5.9 節末問題 5.9 の解答

問題 5.9.1

コード 5.9.1 では、以下のように while 文を用いて「ギリギリまで払ったときの紙幣の枚数」を数えていました。

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

これは割り算を用いて計算することもできます。残り金額が N 円のとき、使える紙幣の枚数の最大値は以下の表のようになります。

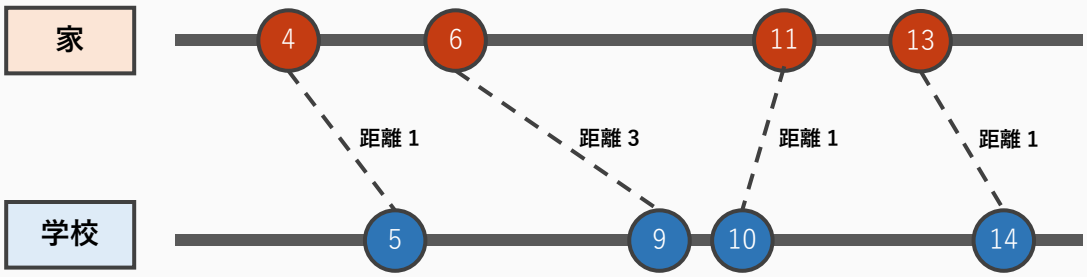
札の種類	10000 円札	5000 円札	1000 円札
使える最大枚数	$\left\lfloor \frac{N}{10000} \right\rfloor$ 枚	$\left\lfloor \frac{N}{5000} \right\rfloor$ 枚	$\left\lfloor \frac{N}{1000} \right\rfloor$ 枚
ギリギリまで使ったときの残り金額	$N \bmod 10000$ 円	$N \bmod 5000$ 円	$N \bmod 1000$ 円

したがって、以下のようなプログラムを書くと、計算量 $O(1)$ で答えを求めることができます。$N = 10^{18}$ 程度の入力をしても、一瞬で実行が終わります。

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

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

    // 10000 円札を支払う
    Answer += (N / 10000); N %= 10000;
    // 5000 円札を支払う
    Answer += (N / 5000); N %= 5000;
    // 1000 円札を支払う
    Answer += (N / 1000); N %= 1000;

