アルゴリズムとデータ構造 2019 第５回　　年　　組　　番　氏名

1. 以下に示したのは，**クイックソート**のプログラム例である。クイックソートは全体を部分に分割しながら同じ処理を繰り返していくという，代表的な**分割統治**アルゴリズムである。
　クイックソートでは，まず配列の中の適当な要素を pivot（**軸**）に選び，pivot 以下の要素を左側，pivot 以上の要素を右側に寄せることで，配列を左右 2 つの部分配列に分割する。さらにそれぞれの部分配列も同様の処理で分割し，要素が 1 つになるまで分割を繰り返すと，元の配列全体のソートが完了する。
　このプログラムでは，この分割を行うために，まず配列を左から走査して pivot 以上の要素を見つけ，次に右から走査して pivot 以下の要素を見つけ，左側で見つけた要素と右側で見つけた要素同士を交換することを繰り返す。空欄を埋めてクイックソートの関数を完成させよ。

```
public static void quicksort(double[] a, int left, int right) {

    if (left >= right) return;

    double pivot = a[(left + right) / 2];  // この例では中央の要素を pivot にしている
                                           // left + (right - left) / 2 のほうが安全
    int l = left, r = right;
    while (true) {
        while (a[l] < pivot) {  // 左から走査して pivot「以上」の要素で止まる
                                //  （<=にすると pivot が最大値のとき止まらない）
        }
        while (a[r] > pivot) {  // 右から走査して pivot「以下」の要素で止まる
                                //  （>=にすると pivot が最小値のとき止まらない）
        }
        if (l >= r) break;

        double t;


        l++; r--;
    }

    quicksort(a,          , l - 1);  // l-1 までは pivot より大きい

    quicksort(a, r + 1,          );  // r+1 までは pivot より小さい（l==r の場合 pivot 上）
}
```

2. クイックソートの計算量を考える。簡単のため配列の要素数は$n = 2^k$とする。
(a) クイックソートで理想の pivot が選ばれると配列はちょうど中央で 2 分割される。理想的な pivot が選ばれ続けた場合，k段階の分割で個々の部分配列の要素数はほぼ $n/2^k$ となる。この要素数が 1 になるまでの分割の段階数を求めよ。

(b) 次に各段階での計算量を考える。クイックソートでは，各段階の分割のために全要素を pivot と比較するので，計算量は要素数nに比例する。これに段階数である(a)の解を掛ければ，クイックソートの全計算量が概算できる。計算量を O 記法で表せ。

(c) クイックソートで最悪の pivot が選ばれ続けた場合，分割の段階数は何段階になるか。

3. **マージソート**も分割統治による高速なソートであり，前回演習の 4.のアルゴリズムに従って配列の中の隣り合う整列済みの部分をマージしていくことで配列全体を整列させる。マージソートには，配列の 2 分割を要素数 1 個になるまで繰り返してから合併して戻していくトップダウンの方式と，隣り合う要素のマージからはじめて，隣り合う部分配列を結合して成長させていくボトムアップの方式がある。
　以下は，トップダウン方式の例である。配列を前半 a[left]～a[mid]と後半 a[mid+1]～a[right]に分割し，それぞれを再帰的にソートしてからマージする。この例の動作を理解し，空欄を埋めて関数を完成させよ。

```
public static void mergeSort(double[] a, int left, int right) {

    if (left >= right) return;

    int mid = (left + right) / 2;   // left + (right - left) / 2 のほうが安全

    mergeSort(a,          , mid);    // 配列の前半部分をソート済みにする

    mergeSort(a,          , right);  // 配列の後半部分をソート済みにする

    // 配列の前半部分を退避する
    double[] b = new double[mid - left + 1];
    for (int h = 0; h < b.length; h++)
        b[h] = a[left + h];

    // 退避した前半部分と後半部分をマージする
    int i = 0, j = mid + 1, k = left;
    while (k <= right) {

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

        else if (j > right) a[k++] =

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

        else a[k++] =

    }
    b = null;  // 退避用の配列を削除
}
```

4. マージソートの計算量を考える。簡単のため配列の要素数は$n = 2^k$とする。
(a) トップダウン方式のマージソートでは，配列はいつも中央で 2 分割される。よって，k段階の分割で個々の部分配列の要素数はほぼ $n/2^k$ となる。この要素数が 1 になるまでの分割の段階数を求めよ。

(b) 次に各段階での計算量を考える。マージソートでは，退避とマージのために全要素をコピーするので，各段階での計算量は n に比例する。これに段階数である(a)の解を掛ければ，マージソートの全計算量が概算できる。計算量を O 記法で表せ。

5. 今回の 1.と 3.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）に並べ替えるように書き換えよ。

6. 【発展】下記のプログラムは，ボトムアップ方式のマージソートの例である。まず先頭から順に隣り合う要素を組にして併合して要素 2 個の整列済み部分配列（連；run）にすることからはじめ，それをさらに隣どうし併合して 4 個，8 個，16 個と成長させていく。空欄を埋めてプログラムを完成させよ。

```
/**
 * 配列をマージソートで整列する
 * @param a 配列
 */
public static void mergeSort(double[] a) {
    int n = a.length;

    // 連（run；ソート済みの部分配列）の長さを 1, 2, 4, 8 と増やしていく
    for (int runLen = 1; runLen < n;              ) {

        // 先頭から順に 2 つずつ連の組を作ってマージ（併合）していく
        for (int i = 0; i < n - runLen; i += runLen * 2) {

            // 後半の部分配列の末尾位置を求める
            int last =

            // それが全体配列からはみ出ないようにする
            if (last > n – 1) last = n – 1;

            // a[i]～a[i+runLen-1]と a[i+runLen]～a[last]をマージ（併合）する
            merge(a, i, i + runLen, last);
        }
    }
}

/**
 * 配列の整列済みの前半と整列済みの後半をマージ（併合）する
 * @param a 配列
 * @param first1 前半部分の開始位置
 * @param first2 後半部分の開始位置
 * @param last 後半部分の終了位置
 */
private static void merge(double[] a, int first1, int first2, int last) {

    // 配列の前半部分を退避する
    double[] b = new double[first2 – first1];
    for (int h = 0; h < b.length; h++)
        b[h] = a[first1 + h];

    // 退避した前半部分と後半部分をマージ（併合）する
    int i = 0, j = first2, k = first1;
    while (k <= last) {
        if (i >= b.length) a[k++] = a[j++];
        else if (j > last) a[k++] = b[i++];
        else if (b[i] <= a[j]) a[k++] = b[i++];
        else a[k++] = a[j++];
    }
    b = null;  // 退避用の配列を削除
}
```