アルゴリズムとデータ構造

コンピュータサイエンスコース
知能コンピューティングコース

第6回

整列データの処理
数値計算の基本的なアルゴリズム

塩浦昭義
情報科学研究科 准教授

shioura@dais.is.tohoku.ac.jp
http://www.dais.is.tohoku.ac.jp/~shioura/teaching

中間試験について

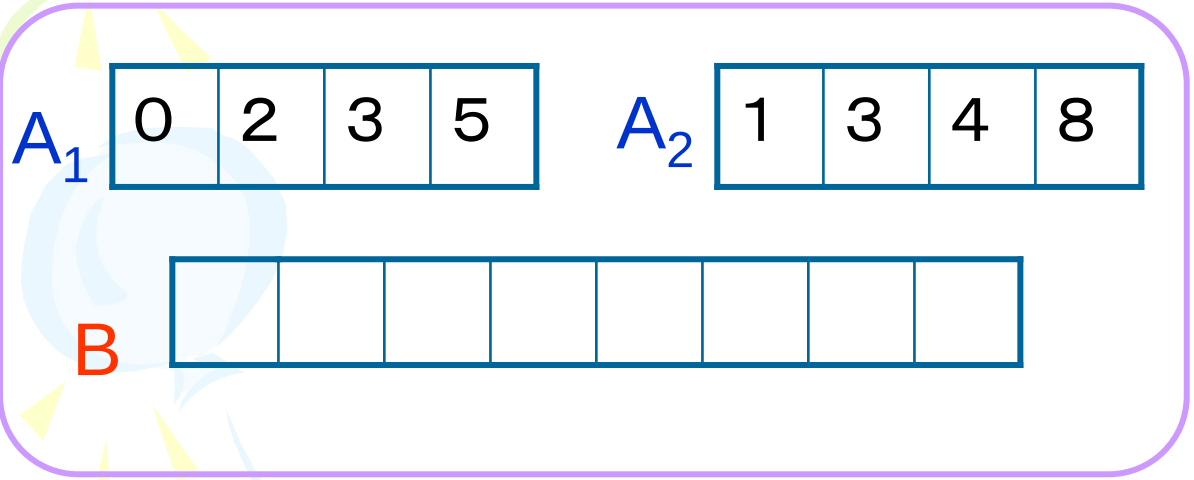
- 日時：6月4日（木）8:50～10:20
- 過去にレポートを一度も出していない場合，受験は不可
- 教科書，ノート等の持ち込みは一切不可
- 座席はこちらで指定
- 試験内容：第5回目までの講義で教えたところ
  - アルゴリズムやデータ構造の挙動
  - 時間計算量の解析，および関連する証明問題
  - 用語の定義，など
- 50点満点，24点以下は追試レポートもしくは単位不可

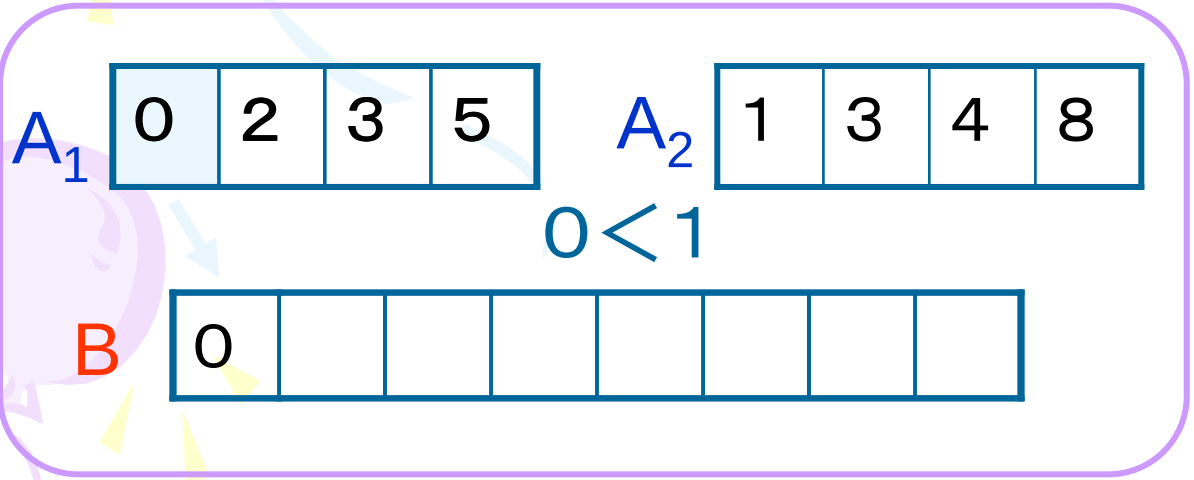
整列データの処理

- 整列(ソート)されたデータの処理は，整列されていない場合に比べて効率よく実行できることが多い
  - 複数の整列データのマージ(併合)
  - 共通要素の列挙
  - 二分探索
  - などなど

