



408530014 資管二 陳品洋

---

**quick/merge/heap 排序比較**

---



## 目錄：

開發環境	p.3
前言	p.4
生成資料方法	p.5
快速排序法	p.6
合併排序法	p.8
推積排序法	p.10
實驗三種排序	p.12
總結	p.14
參考與附錄	p.15

# 開發環境

作業系統：macOS Mojave 10.14.6

cpu 處理器：2.3 GHz i5

記憶體：8 GB 2133 MHz LPDDR3

開發整合環境: vim



# 前言

繼上次的排序法實驗，今天要做的事更加複雜的排序法實驗。

有別以往的遍歷 (traverse) 類的排序，這次更著重在於資料結構的觀念以及應用。像是，二元樹，軸的設定，合併資料元組等觀念。

使用的作業系統與先前使用的 linux 系統不同，是使用類Unix的MacOS開發環境，其指令與linux稍有不同，還需多加適應。

在需投入大量時間的學習，開發，希望閱覽此專案的你能夠更加了解程式設計的魅力。

# 生成資料方法

其實，生成資料並不複雜。先設定一個srand(time(NULL))，值得一提的是srand( (time(NULL) )中time(NULL)函數是得到一個從1900年1月1日到現在的時間秒數，這樣每一次運行程序的時間的不同就可以保證得到不同的隨機數了。

因此利用迴圈產生1,000,000筆資料。

若需要產生英文字母，需要注意的是我們必須先得到0~25之間的亂數，將此亂數%26可以滿足。再將此亂數加上字母首位的ascii碼，就得到了一組英文字母亂數。我們只需再將第100位字母執行換行的動作即可。

