Quick Sort

計算機アルゴリズム特論：2017年度

只木進一

基本的考え方

- リスト（あるいは配列）Sの中の、ある要素x(pivot)を選択
  - xより小さい要素からなる部分リストS_1
  - xより大きい要素からなる部分リストS_2
  - xはS_1またはS_2に含まれる
  - 長さが1になるまで繰り返す
- pivot xの選び方として、中央の要素を選択すると効率が良い

三つの可能性
pivotを中央の要素とした場合

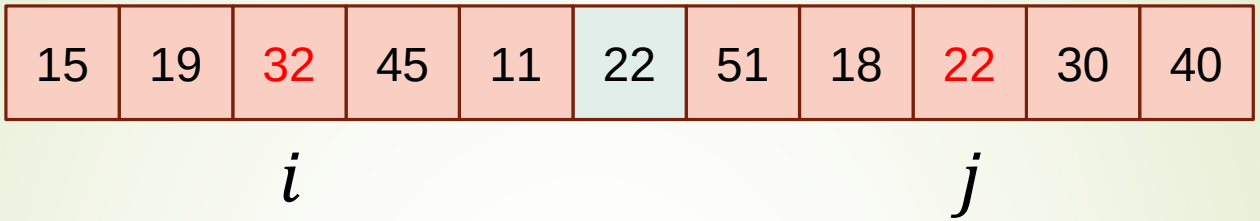
- pivotの両側の入れ替えるべき要素数が同じ場合：例1
- pivotの位置より、左側にある移動すべき要素が右側のそれより多い場合：例２
- pivotの位置より、左側にある移動すべき要素が右側のそれより少ない場合：例3

- 左端から$x_i \geq x$となる要素を探す
- 右端から$x_j \leq x$となる要素を探す
- x_iとx_jを入れ替える
- $i < j$である限り繰り返す

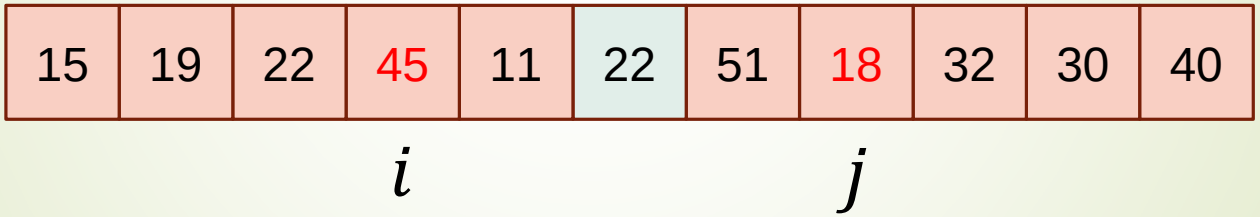
- $S_1 = \{x_k | 0 \leq k < i\}$
- $S_2 = \{x_k | i \leq k < |S|\}$

