Homework 1

0811324 胡政煒

```
輸入你要的input size: 10
Insertion Sort with size 10 took 0 microseconds.
Merge Sort with size 10 took 0 microseconds.
```

```
輸入你要的input size: 100
Insertion Sort with size 100 took 0 microseconds.
Merge Sort with size 100 took 0 microseconds.
```

```
輸入你要的input size: 1000
Insertion Sort with size 1000 took 1013 microseconds.
Merge Sort with size 1000 took 0 microseconds.
```

```
輸入你要的input size: 5200
Insertion Sort with size 5200 took 8003 microseconds.
Merge Sort with size 5200 took 564 microseconds.
```

```
輸入你要的input size: 10000
Insertion Sort with size 10000 took 26000 microseconds.
Merge Sort with size 10000 took 1017 microseconds.
```

```
輸入你要的input size: 100000
Insertion Sort with size 100000 took 2546902 microseconds.
Merge Sort with size 100000 took 18988 microseconds.
```

其實能夠明顯看出，隨著 array size 的增加，insertion sort 所花費的時間會明顯高於 merge sort 很多。而且我認為跑的時間，也跟電腦效能有很大的關係，因為我電腦剛買不久，因此可能處理器比較好一點，在 input size 小的時候，所花費的時間幾乎為 0us，可能要以更小的單位才看的出來，我也試用過別人的電腦跑過，結果是有數據的，因此除了 sort 的方式不同會影響之外，使用的電腦也會有所差異。

因此理論上來說，當 array size 非常小的時候， insertion sort 的速度應該是要快於 merge sort 的，因此會有幾微秒的差異。而我的電腦，差不多在 input size 為 5200 的時候，電腦才開始能計數出 merge sort 所花費的時間。

我使用以下 library 內建功能 來製造亂數與計算時間。
#include <random> 使用 random library 產生亂數陣列
#include <chrono> 使用 chrono library 來計算 sort 的時間

老師在課堂中提到，insertion sort 的時間複雜度是 O(n^2)，而 merge sort 的時間複雜度則是 O(n log n)。少量排序時，適合使用 insertion sort，但在大量排序時，使用 merge sort 會更有效率。從這次作業的實際程式跑出來，也是應證老師上課所說的。

```
void insertion_sort(int* arr, int size) //// 定義一個函式名稱為 "insertion_sort"，接收一個指向整數陣列的指標 "arr" 和陣列的大小 "size"
{
    int i, key, j;
    for (i = 1; i < size; i++) // 使用 for 迴圈，從第二個元素開始比對陣列元素，並遞增迭代
    {
        key = arr[i];// 將目前元素的值存到 key 變數
        j = i - 1; // 將目前迴圈所指向的元素索引值存到 j 變數，並將其減一，用來與前一個元素比對

        while (j >= 0 && arr[j] > key) // 若 j 不小於 0 且前一個元素的值大於目前元素的值，則進入迴圈
        {
            arr[j + 1] = arr[j];// 將前一個元素的值往右移動一位
            j = j - 1;// 減少 j 的值，使其與目前元素繼續比對
        }

        arr[j + 1] = key;// 將目前元素插入到排序好的子序列中
    }
}
```

以上是詳細的介紹程式在進行的過程，因此可以推測，在最糟糕的情況下，即是數字由大到小排列，因此每個數字都需要重新排序。在 while 迴圈中，會需要做 n-1 次排序。整個 insertion sort 的時間複雜度因為總共有 n 個數字，所以為 O(n^2)，與課堂所述相同。因為 n 會指數成長，所以在數字很多時非常耗時。不過，當資料量很少時，因為相較於 merge sort，insertion sort 不具有遞迴形式，因此不需要系統的堆疊，會更有效率。

以下是 merge sort 的程式說明:

```
void merge_sort(int* arr, int p, int q) // 定義一個合併排序的函數，傳入參數包括一個整數數組arr
                                        // 數組的左邊界p，和右邊界q
{
    if (p < q) // 如果左邊界小於右邊界，表示這個子數組還可以進一步切割
    {
        int m = (p + q) / 2;// 計算中間位置，進一步切割數組
        merge_sort(arr, p, m);  // 對左半邊子數組進行遞歸排序
        merge_sort(arr, m + 1, q); // 對右半邊子數組進行遞歸排序
        merge(arr, p, q, m);  // 合併左右子數組
    }
}
```

以上程式碼中使用了遞歸（Recursive）的方法來實現合併排序。該函數將整個數組分為左右兩個子數組，分別對左右兩個子數組進行遞歸排序，然後再將排好序的左右子數組合併。

實際 merge sort 內部運作程式明:

```
void merge(int* arr, int p, int q, int m)
{
    int n1 = m - p + 1;// 計算左側陣列的大小
    int n2 = q - m;// 計算右側陣列的大小

    int* L = new int[n1];// 建立大小為 n1 的左側陣列
    int* R = new int[n2];// 建立大小為 n2 的右側陣列

    for (int i = 0; i < n1; i++) // 將原陣列 arr 中的元素複製到左側陣列 L 和右側陣列 R
    {
        L[i] = arr[p+i];
    }
    for (int i = 0; i < n2; i++)
    {
        R[i] = arr[i+m+1];
    }

    int i = 0, j = 0, k =p ;

    while (i < n1 && j < n2) // 將左側陣列 L 和右側陣列 R 中的元素合併排序
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) // 將左側陣列 L 或右側陣列 R 中的剩餘元素添加到 arr 中
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
    delete[] L;// 釋放動態分配的左側陣列的內存
    delete[] R;// 釋放動態分配的右側陣列的內存
}
```

以上是詳細的介紹程式，這段程式碼的主要功能是將陣列中的元素進行合併排序，並將排序後的結果存儲回原陣列中。

merge 演算法利用遞迴的方式處理，其中 merge sort 函式負責 divide and conquer 中的 divide 部分。p 和 q 代表排序範圍，使用 if 語句確保範圍有效。m 為將原始陣列拆成前半部分和後半部分的索引值，再對前後兩部分分別呼叫 merge sort 遞迴排序，最後使用 merge 函式將兩部分合併並排序。merge 函式負責合併排序部分，n1 為左側子陣列的元素數量，其值為 m-p+1；n2 為右側子陣列的元素數量，其值為 q-m。接著建立兩個 subarray，L[]和 R[]，使用兩個 for 迴圈將原始陣列前後兩部分分別複製到 subarray 中，再使用 while 迴圈進行元素大小的比較，將兩個 subarray 中較小的元素依序放回原始陣列中，最終完成排序合併。總而言之，若要處理 n 個元素的排序，則需要進行 n-1 次分割，並進行 2 的 log n 次合併操作，每次合併的時間複雜度為 O(n)，因此整個程序的時間複雜度為 O(n log n)。

相同結果可用遞迴時間函式來計算：
T(n)
= T(n/2) + T(n/2) + *c*n(其中 c 是一個正整數，c*n 表示合併時間)
= 2*T(n/2)+c*n
=> 因此 T(n) 的時間複雜度為 O(n log n)

根據以上的分析，可以得出結論，這個算法的時間複雜度與課堂上講授的一致。另外，我們也可以發現，當處理的元素數量較少或是數列已經接近排序完成時，使用插入排序算法會比較有效率；但若是處理的元素數量較多且數列亂序時，則使用合併排序算法會有較短的處理時間。