# HW2 排序演算法比較

408410113 王興彥

本次報告內容討論三個高階排序演算法(quick sort, merge sort, heap sort)的實作，

並比較其差異，以及探討優化方法。

GitHub: https://github.com/xingyan0523/pdhw2

## 1. 實作重點與方法

### (1) Quick sort

Quick sort 的重點會在 partition 的部分，如何選擇 pivot 是 Quick sort 的重點，不但會影響時間複雜度，若是使用遞迴的方式實作也會影響到空間複雜度。

每次開始前先將現有 arr 依 pivot 大小將 arr 分成 < pivot 和 > pivot，再利用 recursive 的方式，將左右 subarray 繼續切分。

我的實作方式因為沒有經過特殊處理，因此時間複雜度平均為 O(nlogn)，最差為 O(n^2)，空間複雜度最差為 O(n)，若多加一個 subarray 大小判斷，並選擇小的那邊遞迴可將空間複雜度優化 O(logn)。

```
quickSort(int *arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

### (2) Merge sort

Merge sort 的重點則是放在 merge 的部分，一般情況下需要使用額外的陣列來協助 merge，且 merge 時需要特別注意陣列的 index 範圍，因為變數較多需要小心使用，以免超出可用記憶體範圍。

先使用遞迴將 array 切成一小塊，到最後開始倆倆合併，合併的過程會將兩個 subarray 的 key 依照大小作 merge，合併到最後即可完成排序。

我實作的方式平均的時間複雜度和最差都是 O(nlogn)，空間複雜度是 O(n)。

```
        mergeSort(int *arr, int left, int right){
            if (left < right) {
            int mid = left + (right - left) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
    }
}
```

### (3) Heap sort

Heap sort 的重點在於維持 max(min) heap。

Heap sort 主要是運用二元樹的抽象該念，實際上並不需要建構樹，一開始先用它的特性將 arr 調整成 max-heap(x * 2, x * 2 + 1 會是 x 的兩個 child)，調整好後 root 跟最後一個位置

交換，此時最大值在最後面的 index，再將樹調整成 max-heap，並重複上述動作，最後可以獲得由小到大的 arr。

```
heapSort(int *arr, int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapmaintain(arr, n, i);
    for (int i = n - 1; i > 0; i--) {
        swap(&arr[0], &arr[i]);
        heapmaintain(arr, i, 0);
    }
}
```

## 2. 測試環境與方法

### (1) 環境

**Operating System: Kubuntu 20.04**
**KDE Plasma Version: 5.18.5**
**KDE Frameworks Version: 5.68.0**
**Qt Version: 5.12.8**
**Kernel Version: 5.8.0-49-generic**
**OS Type: 64-bit**
**Processors: 4 × Intel® Core™ i7-6500U CPU @ 2.50GHz**
**Memory: 11.6 GiB of RAM**

### (2) 方法

(a) data_gen.c 產生固定 1,000,000 個測資

使用方式 ./data_gen [-type], type: -n = integer -s = string

String 為隨機長度(1~100)包含隨機字元(數字及英文大小寫)的字串

Integer 為隨機數字字串(0 ~ 2,147,483,647)

(b) cal_avg.c 讀取格式化的測試結果文本，並計算出平均值

(c) ___ sort.c

./___sort [-type], type: -n = integer -s = string

(d) test.c 為自動化測試程式

使用方式 ./test [-type] x(測試次數)

執行後會自動產生需要的側資檔案，並執行 x times 後將結果並以 quicksort mergesort heapsort 的順序輪流測試，隨後將結果輸出到 result，最後會將平均值輸出到 result 的最後面。

**3. 測試結果**

| String | Quick sort | Merge sort | Heap sort |
|---|---|---|---|
| 100 次平均(sec) | 0.785996 | 1.954127 | 2.001967 |
| Integer | Quick sort | Merge sort | Heap sort |
| 100 次平均(sec) | 0.204525 | 0.253072 | 0.420406 |
| Best | Ω(n log(n)) | Ω(n log(n)) | Ω(n log(n)) |
| Average | θ(n log(n)) | θ(n log(n)) | θ(n log(n)) |
| Worst | O(n^2) | O(n log(n)) | O(n log(n)) |
| Space | O(n log(n)) | O(n) | O(1) |

**4. 結果分析 (以自己實作的方式分析，先不討論優化)**

這次測試結果為速度由快到依序為 quick sort, merge sort, heap sort, 但直接這樣看時間和複雜度也不是非常準確，因為這都只是粗略的估計而已，實際上在不同的實作方式和不同的環境下結果都不一定會相同。

## 5. 優化方式

(1) Quick sort

(a) Median of medians algorithm.

(b) Hybrid quick sort.

(2) Merge sort

(a) 若排序的 element 都是整數的話可以使用 inplace 的 merge sort，省空間之外也可省去 malloc 的時間。

(b) Parallel merge sort.

(3) Heap sort

(a) Bottom-up heap sort.

(4) String 在排序的時候可使用 index 或是 pointer 來排序速度會快很多，這邊為了讓差異更明顯，所以用了比較耗時的方法。

## 6. 參考資料


https://www.geeksforgeeks.org/quick-sort/

https://www.geeksforgeeks.org/merge-sort/

https://www.geeksforgeeks.org/heap-sort/