再者，我們可以觀察每行100+1位字母 \* 1,000,000行，足有101,000,000 bytes (101MB)，以文字檔案來說非常驚人。

SFLBAZIUTYFLHVCVL0000STZZPPUDFJPJQUEUXBKYZKSZJUPDAOXNFGEDUDWDV1RQKSHZTBVXAJPZQVAFBXETUVAHQIUDUGNHEPHUJXCZXVQZJDNZYZTOZHLIJIVRLRYUXRRWGOJIGUPEZOWZZODVOUZYXOKGUKEXTMOMDVBYMYHROUGMMICJPUCWFVCKBJDGFGCIXIEGCDXTKAAEKWGMXQFETDPEJJSGJCOVSBLHQYAMWFXIAEJJTLXQAVHNYBBSBITFXUXFXUYQSAAWENCYSSTYDWLODIQTSEGNJLHTNIOGEISUZPIVYXQNQBOSURGRMWNJSQWC0BPEADXJZB0WPXCDOKIJJWURZMWFXAQVRCQKOEVCHNQJJRSLDWW0ZFGJW0SGCEKDSSAAYMXHGAKRNDZJYEPSNXUUCPKXBDUMJNJQRPX0KCJQXFIPADPMKXAWJNUULGMSFUCYICTXNSXNJOZOZJBJLMGKJSWTSAZABOWKTPNVJCMXLGGHFKDEDBVFAQPDHWLSQSNOHPUNTHVTLDZVPLHSWKVYTRORNMESITHGNHXQXVAWSXJJCSPTKSWJGGZWHFHNELFCWRNJJFTKYRZZUKLELMRUFDBJFNGNXMBYHTNBTFIDHFBIZQNQPFAUBCTOWUFXSRBONWFHBNELEXIFWAJVQZWJCIUITUDW0VJCGBXCBCUDXHEPVQKIITFXCPLPKJEDPBVKGSQMSIRWVZYMRAJTMOLEXLOKJTHNGIHQBGGAUWYKSTSPXCGIZSVNDZQWICBUMGHBZBULEPQHPQHLGQNDPTVAXRXFUZYQSEGHOQEJWYN SZUYTNXCGJOLMAQJOPGZEVMPGMZVXZPVR0ZFIIDXHXCWTLWREPQVJHFQZELWXSIVPYKTRIVTRYQQXCVcerJDANUXNTNCBLHBDUEUJAAFDNQAMRNCTZCCCEYUSVXMIOELVPNDQIAJVXLCNCTDUVRAWRASACNDZCFXZNLUFNEOJTDBIBEKYRLWNXIWSRMTKCUBYWKXUVS1MYRZMCYXLYLFVZACETGWSAATVPANKAQODOFQUYTTGPJQDRSYPKTMKNOLKKYVSSHEIWCQRNLDECRVPGUEQZNDYAGXAFNCQBTRBEIYWTIJVDPYCEZGMMBFLOKOZATCMZTDPOLWAEZOLLEOTFWUUNJWUKZIWTAKYUOHFBG0PUVFNFKNRJ0FRNMXTMMUNVCGIJMGCWXKRIPAVLGKPSMGXNUKTCYMUORSESZKVKTRRHLNWJVEEBNKAIZVJEICCOKNTNTSJXXDDMGYZAPPFBSENVDNSFFKRQPOGDZYKLOXIKNRVDSLUPBUUGEMJVVRXKWRAZQPAODSCZZPBDAWNACCHCUVRVCMLCNKLUAEJQSKBIGIFVTUQLBUUXEV0WRMNBCXNKCDYJWCPBDJHNKTCRVPEJMUFGLUJZRAQJHLHNMHYQGQNDKVZKWMTEVWOFLEPMUHOZMWHRAHLIDBYGJOYTCJJAYBNJFONDVHJBGGKZTLMVTAKBXUGXYKVMOFDSOJEMRPQ0JQBMSSHQOGUJLMZAAGUZTLCVHCSKEVYYRLRGWOCGXHKQLHYISZMBQPIRNQMXQZDNWRSXOLBILXUWYAYHTLDTSXQPTHVTXGYQRJXW0GFQYNIFIUYHSBEGVKGTSJLXUEGLK0MHXVAXCDGUWHPOYHEGSNJECUUPYJGAIERTYUCWNUDQEEDEHJBFLHAFZHEELSTPTZCATIHLSMNDYFGRGOXFYRFGQNECNN0WNLORBXIIQGRSMTUWQHSWGKWANWFGEEXGGABNIKWBEZASXLBLQEMDCDYSMYHOZHDZRYNXGCMINYBNBUOLECIVHNNKUMCFREKTFIAZITEDKQAMZQBYTSKRGANTIERXPAPINIGMZTPKNXKGSPABAZVHLYXPLSXDDWBGNZPOHDNUJMFATVAJQVKJ0HDIZFMHTGSXPA0PFBGDZLCCAVNDSEQKOCSCWQKODOUBGADLCPPNBVYNVQYGCBSQIPVTVTCBKHCTSETFSVYTDWNJASDMRYFKSRMCMFEYVATWXNYPWPVGASOCUUQBVGZWTZKSCIEUBEZBWJBHKBUYGDINRLXUZARXQZFNKTRUDPAFPDQYQWYSCXTWPTQRHPGZTHBGMYAWAMCMKYEZAMZQZMDIYJMRZBZKZLMPBCVNFTNWBNLENPMTEICMGDBBDOEGDNJZADEUCKBVJDSBZFGQYC5ZLWGGCVAJATWPTCCQWFHUIPALQYVVVOYBLSPNBBLTWFKEGTCHABIVVFCYLNGLHMRWZNUNYKSVYXQWMEFMKGJMRXLJADGYCSRPNYOTYEZDJTCKOWJFNHFKPKUOVSNPLSJD

