アルゴリズム論

2017年5月8日

樋口文人

目次

- 課題について
- 効率の良い探索
  - 二分探索
  - 数値計算への応用
- 文字列の探索

これまでに見た探索アルゴリズム

- ランダムに探す
- 線形探索：順番に探す
- どちらの方法もデータの並びがランダムか整列済みかに係らず探索の効率は同じ

見つけるまでの手間

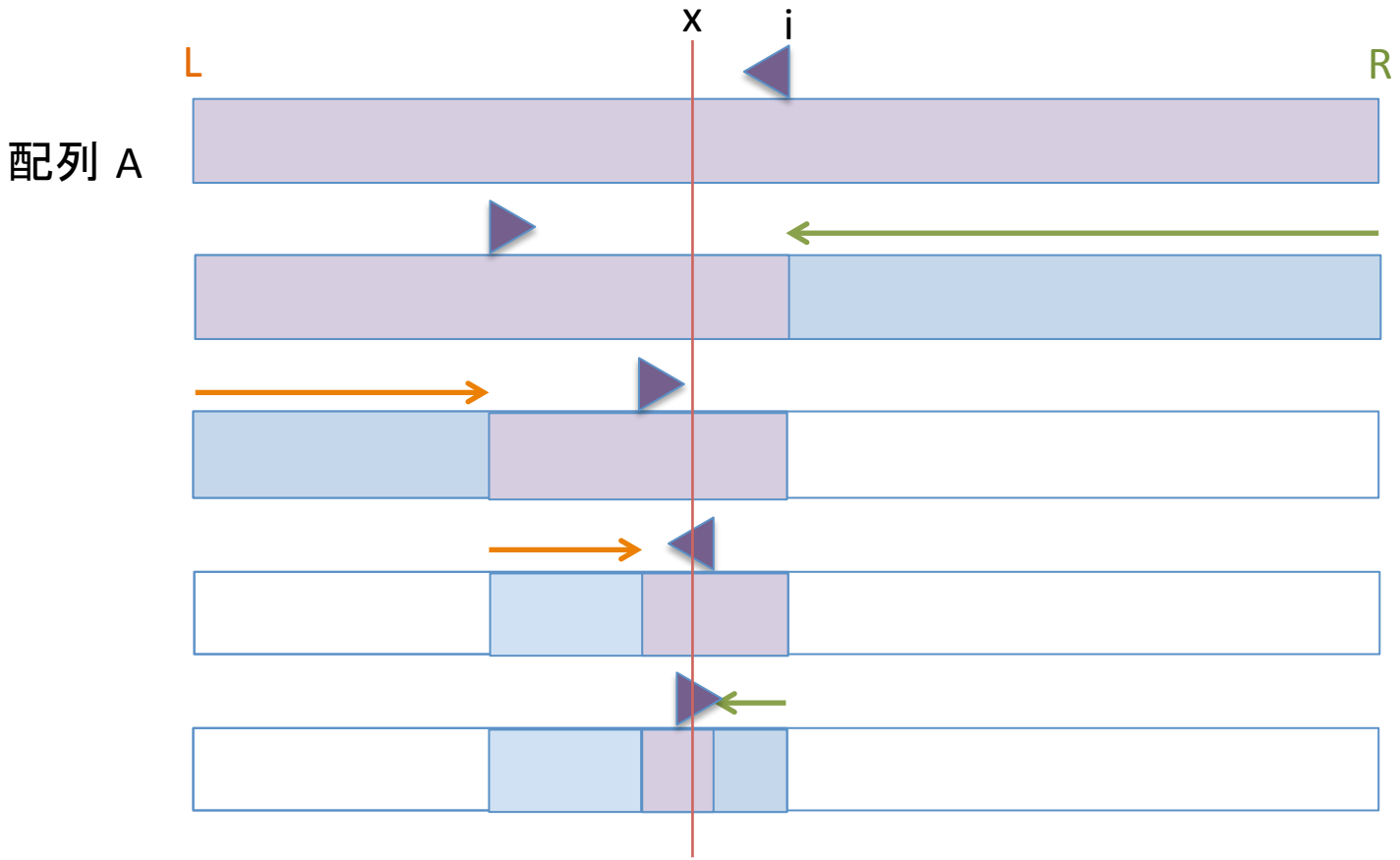
探索手法 ＼ データの並び	ランダム	線形探索
ランダム	最良：1 平均：N 最悪：∞	最良：1 平均：(N+1)/2 最悪：N
整列済み	最良：1 平均：N 最悪：∞	最良：1 平均：(N+1)/2 最悪：N

