アルゴリズムとデータ構造 2015 第３回　　年　　組　　番 氏名

1. 要素が既に小さい順に並んでいる配列 data に対して，次の手順で key を探索するメソッドを作成せよ。(1) まず配列の中央の値と key を比較し，もし等しければ探索終了。(2) そうでなければ， key が配列の前半と後半のどちらに含まれるか分かるので，それを新しい探索範囲とする。(3) 新しい探索範囲の中央の値と key を比較する。(4) 以上の手順を繰り返し，範囲を次々に半分に絞り込んで探索していく。

```
public static int binarySearch(int key, int[] data) {

    int left = 0;  // 探索範囲の左端
    int right = data.length - 1;  // 探索範囲の右端

    while (left <= right) {

        int mid =

        if (key == data[mid]) {


        } else if (key < data[mid]) {


        } else {


        }
    }
    return -1;
}
```

2. この **2 分探索**の最大計算量（最悪計算量）を考えてみる。配列の要素数を n とする。
   (a) 最初の探索範囲には n 個の要素がある。(1)～(3)のループを 1 回行うごとに探索範囲はほぼ α 分の 1 に狭まるので，k 回目では n の α^k 分の 1 個になる。α の値を示せ。

   (b) 最悪の場合のループ回数を k 回とすると，k−1 回目に探索範囲が最後の 1 個になって k 回目に最後の比較をするので，およそ $n \cdot \left(\frac{1}{\alpha}\right)^{k-1}=1$ という式が成り立つ。この式から最大計算量 k を求め，n を横軸としたグラフを描いて線形探索と比較せよ。

3. この **2 分探索**の平均計算量を考えてみる。ただし，簡単のため $n = 2^k - 1$ とする。
   (a) key が配列の中央の値と同じなら 1 回目で見つかるので，1 回目で見つかる値は 1 個である。以下，2 回目で見つかる値は 2 個，3 回目は 4 個と続く。最後の k 回目で見つかる値は何個か示せ。

   (b) 平均計算量が以下の式で求められる理由を考えよ。

$$C = \frac{1}{n}\{1 \cdot 1 + 2 \cdot 2 + 3 \cdot 4 + \cdots + (k-1) \cdot 2^{k-2} + k \cdot 2^{k-1}\}$$

   【発展】この値は，$2C - C$を計算することによって求めることができる。難しいので説明は省略…。

$$2C - C = \frac{1}{n}\left\{k \cdot 2^k - \left(1 + 2 + 2^2 + \cdots + 2^{k-1}\right)\right\} = \frac{1}{n}\left(k \cdot 2^k - \frac{1-2^k}{1-2}\right) = \cdots$$

4. アルファベット順に並んだ文字列（String）に対する 2 分探索のプログラムを作成せよ。クラス型では関係演算子（不等号）による大小比較ができないので，代わりに compareTo メソッドを利用する。その際，compareTo と equals を両方使うのは冗長であるから，equals は使用しないこと。

5. 下記のプログラムは，2 分探索を**再帰**を用いて記述したものである。1 とは引数が異なり，配列の中の data[first]～data[last]の範囲から値 key を探索する。空欄を埋めてプログラムを完成させよ。

```
public static int binarySearchR(int key, int[] data, int first, int last) {

    if (first > last)
        return -1;

    int mid = (first + last) / 2;
    if (key == data[mid])
        return mid;
    if (key < data[mid])
        return binarySearchR(key, data, first,         );
    /* else */
        return binarySearchR(key, data,          , last);
}
```

6. 【発展】下記のプログラムは，2 分探索を改良した**内挿探索（補間探索）**と呼ばれるものである。これは，配列内で添字が left から right まで増えるにつれて，値が data[left]から data[right]までほぼ一定の増加率で増えると仮定すると，探索値 key は次の比例式の mid の位置付近にあるだろうという推測に基づく。

$$right - left : data[right] - data[left] = mid - left : key - data[left]$$

内挿探索は非常に高速だが，データによっては非常に遅くなってしまう。どのような場合か考えてみよ。

```
public static int interpolationSearch(int key, int[] data) {

    int left = 0;  // 探索範囲の左端
    int right = data.length - 1;  // 探索範囲の右端

    while (left <= right) {
        int ldata = data[left];
        int rdata = data[right];

        // mid の計算式で分母が 0 にならないようにする（重複値対策）
        if (ldata == rdata) {
            if (key == ldata) return left;
            /* else */ break;
        }
        int mid = left + (right - left) * (key - ldata) / (rdata - ldata);

        if (key == data[mid]) {
            return mid;
        } else if (key < data[mid]) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return -1;
}
```