# 快速排序法

快速排序法，其概念是先設定一中心軸 (pivot)，比其大的放右邊，比其小的放左邊。如次循環可以完成排序。

**最佳時間複雜度：** $O(n \log n)$

所選pivot正好是中位數

**最差時間複雜度：** $O(n^2)$

所選pivot是最大或最小

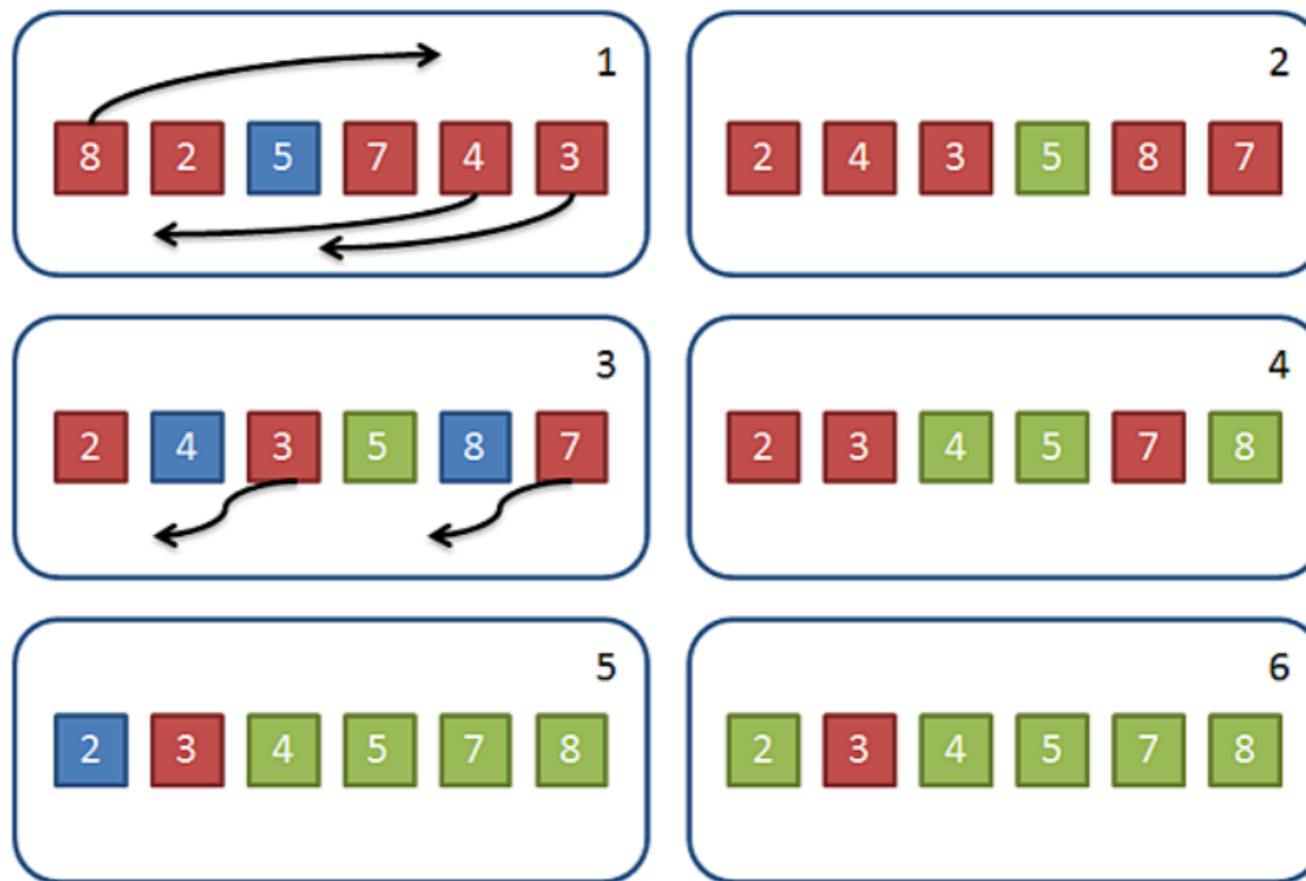
**最佳空間複雜度：** $O(\log n)$

遞迴呼叫的深度為 $\log n$

**最差空間複雜度：** $O(n)$

遞迴呼叫的深度為 $n-1$

**穩定性：**不穩定(Unstable)



- 快速排序法的空間複雜度依實作方式而不同
- 遞迴呼叫需要額外的堆疊空間  $\Rightarrow$  因遞迴的深度而異

# 快速排序核心

```
while(1){  
    while(i<=right){  
        if(arr[i]>pivot){  
            break;  
        }  
        i++;■  
    }  
    while(j>left){  
        if(arr[j]<pivot){  
            break;  
        }  
        j--;  
    }  
    if(i>j){  
        break;  
    }  
    swap(&arr[i],&arr[j]);  
}  
swap(&arr[left],&arr[j]);  
DQuickSort(arr,left,j-1);  
DQuickSort(arr,j+1,right);
```

