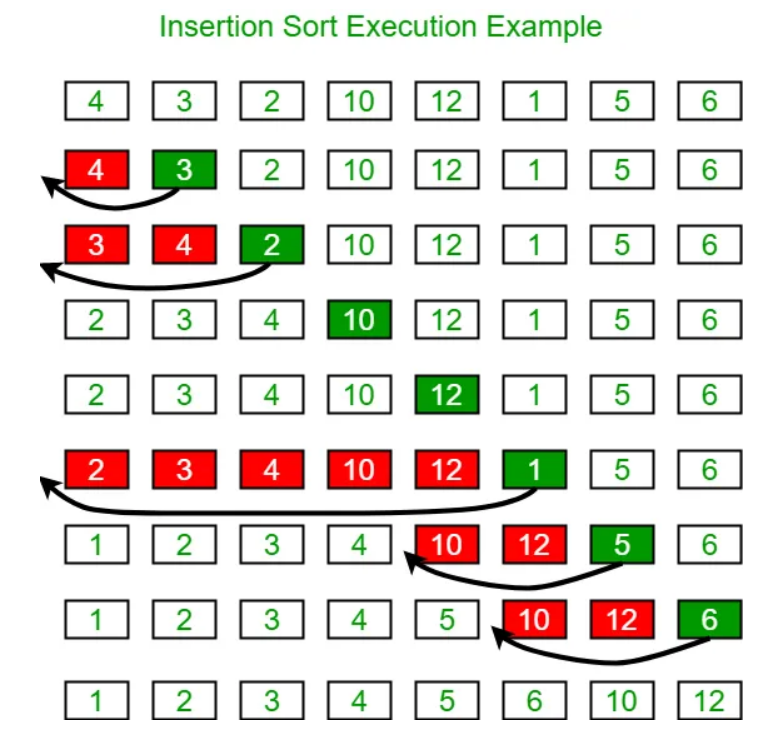
演算法導論 HW1

110511254 徐煜絨

1. 程式架構

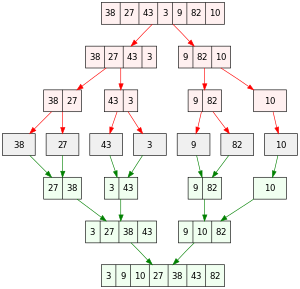
除了主程式，另外還有四個函式，包含在陣列中生成隨機變數、insertion sort, merge sort和merge。

首先是insertion sort，其排序法為預設第i位前已經排好，再將第i+1位的值與前面（[0]~[i]做比較）。若希望最後結果是由小到大，第i+1位的值會被令為key，當key和第i位比較而key<a[i]時，那a[i]的值就會被寫入a[i+1]，而key會繼續和a[i-1]比大小。當key>=a[n]，key會被寫到a[n+1]，此時i=i+1，新的a[i+1]被令成key…，持續重複以上動作直到i=size-1（當陣列大小=size，最後一個位子的編號會是size-1）。



（圖源：https://reurl.cc/qrAv9N）

Merge sort則是把一開始的陣列多次分成兩部分，最後再組合起來。從下圖可以看到拆解的過程是recursive的，拆分的程度取決於程式碼，下圖的base case和我的程式碼都是當拆解到剩一個元素再開始merge。

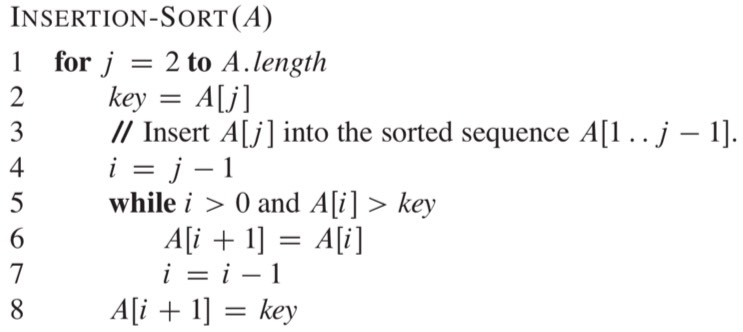


（圖源：https://en.wikipedia.org/wiki/Merge\_sort）

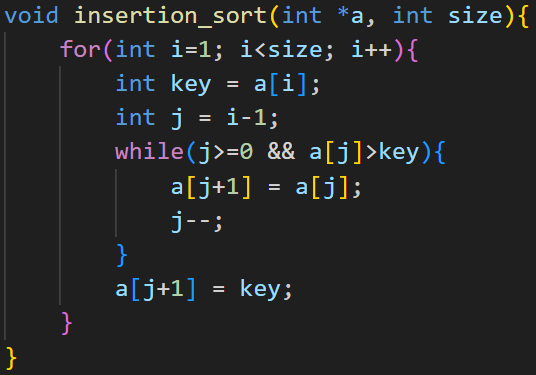
　　最後是主程式，我用srand(time(NULL))以及gen函式中的rand()生成陣列內的變數。在時間計算上則是在呼叫insertion sort和merge sort函式的前後，用clock()/CLOCKS\_PER\_SEC去取當下的時間，再把兩者相減而得。