    // 答えを出力
    cout << Answer << endl;
    return 0;
}
```

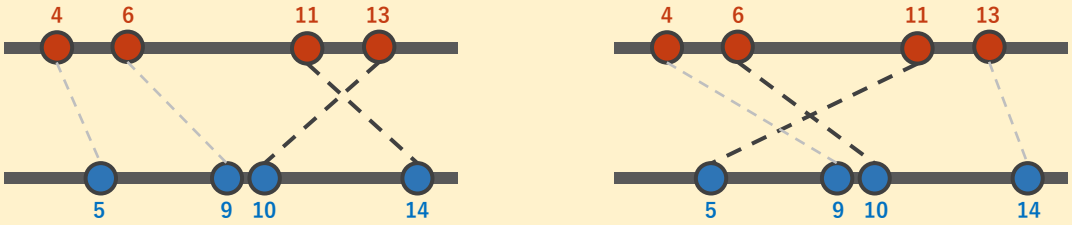
問題 5.9.2

下図のように、「左から 1 番目の家と左から 1 番目の小学校」「左から 2 番目の家と左から 2 番目の小学校」…「左から N 番目の家と左から N 番目の小学校」を繋ぐと、家と通う学校の距離の合計が最小となります。

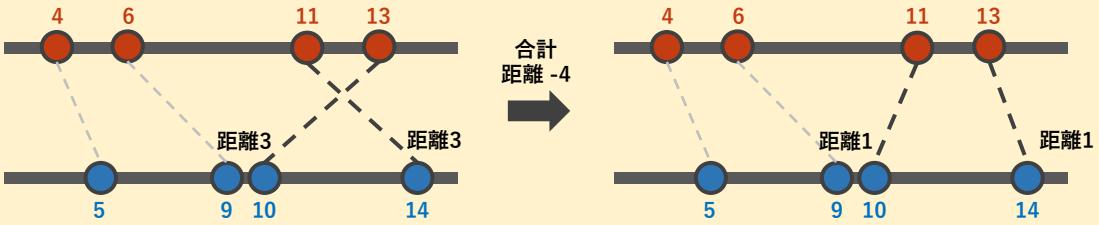


この事実は、以下のようにして証明することができます。（少し難易度が高いので読み飛ばしても構いません）

まず、上で述べた方法（以下、方法 A と記す）を除くすべての繋ぎ方には「交差する箇所」が少なくとも 1 つ存在します。



また、交差する箇所を繋ぎ換えて交差を消すと、合計距離は減るか変わらないかのいずれかになります。下図がその一例です。



そして、交差を消す操作ができなくなるまで繰り返すと、最終的には必ず方法 A になります※。このことから、方法 A より距離の合計が短い繋ぎ方（家と学校の割り当て方）が存在しないことが証明できました。

※交差する点線の組の数（最大 ${}_N\mathrm{C}_2$ 組）が必ず 1 以上減ることから証明できます。

したがって、配列 $(A_1, A_2, \ldots, A_N)$ を小さい順にソートしたものを $(A'_1, A'_2, \ldots, A'_N)$、配列 $(B_1, B_2, \ldots, B_N)$ を小さい順にソートしたものを $(B'_1, B'_2, \ldots, B'_N)$ とするとき、距離の合計は以下の式で表されます。

$$\sum_{i=1}^{N} |A'_i - B'_i| = \underbrace{|A'_1 - B'_1|}_{\text{左から 1 番目の家と左から 1 番目の学校の距離}} + |A'_2 - B'_2| + \cdots + \underbrace{|A'_N - B'_N|}_{\text{左から } N \text{ 番目の家と左から } N \text{ 番目の学校の距離}}$$

よって、以下のようなプログラムを提出すると、正解が得られます。

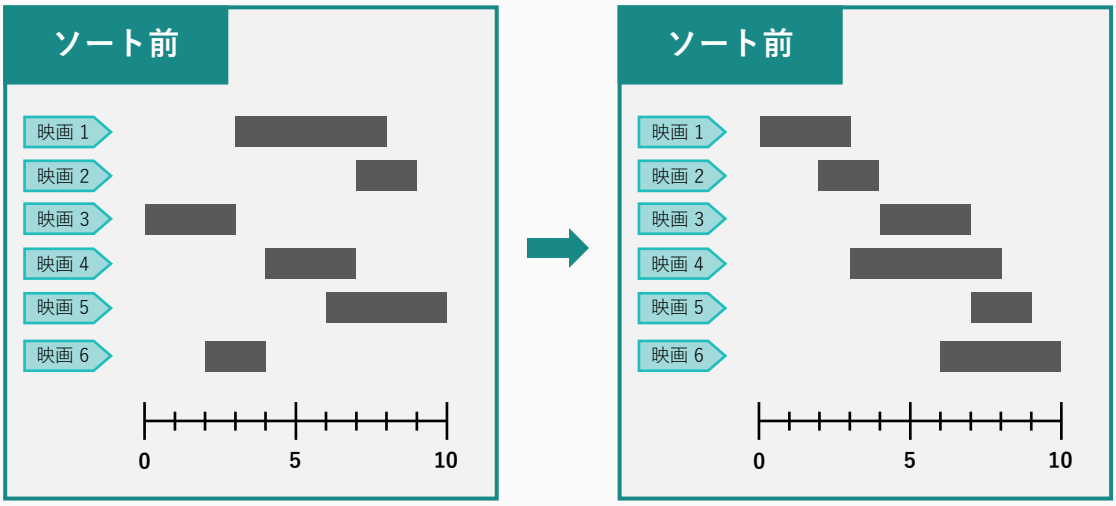
```cpp
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

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

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

    // ソート
    sort(A + 1, A + N + 1);
    sort(B + 1, B + N + 1);

