5.9 節末問題 5.9 的解答

問題 5.9.1

在程式碼 5.9.1 中，如下使用了 while 敘述來計算「支付到極限時的紙幣數量」。

```
while (N >= 10000) { N -= 10000; Answer += 1; }
while (N >= 5000) { N -= 5000; Answer += 1; }
while (N >= 1) { N -= 1000; Answer += 1; }
```

這也可以用除法來計算。當剩餘金額為 N 日元時，可用的紙幣數量最大值如下表所示。

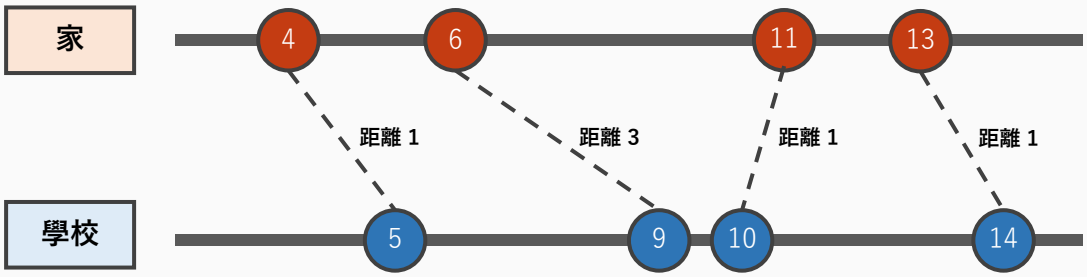
紙幣種類	10000 日圓鈔	5000 日圓鈔	1000 日圓鈔
可用最大數量	$\left\lfloor \frac{N}{10000} \right\rfloor$ 枚	$\left\lfloor \frac{N}{5000} \right\rfloor$ 枚	$\left\lfloor \frac{N}{1000} \right\rfloor$ 枚
支付到極限時的剩餘金額	$N \bmod 10000$日圓	$N \bmod 5000$日圓	$N \bmod 1000$日圓

因此，撰寫如下程式，即可以計算複雜度 $O(1)$求出答案。即使是 $N = 10^{18}$ 程度的輸入，也能瞬間執行完畢。

```
#include <iostream>
using namespace std;

int main() {
    // 輸入
    long long N, Answer = 0;
    cin >> N;

    // 支付 10000 日圓鈔
    Answer += (N / 10000); N %= 10000;
    // 支付 5000 日圓鈔
    Answer += (N / 5000); N %= 5000;
    // 支付 1000 日圓鈔
    Answer += (N / 1000); N %= 1000;

    // 輸出答案
    cout << Answer << endl;
    return 0;
}
```

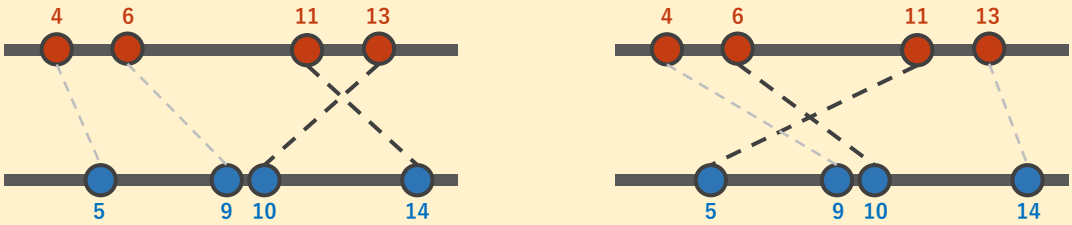
問題 5.9.2

如下圖所示，若將「從左數來第 1 家與從左數來第 1 所小學」、「從左數來第 2 家與從左數來第 2 所小學」……「從左數來第 N 家與從左數來第 N 所小學」連接，則家與就讀學校的距離總和為最小。

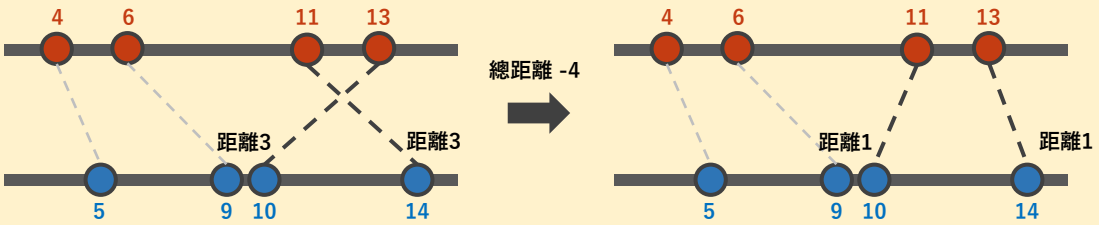


這一事實可以如下證明（因為難度有點高，可跳過不讀）。

首先，除了上述方法（以下記為方法 A）之外，所有的連接方法都至少存在一處「交叉點」。



此外，若將交叉點的連接更換而消除交叉，總距離會減少或不變。下圖為其中一例。



然後，反覆進行消除交叉的操作直到無法進行，則最終一定會變成方法 A※。因此，可以證明不存在比方法 A 距離更短的連接方式（家及學校的分配方式）。

※可以從交叉點線的組數（最大 ${}_{N}C_{2}$ 組）一定減少 1 組以上來證明。

因此，將陣列$(A_1, A_2, \ldots, A_N)$依小到大的順序排列為 $(A'_1, A'_2, \ldots, A'_N)$，將陣列 $(B_1, B_2, \ldots, B_N)$ 依小到大的順序排列為 $(B'_1, B'_2, \ldots, B'_N)$，距離總和可以用下式表示。

