Binary Search Magic!!!!

高校 2 年 赤澤侑

2020/08/10

目 次

はじめに

皆さんこんにちは. 高校 2 年の赤澤です. 皆さんはアルゴリズムという言葉を聞いたことはありますか？アルゴリズムというのは大まかに言うと “問題を解決するための手法”です. この言葉は特にコンピュータにどうやって解かせるかという文脈で語られることが多いので, 「うわぁー僕プログラミングできないよ〜〜」とか, 「パソコン使ったことないからどうせ無理だろうな〜〜」とか思っている人も多いでしょう. しかしそんなことは全くありません. アルゴリズムというのは “やり方”に特化した話ですから, むしろ, 自分の頭でしっかりと考えようとする姿勢が重要です.

本稿では計 4 つの問題を使って基礎的なアルゴリズムである二分探索への理解を深めていきます. 第 I 部では二分探索がどのようなものであるか掴めるように日常的なゲームを取り上げました. 第 II は第 I 部で得られた二分探索のイメージをより形式化・抽象化していきます. 第 III 部は二分探索を用いる発展的な問題を解いていきます. 問題文のすぐ下の行から解説が始まっていますが, どうぞ解説を読む前に「自分だったらどうやって解くか」を考えてみてほしいです.

なお, 本稿では高校範囲やそれ以上の数学的表現を多分に含みます. しかしながら, そのような表現には必ず説明や例示をしましたから, 見た目のいかつさに騙されず, それらの記号が何を表現しようとしているのか意識して読んでほしいです.

それでは深遠なる二分探索の世界へようこそ！

第 I 部
二分探索ってなんだろう

ショートストーリー：めぐるちゃんの数あてゲーム

ある暑い夏の日の昼下がりのこと. 小学 4 年生のめぐるちゃんとクラスメイトの AtSum くんがお話をしています.

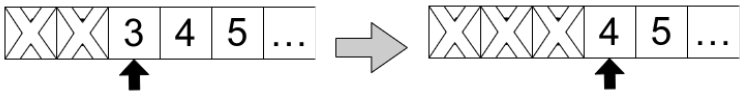
めぐる「ねえ AtSum！私, 好きな数字があるの！その数字をあててみてよ！」
AtSum「うん, いいよ！ えっと……その数字は 1？」
めぐる「ちがうよ」
AtSum「じゃあ 2？」
めぐる「ううん」
AtSum「なら 3 は？」
めぐる「もう！そんなふうに全部聞いてたらすごく時間がかかっちゃうよ！もっと効率的な方法はないの？」
AtSum「う〜ん, あっそうだ. その数は 2 の倍数？」
めぐる「いいえ」
AtSum「じゃあ 3 の倍数！」
めぐる「うん, その条件は満たしているよ. 」
AtSum「えっと…それなら……」

???「ふたりとも何をしているの？」

めぐる「あっ, Nyapo お姉ちゃん !! 」
AtSum「Nyapo 姉！実はめぐるに数字あてクイズを出されていてさ, いくつか質問しているけど全然わからないんだ……」
Nyapo「へぇ, それは面白そうね. めぐるちゃん, 私も質問していいかな？」
めぐる「うん, いいよっ！」
Nyapo「じゃあ, その数字は 1e9+7 より小さい数字かな？」
めぐる「??? いちいーきゅうってなに？」
Nyapo「あっ, ごめんね. えっと 10 億より小さい数字かな？」
めぐる「うん, そうだよ！っていうか 30 よりも小さいよ !! 」
Nyapo「そうなのね. じゃあ 15 よりも小さい数字？」
めぐる「うん♪」
Nyapo「7 以下？」
めぐる「ちがうよ」
Nyapo「11 以下かな？」
めぐる「そうだよ」
Nyapo「10 よりも小さい？」
めぐる「うん」
Nyapo「8 以下っていうのはどう？」
めぐる「それはちがうよ」
Nyapo「なるほど, ということはめぐるちゃんは 9 が好きなのね」
めぐる「正解だよ, Nyapo お姉ちゃん！」

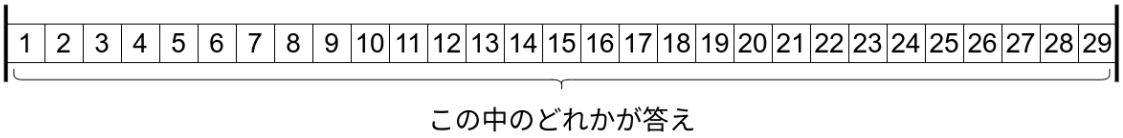
AtSum　「Nyapo 姉すげぇ！でも, なんでこんな質問したんだ？」
Nyapo　「それはね……」

Nyapo さんの方針を考える前にまずは AtSum くんの方法を考えてみましょう．AtSum くんは 1 から順番にある整数が答えであるかどうかを聞いていきました．図にすると以下のようになります．

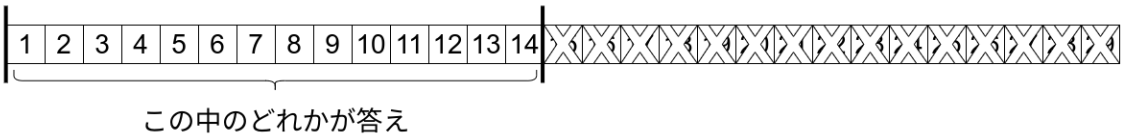


このように前から (あるいは後ろから) 順番に調べていく方法を**線形探索**といいます．

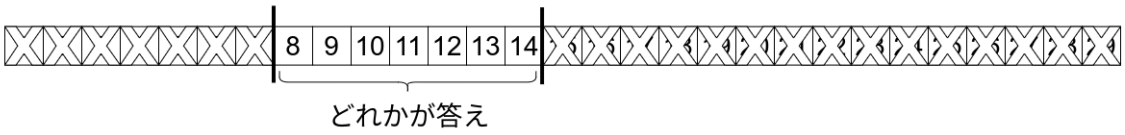
次に Nyapo さんの方法について考えてみます．Nyapo さんの最初質問とそれに関するやりとりで答えが 1〜29 のどれかであると判りました．図にすると次のようになります．



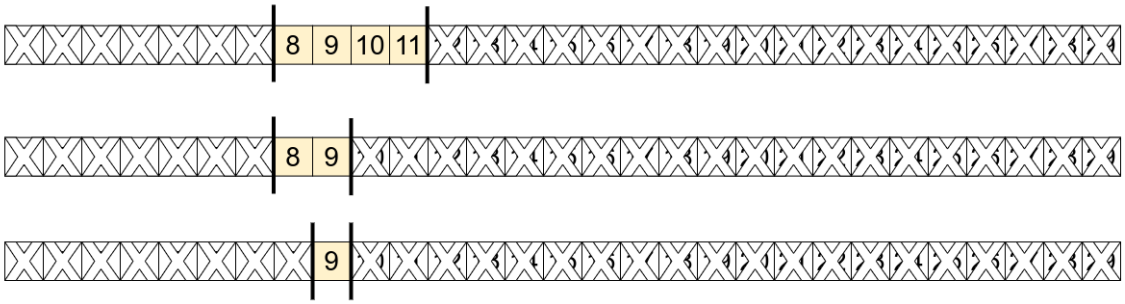
そのあと Nyapo さんは「答えは 15 よりも小さいか」という質問をしました．これに対する回答は “Yes” だったので答えが 1〜14 の範囲にあることが判ります．