整列済みデータの探索

二分探索

二分探索

- 探索範囲を順に2分の1に狭めていく

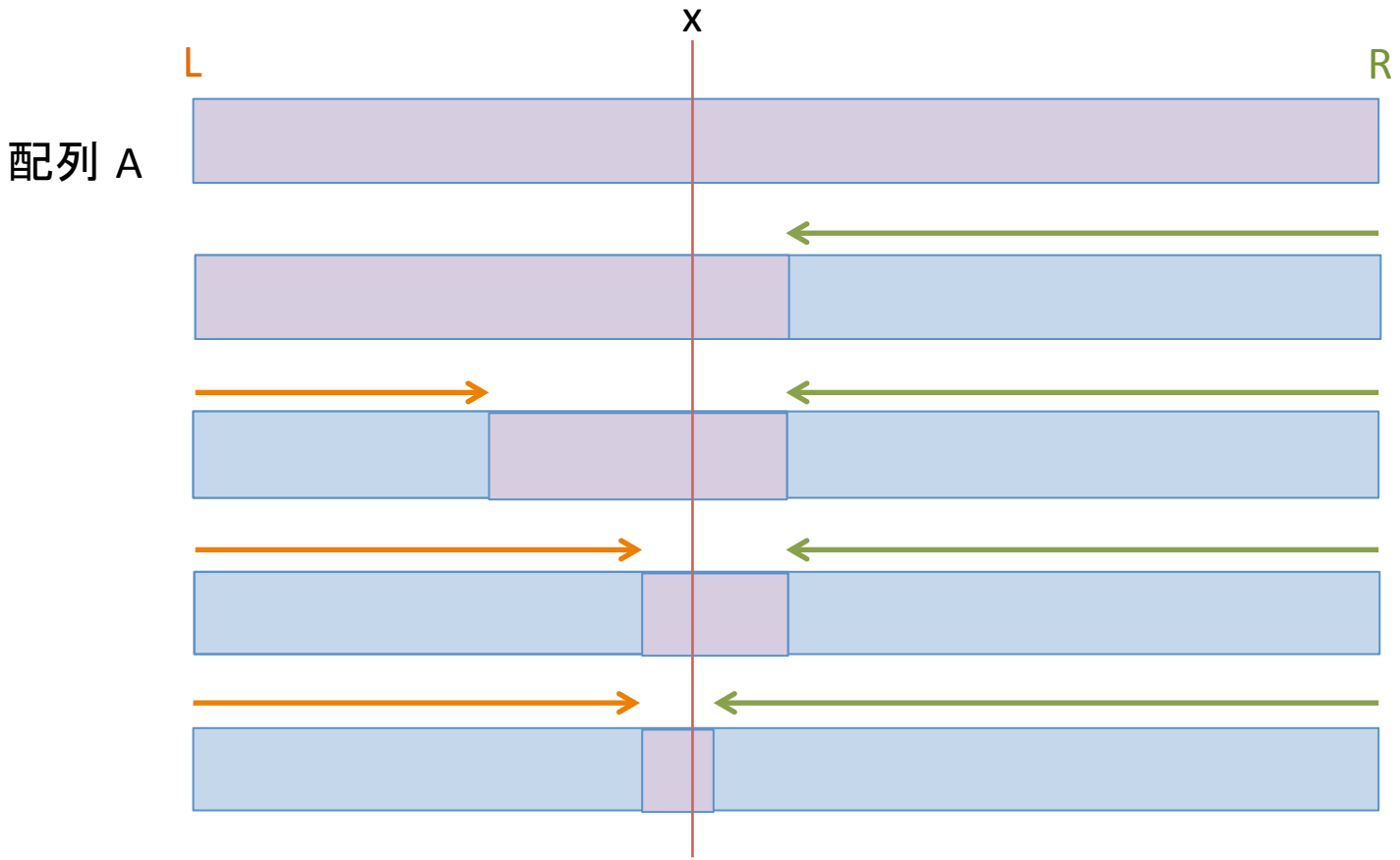


探索範囲と比較対象

- 区間 : L=1, R=n 比較対象 : i=[(L+R)/2] = [(1+n)/2]
  - x < A[i] : L = L, R = i, i = [(L+R)/2]
  - A[i] < x : L = i+1, R = R, i = [(L+R)/2]
- 条件
  - L >= R なら終了
- log n で探索を終了できる