(沒超過20行) 設定左右指標。左指標往右，右指標往左，一一對比pivot。

左指標得到一直大於pivot的值，右指標得到小於pivot的值，交換。

用迴圈將大於pivot的放右邊，小於的放左邊。

將pivot的值 (arr[left]) 與重疊的左右指標交換。

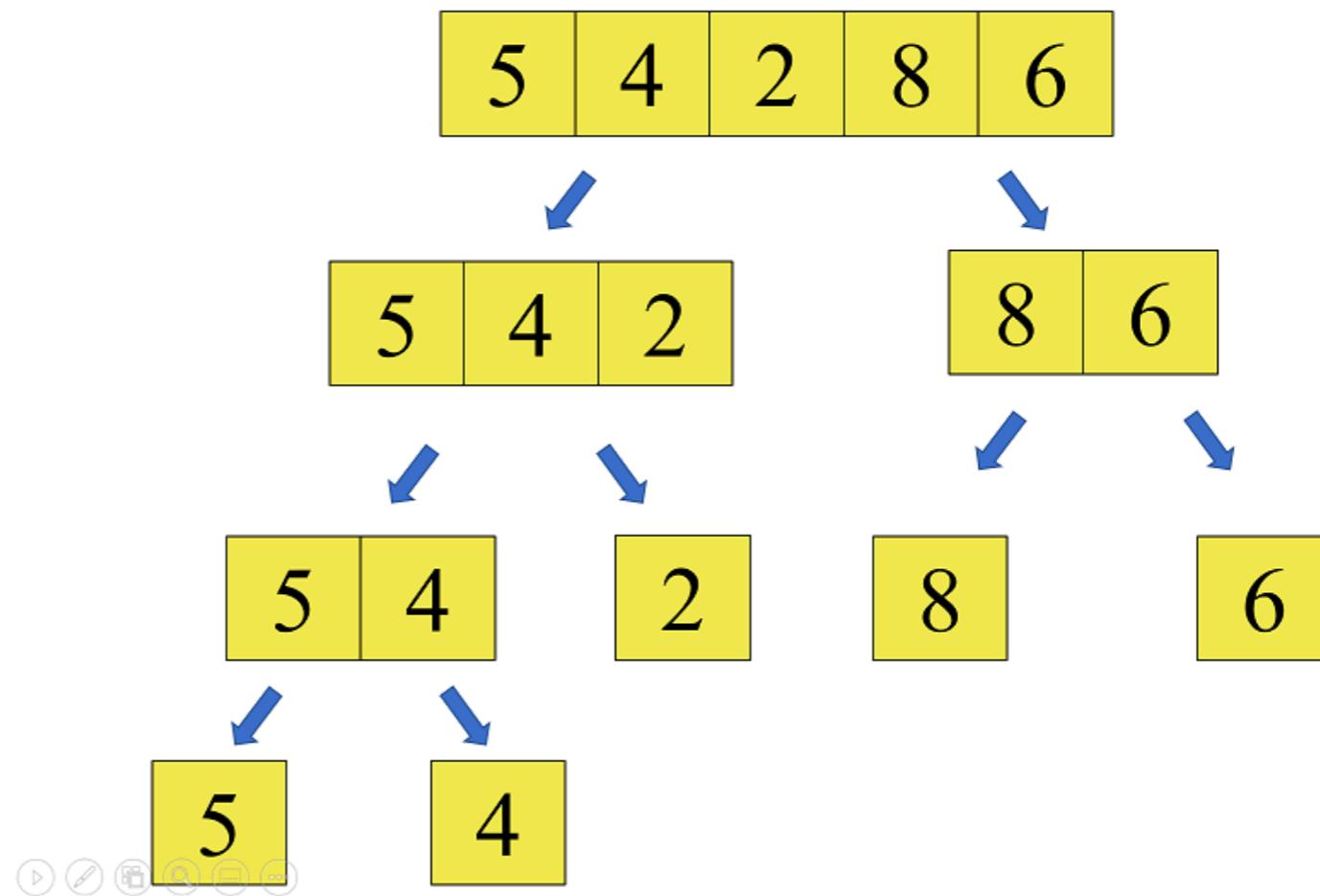
分左右子陣列重複上述操作

# 合併排序法

合併排序法分為2個部分：拆分與合併。

拆分：先將大陣列分為兩個小陣列，再將小陣列各自切一半直到每個小陣列切到剩一個元素。

合併：排序兩個只剩一個元素的陣列並且合併，把兩邊排序好的小陣列合併並且排序成一個陣列直到形成一的最終大陣列。



最佳時間複雜度： $O(n \log n)$

最差時間複雜度： $O(n \log n)$

空間複雜度： $O(n)$

每回合需要儀暫存空間存放合併的結果