次の質問は「7 以下かどうか」で回答は “No” なので答えは次のような範囲になります．



以降の質問も同様に範囲を絞っていくと次のようになります．(答えとしてあり得る範囲を黄色で塗ってあります．)



Nyapo さんはこのように考えることでめぐるさんの好きな数字をあてたのですね．ここで 1 回の質問ごとに答えの範囲がどのように絞られていくかを見てみます．次に書かれているものは “答え候補” の個数です．

$$29 \to 14 \to 7 \to 4 \to 2 \to 1$$

このように見てみると 1 回の質問で答え候補の個数が半分になっていることが判ります. なるほど, Nyapo さんの方法は答えの候補を半分に絞っていくような手法だったのですね. このように答え候補の真ん中が答えになるかどうかを質問することで答え候補の個数を半分に絞っていくやり方のことを**二分探索**といいます.

さて, この数あてゲームの解き方として 2 つの手法を挙げました. 線形探索と二分探索です. また, ある問題を解くための手法・やり方のことをアルゴリズムといいます. 線形探索や二分探索は基本的なアルゴリズムのうちの 1 つです. 第 II 部ではこれらのアルゴリズムの特徴を捉えていきます.

まとめ

アルゴリズム……ある問題を解決するための手法のこと.

線形探索……順番に確認していって条件に合うものを探すアルゴリズム.

二分探索……答え候補を半分に絞っていくことで答えを求めるアルゴリズム.

第 II 部
抽象化それは神の営みです

1　二分探索の計算量について

ショートストーリー：めぐるちゃんの数あてゲーム

AtSum くんは小学生でありながら “全日本指パッチン選手権” の運営委員です. この大会の予選は運営元に 1 分間指パッチンをする動画を送り, 運営がそれを審査することで行われます. 動画内での指パッチン回数が規定の予選通過ボーダーを超えていれば本戦出場です. さあ, 今年も AtSum くんのもとに数多の指パッチン動画が送られてきました. そんな彼のもとに運営委員長から電話がかかってきました.

AtSum　「もしもし, こちら AtSum です. 」
委員長　「おお, AtSum くん. 良かった. 今年の予選についてなんだが, 今時間はあるかな？」
AtSum　「はい, 大丈夫ですよ. 委員長. 」
委員長　「実は困ったことになっていてな……毎年本選を開催している本郷マイコンホールがあるだろ？何でも今年はホール内に入る人数を制限する必要があるらしくて, 本戦出場者の数を知る必要があるんだ. 」
AtSum　「なるほど」
委員長　「そこで, 本戦出場資格をもつ人数を教えてもらうことはできないか？」
AtSum　「わかりました. いいですよ. すこしお待ちください. 」
委員長　「ありがとう. 」
AtSum　「……とは言ったものの, 今年は N 件も応募があったんだよなぁ. 幸い昨日までに全動画の指パッチン回数を書き出して小さい順に並べておいたからそんなに大変な作業ではないけど……. どうしようかな？」
めぐる　「どうしたの, AtSum？そんなに困った顔をして. 」
AtSum　「めぐる！実はね……」

～少年説明中～

めぐる　「それならこの前 Nyapo お姉ちゃんに習ったアレをつかえば いいんじゃない？」
AtSum　「Nyapo 姉に習ったアレって……あっそうか！」
二人　「「二分探索 !!! 」」

この問題の解法を考える前にまずは問題の本質部分を抜き取ってみましょう. 次のようになるはずです. なお予選通過ボーダーを K としています.

問題

ここに N 個の数字が昇順に並んでいる. 数字の個数 N と数列 $(A_1, A_2, ..., A_N)$, 整数 K が与えられるので数列中に K 以上の数がいくつあるか求めよ.
ただし N, $(A_1, A_2, ..., A_N)$, K は次の制約を満たす.

- $1 \leq N \leq 1,000,000$
- $0 \leq A_i \leq 10,000$ $(1 \leq i \leq N)$
- $A_i \leq A_{i+1}$ $(1 \leq i < N)$
- $0 \leq K \leq 10,000$

例えば $N = 10$ で数列が (1, 2, 3, 5, 7, 8, 9, 13, 14, 14), $K = 10$ のとき, 答えは 3 になります. 以下にいくつか追加でサンプルを提示しておきます.

$N = 20$, $K = 0$
数列：(3, 3, 4, 5, 7, 7, 8, 8, 10, 11, 16, 19, 19, 21, 22, 22, 23, 24, 24, 28)
Ans. 20　ボーダーが 0 回なので全員が予選通過です.

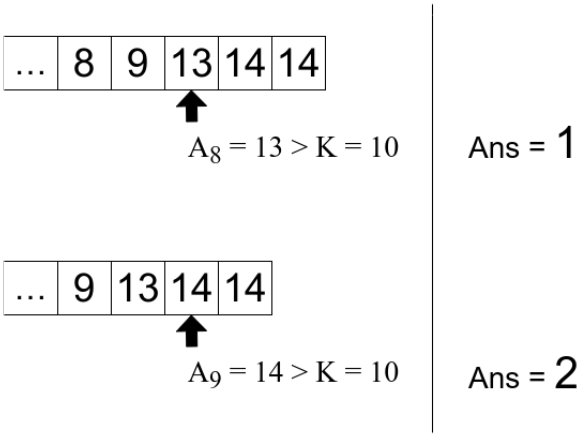
$N = 8$, $K = 10000$
数列：(24, 107, 1000, 2724, 3006, 8063, 9920, 9996)
Ans. 0　ボーダーは大変厳しかったようで, 今年本選に行ける人は誰もいませんでした.

さて, それではまず線形探索で考えてみましょう. 第 I 部で説明したとおり, 線形探索は前から順番に見ていくアルゴリズムでした. そこで次のようなアルゴリズム α が考えられます.

アルゴリズム α

前から順番に数字を見ていき, K 以上であれば答えの値を加算していく.

図で表すと次のようになります.



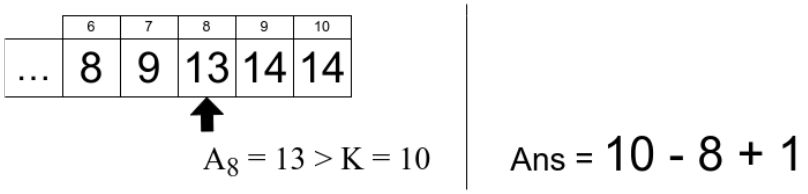
このようにすれば, 答えを求められることは明らかでしょう. しかしアルゴリズム α にはすこし無駄な部分があります. すでに数列は昇順に並んでいることが解っているので, 一度 K 以上の数が出てくればそこから先は全て K 以上になっているはずです. 即ち, AtSum くんは全ての要素を調べる必要はなく, 初めて K 以上となる数字が出てくるところを見つけることさえできればよいということで

す. このことを用いて次のアルゴリズム β を考えてみます.