考え方

- 探索除外範囲を全体にまで広げる



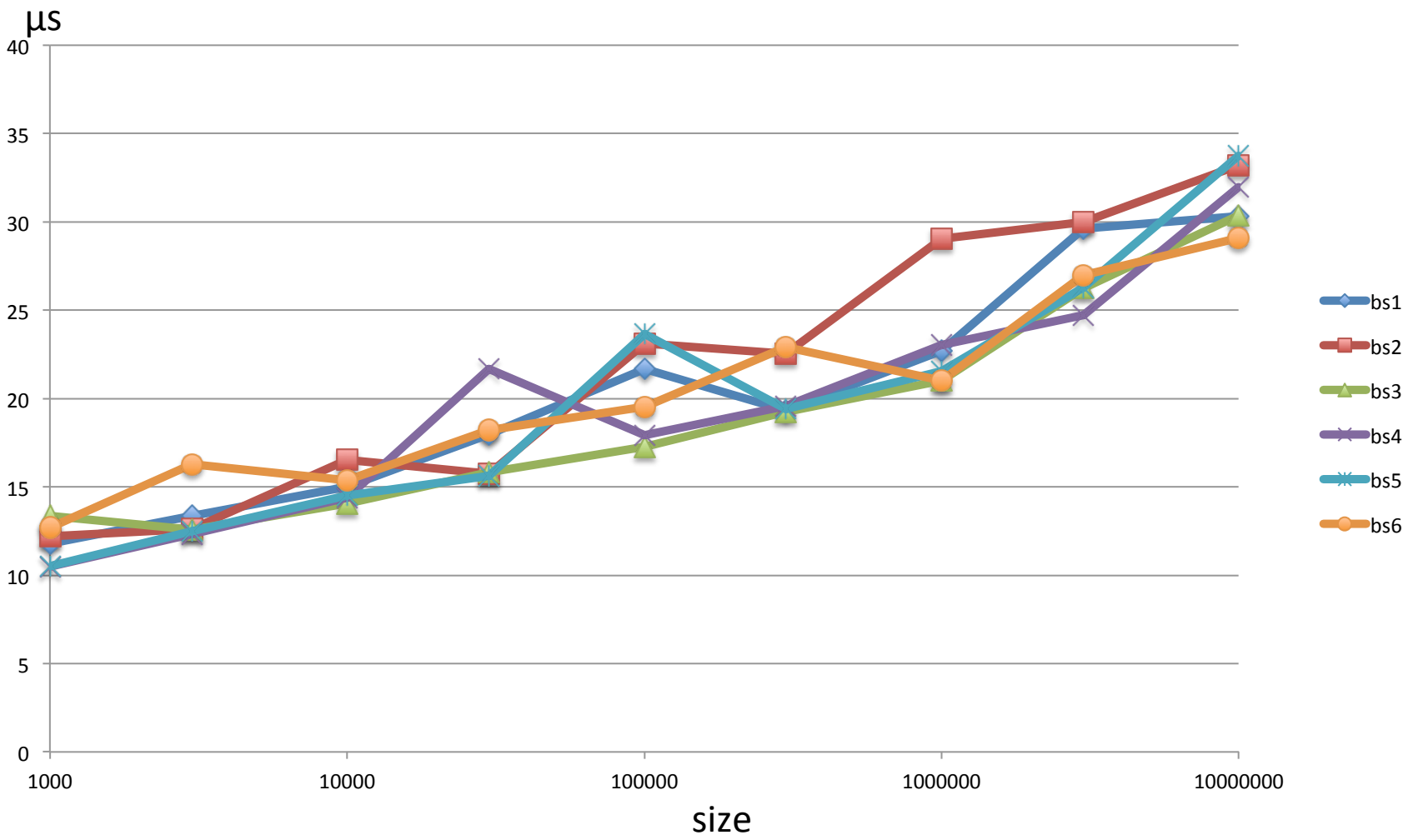
Python: 二分探索

```
import math
import time
import array
import random

repeats = [1000, 3000, …, 10000000];
tries = 10000;

for n in repeats:
  base = array.array('l', range(n));
  ary = [];
  for i in range(n):
    ary.append(random.choice(base));
  ary.sort()
  pos = 0; step = 0;
  start_time = time.time();
  for i in range(tries):
    target = random.randint(0, n-1);
    low = 0;
    high = n-1;
    found = False;
    while (not found and low < high):
      mid = int(math.ceil((low + high)/2));
      step = step + 1;
      if (ary[mid] == target):
        found = True;
      else:
        if (ary[mid] > target):
          high = mid-1;
        else:
          low = mid+1;
    pos = pos + mid;

  print "time: binary search ", n,
(time.time()-start_time)*1e6/tries, "[us]",
" found at ", float(pos)/tries/n, " with ",
float(step)/tries, " steps.";
```