整列配列のマージ

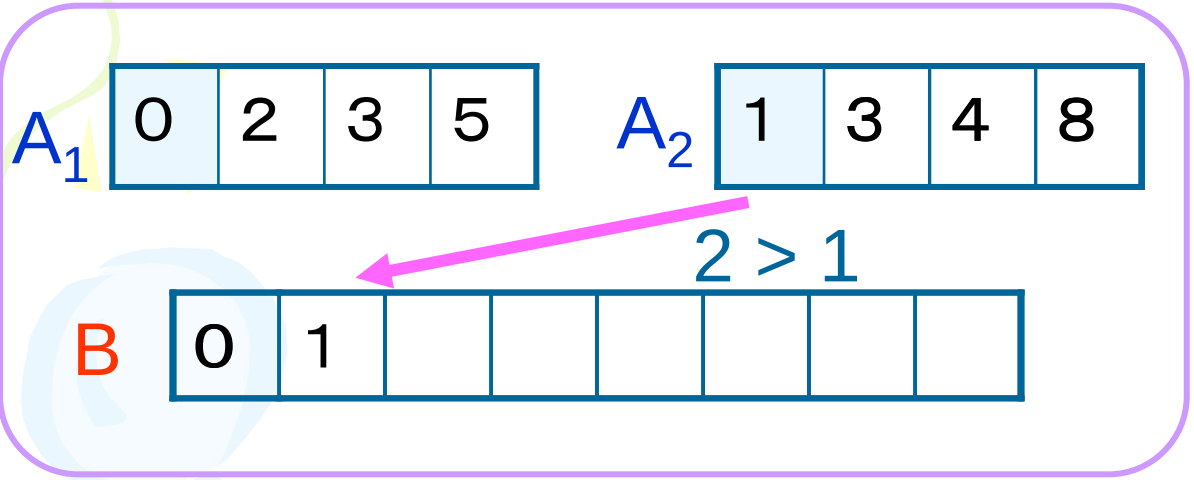
- 2つの整列配列のマージは線形時間で可能



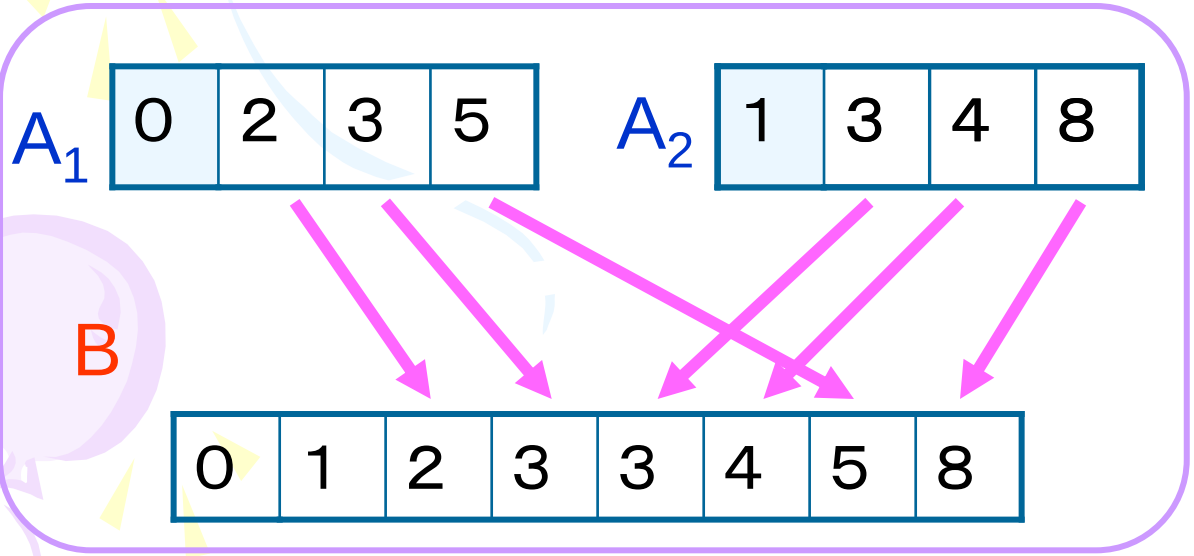


A_1 と A_2 の先頭の
数字を比較
小さいほうを B の
空欄の先頭へ移動

整列配列のマージ



A_1 と A_2 の先頭の数字を比較
小さいほうを B の空欄の先頭へ移動



この作業を繰り返す

整列配列のマージの時間計算量

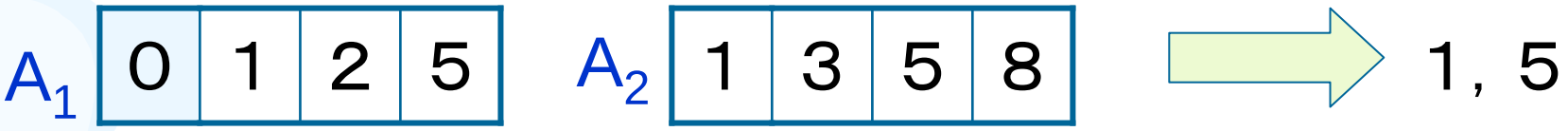
- 2つの配列の長さをそれぞれ n_1, n_2 とする
- 各反復は O(1) 時間で実行可能（比較とデータ挿入）
- 一回の反復の後，配列Bの要素は1つ増える
  ➔反復回数は高々 n_1+n_2

∴ 合計の計算時間は $O(n_1 + n_2)$

※参考：2つの配列がソートされていないときは
$O(n_1 \log n_1)+O(n_2 \log n_2)+O(n_1+n_2)$時間

共通要素の列挙

- 与えられた2つの配列の両方に共通して存在する要素を全て出力する



配列のマージと同様のやり方により，
$O(n_1+n_2)$時間で実行可能

アルゴリズムの流れ：

A_1 と A_2 の先頭の数字を比較

同じ値➔その値を出力し，両方の値を削除

異なる値➔小さい方を削除

共通要素の列挙

0と1を比較
異なる値なので，0を削除

1と1を比較
同じ値なので，1を出力し，両方の値を削除

2と3を比較