アルゴリズム β

前から順番に数字を見ていき, 初めて K 以上になる要素がどこにあるかを調べる. A_j がこれを満たすとき $N-j+1$ を答えとする.

図で表すと次のようになります.



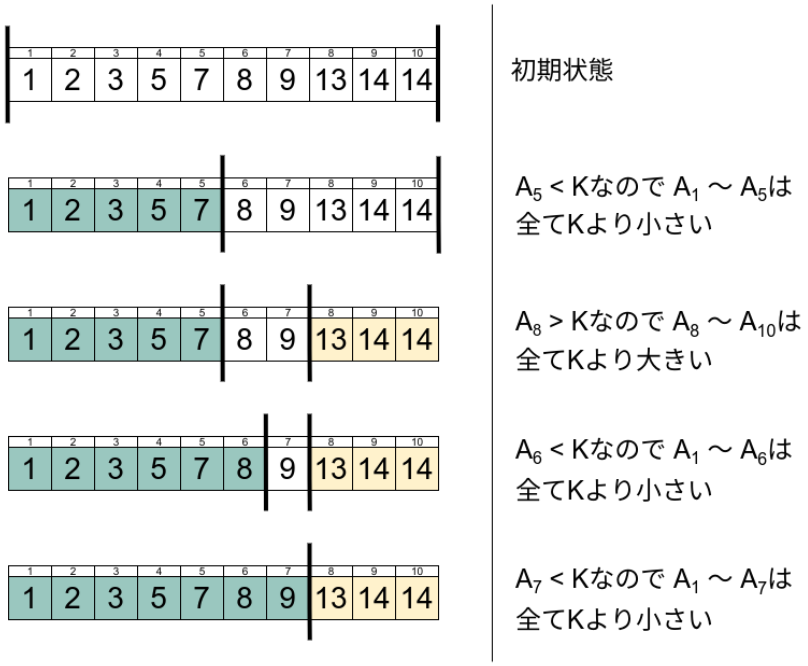
前から j 番目の要素以降は全て条件を満たすので $N-j+1$ が答えになります.

では二分探索で考えてみるとどうなるでしょうか. 第 I 部でやったときとはすこし異なりますが, 「ある数は K 以上か」という観点で処理をすれば初めて K 以上になる場所を見つけることができそうです. その場所が見つかればアルゴリズム β のときと同様に簡単な計算で個数を求めることができます. 二分探索によるアルゴリズム γ は次のようになります.

アルゴリズム γ

二分探索をすることで初めて K 以上になる場所を見つける. A_j がこれを満たすとき $N-j+1$ を答えとする.

アルゴリズム γ が動くようすを図にすると次のようになります. (図の各段階で K 以上であることが判っているものを黄色に, K より小さいことがわかったものを緑色に塗っています.)



具体的にどのように二分探索をするかというのは少し実装寄りな話になるので第 II 部第 3 節に載せておきます. 余力のある人や興味のある人はそちらを読んでください.

こうして私達は AtSum くんを助けるための方法を 3 つも得ることができました !!! では, 彼はど

のアルゴリズムを使うべきでしょうか？物語の中では二分探索と言っていますが, その理由を考えてみましょう.

少し唐突ですがここで**計算量** (complexity) という概念を導入します. 計算量というのはアルゴリズムの良し悪しを判断する際に用いる指標の一つで「そのアルゴリズムがどれくらいの時間で計算をするか[*1]」をざっくりと指し示すものです[*2]. さて, まずはアルゴリズム α, β, γ の中で 1 番たくさん繰り返される動作を考えてみます. 解りますか？ぱっと思い浮かばない人はもう 1 度 3 つのアルゴリズムを見返してみましょう. では正解です. α, β, γ は次の動作を最も多く行います.

- ある数が K より大きいかどうか比較する

ここでこの動作を処理 X とおき, 処理 X の所要時間を t とおきましょう. ここからは各アルゴリズムが処理 X を何回行うかを考えていきます.

まずはアルゴリズム α です. アルゴリズム α は N 個のデータに対して 1 回ずつ処理 X を行うので $t \times N$ の時間がかかることが解ります. ここで具体的な t の値を求めることはしません. なぜなら使うコンピュータのスペックや実装する言語によって異なる値を取るからです. そういった値はアルゴリズムの評価に組み込むにはふさわしくない要素です. とすると真に重要な事実はアルゴリズム α がどの程度の処理をするかは N の値に依存しているということです. このことを $\boldsymbol{O(N)}$ と書きます. この O(オーダー) という記号ですが, これはあるアルゴリズムがデータの数 N に対してどれくらいの計算をするかを表します. 即ち $O(N)$ というのは「**全ての動作を終えるのにかかる時間は N にだいたい比例しますよ**」ということを言っているのです. 例えば $O(N^2)$[*3]というアルゴリズムがあればそのアルゴリズムが処理を終えるまでにかかる時間は N^2 に比例して大きくなるということを表しています. 計算量オーダーを考えるときには定数 (与えられるデータによらず決定するもの. ここでは処理 X にかかる時間 t が定数です) は取り払って考えます. これは個別の動作に対する所要時間はコンピュータや環境によって異なるからといった理由やオーダーが違えば定数の違いはほとんど影響がなくなってしまうという理由によるものです. また次数 (N の肩に乗っている数字のことです) が最も大きいもの以外は無視します. 例えば $N^2 + N$ の処理をするようなアルゴリズムがあったとするとこの計算量は $O(N^2)$ です. その理由は次の表をみると明らかです.

N	100	10,000	1,000,000
$N^2 + N$	10,100	100,100,000	1,000,001,000,000
N^2	10,000	100,000,000	1,000,000,000,000

N の値が大きくなればなるほど $+N$ の影響が相対的に小さくなっています.

O 記法についてはこれくらいにして実際にアルゴリズム α がどれくらいの計算をするのか見てみます. ここに C++という言語で書かれたプログラムがあります. プログラム自体を読み解く必要は全くありませんので, 興味のある人だけ見てください.

x

```
#include<iostream>
#include<chrono>
#include<random>
#include<algorithm>

std::chrono::system_clock::time_point start, end;

double alpha(int N, int K, int A[]){
```

[*1] 計算量には本稿で扱う時間計算量の他に空間計算量 (領域計算量) や通信計算量などがあります.

[*2] 計算量について厳密に考えるためには, 本来チューリングマシンという概念を必要としますが, 今回それは用いず大雑把に説明します. 興味のある方はチューリングマシンについて調べてみるとよいでしょう.