# 合併排序核心

```
int mid = (low + high) / 2;
DMergeSort(arr, low, mid);
DMergeSort(arr, mid + 1, high);
merge(arr, low, mid, high);
```

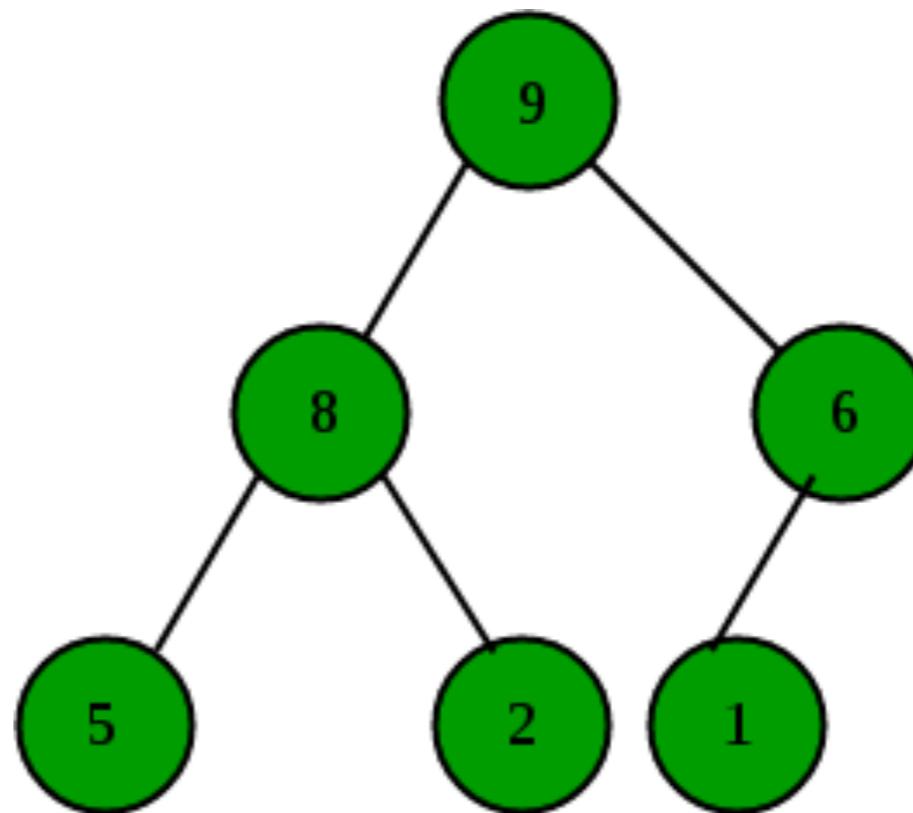
將陣列分割，至單一元素，接著套入合併函數。

```
for(k=0; left_low<=left_high && right_low<=right_high; k++){
    if(arr[left_low]<=arr[right_low]){
        tmp[k] = arr[left_low++];
    }else{
        tmp[k] = arr[right_low++];
    }
}
```