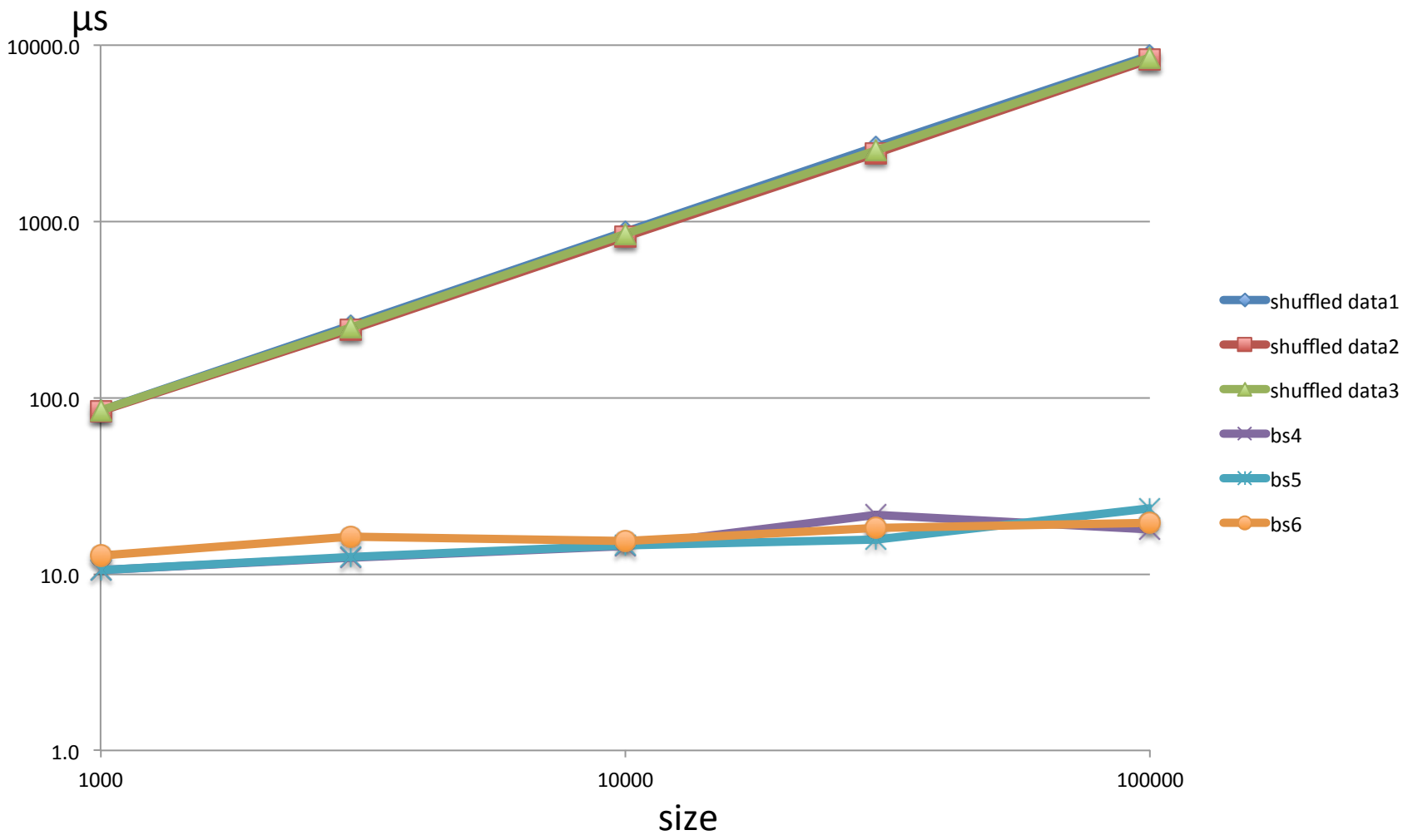
Python: 二分探索 主要部

```
target = random.randint(0, n-1);
low = 0;
high = n-1;
found = False;
while (not found and low < high):
  mid = int(math.ceil((low + high)/2));
  step = step + 1;
  if (ary[mid] == target):
    found = True;
  else:
    if (ary[mid] > target):
      high = mid-1;
    else:
      low = mid+1;
```