異なる値なので，2を削除

共通要素の列挙

5と5を比較
同じ値なので，5を出力し，両方の値を削除

A_1の要素を全て調べ終わったので終了
共通する要素は1，5のみ

二分探索の考え方

教科書の記述と
少し違います

- 昇順にソートされた配列 A[0], A[1], …, A[n-1] の中に整数 x が存在するかどうか，調べたい

0	1	2	3	4	5	6	7	8	9
?	?	?	?	?	?	?	?	?	?

真ん中の要素（例えばA[4]）を調べる
A[4] = x ➔ 終了，整数 x が配列の中に存在
A[4] < x ➔ 整数 x が存在するならば A[4] より右側
➔ A[5], …, A[9] の中を調べればよい
A[4] > x ➔ 整数 x が存在するならば A[4] より左側
➔ A[0], …, A[3] の中を調べればよい
探索する範囲が半分になる---二分探索と呼ばれる

二分探索の流れ

$A[0] \leq A[1] \leq \cdots \leq A[n-1]$ を仮定

(1): $i_L := 0$, $i_R := n-1$ とおく.

(2): $i_M := \lfloor (i_L + i_R)/2 \rfloor$ とおく.

(3): $A[i_M] = x$ ならば終了，配列の中にxが存在.

$A[i_M] > x$ ならば $i_R := i_M - 1$ とおく.

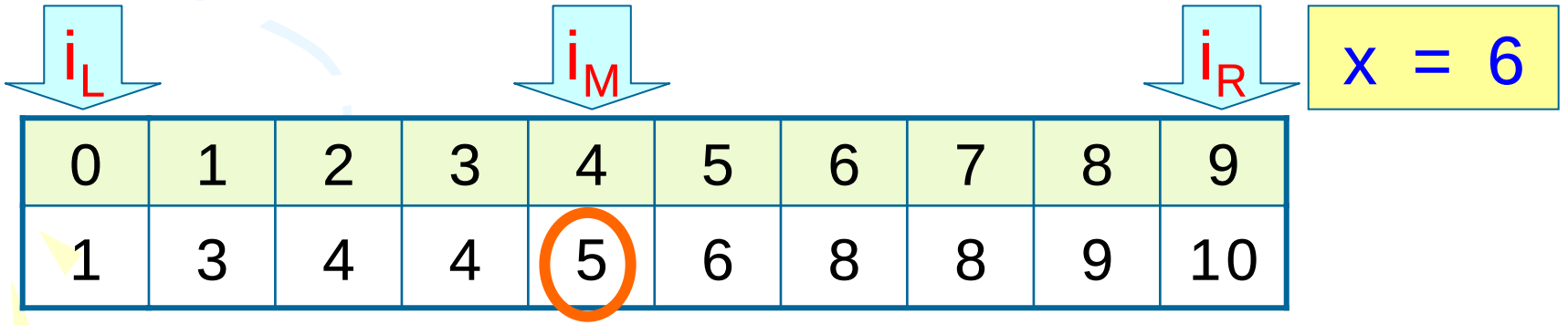
$A[i_M] < x$ ならば $i_L := i_M + 1$ とおく.

