第 3 章

二分探索

3.1 問題 B11：Binary Search 2 (難易度：★3相当)

この問題は、以下のような方針で解くことができます。

- **手順1：**配列 $A = [A_1, A_2, \ldots, A_N]$ を小さい順にソートする。
- **手順2：**二分探索を使って、それぞれの質問に答える。

C++ の場合、手順 1 は `sort` 関数を使って処理することができます。また、手順 2 は `lower_bound` 関数を使って処理することができます。

なお、20 行目の `lower_bound(A + 1, A + N + 1, X) - A` の値について、本の 86 ページには「$A_i \geq X$ を満たす最小の i である」と書かれていますが、これは「配列 A の中に X より小さい要素がいくつあるか」と一致することに注意してください。

解答例（C++）

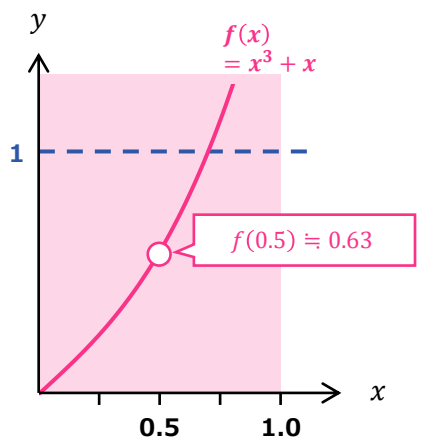
```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int N, A[100009];
int Q, X[100009];

int main() {
    // 入力
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> A[i];
    cin >> Q;
    for (int i = 1; i <= Q; i++) cin >> X[i];
    // 配列 X をソート
    sort(A + 1, A + N + 1);

    // 質問に答える
    for (int i = 1; i <= Q; i++) {
        int pos1 = lower_bound(A + 1, A + N + 1, X[i]) - A;
        cout << pos1 - 1 << endl;
    }
    return 0;
}
```

※Python のコードはサポートページをご覧ください

3.2 問題 B12：Equation (難易度：★4相当)

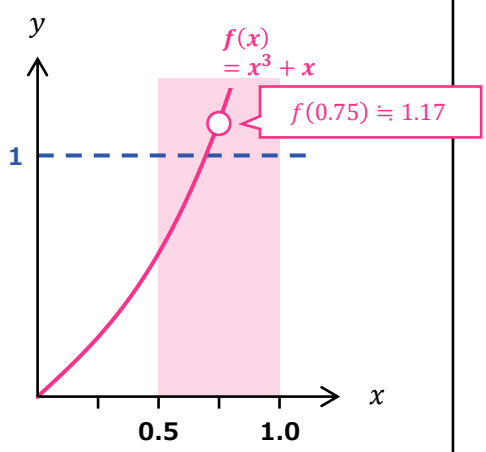
この問題は、**答えで二分探索**をして解くことができます。手始めに、$N = 1$ のときの答えを求めることを考えましょう。まずは答えが 0 以上 1 以下の範囲であることが分かっているとします。

1



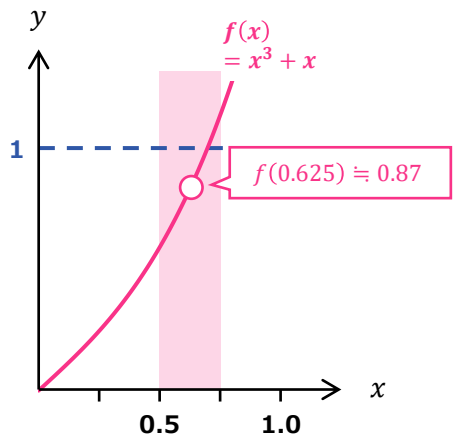
現在の範囲は 0.0～1.0（中央 0.5）
$f(0.5) < 1$ →答えは 0.5 以上

2



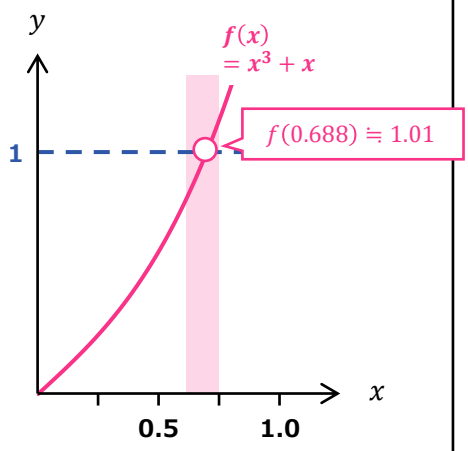
現在の範囲は 0.5～1.0（中央 0.75）
$f(0.75) > 1$ →答えは 0.75 未満

