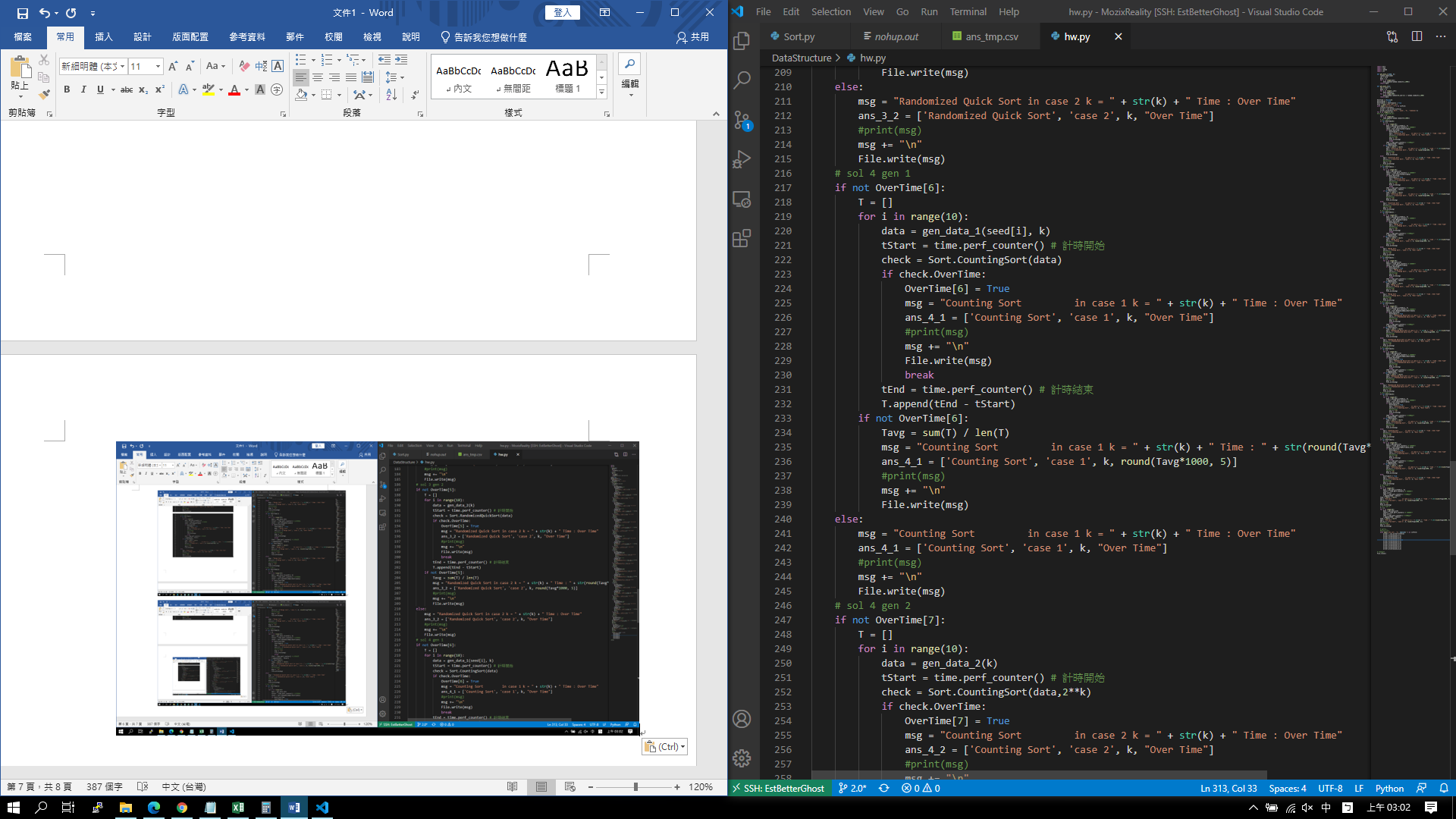


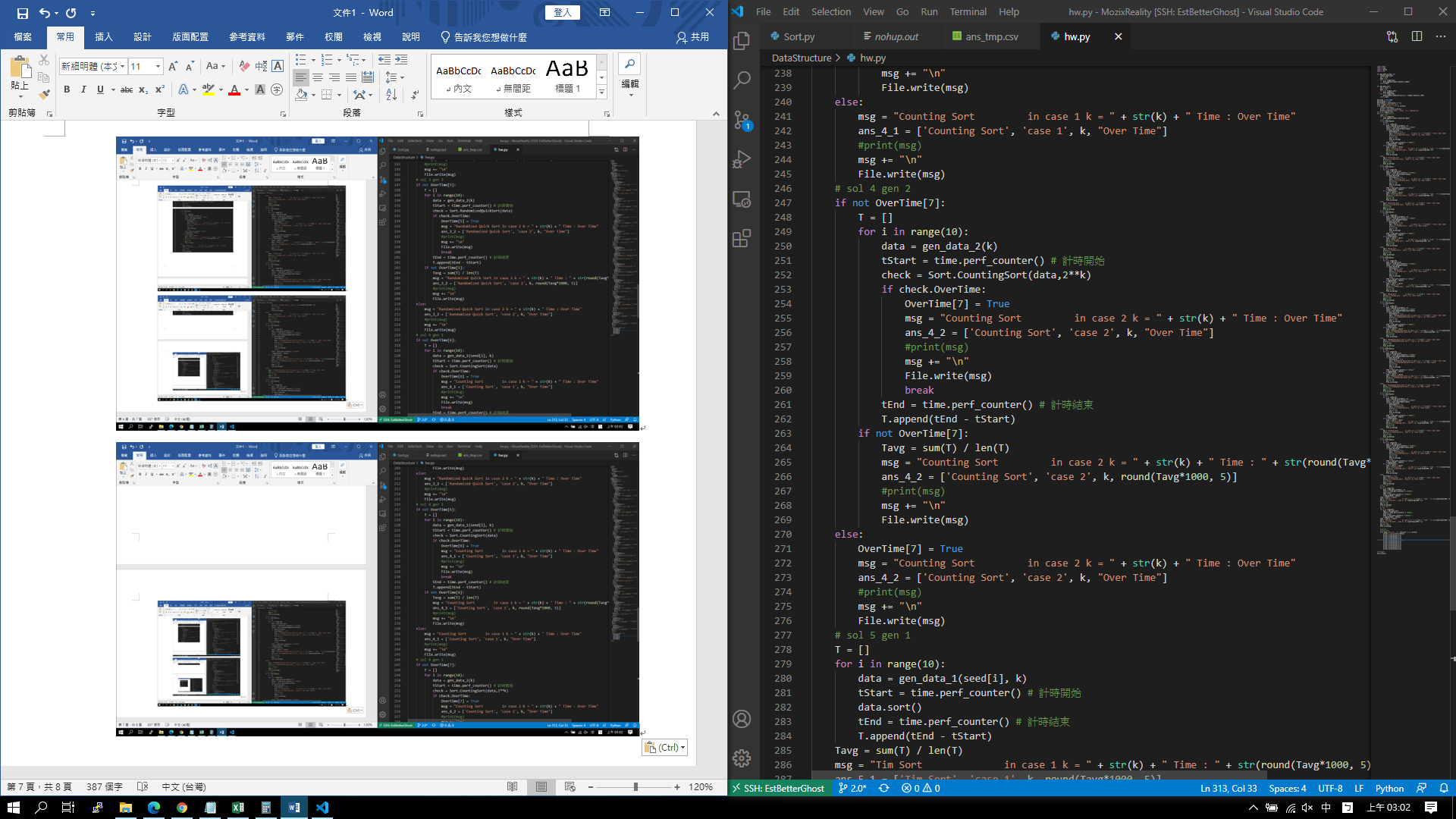


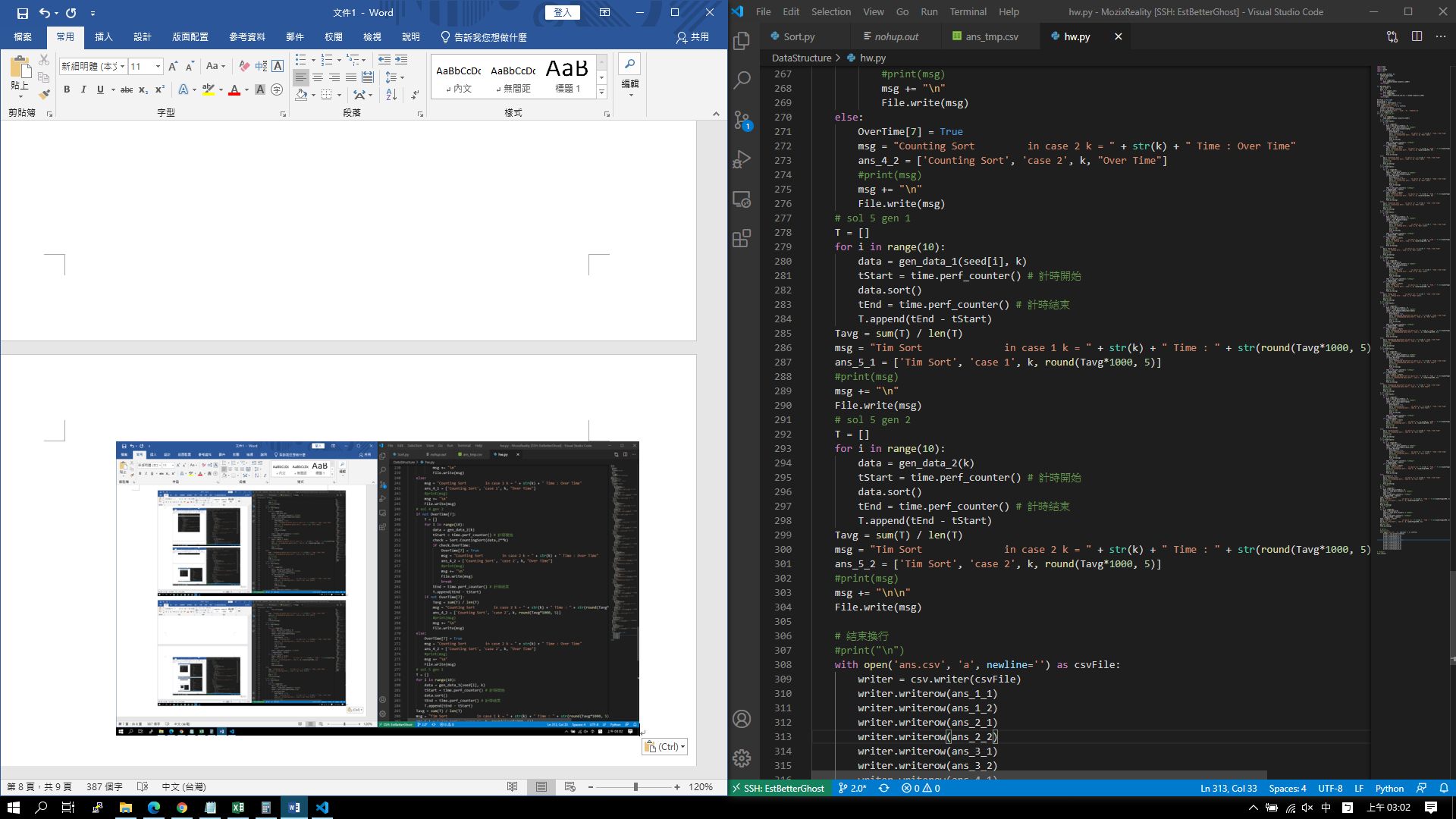












說明 :

由迴圈跑k自10 – 30，過程中，於迴圈最一開始，隨機擲出10個種子碼並儲存起來。隨後再分別跑5種排序、每種排序兩種陣列。

在2種陣列五種5排序的過程中，每次10組陣列排序的時間分開計算，最後加總平均後即會記錄該數值，值得注意的是，在10次排序過程中，若有任何一次超過1個小時，則會因為判定超時，使得剩下的排序動作以及日後k增加的排序皆不會進行。

超時的判定寫在Sort的排序內部，每進行一步動作皆會做超時判定，倘若超時，則會直接返回主程式，並進行超時檢查。

這裡附上 Github 連結，可檢視此次試驗所有 code、csv以及錯誤訊息 :