例1

基準値



左端から基準値以上となる最初の要素(32)の位置i
右端から基準値以下となる最初の要素(22)の位置j
二つの要素を入れ替える



位置iから基準値以上となる最初の要素(45)の位置i
位置jから基準値以下となる最初の要素(18)の位置j
二つの要素を入れ替える

15	19	22	18	11	22	51	45	32	30	40

i j

位置iから基準値以上となる最初の要素(22)の位置i
位置jから基準値以下となる最初の要素(22)の位置j
$i \geq j$となったので、iで分割

15	19	22	18	11

基準値より小さい

22	51	45	32	30	40

基準値以上

例2

基準値

i j

左端から基準値以上となる最初の要素(25)の位置i
右端から基準値以下となる最初の要素(22)の位置j
二つの要素を入れ替える

i j

位置iから基準値以上となる最初の要素(32)の位置i
位置jから基準値以下となる最初の要素(23)の位置j
二つの要素を入れ替える

22	19	23	45	11	32	51	37	25	30	40

i　　j

位置iから基準値以上となる最初の要素(45)の位置i
位置jから基準値以下となる最初の要素(11)の位置j
二つの要素を入れ替える

22	19	23	11	45	32	51	37	25	30	40

j　　i

位置iから基準値以上となる最初の要素(45)の位置i
位置jから基準値以下となる最初の要素(11)の位置j
$i \geq j$となったので、iで分割

22	19	23	11

基準値以下

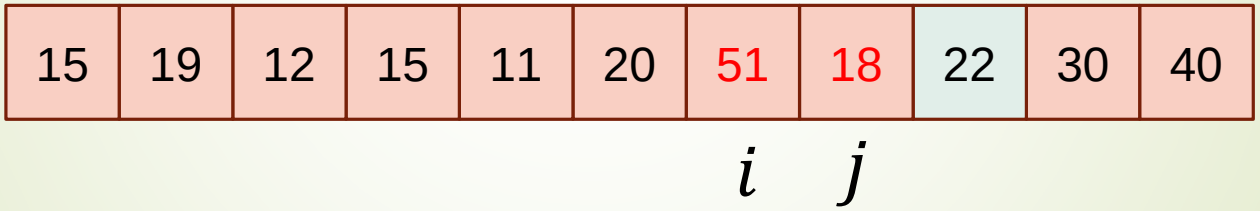
45	32	51	37	25	30	40

基準値より大きい

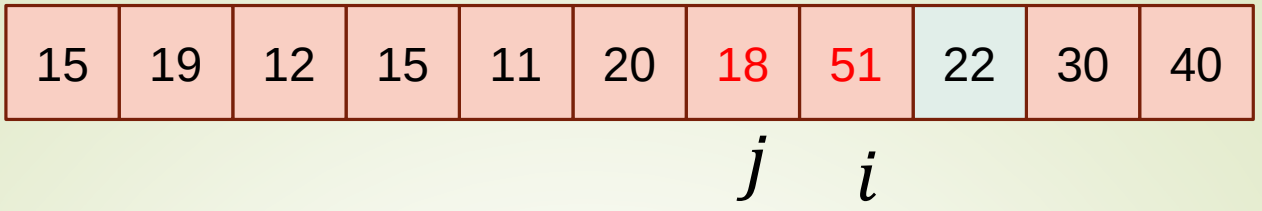
例3

基準値

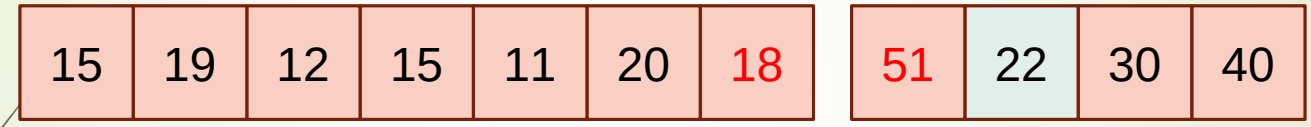
左端から基準値以上となる最初の要素(22)の位置i
右端から基準値以下となる最初の要素(20)の位置j
二つの要素を入れ替える



位置iから基準値以上となる最初の要素(51)の位置i
位置jから基準値以下となる最初の要素(18)の位置j
二つの要素を入れ替える



位置iから基準値以上となる最初の要素(51)の位置i
位置jから基準値以下となる最初の要素(18)の位置j
$i \geq j$となったので、iで分割



基準値より小さい　　基準値以上

```
public T[] doSort() {
    sortSub(0, data.length);
    return data;
}
 protected void sortSub(int begin, int end) {
    if (end <= begin + 1) {
        return;
    }
    int boundary = partition(begin, end);
    sortSub(begin, boundary);
    sortSub(boundary, end);
}
```

```
protected int partition(int begin, int end) {
   int fromLeft = begin - 1;
   int fromRight = end;
   int m = (begin + end) / 2;
   T v = data[m];
   for (;;) {
      while (less(data[++fromLeft], v));
      while (less(v, data[--fromRight])) {
         if (fromRight == begin) {  break; }
      }
      if (fromLeft >= fromRight) {break; }
      exch(fromLeft, fromRight);
   }
return fromLeft;
}
```