3



現在の範囲は 0.50～0.75（中央 0.625）
$f(0.625) < 1$ →答えは 0.625 以上

4



現在の範囲は 0.625～0.750（中央 0.688）
$f(0.688) > 1$ →答えは 0.688 未満

このように、たった 4 回の比較で、範囲が 0.625～0.688 まで絞られました。

それでは、本問題では何回の比較が必要なのでしょうか。まず、制約より $N \leq 10^5$ であるため、明らかに答えは 0 以上 100 以下です（$f(100) = 1000100$ ですので、既に 10^5 を超えています）。

また、答えと出力の絶対誤差が 0.001 未満であれば正解となるので、**元々 100 あった幅を、二分探索によって 0.001 未満まで縮めなければなりません**。そこで 20 回の比較を行った場合は次のようになります。

> 範囲の幅は $100 \div 2^{20} = 0.000095 \ldots < 0.001$ より、十分である。

したがって、以下の解答例のように 20 回のループを行うプログラムを書くと、正解が得られます。

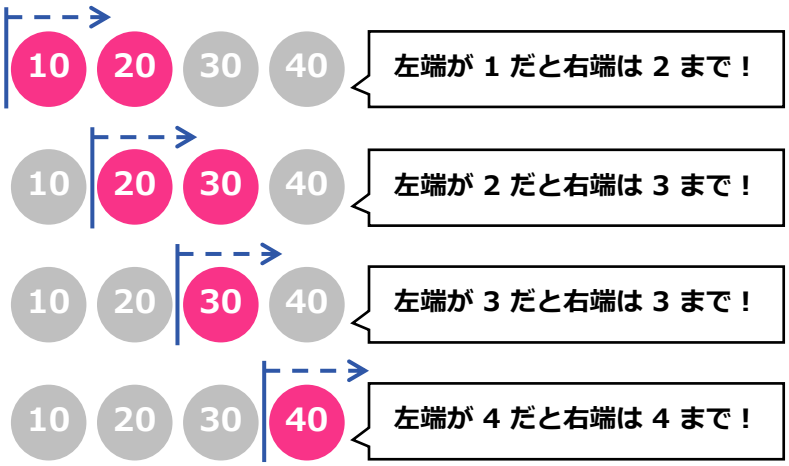
解答例（C++）

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

// 関数 f
double f(double x) {
    return x * x * x + x;
}

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

    // 二分探索
    double Left = 0, Right = 100, Mid;
    for (int i = 0; i < 20; i++) {
        Mid = (Left + Right) / 2.0;
        double val = f(Mid);

        // 探索範囲を絞る
        if (val > 1.0 * N) Right = Mid; // 左半分に絞られる
        else Left = Mid; // 右半分に絞られる
    }