[*3] N^2 は $N \times N$ を意味します. N^2 の 2 のように, 数字の肩に乗っている数字のことを指数といいます. 指数はその数字を何回掛け合わせるかを表しているので例えば N^3 であれば $N \times N \times N$, 10^4 であれば $10 \times 10 \times 10 \times 10 = 10000$ を表します.

```
  int res = 0;
  start = std::chrono::system_clock::now();

  for(int i = 0; i < N; ++i){
    if(A[i] >= K)++res;
  }

  end = std::chrono::system_clock::now();
  return std::chrono::duration_cast<
      std::chrono::nanoseconds>(end-start).count();
}

int main(){
  int trial;
  int array_len;
  double sum_times_alpha = 0.0;
  int nk;
  int narray[1001000];
  std::random_device seed_gen;
  std::mt19937 engine(seed_gen());

  scanf("%d %d", &trial, &array_len);
  for(int i = 0; i < trial; ++i){
    for(int j = 0; j < array_len; ++j){
      narray[j] = engine() % 10001;
    }
    std::sort(narray, narray + array_len);
    nk = engine() % 1000001;
    sum_times_alpha += alpha(array_len, nk, narray);
  }

  printf("algorithm_alpha -> [%f nsec]\n", sum_times_alpha/trial);
  return 0;
}
```

プログラムの内容は数列の長さ array_len と試行回数 trial を受け取り, 試行回数分だけランダムに条件に適う数列を生成してアルゴリズム α を実行し, アルゴリズム α の平均実行時間を返すものです. それでは実際にこのプログラムを使って, N の値を変えながらその処理時間を調べてみましょう. 実行すると次のようになります.

N(個)	5,000	10,000	20,000	30,000	100,000	1,000,000
α 実行時間 (ns)	8,674	17,329	34,308	51,003	172,270	1,734,077

なお, 試行回数は N の値によらず 100 回としています. さあ, $O(N)$ というのは「実行時間は N に比例する」ということを表しているのでした. ということは $N = 10,000$ だったものが $N = 20,000$ になれば実行時間は 2 倍になるはずです. 実際に表で確認してみると, 確かにそのような値が得られています！他の部分を見ても N が 3 倍になれば実行時間も 3 倍に, N が 10 倍なら実行時間も 10 倍になっていることが解るでしょう[*4] !! ということでアルゴリズム α は $O(N)$ のアルゴリズムでした.

次はアルゴリズム β です. これは少し厄介ですね. 処理 X を行う回数は N と K それに数列 $(A_1, A_2, ..., A_N)$ に完全に依存しています. 仕方がないのでアルゴリズム β は最も少ない場合で 1 回, 最も多い場合で N 回の処理 X が必要, ということになります. 平均をとって $\frac{N+1}{2}$ 回程度の処理 X を行うと考えれば平均の実行時間は $t \times \frac{N+1}{2}$ になります. N がある程度大きくなってくると N+1 の+1 は誤差の範疇ですから $\frac{t}{2} \times N$, オーダーを考えるときには定数は無視するのでアルゴリズム β も $O(N)$ のアルゴリズムといえるでしょう. では実際にアルゴリズム β を動かしてその実行時間を見てみます. 先程のコードに次のように実装した（これも読む必要はありません）アルゴリズム β で答えを求めるプログラムを追加します.

[*4]表中の値は厳密に 2 倍, 3 倍……となっているわけではありません. これは計測に使った機能の誤差や同時に動いていたプログラムの影響など様々な要因が考えられます.

x

```
double beta(int N, int K, int A[]){
  int res = 0;
  start = std::chrono::system_clock::now();

  for(int i = 0; i < N; ++i){
    if(A[i] >= K){
      res = N + 1 - i;
      break;
    }
  }

  end = std::chrono::system_clock::now();
  return std::chrono::duration_cast<
      std::chrono::nanoseconds>(end-start).count();
}
```

そしてアルゴリズム α と同じようにして実行時間を計測すると次の表のようになります[*5].

N(個)	5,000	10,000	20,000	30,000	100,000	1,000,000
α 実行時間 (ns)	8,674	17,329	34,308	51,003	172,270	1,734,077
β 実行時間 (ns)	5,184	9,708	18,710	30,436	109,097	931,366

比較のため, アルゴリズム α の実行時間も載せました. 試行回数が 100 回と少ないのでアルゴリズム α ほどきれいな関係にはなっていませんが, それでも N の変化率と同じくらい実行時間も変化していることが解ります. また α と β を比較してみると, これも試行回数の少なさが原因となってきれいに半分とはいきませんでしたが, β は α の $\frac{1}{2}$〜$\frac{2}{3}$ 程度の実行時間になっていることが解ります. 以上のことから①アルゴリズム α も β も $O(N)$ のアルゴリズムであるということ, ②同じオーダーのアルゴリズムは定数の小さいほう (α の定数は t, β の定数は $\frac{t}{2}$) が短時間で処理できるという 2 点が実験からも確かめられました.

それでは待ちに待った二分探索解法, アルゴリズム γ の計算量を考えてみます. アルゴリズム γ は範囲を半分に絞るという動作を繰り返すことで境界線を求めるやり方でした. これは 1 回の処理 X で答えとしてあり得る範囲を半分にするというふうに言い換えることができます. 即ち, その実行時間は $t \times$ (何回 2 で N を割れば N を 1 にすることができるか)[*6]です. この (何回 2 で N を割れば N を 1 にすることができるか) を数学の世界では $\log_2 N$ と書きます[*7]. $\log_2 N$ は以下の表のような値をとります.

N	4	8	16	32	64	256	1,024	16,384	262,144	16,777,216	1,073,741,824
$\log_2 N$	2	3	4	5	6	8	10	14	18	24	30

この表を見てみると N が大きくなっても $\log_2 N$ の値の変化は非常に小さいように見えますね ! 以上のことからアルゴリズム γ の計算量オーダーは $O(\log N)$[*8]であるということができるでしょう. また $O(N)$ と $O(\log N)$ のアルゴリズムとでは $O(\log N)$ のほうが圧倒的に少ない回数の計算で済むことが予想できますね. ではアルゴリズム γ の実行時間を計測したものを以下の表に記します. なおアルゴリズム γ のソースコードは次のようになっています (例の如く読む必要はありません).

[*5]編集の都合上 1 つずつ出てきていますが, アルゴリズム α, β, そしてこれから出てくる γ も全て同じケースを用いて実行時間を計測しています. なお, 実際に使用したコードの完全版は https: にて公開していますので, もし興味がありましたら動かしてみてください.

[*6]小数点以下は切り捨てて考えます.

[*7]これは正しい log の定義からずれたものであり, 非常に狭い範囲の話になっていることに注意してください.

[*8]$\log_x$ というときの x のことを底と呼びますが, オーダを考えるときには底も省略するのが一般的です.

x

```
double gamma(int N, int K, int A[]){
  int res = 0;
  start = std::chrono::system_clock::now();

  int* pos = std::lower_bound(A, A+N, K);
  res = N - (int)(pos-A) + 1;

  end = std::chrono::system_clock::now();
  return std::chrono::duration_cast<
      std::chrono::nanoseconds>(end-start).count();
}
```

また, 比較のためアルゴリズム α と β の実行時間も付してあります.

