110753201 資科碩一 曹昱維

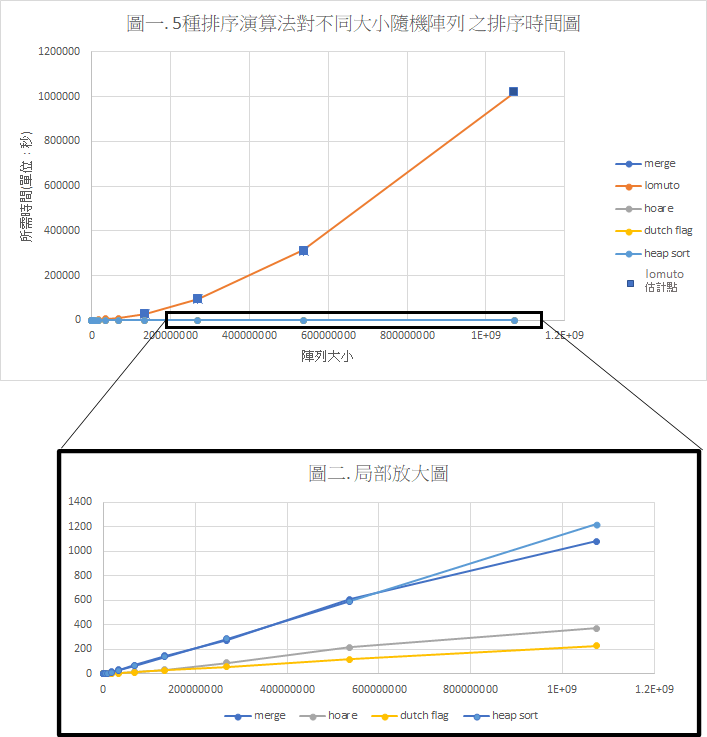
1. **作業目標：**比較merge sort, 三種randomized quick sort (Lomuto Partition, Hoare Partition, 與Dutch Flag Partition)與任意一個時間複雜度為的排序演算法。
2. **比較方法：**計算上述演算法在input array 大小為時（ = 10, 11, 12, …, 30）的平均執行時間。針對每個input array 大小，分別產生10 個input array，最後取10 次執行時間的平均。若執行時間超過1小時，請估計執行時間，並解釋你的估計方法。

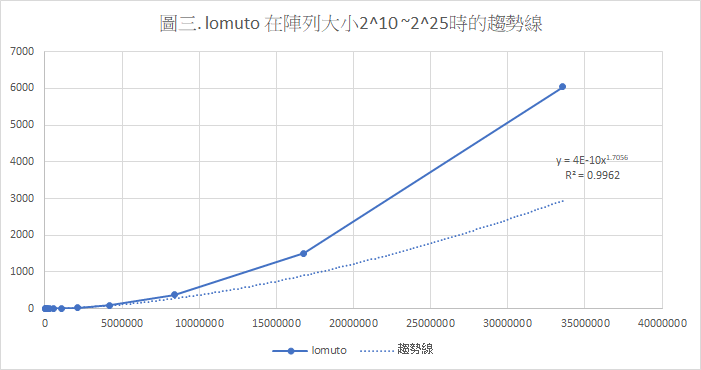
**Input array** 產生方法：陣列的每個數字都從1~1000 中的正整數中隨機選擇，且每個數字被選到的機率都相同（0.001）。

**1. (70%) 一個折線圖（包含5 條折線）。折線圖的x 軸為陣列大小，y 軸為對應的平均執行時間。每個排序方法用一條折線表示其執行時間。請解釋你的實驗結果。**

1. **圖一**即為本次實驗的結果，其中可以看到 Partition 方法為*lomuto*的Quick sort 所需的時間**隨 著陣列變大而遠大於**其他排序演算法
2. 原始數據(如**表一**) ，其中 使用Partition 方法為lomuto的Quick sort在陣列執行到 **2^25 大小**時，平均時間已達約 150 分鐘，即暫停後續執行，並按其在 EXCEL 趨勢圖公式(如**圖三**) y =( 4\*10^(-10))\*(X^1.7056)，推估相關數據(藍底橘字)。
3. 推測 Partition 方法為*lomuto*的Quick sort 所需時間遠大於其他演算法的原因:  
   因為 *lomuto* 在面對**陣列裡的重複元素**時的**時間複雜度是O(n^2)**，而其他演算法在面對**重複元素**時則依舊是O(nlog(n)) 或是 O(n) ，如merge sort演算法就是O(nlogn)，*dutch flag*演算法就是O(n)