從兩種陣列的第一項開始比較，較小的先放入暫存空間，較大的放右邊。利用迴圈執行至合成一個較大陣列。

# 堆積排序法

以max heap為例：重複從最大堆積取出數值最大的結點(把根結點和最後一個結點交換，把交換後的最後一個結點移出堆積)，並讓殘餘的堆積維持最大堆積性質。我們可以以一維陣列建構二元樹（每一父結點n，都有 $2*n+1$ 與 $2*n+2$ 子結點），每一個父結點都有兩個子結點，目標在於建構完全二元樹（每一個父結點一定大於子結點）。



時間複雜度： $O(n \log n)$

建立MaxHeap :  $O(n)$

執行 $n-1$ 次Delete Max :  $(n-1) \times O(\log n)$   
 $= O(n \log n) \Rightarrow O(n) + O(n \log n) = O(n \log n)$

空間複雜度： $O(1)$

原地置換

穩定性：不穩定

# 堆積排序核心

```
int child1 = 2*i+1;
int child2 = 2*i+2;
int max = i;
if(child1<n && tree[child1]>tree[max]){
    max = child1;
}
if(child2<n && tree[child2]>tree[max]){
    max = child2;
}
if(max!=i){
    Swap(tree,max,i);
    heapify(tree,n,max);
}
```

將三個為一組的二元樹，製作成完全二元樹，即父節點最大。利用迴圈將所有“堆”完全化。（確保子節點不會出界child<n）

找到最大值後，拿最大值與父結點交換完成分堆。

```
void DHeapSort(int *tree,int n){
    buildHeap(tree,n);
    int i;
    for(i=n-1;i>=0;i--){
        Swap(tree,i,0);
        heapify(tree,i,0);
    }
}
```

將root與最後一節點交換，並單獨拿出來。且堆化除了剛剛拿出拿的節點，重複執行。

一一拿出，產生一個完成的排序。

# 實驗三種排序法（1） 數字排序

快速排序法：cat database1.txt | ./hw11 -d -q 單位：毫秒

```
using time:153616  
using time:152954  
using time:153901  
using time:152577  
using time:153492  
using time:152937  
using time:153993
```

合併排序法：cat database1.txt | ./hw11 -d -m 單位：毫秒 推積排序法：cat database1.txt | ./hw11 -d -h 單位：毫秒

```
using time:284138  
using time:287928  
using time:282595  
using time:289659  
using time:287792  
using time:288218  
using time:290523
```

```
using time:330849  
using time:338695  
using time:338695  
using time:349420  
using time:333605  
using time:368887  
using time:336594
```