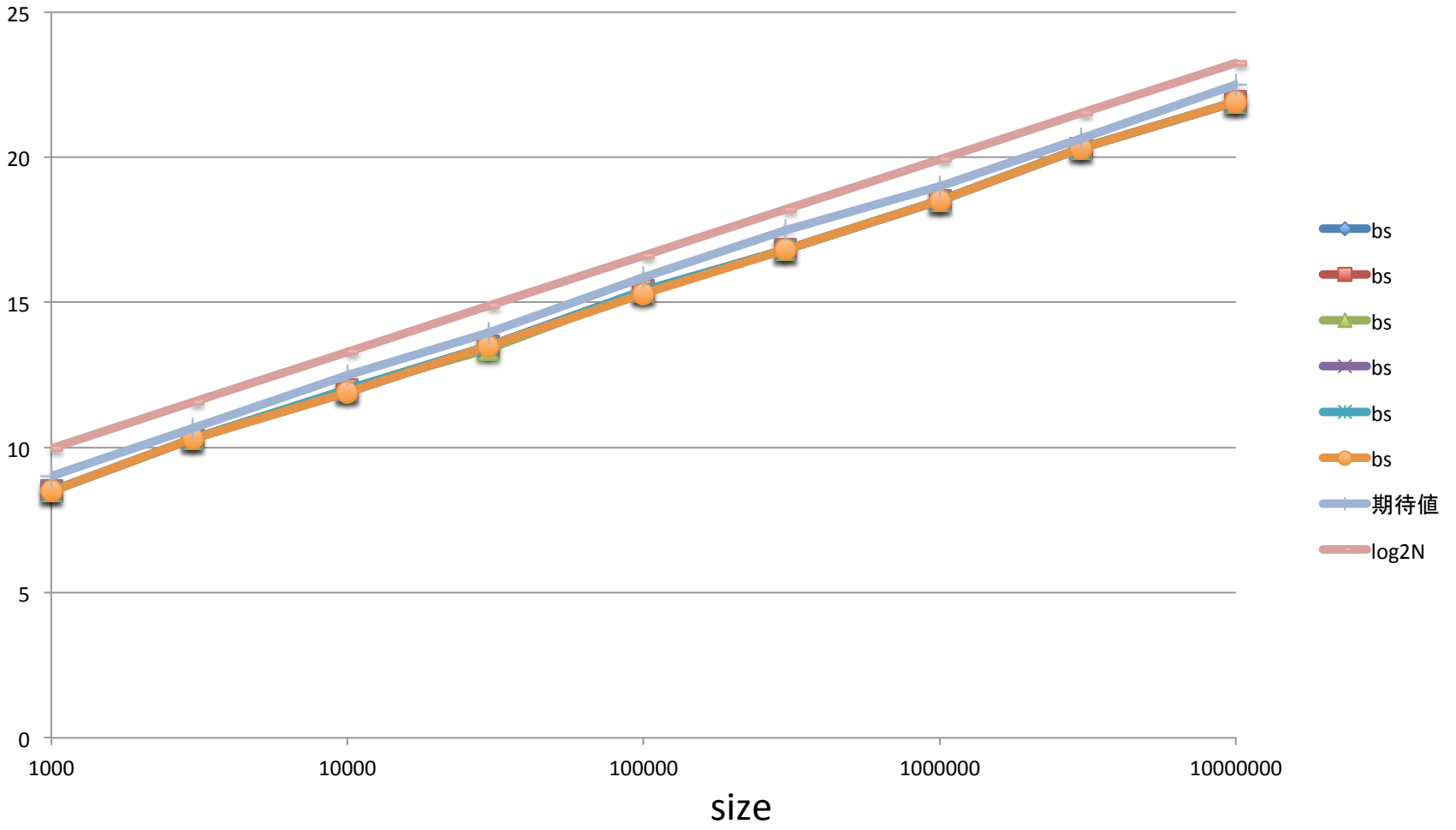
二分探索の探索時間


μs
40
35
30
25
20
15
10
5
0
1000
10000
100000
1000000
10000000
size
bs1
bs2
bs3
bs4
bs5
bs6

線形探索との比較


μs
10000.0
1000.0
100.0
10.0
1.0
1000
10000
100000
size
shuffled data1
shuffled data2
shuffled data3
bs4
bs5
bs6

二分探索の探索ステップ数

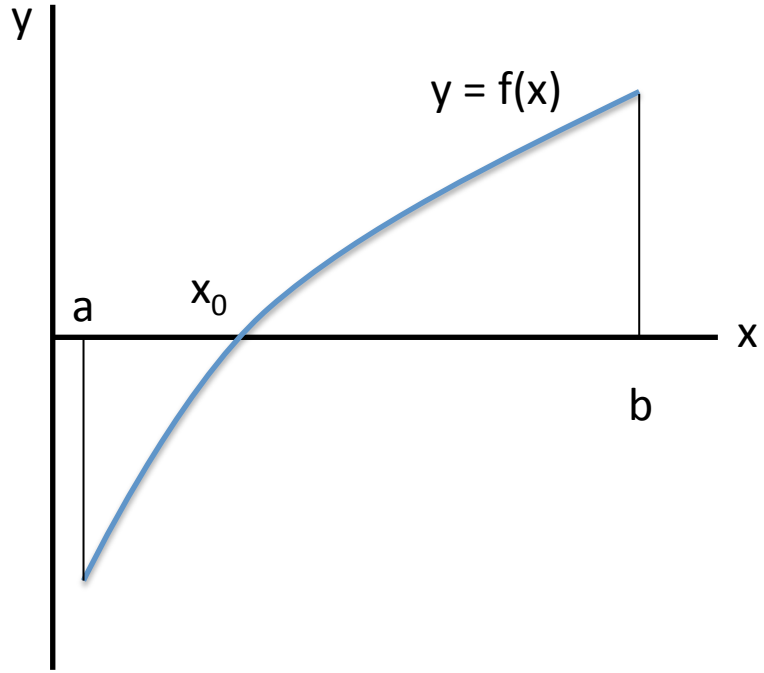


質問

- 以下の条件はどのように表現できるか？
  - 凡例:
    - 配列 A, 添字 i = 1…n, 探索区間 [L, R], 探索対象 x
  1. 配列の要素（データ）が整列済みである
  2. 探索除外範囲に探索対象は無いという条件はどう表せる？
  3. 探索対象となる複数のデータが配列に含まれる場合はどうか？