    // 出力
    printf("%.12lf¥n", Mid);
    return 0;
}
```

※Python のコードはサポートページをご覧ください

3.3 問題 B13：Supermarket 2 (難易度：★4相当)

まず、「選ぶ品物の左端の番号を i とするとき、右端の番号はどこまで許されるか」を R_i とします。たとえば $A = [10, 20, 30, 40], K = 50$ の場合、$R_1 = 2$、$R_2 = 3$、$R_3 = 3$、$R_4 = 4$ です。



このとき、本問題の答え（何通りの選び方があるか）は次式で表されます。

$$\underbrace{(R_1 - 1 + 1)}_{\text{左端が 1 のときのパターン数}} + \underbrace{(R_2 - 2 + 1)}_{\text{左端が 2 のときのパターン数}} + \cdots + \underbrace{(R_N - N + 1)}_{\text{左端が } N \text{ のときのパターン数}}$$

しゃくとり法で R_i を求める

さて、本問題では明らかに $R_1 \leq R_2 \leq \cdots \leq R_N$ を満たすため、R_i の値は以下のしゃくとり法（→本の 93 ページ）によって求めることができます。

- $R_i = R_{i-1}$ からスタート（$i = 1$ の場合は $R_i = 1$ から）
- 合計が K を超えないギリギリまで、R_i を 1 ずつ増やしていく

問題は、「これ以上 R_i を増やすと合計価格が K 円を超えるかどうか」を高速に判定することなのですが、これは 2 章で学んだ累積和を使えば良いです。

累積和 $A_1 + A_2 + \cdots + A_i$ の値を S_i とするとき、L 番目から R 番目の品物の合計価格は $S_R - S_{L-1}$ となり、この値は計算量 $O(1)$ で求められます。

したがって、以下の解答例のような実装をすると、計算量 $O(N)$ で答えを求めることができます。なお、しゃくとり法では、「R_i を 1 増やす操作」が合計約 N 回しか行われないことに注意してください。

解答例（C++）

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

long long N, K;
long long A[100009];
long long S[100009]; // 累積和
long long R[100009]; // 左端が決まったとき、右端はどこまで行けるか

// A[l] から A[r] までの合計値
long long sum(int l, int r) {
    return S[r] - S[l - 1];
}

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

    // 累積和をとる
    S[0] = 0;
    for (int i = 1; i <= N; i++) S[i] = S[i - 1] + A[i];

    // しゃくとり法
    for (int i = 1; i <= N; i++) {
        if (i == 1) R[i] = 0;
        else R[i] = R[i - 1];
        while (R[i] < N && sum(i, R[i] + 1) <= K) {
                R[i] += 1;
        }
    }

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

※Python のコードはサポートページをご覧ください

3.4 問題 B14：Another Subset Sum (難易度：★5相当)

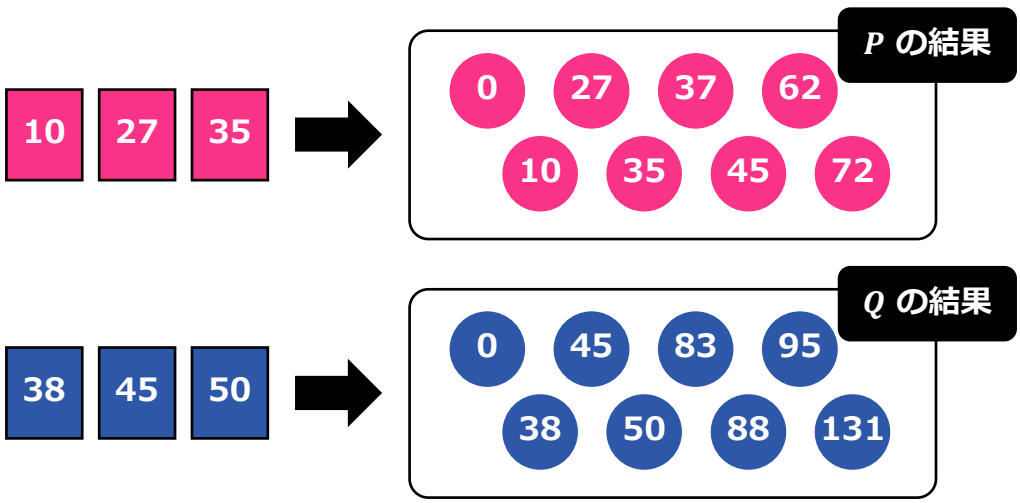
この問題は、半分全列挙を使った以下のような方針で解くことができます。

- **手順1：**「前半 $N/2$ 枚の中から何枚か選んだときの、カードに書かれた整数の総和として何があり得るか？」を全探索で求める
- **手順2：**「後半 $N/2$ 枚の中から何枚か選んだときの、カードに書かれた整数の総和として何があり得るか？」を全探索で求める
- **手順3：**手順 1・2 の結果を合成させて、答えを求める

この説明だけではよく分からないと思うので、例として $N = 6, S = 100, A = [10, 27, 35, 38, 45, 50]$ のケースを考えてみましょう。手順 1 の結果を P、手順 2 の結果を Q とするとき、

- $P = [0,10,27,35,37,45,62,72]$
- $Q = [0,38,45,50,83,88,95,131]$

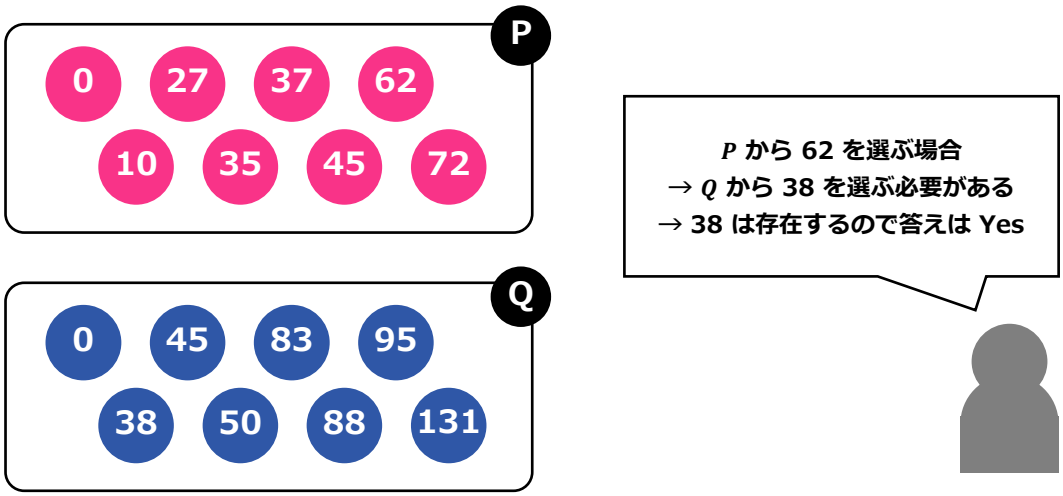
となります。イメージ図を以下に示します。



ここで、もし **P, Q から合計が 100 になるように 1 つずつ取り出す方法**が存在すれば、本問題の答えは Yes になるのですが、これはどうやって判定すれば良いでしょうか。もちろん $8^2 = 64$ 通りを全探索しても良いのですが、

計算に時間がかかってしまいます。そこで、「P の中からどれを選ぶか」を全探索するという方針が効率的です。

P から選ばれた要素を x とするとき、Q からは $100 - x$ を選ぶ必要があり、それが可能かどうかは配列の二分探索によって判定できるので※1、およそ $8 \log 8$ 回程度の計算しかする必要がありません（今回のケースでは、P の要素数も Q の要素数も 8 です）。



一般のケースでは？

ここまでは $N = 6$ の例を説明しましたが、一般のケースでも同じように解くことができます。これを実装すると解答例のようになり、各ステップでの計算量は以下のようになります。

- **手順1**：ビット全探索を使って $O(N \times 2^{N/2})$
- **手順2**：ビット全探索を使って $O(N \times 2^{N/2})$
- **手順3**：P, Q の要素数は $2^{N/2}$ なので、 $O\left(2^{N/2} \log 2^{N/2}\right) = O(N \times 2^{N/2})$

したがって、プログラム全体の計算量は $O(N \times 2^{N/2})$ です。本問題の制約は $N \leq 30$ であるため、$2^{15} = 32768$ より十分余裕を持って間に合います。

解答例（C++）

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

※1　Q が小さい順にソートされている必要があることに注意してください