# 實驗三種排序法 (2) 字串排序

快速排序法 : cat database2.txt | ./hw11 -s -q 單位 : 毫秒

using time:410810

using time:354116

using time:354431

using time:357399

using time:358734

using time:361320

using time:364579

合併排序法 : cat database2.txt | ./hw11 -s -m 單位 : 毫秒      堆積排序法 : cat database2.txt | ./hw11 -s -h 單位 : 毫秒

using time:1604925

using time:1408687

using time:1421874

using time:1563179

using time:1484731

using time:1555590

using time:1419886

using time:828202

using time:869492

using time:1126457

using time:852855

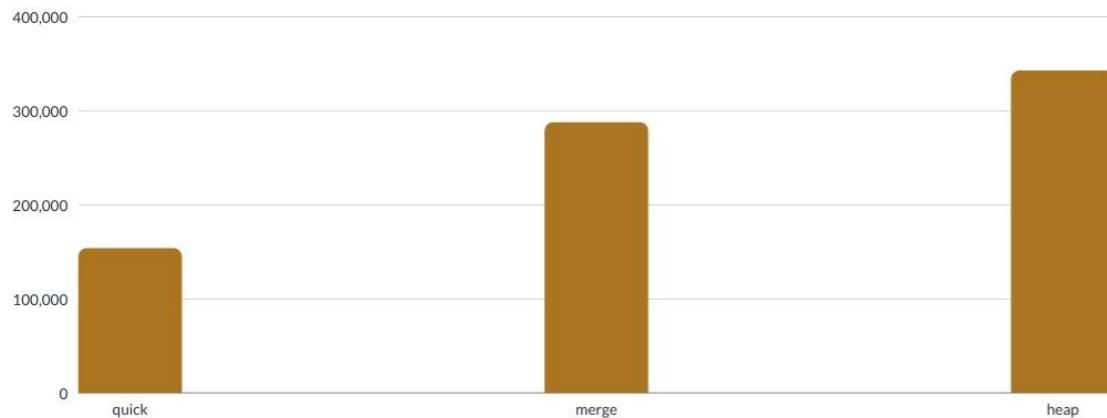
using time:1037377

using time:1033186

using time:1078000

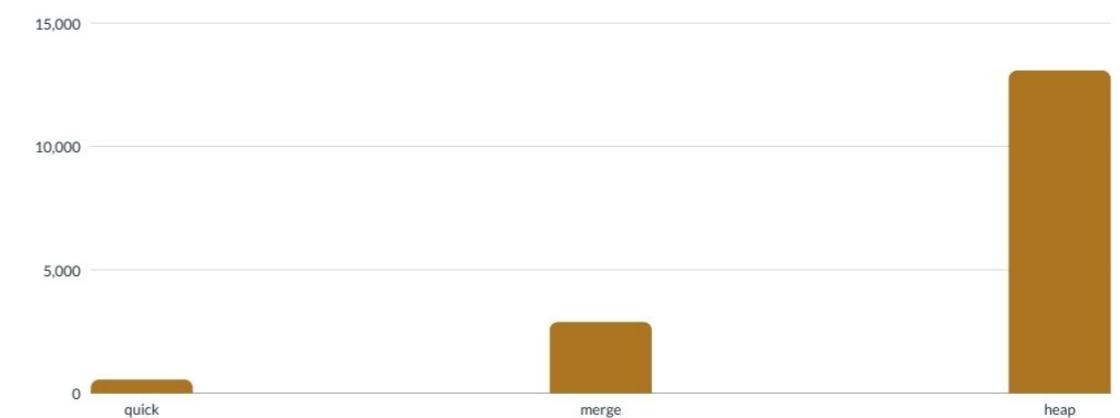
# DIGIT SORT

## AVERAGE PERFORMANCE



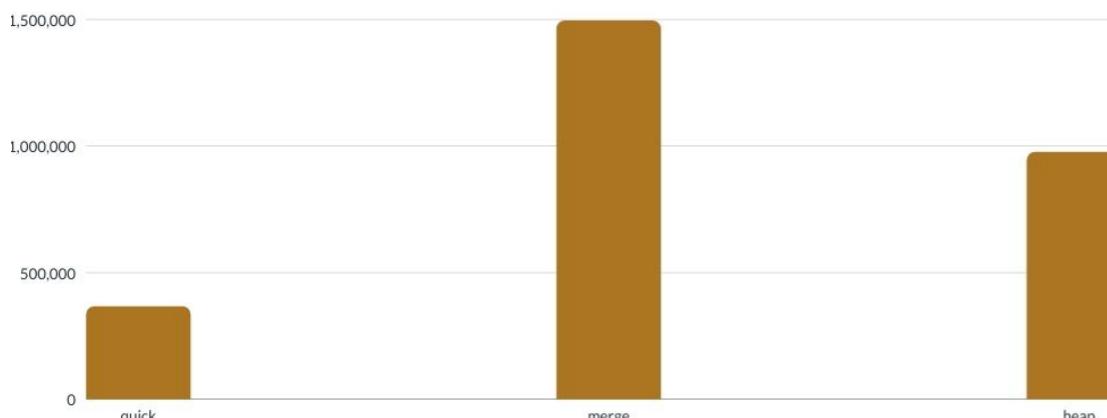
# DIGIT SORT

## SAMPLE STANDARD DEVIATION



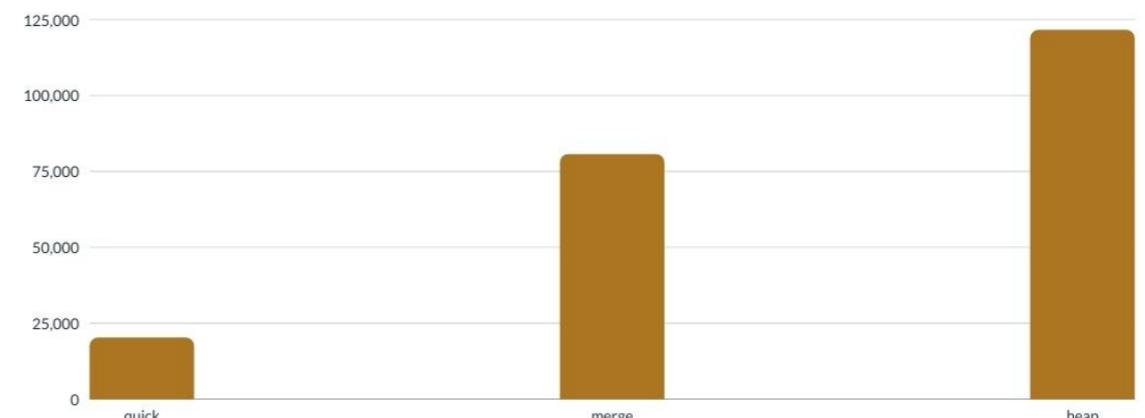
