HW3 程式題

111504504 楊愷晴

一、程式碼

Way1: 使用迴圈遍歷

```
5   long long int count_inver(int *a, int a_size){
6       long long int inversion = 0;
7       for(int i = 0; i < a_size; ++i){
8           for(int j = i+1; j < a_size; ++j){
9               if(a[i] > a[j])
10                  inversion += 1;
11          }
12      }
13      return inversion;
14  }
```

每次都看第 i 個元素後面的 element 有沒有比他小，有的話就是一個 inversion，時間複雜度 O(n^2)

Way2: modify merge sort

```
8   long long int conut_inversion(int *a, int a_size){
9       int * temp = (int*)malloc(sizeof(int) * a_size);
10      return merge_aux(a, temp, 0, a_size-1);
11  }
```

```cpp
long long int merge_inver(int *a, int *temp, int p, int q, int r){
    int i = p;// p to q-1
    int j = q;// q to r
    int k = p;
    long long int inversion = 0;

    while(i <= q-1 && j <= r){
        if(a[i] <= a[j]){// the left one is smaller
            temp[k++] = a[i++];
        }
        else{// the right one is smaller
            temp[k++] = a[j++];
            inversion = inversion + (q - i);// this one must go across the whole left subarray
        }
    }

    while(i <= q-1)// copy the left elements from left subarray to temp
        temp[k++] = a[i++];

    while (j <= r)// copy the left elements from right subarray to temp
        temp[k++] = a[j++];

    for(i = p; i <= r; i++)// copy the data from temp back to a, then a is sorted
        a[i] = temp[i];

    return inversion;
}
```

```cpp
long long int merge_aux(int *a, int *temp, int p, int r){
    // array a, index from p to r
    long long int inver = 0;
    int q;
    if(p < r){
        q = (p+r)/2;// middle index, separate array into two parts, left and right
        inver += merge_aux(a, temp, p, q);// index from p to q
        inver += merge_aux(a, temp, q+1, r);// index from q+1 to r
        inver += merge_inver(a, temp, p, q+1, r);// merge means combine left and right subarray
    }
    return inver;
}
```

遵守 merge sort 的精神，把 array 分兩段，合併階段時如果左邊 subarray 的元素大於右邊 subarray 的元素，就更新 inversion 的數量。

時間複雜度 O(nlogn)

二、實驗測試兩種方法

測試環境

CPU: 13th Gen Intel® Core™ i7-1360P

Operating System: Windows 11

Compiler: gcc 13.2.0

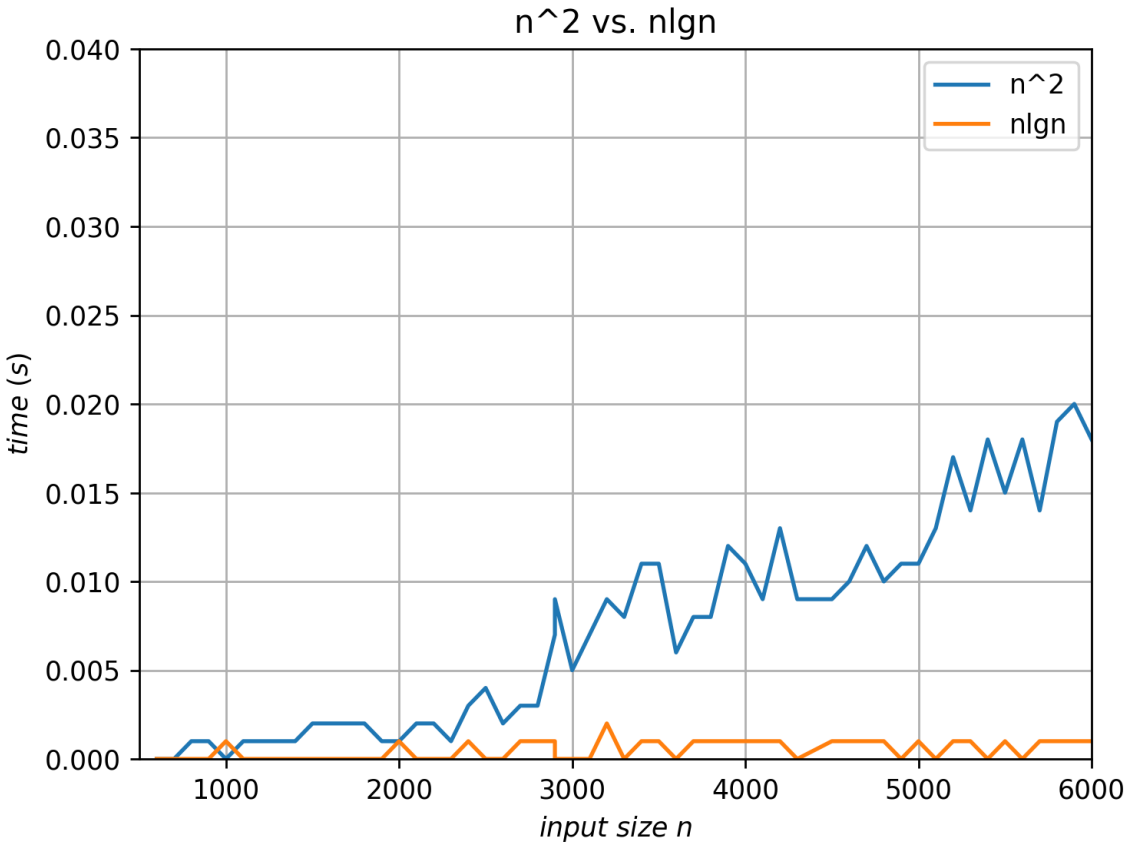
產生測資

用 Python 的 Random 套件隨機生成大小 n=600~6000 的整數 list，寫進”testdata.txt”（在 vs code 執行）

在 c 的主函式加入讀檔功能，讀取”testdata.txt”，使用 clock_t 進行執行時間的運算，再把結果寫到檔案”result.txt”（在 vs code 執行）

寫一個 plot.py，讀取”result.txt”，用 matplotlib 畫圖（在 google colab 執行）

繪圖結果



根據圖表，大概在 n=2000 後，nlgn 的方法所需的時間明顯小於 n^2 的方法