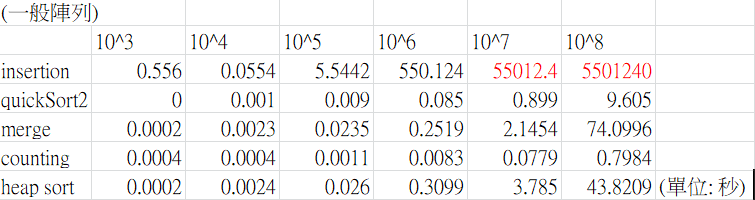
DS HW1

財政四 105205039 方文忠

實驗1: 未排序陣列 (為了更好程現結果，額外附上~10^5 、除去極端值結果圖)

註:由於舊版quicksort似乎有效能問題，我另外找了另一版quicksort替代

數據:



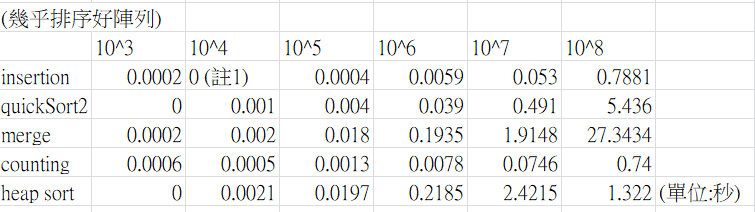
\*紅字為推論時間

\*0為超越小數精度，小於0.0001

結果圖: (縱軸單位: 秒，橫軸單位: 陣列長度)

(1-6 是10^3 ~10^8)

實驗二: 幾乎排序好的陣列(為了更好程現結果，額外附上~10^6結果圖)



\*0為超越小數精度，小於0.0001

結果圖: (縱軸單位:秒，橫軸單位:陣列長度)

程式碼來源:

1. insertion sort來源: <https://www.geeksforgeeks.org/insertion-sort/>
2. merge sort來源: <https://www.geeksforgeeks.org/merge-sort/>
3. quick sort 來源: <https://www.geeksforgeeks.org/quick-sort/>
4. quicksort 2 來源: <http://www.algolist.net/Algorithms/Sorting/Quicksort>
5. counting sort 來源: <https://www.geeksforgeeks.org/counting-sort/>
6. heap sort 來源<https://www.geeksforgeeks.org/heap-sort/>

實驗環境

Windows作業系統

C++ 語言

Vscode 搭配 bash

實驗程式碼

實驗程式碼放在附檔資料夾中

大概實作的部分是兩個陣列生程器

Sorting的部分修改成吃進輸入，然後sorting前後計當下時間，相減得到運行時間

還有一些shell script簡化工作

===========================補充討論==================================

DS 補充筆記 --- 為甚麼quickSort差異這麼大?

**A.概述**

兩個網路上找的演算法效能差了非常多



Source:

表現差quicksort 來源: <https://www.geeksforgeeks.org/quick-sort/>

表現好quicksort 來源: <http://www.algolist.net/Algorithms/Sorting/Quicksort>

**B.演算法差異比較**

演算法1.

定義:

swap()

partition()

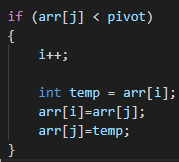
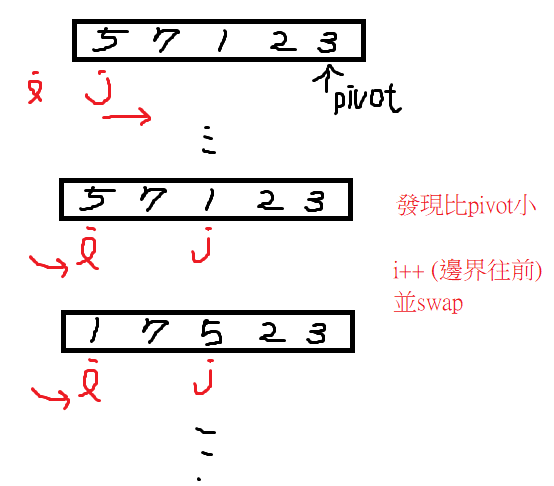
quickSort()

大概邏輯是

\*Partition()決定pivot => (最後一位)

然後關鍵是partition的做法是

兩指標法，一個掃當前位置，一個將紀錄邊界，所以當前值小於pivot，就讓邊界值+1，並交換當前值與邊界值

最後再交換pivot (最後一位)與邊界值即完成partition

\*quicksort()則負責遞迴呼叫 pivot的上下半部

演算法2:

只定義quickSort()

Pivot取陣列中間

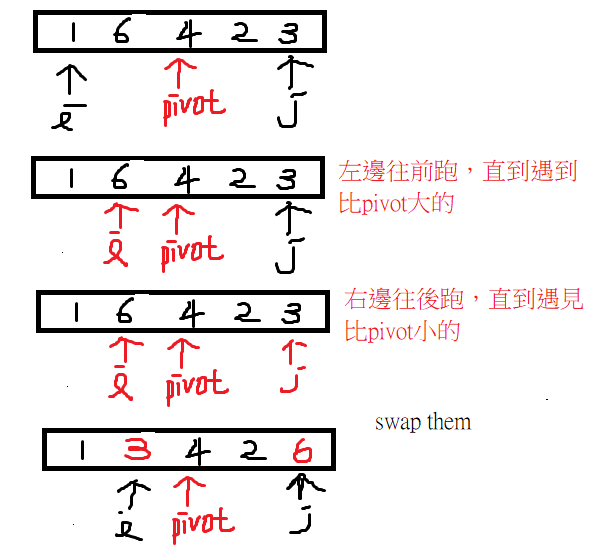
一個指標從前掃到後，一個指標從後掃到前

先讓前面跑到大於pivot的值

再讓後面跑到小於pivot的值

兩者交換

直到 兩指標相會



然後recursive呼叫Qicksort

問題點:

兩者速度差了快1000倍，且演算法1在大資料下會爆成n^2

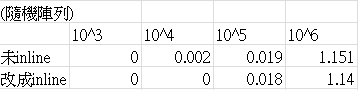
**C.探討:**

可能性1.

前者過於頻繁的呼叫function call? (CPU跳指標因素)

首先加上inline、甚至直接把程式碼merge，改善結果不是很理想

如下圖實驗所示，效能有提升且與資料量成正比，但提升效果十分有限



**小結:** 頻繁call function 的確對大量資料有影響，但相對不是造成此差異的主因

可能性2.

Pivot選擇位置有差別?

理論上此為隨機陣列，選擇哪一個作為pivot應沒有差異，實際結果也證實如此(試過將演算法1改成取中間值、第一個作為pivot，效能沒有提升)，至少在完全隨機陣列下，選哪個位置做pivot沒有差異

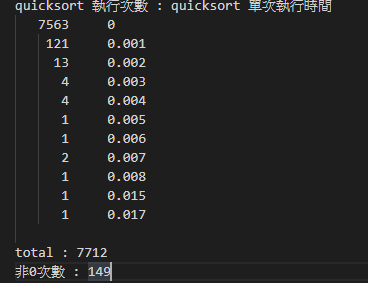
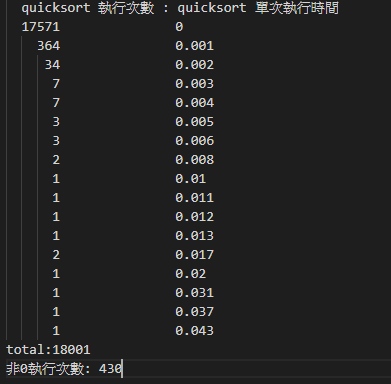
小結: 固定位置當pivot，中間與最後一個只是實作方便，不會造成差別

可能性3.

Partition演算法效能不同?

實驗: 統計每次partition 的時間，並且統計下來

左圖為演算法1，右圖為演算法2



可以比對兩個演算法，發現演算法1 不論是執行次數還是非0執行次數都較多

並且單次排序時間最大值也較多

問題: 會不會是陣列順序導致演算法2比較容易選到好的pivot?

不是，經過重複隨機陣列，兩者依然存在顯著差異

小結:幾乎可以確定是quicksort 不同partition方式會造成差異

因此上網查了資料，發現是真的有兩種partition方式，並且因為swap次數差異而有效能差異

演算法1使用Lomuto’s partition

演算法2使用Hoare’s partition

<https://cs.stackexchange.com/questions/11458/quicksort-partitioning-hoare-vs-lomuto>

差別比較重點:

1. Hoare’s partition 平均省了三倍的swap 次數
2. Lomuto’s partition 在資料幾乎排序好很容易變成n^2