(4): (2) に戻る

$A[i_L]$と$A[i_R]$の間を探索

$A[i_M]$は真ん中の要素

$A[i_M] = 5 < 6 = x$
➔ $i_L := i_M + 1 = 5$



二分探索の流れ

(2): $i_M := \lfloor (i_L + i_R)/2 \rfloor$ とおく.

(3): $A[i_M] = x$ ならば終了，配列の中にxが存在.

$A[i_M] > x$ ならば $i_R := i_M - 1$ とおく.

$A[i_M] < x$ ならば $i_L := i_M + 1$ とおく.

(4): (2) に戻る

$A[i_M] = 8 > 6 = x$
➔ $i_R := i_M - 1 = 6$

i_L i_M i_R x = 6

0	1	2	3	4	5	6	7	8	9
1	3	4	4	5	6	8	8	9	10

二分探索の流れ

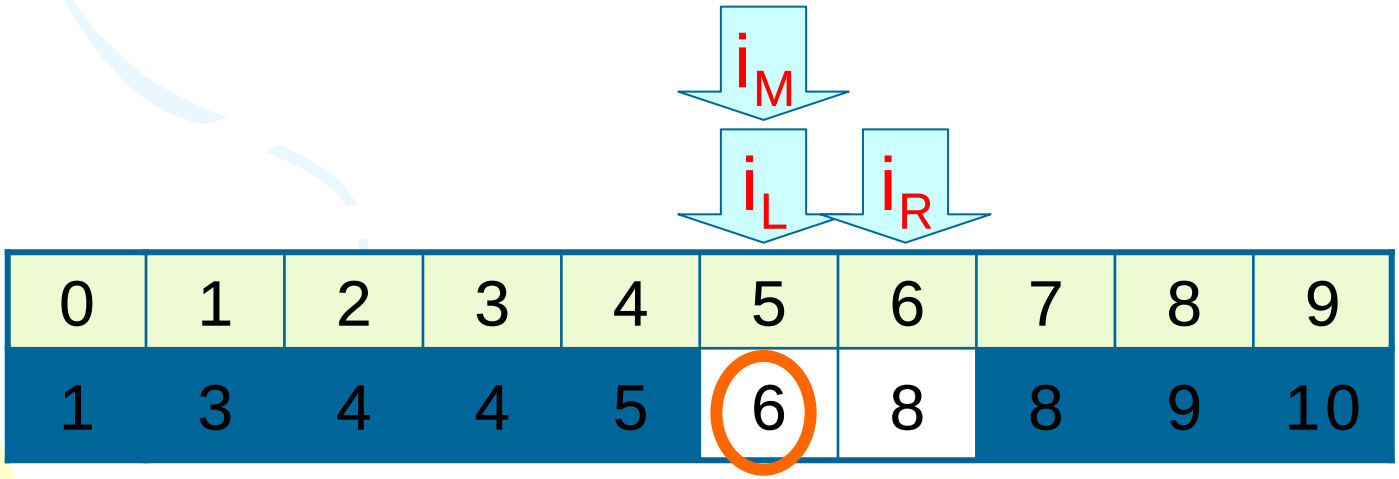
(2): $i_M := \lfloor (i_L + i_R)/2 \rfloor$ とおく.

(3): $A[i_M] = x$ ならば終了，配列の中にxが存在.

$A[i_M] > x$ ならば $i_R := i_M - 1$ とおく.

$A[i_M] < x$ ならば $i_L := i_M + 1$ とおく.

(4): (2) に戻る



x = 6

二分探索の時間計算量

- 配列の長さをそれぞれ n とする
- 各反復は O(1) 時間で実行可能
- 一回の反復の後，i_R-i_L の値は半分以下に減少

  ➔反復回数は高々 $\log_2 n$

∴ 合計の計算時間は $O(\log_2 n)$

※参考：配列がソートされていないときはO(n)時間

数値計算

- 数値計算の問題---所望の数値を計算する問題
  - 連立一次方程式や非線形方程式の解の計算
  - 微分不等式の解の計算
  - 行列の固有値の計算
- 答え，および計算の途中で表れる数値は一般の場合，実数（複素数の場合もあり）

  ➔ 数値誤差を考慮する必要がある
  - コンピュータ上で無理数を厳密に表現することは不可能
  - 有理数でも表現可能な桁数に制限有り

    (例：1/3＝0.333333 と途中で打ち切り)