# STRING SORT

## AVERAGE PERFORMANCE



# STRING SORT

## SAMPLE STANDARD DEVIATION



## 總結

以上是本次實驗結果，快速排序相對於其他兩種排序快速且更穩定。

合併排序儘管穩定但對於非常大筆資料的處理相較推積排序更慢。

推積排序是相對較最不穩定的。

## 參考與附錄：

Github: <https://github.com/aStudentLoningToGrow/QUICK-MERGE-HEAP.git>

<https://kopu.chat>

<https://alrightchiu.github.io/SecondRound/comparison-sort-merge-sorthe-bing-pai-xu-fa.html>

<http://notepad.yehyeh.net/Content/Algorithm/Sort/Heap/Heap.php>

[https://www.youtube.com/watch?v=j-DqQcNPGbE&ab\\_channel=%E9%BB%84%E6%B5%A9%E6%9D%B0](https://www.youtube.com/watch?v=j-DqQcNPGbE&ab_channel=%E9%BB%84%E6%B5%A9%E6%9D%B0)

<https://emn178.pixnet.net/blog/post/88613503>

<https://medium.com/codeda/%E6%BC%94%E7%AE%97%E6%B3%95%E7%AD%86%E8%A8%98-%E4%B8%80-merge-sort-and-insert-sort%E5%AF%A6%E4%BD%9Cin-java-cb3ca6f7e22b>