    // 答えを求める
    long long Answer = 0;
    for (int i = 1; i <= N; i++) Answer += abs(A[i] - B[i]);
    cout << Answer << endl;
    return 0;
}
```

※ Python などのソースコードは chap5-9.md をご覧ください。

問題 5.9.3

コード 5.9.2 のアルゴリズムは確かに正しい答えを出すことができますが、計算が遅いです。「いま選べる中で最も終了時刻が早い映画」を調べるのに計算量 $O(N)$ かかるため、N 個すべての映画を選べるようなケースでは、計算量が $O(N^2)$ となってしまいます。一体どうすれば高速化できるのでしょうか。

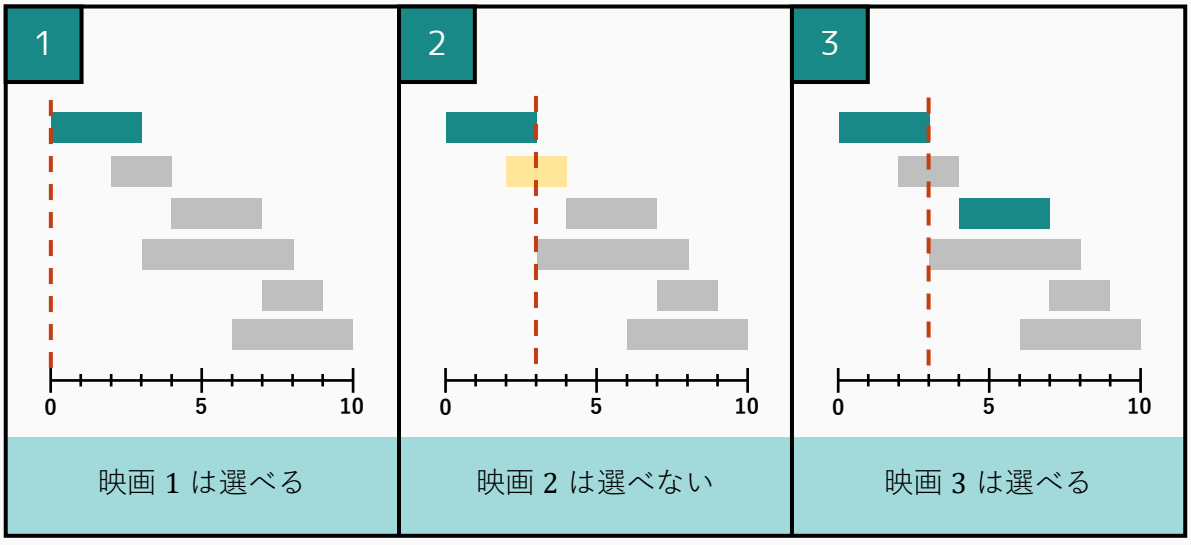
そこで、映画を終了時刻の早い順にソートし、最も終了が早いものを「映画 1」、最も終了が遅いものを「映画 N」としましょう。

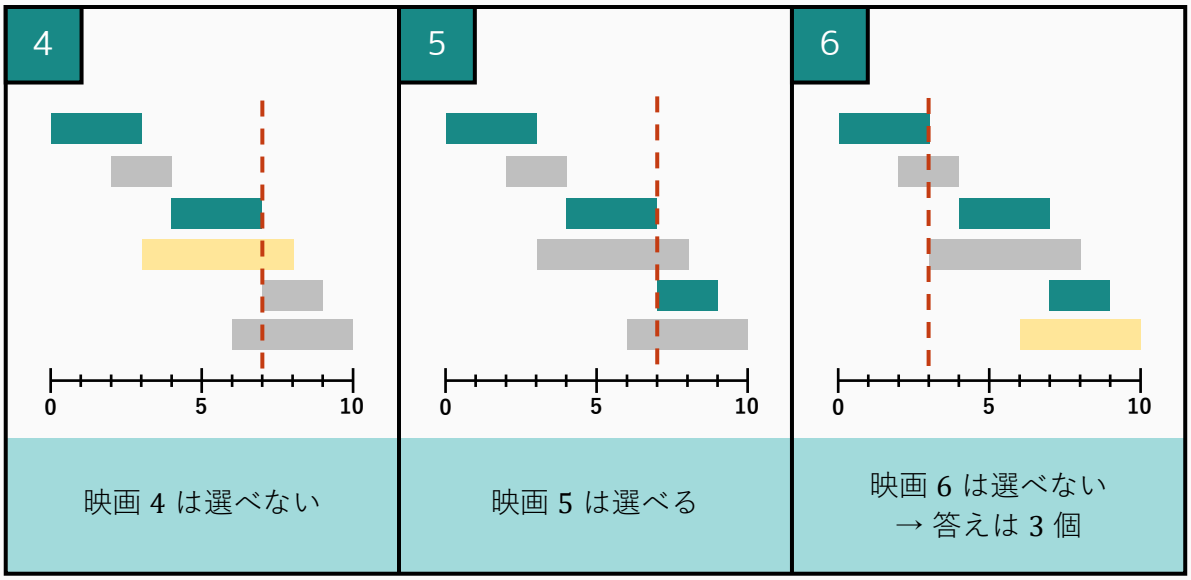


そうすると、以下のようなアルゴリズムで「最も終了が早いもの」を効率的に選び続けることができます。

- 映画 1 を選ぶ。
- （開始時刻的に）映画 2 を選べるならば、選ぶ。
- （開始時刻的に）映画 3 を選べるならば、選ぶ。
- 　　:
- （開始時刻的に）映画 N を選べるならば、選ぶ。

このアルゴリズムを具体的な例に適用すると、以下のようになります。





たとえば C++ での実装例は以下のようになります。なお、終了時刻の早い順にソートするために、`Movie` という型を使っています。`Movie` 型の変数 `A` があったとき、

- `A.l`：映画の開始時刻
- `A.r`：映画の終了時刻

を意味し、`A.r` の大きい方が「大きい」と判定されます（`bool operator<` の部分）。そのため、`sort` 関数によって `A.r` の小さい順に整列されます。

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

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

// Movie 型の比較関数
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; // 現在時刻（最後に選んだ映画の終了時刻）
int Answer = 0; // 現在見た映画の数

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

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

    // ソート
    sort(A + 1, A + N + 1);

    // 終了時刻が最も早いものを選び続ける
    for (int i = 1; i <= N; i++) {
        if (CurrentTime <= A[i].l) {
            CurrentTime = A[i].r;
            Answer += 1;
        }
    }

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

※ Python などのソースコードは chap5-9.md をご覧ください。