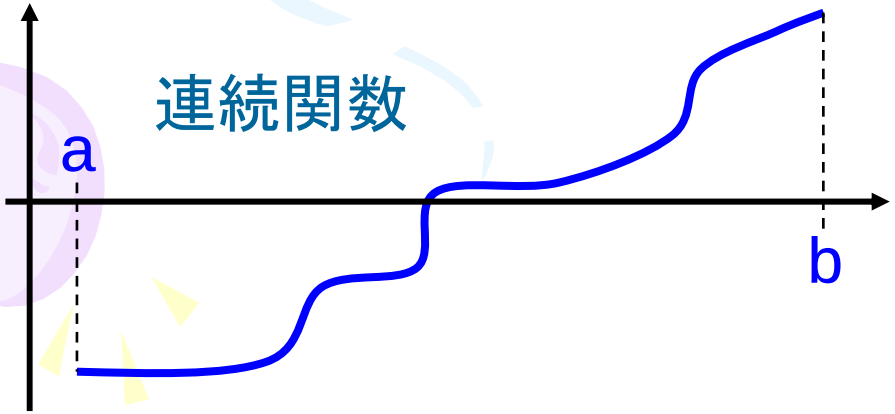
数値計算のアルゴリズム

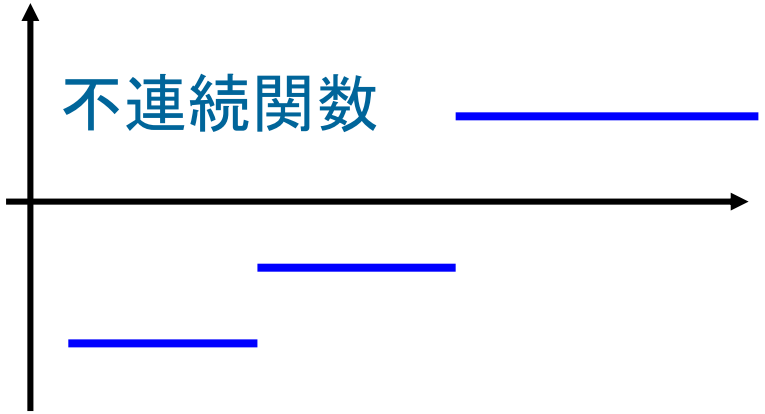
- 数値計算のアルゴリズムの要件
  - アルゴリズムがわかりやすい，プログラムを作りやすい
  - 計算時間が理論・実用の両面で少ない
  - 空間計算量が少ない
  - 計算結果の誤差が小さい
  - 計算結果の誤差の範囲が保証されている

単調関数の零点を求める

- 授業で扱う問題
  - 単調非減少な連続関数の零点を求める
  - x は関数 f の零点 ←→ f(x) = 0

中間値定理: 一変数の連続関数 f と実数 a, b（ただし a < b）が f(a) < 0 < f(b) を満たす
➔区間 [a, b] において関数 f は零点をもつ
（∃x*∈[a, b]: f(x*) = 0）

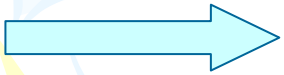




単調関数の零点を求める問題

零点を求める問題（数学版）
入力: 一変数の関数 f, 実数 a, b, ただし $a < b$, $f(a) < 0 < f(b)$
区間 [a, b] において関数 f は単調非減少かつ連続
出力: 関数 f の零点 $x \in [a, b]$

しかし，零点は実数値➔厳密に計算することは困難
零点であること ($f(x)=0$) を厳密に判定することも困難



誤差を許すことにする（近似解で我慢する）

零点を求める問題（数値計算版）
入力: 一変数の関数 f, 実数 a, b, ただし $a < b$, $f(a) < 0 < f(b)$
区間 [a, b] において関数 f は単調非減少かつ連続
正の実数 ε , δ
出力: 関数 f の零点 x^* に対して $|x - x^*| \leqq \varepsilon$ を満たす $x \in [a, b]$
もしくは $|f(x)| \leqq \delta$ を満たす $x \in [a, b]$

零点の計算に対する二分探索

- 二分探索により零点の近似解は計算可能

(1): $x_L := a$, $x_R := b$ とおく.

(2): $x_M := (i_L + i_R)/2$ とおく.

(3): $|f(x_M)| \leq \delta$ ならば終了. x を出力.