N(個)	5,000	10,000	20,000	30,000	100,000	1,000,000
α 実行時間 (ns)	8,674	17,329	34,308	51,003	172,270	1,734,077
β 実行時間 (ns)	5,184	9,708	18,710	30,436	109,097	931,366
γ 実行時間 (ns)	241	248	267	285	354	617

……すごいですね. ちょっとびっくりしてしまいますが, それもそのはず, 先程の表より $10 < \log_2 5000 < 14$ ですからたった数十回程度の処理 X だけで答えを求められます.

さてここまでの内容をまとめると

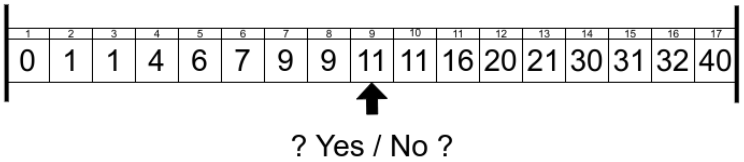
アルゴリズム α……$O(N)$
アルゴリズム β……$O(N)$
アルゴリズム γ……$O(\log N)$

であることが判りました. このように考えてみると AuSum くんが使うべきアルゴリズムは一目瞭然ですね. アルゴリズム γ, 二分探索によるアルゴリズムです. 計算量オーダーは「入力サイズに対してだいたいどれくらいの計算が必要か」を表すものでしたから, ほとんどの場合においてオーダーの小さいもののほうが早いということになります. また同じオーダーの場合は定数部分が実行時間に影響してきます. この “計算量が $O(\log N)$ である” という点が二分探索の大きな強みです. しかし, どんな場合でも二分探索を使えるかというとそうではありません. 二分探索を使うための条件があるのです. 次の節ではどういう状況なら二分探索を使えるかについて考えてみましょう.

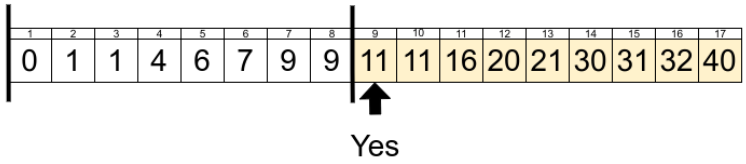
まとめ
二分探索の計算量オーダーは $O(\log N)$
線形探索の $O(N)$ に比べて圧倒的に高速

2 二分探索が使える条件について

二分探索が使える条件を考えるために第 1 節の問題を用いて二分探索とはどのような手法であったかもう一度確認してみましょう. 二分探索では下図のように解が存在しうる範囲の真ん中を調べてみるのでした.



そして例えばこの結果が “Yes”であればそれより大きいものは全て “Yes”であると判断するのです.

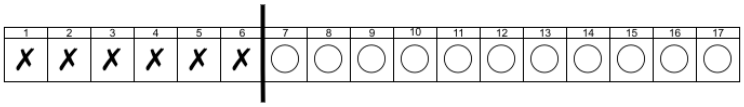


このような判断をするために二分探索という手法はデータに次のような性質を要求します.

性質 I

二分探索を行う区間に条件を満たす・満たさないが切り替わるような境界がただ一つ存在する.

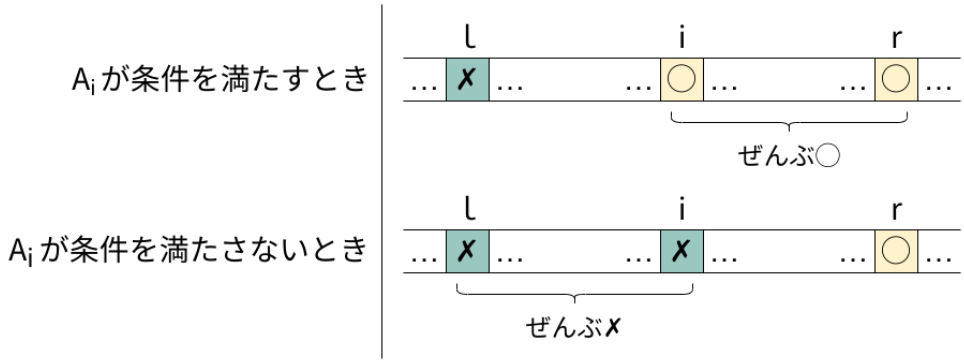
先程の数列の各要素に対して条件を満たすかどうかをマル印とバツ印で表したものが下の図です. (ここでは $K = 8$ としています)



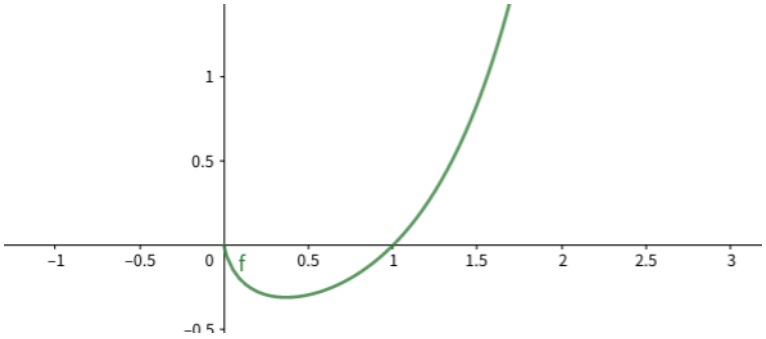
確かに境界線がただひとつだけ存在しています. このような性質を “**単調性**”といい, これがあるものに対しては二分探索を用いることができます. 単調性があれば, 以下のことが従うからです.

性質 II

l 番目の要素が条件を満たさず, r 番目の要素が条件を満たすことが判っているとき, $i(l \leq i \leq r)$ 番目の要素が条件を満たさないなら l〜i 番目の要素は全て条件を満たさず, i 番目の要素が条件を満たすなら i〜r 番目の要素は全て条件を満たす.



ここでは専ら整数という飛び飛びの数を考えましたが連続な数—実数—についても同じように考えることが出来ます. 例えば以下の座標平面上におけるグラフと x 軸との交点を求める (関数 $f(x) = 0$ の解を求める) 場合にも単調性があるときには二分探索が可能です.



この節では二分探索を使うための条件, 単調性について説明しました. 第 III 部では今までとは毛色のことなる問題に対して単調性を “見出す”, 謂わば二分探索実践編となっています. 二分探索という非常に強力な探索手法が様々な場面に活かせることが解るでしょう. また, プログラムが書ける方や, どうやってコンピュータに処理させるのかが気になる方のために第 II 部第 3 節にて, 実装方針と C++ での実装例を紹介します. これらは少し本質からそれるので興味のある方のみ読んでください.

まとめ
二分探索を使うためには単調性が必要.
逆に単調性があるものに対しては二分探索を使うことができる.