1. 因為陣列元素的1~1000隨機選擇，所以陣列的組成會**隨著陣列大小提升而有更多的重複元素。**所以當陣列大小越來越大， *lomuto* 的計算時間就會與其他演算法差距越大，而從趨勢圖公式(如**圖三**) 的X的冪次為1.7056，也可以看出 *lomuto* 的時間複雜度是介於1.5~2之間





|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **表一** | | | | | |
| **陣列大小** | **merge** | **lomuto** | **hoare** | **dutch flag** | **heap sort** |
| **1024** | **0.00124** | **0.000159** | **0.00033** | **0.000225** | **0.000481** |
| **2048** | **0.00192** | **0.0003761** | **0.00039** | **0.000497** | **0.000868** |
| **4096** | **0.00401** | **0.0007916** | **0.00077** | **0.000848** | **0.001929** |
| **8192** | **0.00718** | **0.0018723** | **0.00166** | **0.002069** | **0.004373** |
| **16384** | **0.01318** | **0.0047524** | **0.00352** | **0.003838** | **0.008783** |
| **32768** | **0.0262** | **0.0124647** | **0.00711** | **0.006993** | **0.018447** |
| **65536** | **0.05392** | **0.0364426** | **0.01232** | **0.01342** | **0.038865** |
| **131072** | **0.10525** | **0.121529** | **0.02463** | **0.026401** | **0.081376** |
| **262144** | **0.21017** | **0.435054** | **0.04941** | **0.051149** | **0.172288** |
| **524288** | **0.42909** | **1.61503** | **0.09917** | **0.10688** | **0.366932** |
| **1048576** | **0.89976** | **6.16745** | **0.20045** | **0.198443** | **0.743366** |
| **2097152** | **1.91465** | **23.807** | **0.41774** | **0.405499** | **1.62936** |
| **4194304** | **3.87942** | **94.2771** | **0.88912** | **0.818505** | **3.23672** |
| **8388608** | **7.6567** | **375.717** | **1.88283** | **1.69178** | **7.0998** |
| **16777216** | **17.8069** | **1503.2** | **3.81763** | **3.51543** | **15.2294** |
| **33554432** | **34.5675** | **6038.04** | **8.12911** | **7.05651** | **32.0047** |
| **67108864** | **72.2244** | **9006.3014** | **16.5076** | **14.5443** | **66.9086** |
| **134217728** | **147.835** | **29383.509** | **34.6396** | **31.0774** | **138.695** |
| **268435456** | **273.254** | **95865.165** | **90.5542** | **57.3683** | **285.327** |
| **536870912** | **608.27** | **312764.89** | **218.349** | **120.13** | **594.343** |
| **1073741824** | **1081.05** | **1020411.1** | **372.665** | **228.428** | **1222.23** |

**2. (5%) 每個Sorting 演算法的程式碼來源。**

|  |  |
| --- | --- |
| 演算法 | 程式來源 |
| Merge sort | http://alrightchiu.github.io/SecondRound/comparison-sort-merge-sorthe-bing-pai-xu-fa.html |
| Quick sort-lomuto | https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/ |
| Quick sort-hoare | https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/ |
| Quick sort-dutch flag | https://www.geeksforgeeks.org/3-way-quicksort-dutch-national-flag/?ref=rp |
| Heap sort | https://www.cplusplus.com/reference/algorithm/sort\_heap/ |

