アルゴリズムとデータ構造 2019 第４回　　年　　組　　番 氏名

1. 以下に示すのは，配列 a の中身を小さい順に並べ替える**バブルソート**である。
(1) まず，このアルゴリズムが動作する仕組みを理解せよ。コメントアウトされている println を有効にすると n=3 と n=4 のときに、それぞれどう表示されるか考えよ。

```
public static void bubbleSort(double[] a) {
    int n = a.length;

    for (int i = 0; i < n - 1; i++) {
        for (int j = n - 1; j > i; j--) {
            if (a[j-1] > a[j]) {
                double t;
                t = a[j-1];  a[j-1] = a[j];  a[j] = t;
            }
            // System.out.println("a[" + (j-1) + "]と a[" + j + "]を比較した。");
        }
    }
}
```

(2) 2 つの要素の比較は， i が 0 のときは n-1 回行われ，i が 1 のときは n-2 回行われ，i が n-2 のときは 1 回行われる。計算量（i=0 から i=n-2 までの総比較回数）を n で表せ。

2. **選択ソート**は，i=0 から始めて i を 1 つずつ増やしながら，a[i]～a[n-1]の範囲から最小値 a[k]を選択し，k≠i ならば a[k]と a[i]を交換していくことよって，配列全体の整列を行う。
(1) 以下のプログラムの空欄を埋めて，選択ソートのメソッドを完成させよ。

```
public static void selectionSort(double[] a) {
    int n = a.length;

    for (int i = 0; i < n - 1; i++) {
        // a[i]～a[n-1]の範囲から最小の値 a[k]を見つける
        int k = i;
        for (int j = i + 1; j < n; j++) {

            if (                    ) {

                k = j;
            }
        }
        if (k != i) {
            double t;
            t = a[k];  a[k] = a[i];  a[i] = t;
        }
    }
}
```

(2) 2 つの要素の比較は i が 0 のときは n-1 回行われる。計算量（総比較回数）を n で表せ。なお，選択ソートはバブルソートよりも値の交換回数が少ないので高速である。

3. **挿入ソート**は，i=1 から始めて i を 1 つずつ増やしながら，a[i]を整列済みの a[0]～a[i-1]の適切な位置に挿入することを繰り返していくことよって，配列全体の整列を行う。
(1) 以下のプログラムの空欄を埋めて，選択ソートのメソッドを完成させよ。

```
public static void insertionSort(double[] a) {
    int n = a.length;

    for (int i = 1; i < n; i++) {
        double t = a[i];

        // a[i-1] → a[i], a[i-2] → a[i-1], a[i-3] → a[i-2],... と
        // a[0]～a[i-1]の中身を後ろから順にずらしながら t の挿入位置を探る
        int j = i;

        while (j >=    ) {
            if (a[j-1]    t) break;  // 挿入位置を発見（while の条件式に組み込んでみよ）


            j--;
        }


    }
}
```

(2) 挿入ソートは，ほとんど整列済みの配列に対しては非常に高速だが，逆順に並んでいる配列に対しては最悪の性能になる。最悪の場合，i が 0 のとき a[0]の 1 つをずらし，i が 1 のとき a[0]と a[1]の 2 つをずらし，i が n-2 のときは a[0]～a[n-2]の n-1 個をずらす。ずらす回数の合計は何回になるか。

4. 以下は，ともに小さい順に整列された配列 a と配列 b を合併（マージ）し，全体が小さい順に整列された配列 c を構成して返す関数である。空欄を埋めて関数を完成させよ。

```
public static double[] merge(double[] a, double[] b) {

    double[] c = new double[a.length + b.length];

    int i = 0, j = 0, k = 0;
    while (k < a.length + b.length) {

        if (i >= a.length) c[k++] =

        else if (j >= b.length) c[k++] =

        else if (a[i] <= b[j]) c[k++] =

        else c[k++] =
    }
    return c;
}
```

5. 今回の 1.～4.のプログラムを，文字列（String）の配列を辞書順（アルファベット順・あいうえお順）に整列するように書き換えよ。

6. 【発展】下記は挿入ソートの改良版である**シェルソート**である。このアルゴリズムが挿入ソートよりも高速である理由を考察せよ。

シェルソートでは，配列の中の h 個（ギャップ）おきの要素列を考え，それぞれに対して挿入ソートを行う。例えば，h=3 のときには { a[0], a[3], a[6], … }，{ a[1], a[4], a[7], … }，{ a[2], a[5], a[8], … } の 3 本の要素列に対して別々に挿入ソートを行う。この手順を，ギャップ h を適当な値から始めて適当な方法で 1 まで減らしながら繰り返す。最後は h=1 で通常の挿入ソートを行う。

以下の例では，ギャップ h は [n/2], [n/4], [n/8],..., 1 と変化する。この方式はプログラムが単純になるのが利点だが，より高速なギャップ列も提案されている。他のギャップの計算方法について調べてみよ。

```
public static void shellSort(double a[]) {
    int n = a.length;

    for (int h = n/2; h > 0; h /= 2) {
        for (int i = h; i < n; i++) {
            double t = a[i];
            int j = i;
            while (j >= h && a[j-h] > t) {
                a[j] = a[j-h];
                j -= h;
            }
            a[j] = t;
        }
    }
}
```