二分探索の応用

- f(x)=0 の解 x_0 を（数値的に）見つける
  - 関数 f(x) が区間 [a, b] で連続
  - f (a) * f(b) < 0



$d = 10^{-15}$

$x_i = a$

repeat

　　prev = x_i

　　$x_i = (a+b)/2$

　　もし $f(x_i) < 0$ なら $a=x_i$

　　もし $f(x_i) > 0$ なら $b=x_i$

until $|prev - x_i| < d$

$x_0 = x_i$

文字列探索

文字列の探索

- 検索エンジン
- テキストマイニング
- 遺伝子　塩基配列の照合

文字列の探索

S : テキスト i.e. 探索範囲

P: パターン i.e. 探索の対象

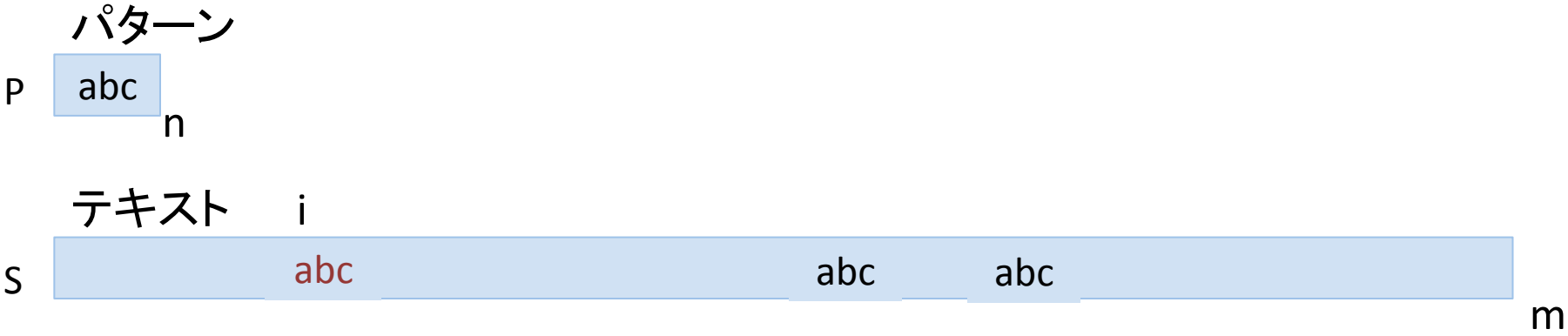
m: テキストの長さ

n: パターンの長さ

i: テキスト中の最初のパターンの先頭文字の位置

n <= m （通常は n << m）

文字列の探索



探索の範囲：先頭から m-n+1 まで
または：テキストの長さはパターンと同じか，それよりも長い n <= m

合致（マッチ）の条件：j = 1, …, n に対して S[i + j -1] == P[j]
位置 i でこの条件が満たされることを G[i] と記すことにする

最初の合致（マッチ）である条件：k = 1, …, i-1 に対して not G[i]

素朴な文字列探索

- テキストの先頭文字から始め
- パターンの先頭文字とテキストと照合
- 不一致ならテキストの次の文字へ移る
  - 一致ならテキストの次の文字とパターンの次の文字を照合
  - 不一致が見つかるか，パターンのすべての文字の一致を確認するまで繰り返す
- テキストの残りの長さがパターンより短くなったら終了

素朴な探索

- テキスト：中野区の中野駅には中野区役所や明治大学の中野キャンパスがある
- パターン：中野区役所

中野区の中野駅には中野区役所や明治大学の中野キャンパスがある

中野区役所　中野区役所　中野区役所　中野区役所　中野区役所
中野区役所　中野区役所　中野区役所　中野区役所　中野区役所
中野区役所　中野区役所　中野区役所　中野区役所
中野区役所　中野区役所　中野区役所　中野区役所
中野区役所　中野区役所　中野区役所　中野区役所
中野区役所　中野区役所　中野区役所　中野区役所
中野区役所　中野区役所　中野区役所　中野区役所

sample: Knuth-Morris-Pratt

from Niklaus Wirth

```
Hoola-Hoola girls like Hooligans.
Hooligan
     Hooligan
      Hooligan
       Hooligan
           Hooligan
             .......
                          Hooligan
```

Knuth-Morris-Pratt

```
import sys
import time
pattern = "マークの加工"
print pattern # 検索文字列
tg = open("target.txt", "r")
s = tg.read(-1) # negative means "all”
for n in range(20):
  tries = 1000
  start_time = time.time()
  for k in range(tries):
  if (tg.read() == “”):
    if (len(s) < len(pattern)):
    sys.exit()
    p = 0
    start = -1
    found = False
    match = False
    end = len(s)+1-len(pattern)
    for i in range(0,len(s)):
    ch = s[i]
    if ( ch == pattern[p]):
      match = True
      if (p == 0):
      start = i
      if (p < len(pattern) - 1):
      p = p + 1
      else:
      found = True
      break
    else:
      match = False
      p = 0
print "time: text search basic",
(time.time()-start_time)*1e3/tries,
"[ms]", found;
```

パターンの構造を分析

- パターン後方からの各文字の位置
  - 区所中野役
    ３１５４２
- 不一致となったテキスト上の文字とパターンを比較

中野区の中野駅には中野区役所や明治大学の中野キャンパスがある

中野区役所　中野区役所　中野区役所　中野区役所　中野区役所

中野区役所　中野区役所　中野区役所　中野区役所　中野区役所

中野区役所　中野区役所　中野区役所　中野区役所

中野区役所　中野区役所　中野区役所　中野区役所

中野区役所　中野区役所　中野区役所　中野区役所　中野区役所

中野区役所　中野区役所　中野区役所　中野区役所

sample: Boyer-Moore

from Niklaus Wirth

```
Hoola-Hoola girls like Hooligans.
Hooligan
     Hooligan
       Hooligan
                Hooligan
                         Hooligan
```