1. Pseudo code和程式碼比較與解釋
2. Insertion sort

Pseudo code:



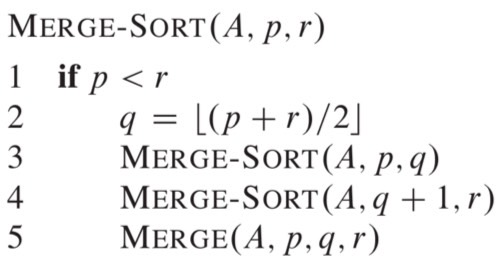
Code in practice:



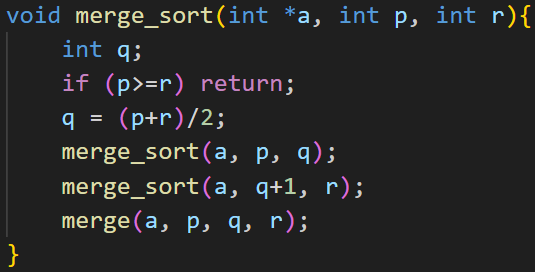
　　兩者的差別在於上面是將a[1]當作array的第一位，而我是把a[0]當作第一位，所以pseudo code從j=2 to A.length，我則是寫i=1; i<size; i++。另外在while-loop的條件式中，跟前面的原因相同，所以當j為0時，也要執行while迴圈。

1. Merge sort

Pseudo code:

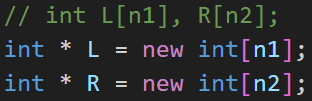


Code in practice:

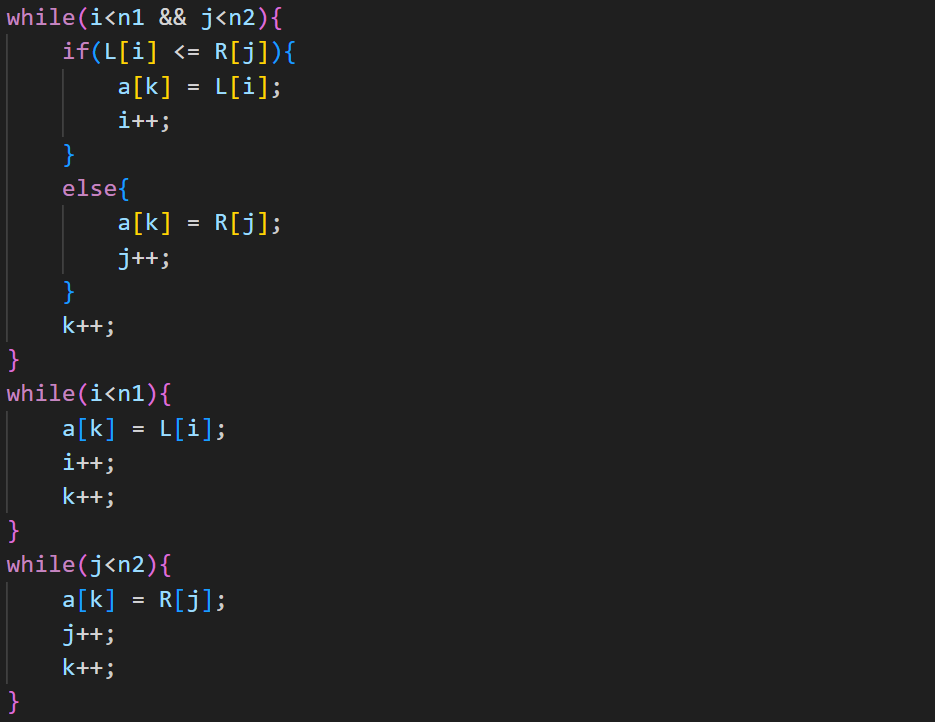


相比於pseudo code，實際上會在有遞迴的程式中加入base case，以merge sort為例，加上if (p>=r) return，可以讓拆分到剩下一個element時，因為p==r，所以結束遞迴並開始merge。

在merge時會因為陣列設定的方式不同，導致執行可能出現segmentation fault，如下圖。一開始是使用第一行做設定，然而當陣列大小達到1000000，在執行時就會跳出警告，原先以為是電腦容量不夠，在查完資料後發現若是以Array\_name[size]產生新陣列，因為是靜態陣列，其大小會有一定限制，所以最後改用動態陣列，並順利解決問題。



　　老師上課提供merge的pseudo code，但因為在寫程式時不確定要怎麼讓L[n1], R[n2] -> ，所以我這裡用另外一種方法，讓L或R其中一方被完全排進A後，另外一方剩下的值能全部進到A。