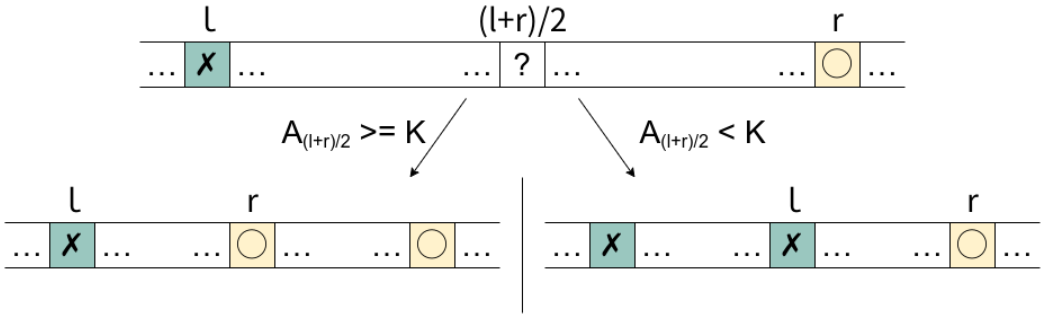
3 二分探索の実装について

この節ではどうやって二分探索を実装するかということについて第 II 部第 1 節の問題を例に説明します. なお, 今回はアルゴリズム γ ではなく, 初めて K 以上になる要素の番号を求めるという部分のみ考えます.

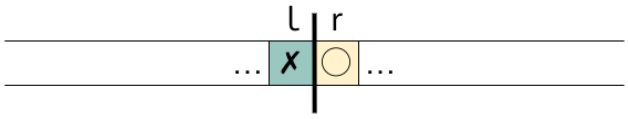
早速ですが二分探索の実装のイメージを以下に記します.

- はじめて K 以上になる要素が l より大きく r 以下の位置に存在していることが解っているとき, $A_{\frac{l+r}{2}}$ [*9]が条件を満たすかどうか確認する.
- $A_{\frac{l+r}{2}}$ が条件を満たすなら r を $\frac{l+r}{2}$ に置き換え, 条件を満たさないなら l を $\frac{l+r}{2}$ に置き換える[*10].
- 以上の操作を $r-l>1$ である間繰り返す.
- r を答えとする.

図にすると以下のようになります.



こうすると l, r にはそれぞれ条件を成立させない要素の番号と条件を成立させる要素の番号が常に入っていることになります. また, それらの値はその段階で解っている中で最も境界線に近いものです. $r-l=1$ であるとき, それぞれの値は連続しているので r が初めて K 以上になる要素の番号だと判ります.



[*9] $\frac{l+r}{2}$ は小数点以下を切り捨てて考えます.
[*10]なお, $\frac{r+l}{2}$ が r または l になることはありません. なぜなら $l-r\geq 2$ のとき $\frac{r+l}{2}\geq\frac{r+(r+2)}{2}=r+1$ であり, $\frac{r+l}{2}\leq\frac{(l-2)+l}{2}=l-1$ であるからです.

ここで気を付けないといけないのは境界線が以下のような位置にあるときです.

境界が左端にあるということは全ての要素が条件を満たすということで, 右端にあるということは全ての要素が条件を満たさないということです. 例えば $l = 1, r = N$ として始めてしまうとこれらの状況を表すことが出来ません. そこで, $l = 0, r = N + 1$ とします. すると上図の状態も表現することができるようになります. そもそも l, r はそれぞれ条件を満たさない, 満たす要素が常に入り続けるはずですから, 数列の 0 番目に絶対に条件を満たさない数 $-\inf$[*11]を, $N + 1$ 番目に絶対に条件を満たす数 inf を “番兵”として入れたと考えてもよいです.

これらの方針を更に抽象的に捉えて実装したものが次に紹介する “めぐる式二分探索”です. l, r を ok, ng と捉えることによって降順になっている (あるいは小さい値ほど条件を満たすような) 状況にも適用できる他に, 解の存在範囲を半開区間で持っているために紛らわしい計算がないのが特徴です.

x

```
1  bool isOK(int a[], int index, int key){
2      if(a[index] >= key) return true;
3      else return false;
4  }
5
6  int binary_search(int a[], int nax, int key){
7      int ng = -1;
8      int ok =nax;
9      while(abs(ok-ng) > 1){
10         int mid = (ok + ng)/2;
11         if(isOK(a,mid,key))ok = mid;
12         else ng = mid;
13     }
14     return ok;
15 }
```

少し難しかったかもしれませんがいかがでしたか？以上が二分探索の実装についてです. もし, プログラムを書いて実行できる環境があれば実際に書いてみるとより深く内容を理解できるでしょう.

[*11] inf は大きな値を意味します. この場合 A_i は 10,000 までの値なので $\inf = 10,000,000$ とでもしておけば十分でしょう.

第 III 部
こんなところにも二分探索

第 Ⅲ 部では AtCoder の問題を使用して二分探索がどのように使えるかを見ていきます. 解法に関する言及があるのでご注意ください. なお, 本稿では問題設定を楽しく理解できるようショートストーリーを問題の前に付してあります.

1　単調性がないなら作ってしまえばいいじゃない

ショートストーリー：積読家の Nyapo さん

Nyapo さんは読書好きで週に 1 度は必ず古書店に足を運びます. 今日もまた己の知的欲求を抑えることができず, 多くの本を買ってしまいました. 満足げな表情をたたえて帰宅する Nyapo さんは, しかし, 机の上に積み上げられた高い高い, 天井にも届くかという本の山を見て “ふうっ ”と静かに溜息をつくのでした. 読書家の二面性ーとでもいいましょうか, 彼女もまた例にもれず積読家でもあったのです.

Nyapo　「う〜ん, 流石に困ったわね. 2 台目の机も全て本で埋まってしまうなんて……. 仕方ない, 明日は一日中本を読むことにしましょう. 」

Nyapo　「どうせならなるべく多くの本を読みたいわね. 私は天才美少女だからどの本を読むのにどれくらい時間がかかるかは判るけど, 崩れちゃうと危ないから上からしか読めないのね……. すぐに読める本から順に読む作戦は使えないなあ. 一体何冊読めるのかしら. 」

問題 (AtCoder Beginner Contest172-C 問題より)

2 つの机 A, B にそれぞれ N 冊, M 冊の本が積んである. 読書に使える時間は T であり, 机 B の上から i 番目 $(1 \leq i \leq N)$ にある本は A_i, 机 B の上から i 番目 $(1 \leq i \leq M)$ にある本は B_i の時間で読める. このとき Nyapo さんは最大何冊の本を読むことができるか.
ただし N, M, T, A_i, B_i は次の制約を満たす.

- $1 \leq N, M \leq 200,000$
- $1 \leq T \leq 1,000,000,000$
- $1 \leq A_i, B_i \leq 1,000,000,000$

以下にサンプルを一つ置いておきます. これ以降サンプルと書かれているものについては以下のデータが使われています. またこの先, 特に断りなく A, B の上から i 番目の本のことも A_i, B_i と表記することがあります.

$N = 6$, $M = 5$, $T = 10$
$(A_1,\ A_2, ...,\ A_6)$: $(2,\ 3,\ 2,\ 7,\ 1,\ 1)$
$(B_1,\ B_2, ...,\ B_5)$: $(1,\ 5,\ 6,\ 1,\ 4)$
Ans. 4　$A_1 + A_2 + A_3 + B_1 = 8 < T$ でこれより多くは読めないので 4 が答えです.