Boyer - Moore

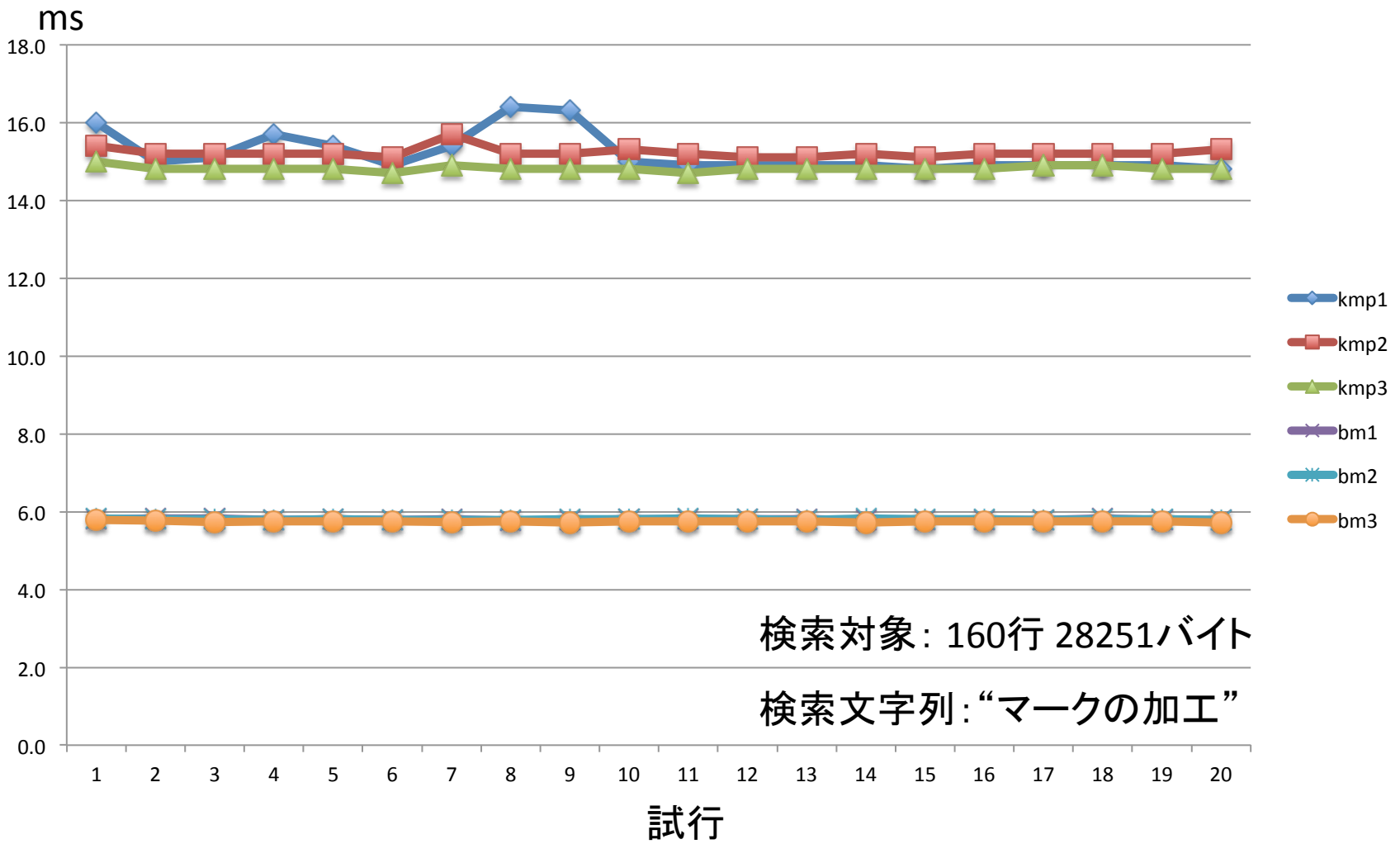
```
import sys
import time

pattern = "マークの加工"
print pattern # 検索文字列
dict = {} # 検索文字列の末尾からの各文字の位置
for i in range(len(pattern)):
  dict[pattern[i]] = len(pattern) - i + 1
tg = open("target.txt", "r")
s = tg.read(-1) # negative means "all”
for n in range(20):
  tries = 1000
  start_time = time.time()
  for k in range(tries):
  if (tg.read() == “”): # ファイルをすべて読み込んだか
  if (len(s) < len(pattern)):
  sys.exit() # 検索文字列より小さければ終了
  p = 0
  start = 0
  found = False
  match = False
  end = len(s)+1-len(pattern)
  i = 0
  while i < end:
  ch = s[i]
  if ( ch == pattern[p]):
    match = True
    if (p == 0):
    start = i
    if (p < len(pattern) - 1):
    p = p + 1
    else:
    found = True
    break
    i = i + 1
  else:
    if match: # マッチした状態から戻るとき
    if dict.has_key(s[start+len(pattern)-1]):
    i = i + dict[s[start+len(pattern)-1]]
    else:
    i = start + len(pattern)
    else:
    i = i + 1
    match = False
    p = 0
print "time: text search bm", (time.time()-
start_time)*1e3/tries, "[ms]", found;
```

Boyer – Moore: 主要部

```
# pattern # 検索文字列
# dict = {} # 検索文字列の末尾からの各文字の位置
# dict[pattern[i]] = len(pattern) - i + 1
# tg = open("target.txt", "r")
# s = tg.read(-1) # negative means "all”

  found = False
  match = False
  start = 0
  end = len(s)+1-len(pattern)
  i = 0
  p = 0
  while i < end:
    ch = s[i]
    if ( ch == pattern[p]):
      match = True
      if (p == 0):
        start = i
      if (p < len(pattern) - 1):
        p = p + 1
      else:
        found = True
        break
    i = i + 1
  else:
    if match: # マッチした状態から戻るとき
      if dict.has_key(s[start+len(pattern)-1]):
        i = i + dict[s[start+len(pattern)-1]]
      else:
        i = start + len(pattern)
    else:
      i = i + 1
    match = False
    p = 0
```