```cpp
// 「配列 A にあるカードからいくつか選んだときの合計」として考えられるものを列挙
// ビット全探索を使う
vector<long long> Enumerate(vector<long long> A) {
    vector<long long> SumList;
    for (int i = 0; i < (1 << A.size()); i++) {
        long long sum = 0; // 現在の合計値
        for (int j = 0; j < A.size(); j++) {
                int wari = (1 << j);
                if ((i / wari) % 2 == 1) sum += A[j];
        }
        SumList.push_back(sum);
    }
    return SumList;
}

long long N, K;
long long A[39];

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

    // カードを半分ずつに分ける
    vector<long long> L1, L2;
    for (int i = 1; i <= N / 2; i++) L1.push_back(A[i]);
    for (int i = N / 2 + 1; i <= N; i++) L2.push_back(A[i]);

    // それぞれについて、「あり得るカードの合計」を全列挙
    vector<long long> Sum1 = Enumerate(L1);
    vector<long long> Sum2 = Enumerate(L2);
    sort(Sum1.begin(), Sum1.end());
    sort(Sum2.begin(), Sum2.end());

    // 二分探索で Sum1[i] + Sum2[j] = K となるものが存在するかを見つける
    for (int i = 0; i < Sum1.size(); i++) {
        int pos = lower_bound(Sum2.begin(), Sum2.end(), K - Sum1[i]) - Sum2.begin();
        if (pos < Sum2.size() && Sum2[pos] == K - Sum1[i]) {
                cout << "Yes" << endl;
                return 0;
        }
    }
    cout << "No" << endl;
    return 0;
}
```

※Python のコードはサポートページをご覧ください