アルゴリズムとデータ構造　第４回演習　　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。
   (a) 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 printf 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。

```
void bubble_sort(double a[], int n) {
    int i, j;
    double t;

    for (i = 0; i < n - 1; i++) {
        for (j = n - 1; j > i; j--) {
            if (a[j-1] > a[j]) {
                t = a[j-1];  a[j-1] = a[j];  a[j] = t;
            }
            /* printf("a[%d]と a[%d]を比較した¥n", j-1, j); */
        }
    }
}
```

   (b) 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。
   (a) 以下のプログラムの空欄を埋めて，選択ソートの関数を完成させよ。

```
void selection_sort(double a[], int n) {
    int i, j, k;
    double t;

    for (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 */
        k = i;
        for (j = i + 1; j < n; j++) {
            if (a[j] < a[k]) {


            }
        }
        if (k != i) {


        }
    }
}
```

   (b) 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=0 から始めて i を 1 つずつ増やしながら，a[i+1]を整列済みの a[0]～a[i]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。
(a) 以下のプログラムの空欄を埋めて，選択ソートの関数を完成させよ。

```
void insertion_sort(double a[], int n) {
    int i, j;
    double t;

    for (i = 0; i < n - 1; i++) {

        t =

        /* a[i] → a[i+1], a[i-1] → a[i], a[i-2] → a[i-1],... と
          a[0]～a[i]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 */
        j = i;
        while (j >= 0 && a[j] > t) {
            a[j+1] = a[j];
            j--;
        }




    }
}
```

(b) 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. 【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えよ。
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。

```
void shell_sort(double a[], int n) {
    int i, j, h;
    double t;

    for (h = (n + 1) / 3; h > 0; h = (h + 1) / 3) {  /* h が 1,0 で終わるように調整 */
        for (i = 0; i < n - h; i++) {
            t = a[i+h];
            j = i;
            while (j >= 0 && a[j] > t) {
                a[j+h] = a[j];
                j -= h;
            }
            a[j+h] = t;
        }
    }
}
```