さて, この問題をどのように考えましょうか？ぱっと思いつくのは次の解法です.

解法 A

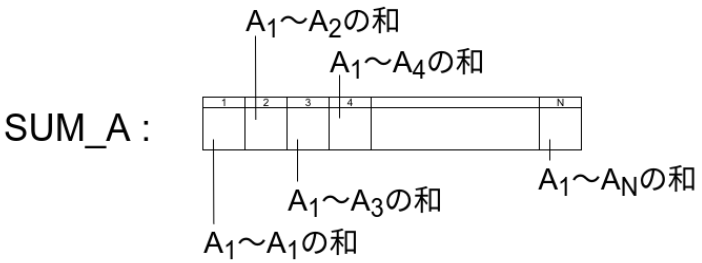
机 A, B のそれぞれの一番上の本の内かかる時間の少ない方を時間が尽きるまで読んでいく.

そうすると先程の例でいうと $B_1 \to A_1 \to A_2 \to A_3$ と選ぶことになり, なるほど, 確かに最大を達成できそうです. では, 次のような状況ならどうでしょうか.

$N = 6,\ M = 5,\ T = 10$
$(A_1,\ A_2, ...,\ A_6)$: $(4,\ 1,\ 1,\ 1,\ 1,\ 1)$
$(B_1,\ B_2, ...,\ B_5)$: $(3,\ 3,\ 3,\ 3,\ 3)$

解法 A でやってみると $B_1 \to B_2 \to B_3$ となり 3 冊読むことが出来ます. しかし, これは読むことのできる本の最大値ではありません. 最大値は机 A の本を選び続けることによって得られる 5 です. よって解法 A は正しくありません. ここでためしに机 A から K 冊読んだという状況を考えてみます. このとき残り時間は $T - \sum_{j=1}^{K} A_j$ です. $\sum_{j=1}^{K} A_j$ という記号が出てきましたが, 恐れることはありませんこれは「A_1〜A_K の総和」を意味します. つまり $A_1 + A_2 + ... + A_K$ と全く同じです. 話をもとに戻しますが, この残り時間で机 B から何冊読めるかを高速に求めることができれば K を 0〜N まで全て試してみることで N×($\sum_{j=1}^{K} A_j$ を求めるための計算回数)×(残り時間が決まったとき机 B から何冊読めるか求めるための計算回数) 回の計算で答えを求められそうです.

まず $\sum_{j=1}^{K} A_j$ を求める方法を考えます. 毎回 K を定めるごとにナイーブに足し算して求めているとこの部分だけで $O(N^2)$, 即ち最大で $200,000 \times 200,000 = 400$ 億回もの計算が発生してとても終わりません[*12]. では, この方法では解けないのでしょうか？そんなことはありません. ここで K 番目の要素が $\sum_{j=1}^{K} A_j$ であるような数列を考えてみます. つまりこういうことです.



この数列に SUM_A という名前を付けます. なお, $\mathrm{SUM\_A}_0 = 0$ とします. このとき SUM_A の i 番目の値は $\mathrm{SUM\_A}_i = A_i + \mathrm{SUM\_A}_{i-1}$ となっていることが解ります. 言い方を変えると (A_1〜A_i の和) = (A_1〜A_{i-1} の和) + A_i ということです. このことから前から順に A_i を足していってその値を次々に格納していくことで SUM_A を $O(N)$ で事前に計算することができると解ります. このように前から要素を足していった値のことを**累積和**といいます. すると $\sum_{j=1}^{K} A_j$ の値を求めるという動作は SUM_A の K 番目の値を参照するという動作になります. コンピュータはこの動作を $O(1)$ 即ち, 定数時間で行うことが出来ます. よって全体として「K を 1〜N へと変化させながら $\sum_{j=1}^{K} A_j$ の値を求めるという動作」を $O(N)$ で行うことが出来ました. ここまでの処理内容を書き表すと次のようになります.

[*12]現在の一般的なコンピューターは 1 秒間に 10^8 回程度, 即ち 1 億回程度のループを回すことができると言われています.

解法 B

- A_i の値を順番に足していくことで SUM_A という数列を作る ← $O(N)$
- K を 1〜N まで変化させる ← $O(N)$
  - $\sum_{j=1}^{K} A_j$ を求める ← $O(1)$
  - 机 A から K 冊読んだとき残り時間で机 B から何冊読めるか求める ← $O(?)$

さて. 次に求めたいのは「机 A から K 冊読んだとき残り時間で机 B から何冊読めるか」です. ここで,「机 A から K 冊読んだとき残り時間」, 即ち $T - \sum_{j=1}^{K} A_j$ を t とおきます. このとき, t という時間を目一杯使って机 B の本を上から順に読んでいけばいいです. しかし, これも毎回足し算をしていくと, この処理は $O(M)$ になってしまいますから全体として $O(NM)$ になってしまいとても終わりません. そこで机 A に対して行ったように B の累積和, SUM_B を作ります. ここでも $\mathrm{SUM\_B}_0 = 0$ とします. ではサンプルにおける SUM_B の第 1 項から第 5 項を見てみましょう.

$$\mathrm{SUM\_B} = \{1,\ 6,\ 12,\ 13,\ 17\}$$

こうしてみると今求めたいものは次のように言うことが出来ます.

SUM_B の中で t 以下になる最大の数があるのは何番目か.

ちょっと見覚えがあると思いませんか？そうです, 第 II 部で何度も考えた問題にそっくりです. そして, 累積和は前から順番に足していった値なので明らかに単調性があります. ということは t という時間でどれくらいの本を読めるかは $O(\log M)$ で求められることが出来ます. 全体としては次のようにしてこの問題が解けるということです.

解法 B

- A_i の値を順番に足していくことで SUM_A という数列を作る ← $O(N)$
- B_i の値を順番に足していくことで SUM_B という数列を作る ← $O(M)$
- K を 1〜N まで変化させる ← $O(N)$
  - $\sum_{j=1}^{K} A_j$ を求める ← $O(1)$
  - 机 A から K 冊読んだとき残り時間で机 B から何冊読めるかを SUM_B に対する二分探索で求める ← $O(\log M)$

よってこの問題は $O(N \log M)$ で解けることが解ります. $N = 200,000$ $M = 200,000$ のときでも全体として 3,400,000 程度なので十分計算可能です.

この問題は前から読んだときにあるところまででどれくらいの時間がかかるか—即ち累積和—が分かれば高速に処理ができるという発想ができるかどうかが鍵でした.[*13] また, 累積和を考えることで二分探索のできるデータに加工するという視点があると, 様々な問題に応用することが出来ます.

まとめ

累積和を事前に求めておくことで i 番目までの和を高速に求めることができる.
累積和の数列は単調性があるので二分探索することができる.

[*13] なおこの問題はよりよい計算量, 具体的には $O(N + M)$ で解くことが可能です. 本稿のメインテーマからそれてしまうのでここでその詳細については触れませんから興味のある方は実際に AtCoder のウェブサイトで調べてみてください.