文字列検索

Apple Mac mini Late 2012
OS X Yosemite 10.10.3
2.6 GHz Intel Core i7
Memory 16GB 1600MHz DDR3



文字列の検索時間

単位：ms

nw1	nw2	nw3	bm1	bm2	bm3
16.0	15.4	15.0	5.81	5.82	5.78
15.0	15.2	14.8	5.82	5.81	5.77
15.1	15.2	14.8	5.82	5.81	5.74
15.7	15.2	14.8	5.78	5.79	5.76
15.4	15.2	14.8	5.80	5.81	5.75
14.9	15.1	14.7	5.79	5.78	5.75
15.4	15.7	14.9	5.81	5.78	5.74
16.4	15.2	14.8	5.77	5.78	5.75
16.3	15.2	14.8	5.79	5.80	5.72
15.0	15.3	14.8	5.80	5.81	5.76
14.9	15.2	14.7	5.80	5.83	5.75
14.9	15.1	14.8	5.80	5.81	5.75
14.9	15.1	14.8	5.80	5.78	5.76
14.9	15.2	14.8	5.78	5.82	5.72
14.8	15.1	14.8	5.79	5.80	5.75
14.9	15.2	14.8	5.79	5.80	5.76
14.9	15.2	14.9	5.79	5.79	5.76
14.9	15.2	14.9	5.82	5.80	5.75
14.9	15.2	14.8	5.78	5.81	5.76
14.8	15.3	14.8	5.78	5.78	5.72

他に利用できる手がかりは？

- または，Boyer-Mooreの方法の効率の良さは何が原因か？
- 検索対象のテキストや検索パターンに関する知識が利用できないか？
  - テキストやパターンが固定している
  - テキストやパターンの一定の傾向がある

作業：やってみよう

1. 二分探索の手順をフローチャートに表現して，プログラムを作成してください
   1. N個のデータをランダムに生成し整列させる
   2. 以下，Q回繰り返し：
      1. ランダムな値を一つ選ぶ
      2. その値を二分探索を使って探し、探索に要する時間を測る
   3. Q回の平均処理時間を求める
   4. 上記をProcessingのプログラムとして作成
   5. Nを変化させて、Nと平均処理時間との関係を確かめてください

宿題：ex04

1. 二分探索の手順をフローチャートに表現して，プログラムを作成してください
    1. N個のデータをランダムに生成し整列させる
    2. 以下，Q回繰り返し：
        1. ランダムな値を一つ選ぶ
        2. その値を二分探索を使って探し、探索に要する時間を測る
    3. Q回の平均処理時間を求める
    4. 上記をProcessingのプログラムとして作成
    5. Nを変化させて、Nと平均処理時間との関係を確かめてください

ヒント

- 探索範囲がステップを追って確実に減るようにする
  - 残りの要素数が2個のときのL，R，i = [(L+R)/2] の関係を考えること
  - 同値のデータが複数あるときにもうまくいくか検討しておく
    - 比較時の等号の取り扱いに注意
- Nを小さくして，配列の内容や動作の経過を出力してみる

ヒント つづき

- Array（配列）については:
  - https://processing.org/reference/Array.html
- 「整列したデータ配列の準備」は今回の課題の主目的ではないので:
  - Processingの持つ IntList, FloatList などのデータ型を利用しても良い
    - これらでは sort() などのメソッド（関数）が利用可能
    - ただし、要素への値の設定に[]に入った添字や代入は使えない
      - get(), set(), append() 等のメソッド（関数）を利用する
  - https://processing.org/reference/IntList.html
  - https://processing.org/reference/FloatList.html

やる気のある人に

- 以下については，提出を求めません。

- 文字列探索について
  - 紹介した各種法をプログラムとして実装し，
  - その処理時間を自分で比較してみてください
- 長文のテキストを使うには：
  - ファイルシステムへアクセスして
  - 既存のテキストファイルを読み込めると便利

コーディング

- コメントとして：
  - プログラムの冒頭にMS, NDの別，クラス，番号，氏名を記述
  - 試した結果，考察，感想など
    - 量的にはA4レポート用紙 ½ ページ以内
    - 個人での努力が読み取れるように！
  - （人的資源を含む）参考にした資料等があれば出典を書いておく
- プログラム本体は上記コメントの後
  - プログラムは実行可能なこと

提出についての注意

- Processing のプログラム名
  - デフォルトでは sketch_yymmdda などだが…
  - lastname_firstname_ex04 のように姓_名_宿題番号に変えること
  - lastname_firstname_c5_ex04 (同姓同名はクラスを付加）
- Processingのプログラムはフォルダに入った状態でzipファイルとしてまとめて提出
  - 必要なら他のファイルもそのフォルダに入れておくように
- Oh-o!Meijiから提出（次回の授業開始までに）

連絡先

樋口文人

wenren@meiji.ac.jp