**4. (5%) 實驗的程式碼。**

**已將本次作業程式碼上傳至github，詳見下列網址:**

**https://github.com/theabc50111/nccu\_cs\_hw/blob/main/DataStructure\_HW/HW1/hw1.cpp**

**5. (20%) 心得、疑問、與遇到的困難**

* 剛開始看到lomuto的計算時間與其他演算法差距甚大，還一直在想是不是那裏有寫錯，可是在經過幾番測試之後，發現測出來的結果都是有正確排序陣列的，直到很後面才想到會有worst case的問題。這讓我對於即使都是演算法之間的區別有很深刻的印象，同時也對時間複雜度的概念也更深刻了，在上課的時候只知道O(n^2)跟O(n)會差很多，但是在執行作業的當下看到這個時間上的差距才真的體會到用到不適合的演算法會造成多大的影響
* 因為我之前都是使用python所以在設計函式的時候很習慣把函式[[lomuto()](https://github.com/theabc50111/nccu_cs_hw/blob/0ffbf4c860cec9ef83ea08a3bab0758ed45f76a7/DataStructure_HW/HW1/hw1.cpp#L87), [hoare()](https://github.com/theabc50111/nccu_cs_hw/blob/0ffbf4c860cec9ef83ea08a3bab0758ed45f76a7/DataStructure_HW/HW1/hw1.cpp#L106), [dutch\_flag()](https://github.com/theabc50111/nccu_cs_hw/blob/0ffbf4c860cec9ef83ea08a3bab0758ed45f76a7/DataStructure_HW/HW1/hw1.cpp#L132)]當成引數傳到其他函式[[quick\_sort()](https://github.com/theabc50111/nccu_cs_hw/blob/0ffbf4c860cec9ef83ea08a3bab0758ed45f76a7/DataStructure_HW/HW1/hw1.cpp#L171)]中，但是因為C++的語法有限定函式的接收引數的型態，所以我最後的做法，只能另外在 [quick\_sort()](https://github.com/theabc50111/nccu_cs_hw/blob/0ffbf4c860cec9ef83ea08a3bab0758ed45f76a7/DataStructure_HW/HW1/hw1.cpp#L171)裡面分別建立不同的引數用來各自接收函式，然後另外建立空的函式[[partition\_form1\_void()](https://github.com/theabc50111/nccu_cs_hw/blob/0ffbf4c860cec9ef83ea08a3bab0758ed45f76a7/DataStructure_HW/HW1/hw1.cpp#L79), [partition\_form2\_void()](https://github.com/theabc50111/nccu_cs_hw/blob/0ffbf4c860cec9ef83ea08a3bab0758ed45f76a7/DataStructure_HW/HW1/hw1.cpp#L83)]， 然後再引用的時候分別[將空的函式(partition\_form2\_void())傳入](https://github.com/theabc50111/nccu_cs_hw/blob/0ffbf4c860cec9ef83ea08a3bab0758ed45f76a7/DataStructure_HW/HW1/hw1.cpp#L181) ，就只為了符合 quick\_sort()的格式，雖然最後有達到我的需求，在主程式裡面我只需要呼叫quick\_sort()並在其引數指定要用哪一個partition的方法即可，但是這個建立空的函式這個解法應該不是最好的方法，**不曉得C++的語法上還有沒有更優化的作法**，**希望助教或是老師有看到這個問題的話，可以給我一些優化建議或方向**  
  *PS. 上述超連結均連結到github作業區*
* 這是我第一次使用C++，所以剛開始在寫作業得時候花了很多時間在語法問題以及套件使用方式上，但是考量到之後的演算法等課程可能還是要用c++來寫，所以還是決定趁現在開始練習
* 之前沒有修過計算機組織以及c語言等課程，所以對stack以及heap，c語言的記憶體配置等不太清楚，所以也需要另外花時間去了解，但我想後續的課程也許也會講到相關的概念，而資料結構後面的課程好像也會講到stack與heap，希望到時候能跟c語言這邊碰到的東西會有共通之處。