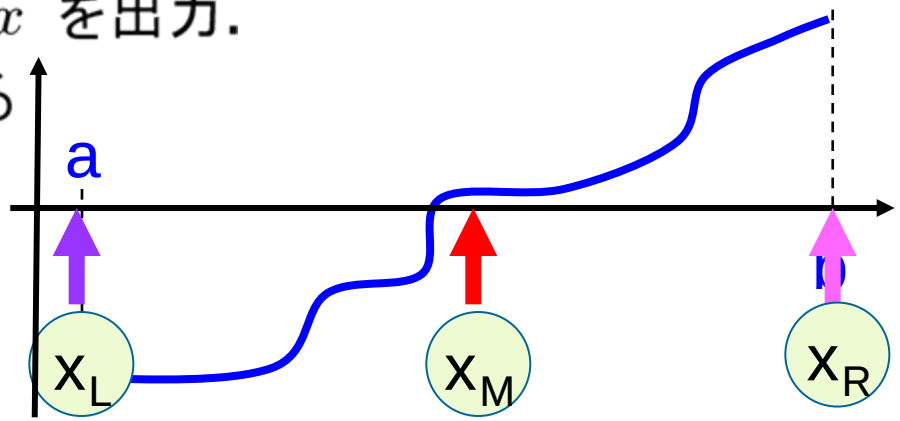
$f(x_M) > \delta$ ならば $x_R := x_M$ とおく.

$f(x_M) < -\delta$ ならば $x_L := i_M$ とおく.

(4): $x_R - X_L \leq \epsilon$ ならば終了. x を出力.

$x_R - X_L > \epsilon$ ならば **(2)** に戻る

常に
$f(x_L) < 0 < f(x_R)$
を満たす



零点の計算に対する二分探索

- 二分探索により零点の近似解は計算可能

(1): $x_L := a$, $x_R := b$ とおく.

(2): $x_M := (i_L + i_R)/2$ とおく.

(3): $|f(x_M)| \leq \delta$ ならば終了. x を出力.

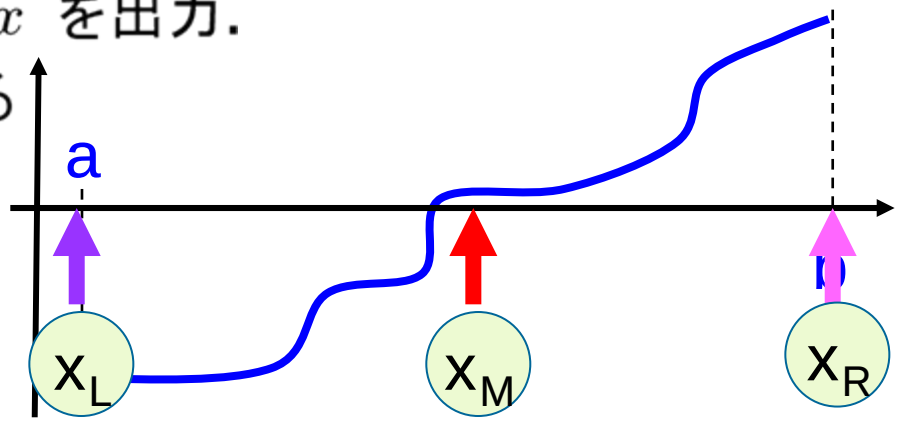
$f(x_M) > \delta$ ならば $x_R := x_M$ とおく.

$f(x_M) < -\delta$ ならば $x_L := i_M$ とおく.

(4): $x_R - X_L \leq \epsilon$ ならば終了. x を出力.

$x_R - X_L > \epsilon$ ならば **(2)** に戻る

常に
$f(x_L) < 0 < f(x_R)$
を満たす



零点の計算に対する二分探索

- 二分探索により零点の近似解は計算可能

(1): $x_L := a$, $x_R := b$ とおく.

(2): $x_M := (i_L + i_R)/2$ とおく.

(3): $|f(x_M)| \leq \delta$ ならば終了. x を出力.

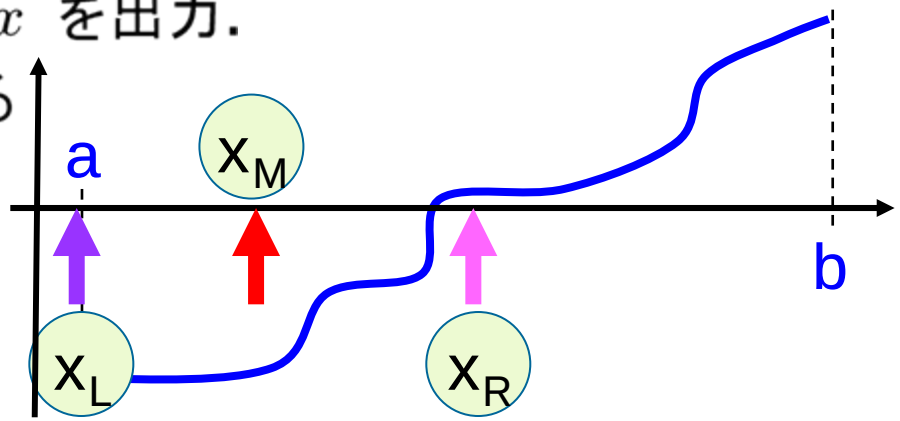
$f(x_M) > \delta$ ならば $x_R := x_M$ とおく.