注：increment/decrement演算子

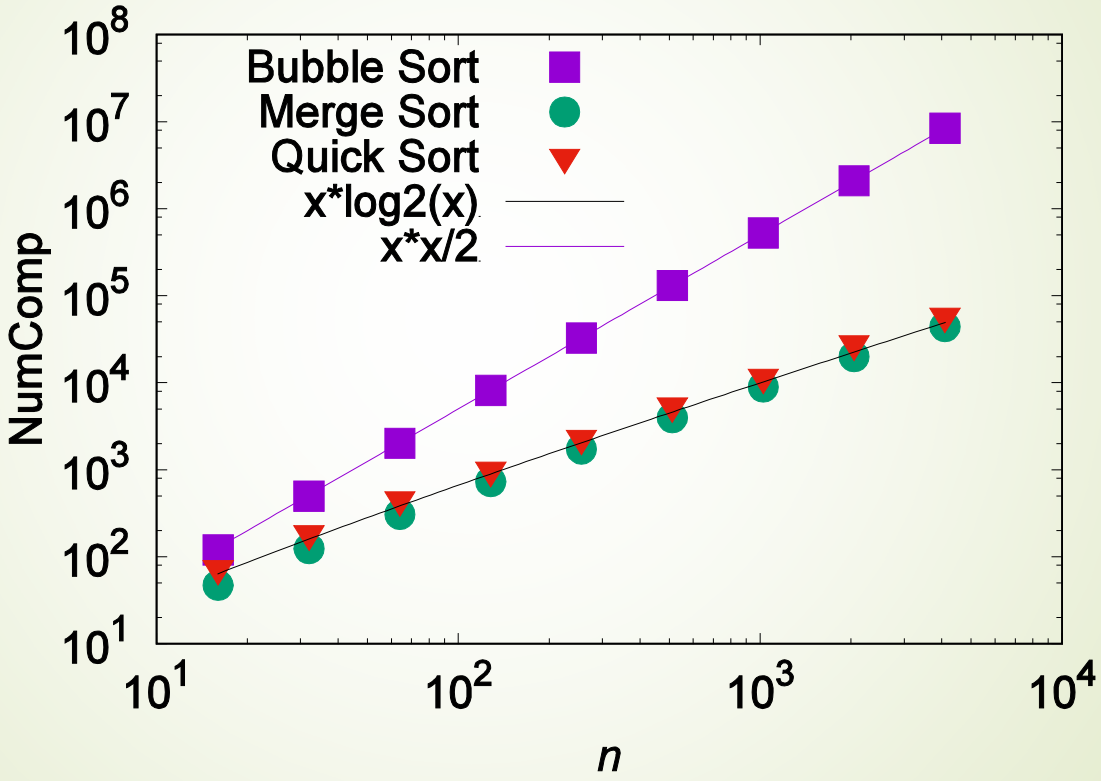
- $b = a + +;$
  - aをbへ代入後、インクリメント
- $b =+ +a;$
  - aをインクリメント後、bへ代入
- while (less(data[++fromLeft], v));
  - vより小さくない要素を見つけるまで、fromLeftをインクリメント

計算量

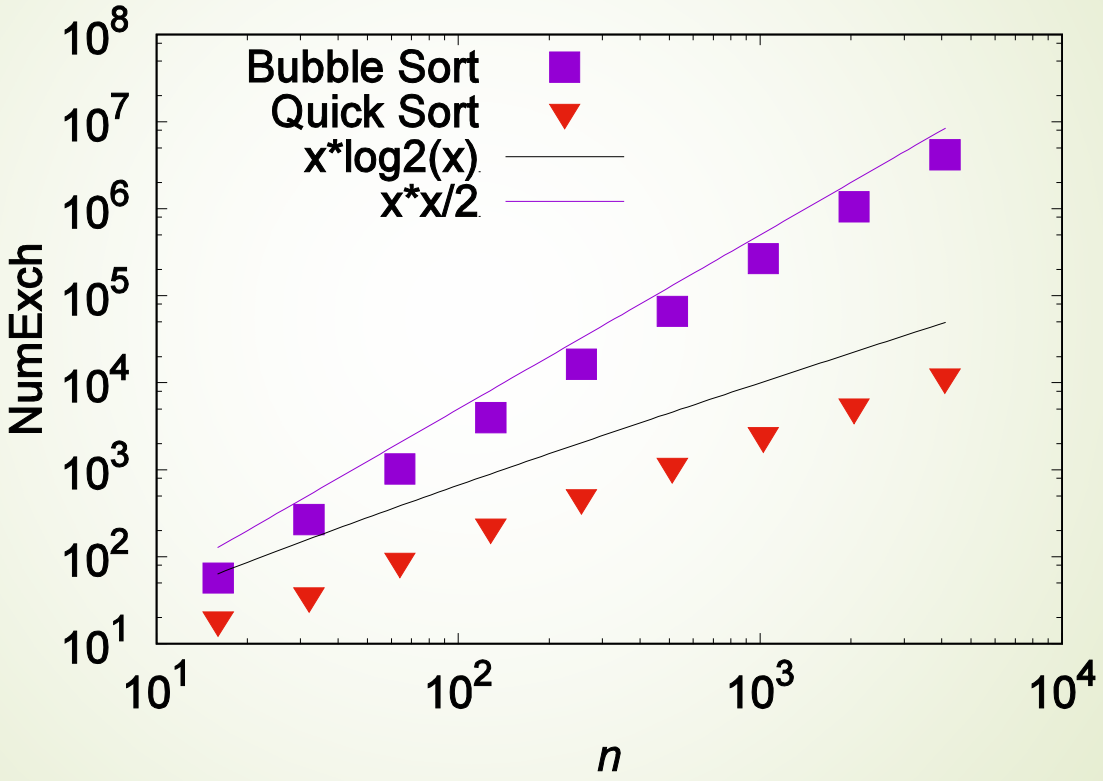
- 一回のpartition操作で、要素の比較はn回
- partitioningは、概ね$\log_2 n$回
- 全体で$n\log_2 n$回の比較