R被取完，剩下L有值

L被取完，剩下R有值

當L和R都還有值

Merge

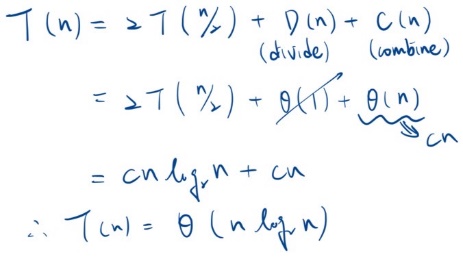
|  |  |
| --- | --- |
| Pseudo | In practice |
|  |  |

1. 時間複雜度
2. Insertion sort

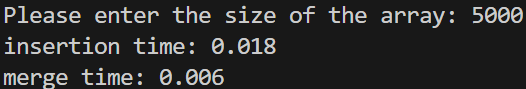
因為有一層while-loop在for-loop中，所以其worst case的時間複雜度T(n) = O(n2)。

1. Merge sort

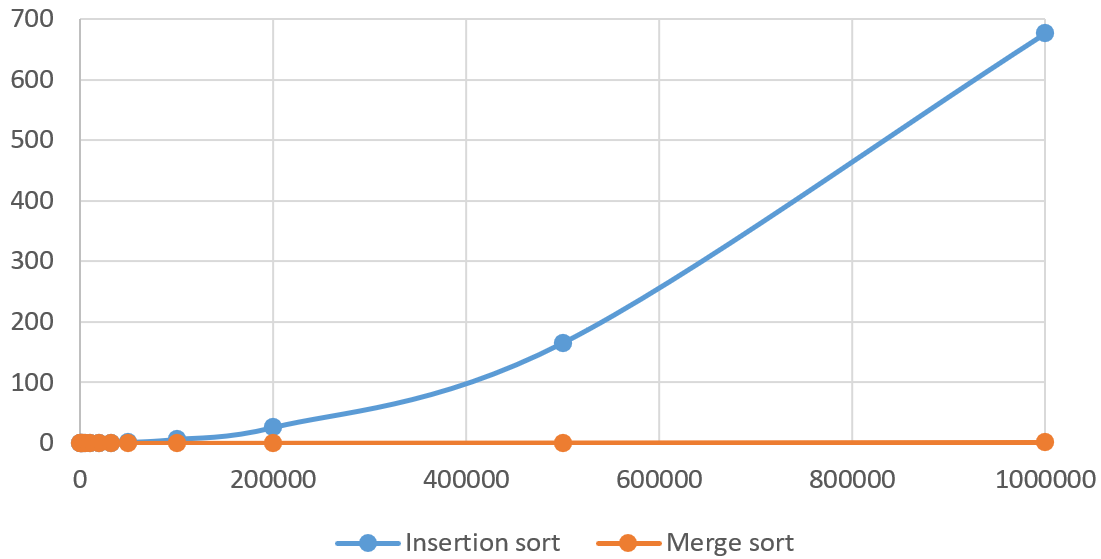
　　Merge sort中有遞迴和排列，因此計算會較為複雜。



1. 實際測試

 <- 輸出結果截圖

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Size of  the array | Insertion sort (sec.) | | Merge sort (sec.) | |
| srand(5) | srand(30) | srand(5) | srand(30) |
| 100 | 0 | 0 | 0 | 0 |
| 500 | 0 | 0 | 0.001 | 0.001 |
| 1000 | 0.001 | 0.001 | 0.001 | 0.001 |
| 2000 | 0.003 | 0.004 | 0.002 | 0.003 |
| 3200 | 0.008 | 0.007 | 0.004 | 0.006 |
| 5000 | 0.018 | 0.017 | 0.006 | 0.007 |
| 10000 | 0.093 | 0.104 | 0.022 | 0.017 |
| 20000 | 0.294 | 0.296 | 0.024 | 0.025 |
| 32000 | 0.716 | 0.691 | 0.04 | 0.043 |
| 50000 | 1.672 | 1.651 | 0.065 | 0.069 |
| 100000 | 6.777 | 6.676 | 0.148 | 0.141 |
| 200000 | 26.351 | 26.152 | 0.278 | 0.275 |
| 500000 | 167.161 | 164.394 | 0.664 | 0.663 |
| 1000000 | 677.967 | 675.519 | 1.365 | 1.348 |



　　上表橫軸為陣列大小，縱軸為執行時間。程式中srand的seed原本是time(NULL)，但為了測試執行時間，我把seed設成5和30並分別測試，圖表中的結果是由兩者做平均所得。Seed設為定值的原因是為了在測試時，陣列內容能有一定相似性，隨著陣列大小增加，陣列會在保有原先內容的情況下被加上其他數值。

　　做成圖後，可以發現其趨勢與上面推導的Θ結果相似，下面兩張圖是nlogn、n2和n的圖，左邊為n[0:100]，右邊則是n[0:1000]。

|  |  |
| --- | --- |
| n[0:100] | n[0:1000] |
|  |  |
| 圖源：  https://stackoverflow.com/questions/23329234/which-is-better-on-log-n-or-on2 | |