$$\sum_{i=1}^{N} |A'_i - B'_i| = \underbrace{|A'_1 - B'_1|}_{\text{從左數來第1家與從左數來第1所小學的距離}} + |A'_2 - B'_2| + \cdots + \underbrace{|A'_N - B'_N|}_{\text{從左數來第 } N \text{ 家與從左數來第 } N \text{ 所小學的距離}}$$

因此，提出如下程式即可得到正解。

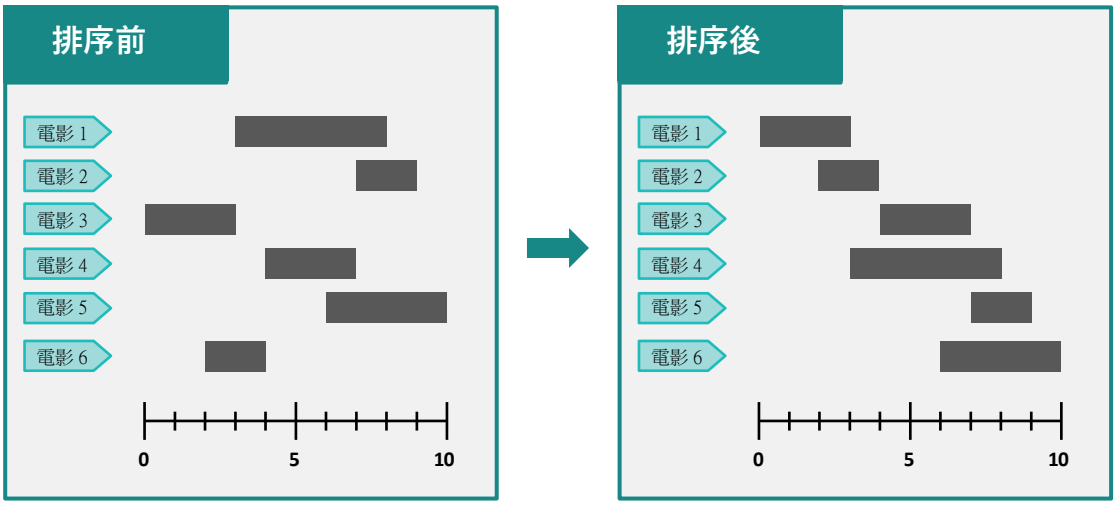
```
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

long long N;
long long A[100009], B[100009];

int main() {
        // 輸入
        cin >> N;
        for (int i = 1; i <= N; i++) cin >> A[i];
        for (int i = 1; i <= N; i++) cin >> B[i];

        // 排序
        sort(A + 1, A + N + 1);
        sort(B + 1, B + N + 1);

        // 求出答案
        long long Answer = 0;
        for (int i = 1; i <= N; i++) Answer += abs(A[i] - B[i]);
        cout << Answer << endl;
        return 0;
}
```

※ Python等原始碼請參閱 chap5-9.md。

問題 5.9.3

程式碼 5.9.2 的演算法確實可以得到正確答案，但計算速度較慢。因為在調查「目前可選擇的電影之中結束時間最早的電影」時需要計算複雜度 $O(N)$，所以在能夠選擇總共 N 部電影的情況下，計算複雜度為 $O(N_2)$。究竟該如何提高速度呢？。

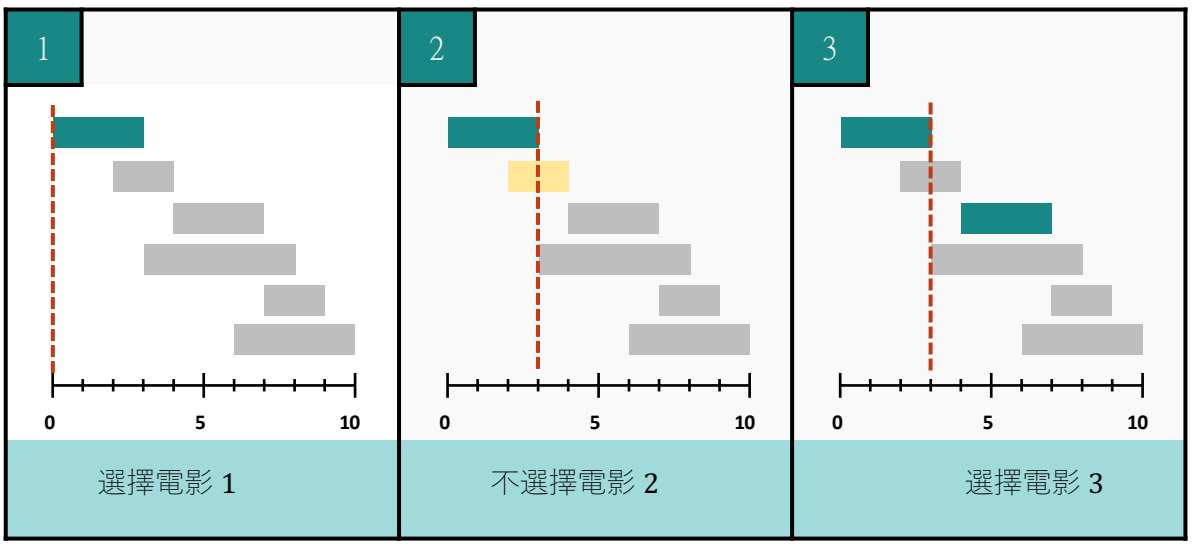
因此，將電影按結束時間早晚排序，將最早結束的電影設為「電影 1」，最晚結束的設為「電影N」。