<https://github.com/MozixReality/DataStruct>

1. 心得、疑問、與遇到個困難

本次作業筆者分成試驗前、試驗中及試驗後三個時間點做討論。

在動手做此次試驗前，筆者考慮使用環境、語言、陣列的產生以及紀錄數據等幾種事項。使用環境選擇同學架設的伺服器，主要是擔心系上伺服器若有過多同時使用，擔心有程式使用效率不彰的問題，加上伺服器系統及規格過於老舊，權衡之下，認為使用同學架設的伺服器較為妥當；其次是選擇使用語言的問題，雖說筆者對C++相對較為熟悉，也認為執行時間較為快速，然而考慮到網路上python時做的排序模板可能稍微較多的情況下，便偷懶的選擇使用python作為本次試驗所使用的語言。不過事後有點後悔，在實作的過程中發現，排序模板沒有想像中的多，數量與C++相差無幾，而筆者對於python相對不熟悉，試驗時出現意料外的狀況也無從處理，這部分筆者稍後再提；再者是陣列產生的情況，筆者以為，10次的陣列產生，若對於5種排序方式都分別以不同的陣列做試驗，有失試驗的公信力。然而，筆者也不希望將過大的陣列放置在記憶體中儲存起來，故在與同學討論之後，決定以儲存種子碼的方式，在每次排序以前重新產生一模一樣的隨機陣列，以達到5種排序每次都能以相同的輸入陣列做試驗，同時這部分筆者檢討自己，Almost Sorted陣列也應當使用相同的方式產生。最後是數據儲存的問題，筆者以為，若僅將程式計算的結果印出在電腦上，再人工紀錄該次數值，此種方式效率太過不彰，因此考慮讓程式於伺服器背景中執行，並再每組k計算完畢之後，直接寫入CSV檔案中，一來提高試驗效率，二來倘若程式途中出現BUG，則筆者可直接追溯CSV檔案中最後一筆k來決定繼續執行之k的範圍。

試驗時，筆者祈禱著程式碼可以正常著執行完畢，但它終究還是不盡人意，首先是程式在k = 20時，Quick Sort中的Partition出了問題，錯誤訊息顯示是在選擇pivot時，randint函式無法正常執行，修改了幾次之後問題仍不見好轉，心灰意冷之際，筆者優先選擇測量其他組數據。然而在k = 27時，程式再度出現問題，意外的是，此次問題並非出現在排序算法中，錯誤紀錄顯示MemoryError，然而筆者所使用的式伺服器裝載32G的RAM，理應不存在記憶體滿出的問題，更何況k=27時尚未是此次試驗的最大值。在反覆檢視了幾次程式碼之後，由於對python不慎熟悉，筆者無法明確找出問題所在，故最終也以失敗收場。

試驗後，筆者最遺憾的非無法將k=30跑完，在檢視同學繪製的圖表後，筆者發現當中許多數據竟是在k > 27以後會有鮮明的差異，因此筆者對於沒有做出最後成果耿耿於懷。其次，是Quick Sort的執行問題，由於k=20以後並沒有Quick Sort的數據，筆者本希望能藉由前幾次的數據推估接下來數次的可能時間，然而推估時卻發現到，Quick Sort恰巧在k=20時與前數次的執行時間有跳躍性的進展，因此在時間的估算上增加了難度，筆者最終妥協在後來數次的執行時間以k=19以及k=18的比值做倍率計算，同時也赫然發現到Quick Sort在理論上來說，複雜度應當是O(nlogn)，但從做圖上看來，表現卻沒有同是O(nlogn)的Merge Sort來的好，同時筆者也見識到，Tim Sort的厲害之處，在理論O(n)的Counting Sort面前，Tim Sort竟然能以貼近線性的排序速度壓過Counting Sort，可見到Tim Sort演算法的厲害之處。而最後的心得則是，本次試驗筆者認為非常有意義，雖然作業時間相當長，工作也相當繁瑣，然而親手做出來的感覺實在是令人印象深刻，比較出了各個算法的理論與實際速度，相信對日後的自己肯定有所助益。