$f(x_M) < -\delta$ ならば $x_L := i_M$ とおく.

(4): $x_R - X_L \leq \epsilon$ ならば終了. x を出力.

$x_R - X_L > \epsilon$ ならば **(2)** に戻る

常に
$f(x_L) < 0 < f(x_R)$
を満たす



零点の計算に対する二分探索

- 二分探索により零点の近似解は計算可能

(1): $x_L := a$, $x_R := b$ とおく.

(2): $x_M := (i_L + i_R)/2$ とおく.

(3): $|f(x_M)| \leq \delta$ ならば終了. x を出力.

$f(x_M) > \delta$ ならば $x_R := x_M$ とおく.

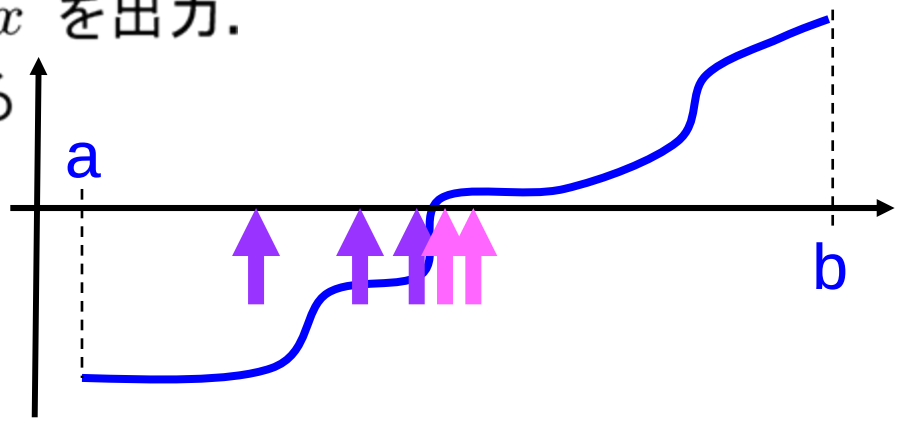
$f(x_M) < -\delta$ ならば $x_L := i_M$ とおく.

(4): $x_R - X_L \leq \epsilon$ ならば終了. x を出力.

$x_R - X_L > \epsilon$ ならば **(2)** に戻る

常に
$f(x_L) < 0 < f(x_R)$
を満たす

x_L と x_R が
十分近くなったら
終了



零点の計算に対する二分探索の時間計算量

- 各反復で $x_R - x_L$ は半分に減少
- 初期値は $b - a$
- $x_R - x_L \leqq \varepsilon$ となったら終了

➔反復回数を k とすると,

$$(b-a)\left(\frac{1}{2}\right)^{k-1} > \varepsilon \geq (b-a)\left(\frac{1}{2}\right)^{k}$$

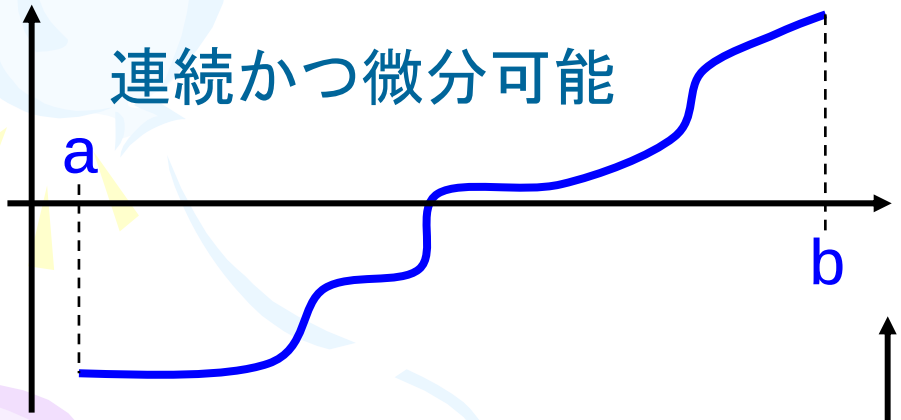
$$\therefore\ k = \left\lceil \log_2 \frac{b-a}{\varepsilon} \right\rceil$$