2　単調性を見出すのは意外と難しいです

ショートストーリー：魔法少女 AtSum !!

Nyapo さんは人々を脅かす悪魔と戦う魔法少女です. 今までは正体を隠していましたが, 偶然にも AtSum くんに自分が魔法少女であるということがバレてしまいました.

AtSum　「Nyapo 姉！魔法少女だったんだな !!! 」
Nyapo　「バレてしまったからには仕方がないわね. ええ, そうよ. 」
AtSum　「すっげぇ！なあ Nyapo 姉, どうやったら, 魔法少女になれるんだ？」
Nyapo　「あなたは魔法少女になりたいの？」
AtSum　「うん！」
Nyapo　「それなら私が魔法を教えてあげるわ. 」
AtSum　「ほんと !? 」
Nyapo　「本当よ. ただし, 私が教えられるのは爆発魔法だから, その習得には阿鼻地獄をみる目になるよ. それでもいい？」
AtSum　「ああ！」

それから Nyapo さんの厳しい特訓が始まりました. 彼の肉体を襲う圧倒的筋肉痛, 疲労……！なるほど爆発魔法とはその効果と筋肉痛のメタファーとのダブルミーニングであったのだとこのとき初めて気づきました. しかしそれを耐え抜いた今, 彼は新たな魔法少女として悪魔と戦える力を得たのです.

Nyapo　「おつかれさま. まさかあなたが本当に魔法少女になれるだなんて思っていなかったわ. よく頑張ったわね. 」
AtSum　「ありがとう, Nyapo 姉」
Nyapo　「今や, あなたの爆発魔法は私のものよりも格上よ. 自信を持って……」

???　「ウキョキョキョーキョエーーーェエェーー!!」

二人　「「 !!! 」」
AtSum　「Nyapo 姉, これは……」
Nyapo　「ええ, ついに来たようね. 悪魔は N 体, 体力はそれぞれ h_i みたいよ. さあ, AtSum, あなたの爆発魔法を見せてやりなさい. 」
AtSum　「うん！」
Nyapo　「またいつ悪魔が来るかわからないから, なるべく少ない回数の魔法で倒すのよ. 」
AtSum　「え, でもどうやって……何回で倒せるかなんてわかんないよ. 」
Nyapo　「あら, 忘れてしまったの？あの夏の日, 私はあなたに強力な武器を授けたはずよ. 」
AtSum　「……そうか, うん, やってみる. 」
AtSum　「そうだ, 僕は悪魔を倒すんだ. Nyapo 姉に教えてもらった爆発魔法と……そして二分探索の力で !! 」

問題 (AtCoder Regular Contest075-D 問題より)

N 体の悪魔が横一列に並んでいる. 左から i 番目の悪魔の体力は h_i であり, 体力が 0 になると消滅する. あなたの爆発魔法が以下のような効果をもつとき最小で何回の攻撃で悪魔を全滅させられるか求めよ.

> ある悪魔を中心に爆発を起こす. 中心にいる悪魔には A のダメージ, それ以外の全ての悪魔に B のダメージを与える. なお $A > B$ である

ただし N, h_i, A, B は次の制約を満たす.

- $1 \leq N \leq 100,000$
- $1 \leq B < A \leq 1,000,000,000$
- $1 \leq h_i \leq 1,000,000,000$

サンプルは以下のとおりです.

$N = 6$, $A = 5$, $B = 2$
$(h_1,\ h_2, ...,\ h_6)$：$(1,\ 6,\ 3,\ 5,\ 4,\ 6)$
Ans. 3 回　左から 2 番目→ 6 番目→ 4 番目と攻撃することで最小攻撃回数を達成できます.

さて, どのように考えましょうか. ぱっと思いつく解法としては体力が最も残っている悪魔を中心に選んで攻撃していくというものですが, この方法では悪魔が 100,000 体いて, 全ての悪魔の体力が 1,000,000,000 であり, $A = 2$, $B = 1$ であるような場合にはざっくり考えてみても $\frac{10^5 \times 10^9}{2}$ くらいの計算が必要になってしまうので終わるはずがありません. ここで当たり前な知見でありながらも非常に重要な発想として

> K 回の爆発魔法で悪魔を全滅させることが可能であるとき,
> それより多い攻撃回数なら必ず悪魔を全滅させることができる.

というものを考えてみます. 即ち, 3 回の攻撃で大丈夫なら 4 回の攻撃でも 5 回でも全滅させることが可能であるということです. 何かピンときませんか？そう, ここには単調性が隠されているのです. ということは「K 回の攻撃で悪魔を全滅させることができるか」という判定問題をある程度高速に解くことができれば攻撃回数 K を二分探索することで答えをもとめられそうです .

では「K 回の攻撃で悪魔を全滅させることができるか」という判定問題を考えてみます. ちょうど K 回攻撃するとき, 全ての悪魔は最低でも $B \times K$ のダメージを負うことに注目します. すると爆発の中心にいた悪魔は $A - B$ のダメージを追加で受けることになります. よって 次のようにして判定問題を解くことができそうです.

判定問題の解法

- 一体一体悪魔を見ていく ← $O(N)$
  - その悪魔の体力から $B \times K$ を引く ← $O(1)$
  - 引いたあとの体力が 0 より大きかったら, 残りの体力が $A - B$ の追加ダメージ何回分に相当するか—即ち何回爆発の中心になる必要があるか—をカウントする. この総和を s とする ← $O(1)$
- s が K 以下であれば全滅させることができる, そうでなければ出来ないという答えを出す. ← $O(1)$

以上よりこの判定問題は $O(N)$ で解くことが出来ますから, K の範囲を R とすると全体として $O(N \log R)$ で解けることが解ります. 10^{18} 回程度攻撃すれば最も攻撃が必要なケースでも問題なく倒せるので $O(N \log R)$ は十分高速な解法であるということが判りました. 以上でこの問題も解くことが出来ました.

このように答えを決め打ってみることで上手く行くようなケースも非常に多いです.

まとめ
答えを決め打って二分探索してみると「判定問題を高速に解けるか？」という問題に言い換えられる場合がある.

おわりに

如何だったでしょうか. 本稿で見てきたように二分探索は非常に単純でありながらも, 強力なアルゴリズムです. もし, 日常生活の中で単調性を見つけたら二分探索してみては如何ですか？

さて, 本稿では二分探索というアルゴリズムを扱いましたが, これ以外にも世の中には様々なアルゴリズムがあります. 例えば「噂が伝わる人数」を考えたり, 「いくつかの町を経由してある町を訪れる場合の最短経路」を考えたり... 様々な状況で様々なアルゴリズムが考えられます. アルゴリズムは皆さんの生活とも密接に関わっているのです. また, 本稿では計算量という概念に触れましたが, その背後には膨大な計算理論や数学があり, 数学の未解決問題 "$P \neq NP$ 予想" にも繋がっています. もし本稿が皆さんの知的好奇心をくすぐり, 情報科学に興味を持っていただけたら筆者としてそれに勝る誉れはありません. 読んでいただきありがとうございました.