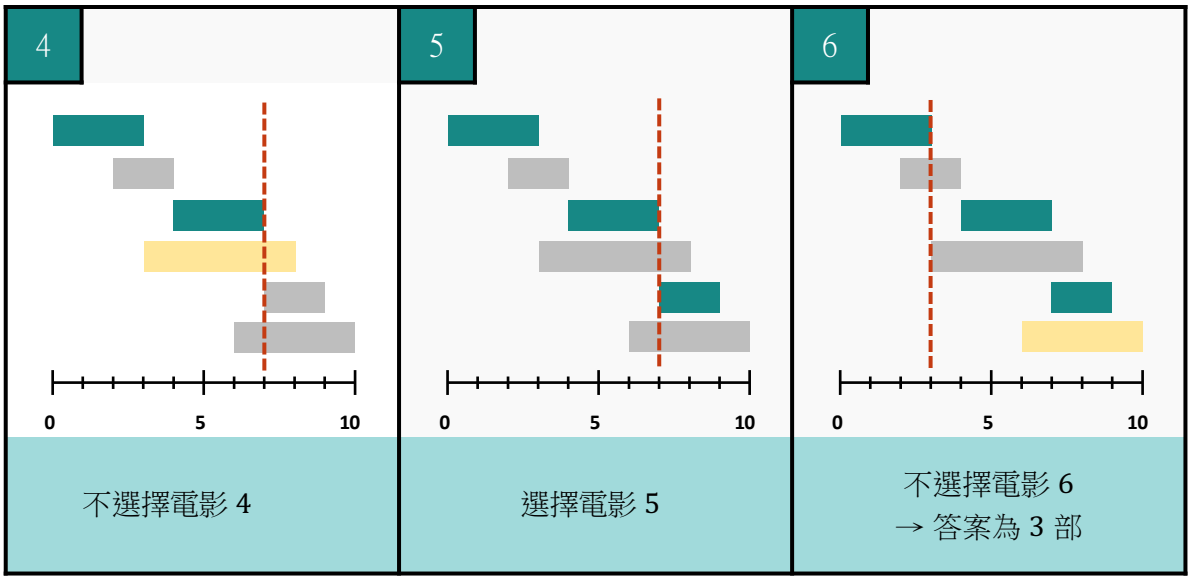


如此，用以下的演算法，可以有效率地持續選擇「最早結束的電影」。

- 選擇電影 1。
- 若（在開始時間）可以選擇電影 **2**，則選擇。
- 若（在開始時間）可以選擇電影 **3**，則選擇。
- ：
- 若（在開始時間）可以選擇電影 N，則選擇。

若將此演算法運用於具體的例子中，會如下所示。





例如， C++ 的實作例如下。為了依照結束時間早晚排序，使用了 `Movie` 型態。當 `Movie` 型態的變數 `A` 存在時，代表：

- `A.l`： 電影的開始時間
- `A.r`： 電影的結束時間

`A.r`較大者會被判定為「大」（ `bool operator<`的部分）。因此，透過 `sort` 函數將 `A.r` 按小到大排序。

```
#include <iostream>
#include <algorithm>
using namespace std;

// Movie 型態
struct Movie {
    int l, r;
};

// Movie 型態的比較函數
bool operator<(const Movie &a1, const Movie &a2)
    { if (a1.r < a2.r) return true;
    if (a1.r > a2.r) return false;
    if (a1.l < a2.l) return true;
    return false;
}

int N;
Movie A[300009];
int CurrentTime = 0; // 現在時間（最後所選電影的結束時間）
int Answer = 0; // 在現在已看過的電影數

int main() {
    // 輸入
    cin >> N;
```

```
    for (int i = 1; i <= N; i++) cin >> A[i].l >> A[i].r;

    // 排序
    sort(A + 1, A + N + 1);

    // 持續選擇結束時間最早的電影
    for (int i = 1; i <= N; i++) {
        if (CurrentTime <= A[i].l) {
            CurrentTime = A[i].r;
            Answer += 1;
        }
    }

    // 輸出
    cout << Answer << endl;
    return 0;
}
```

※ Python 等原始碼請參閱 chap5-9.md 。