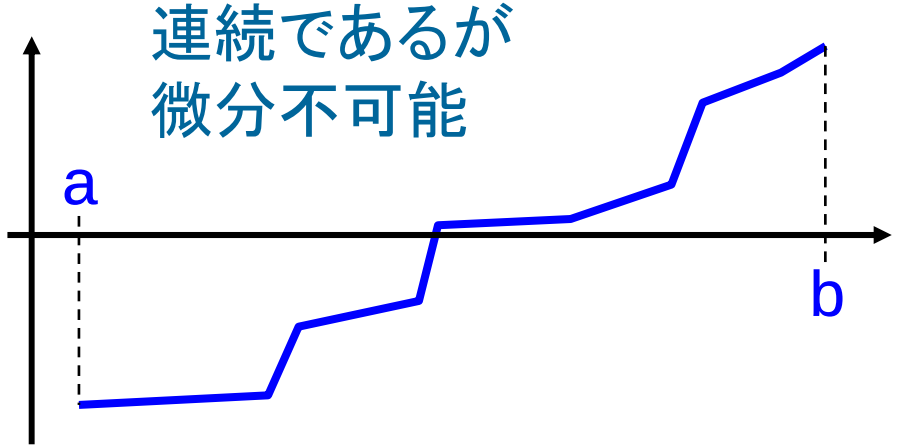
関数値 $f(x)$ の計算が定数時間で出来るならば,

時間計算量は $O\left(\log_2 \frac{b-a}{\varepsilon}\right)$

零点の計算に対するニュートン法

- 関数 f が単調増加かつ微分可能ならば，ニュートン法が利用可能
  - f が単調増加 ⇔ x < y ならば f(x) < f(y)



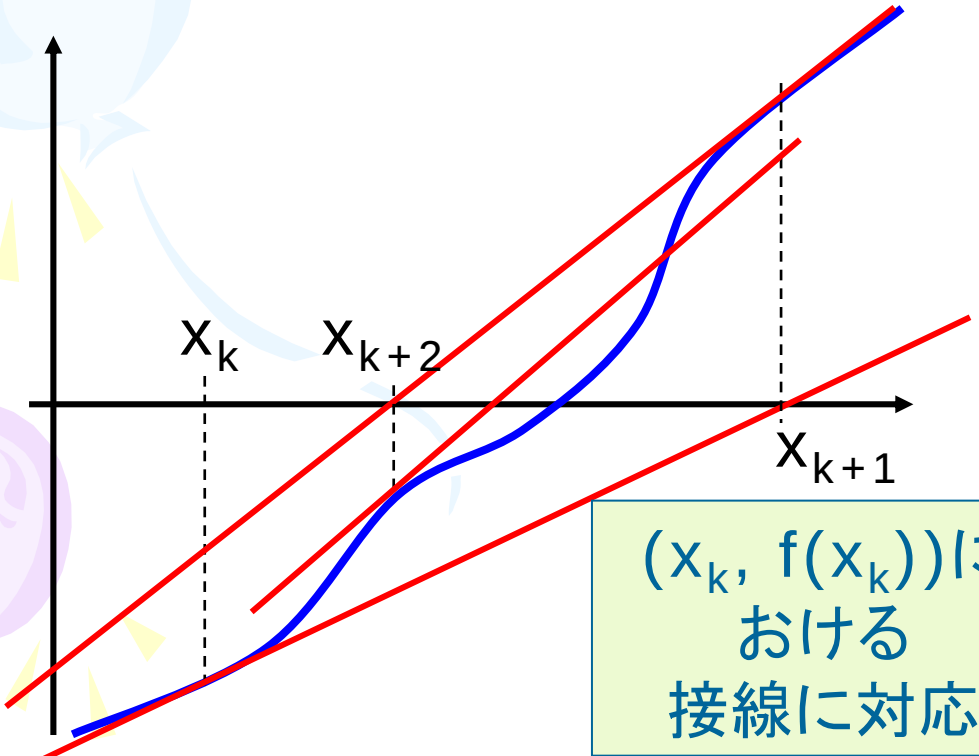


ニュートン法の考え方

- 現在の $x = x_k$ において関数を一次近似

$$f(x) \Rightarrow f(x_k) + f'(x_k)(x - x_k)$$

- 近似した関数の零点を次の $x = x_{k+1}$ とおく



$(x_k, f(x_k))$における接線に対応

x_kとx_{k+1}は次の式を満たす

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

アルゴリズムでは，この式を使って繰り返しx_kを更新

ニュートン法の収束性

- ニュートン法のように，解候補 x_k を繰り返し生成するアルゴリズムの性能は，

  解候補が解 x^* に収束する速さ

  によって評価
  - 正の定数 α が存在して，各反復で
    $|x_{k+1} - x^*| \leqq \alpha |x_k - x^*|$ が成り立つ➔一次収束
  - 正の定数 α が存在して，各反復で
    $|x_{k+1} - x^*| \leqq \alpha |x_k - x^*|^2$ が成り立つ➔二次収束
  - 二次収束の方が一次収束より速い
  - ニュートン法は（ある仮定の下で）二次収束