- 要素をswapするために、作業領域を必要としない

比較回数


Bubble Sort
Merge Sort
Quick Sort
x*log2(x)
x*x/2
NumComp
n
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^1
10^2
10^3
10^4

入替回数


Bubble Sort
Quick Sort
x*log2(x)
x*x/2
NumExch
n
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^1
10^2
10^3
10^4

pivotの選択を最後の要素にした場合

基準値

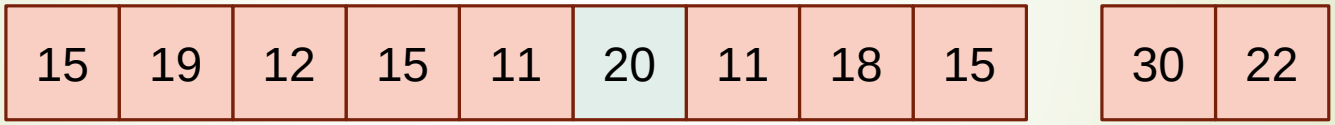
i j

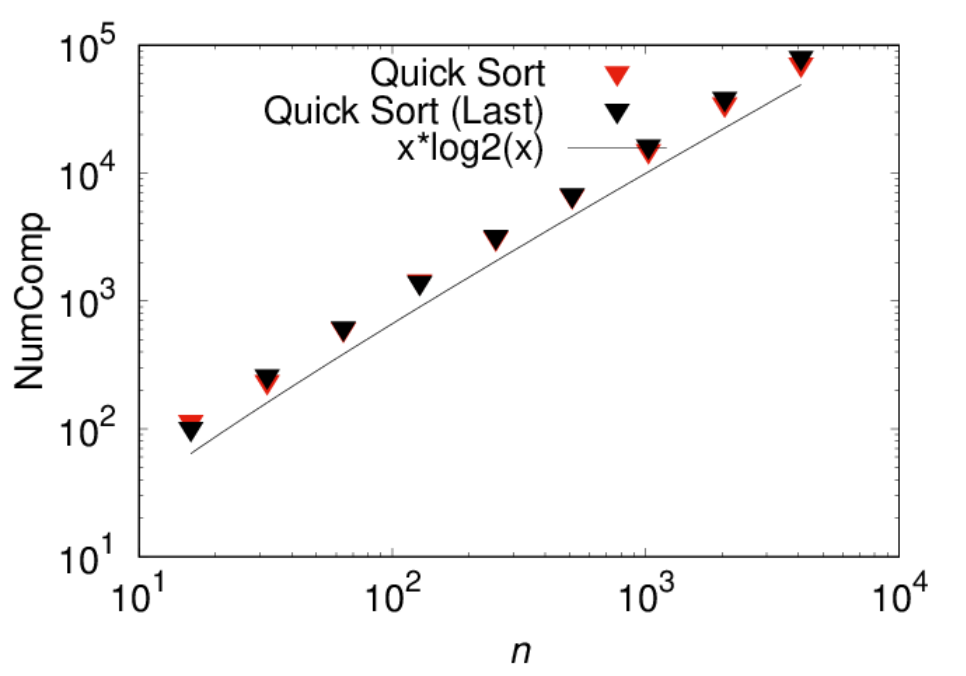
左端から基準値以上となる最初の要素(22)の位置i
右端から基準値以下となる最初の要素(20)の位置j
二つの要素を入れ替える

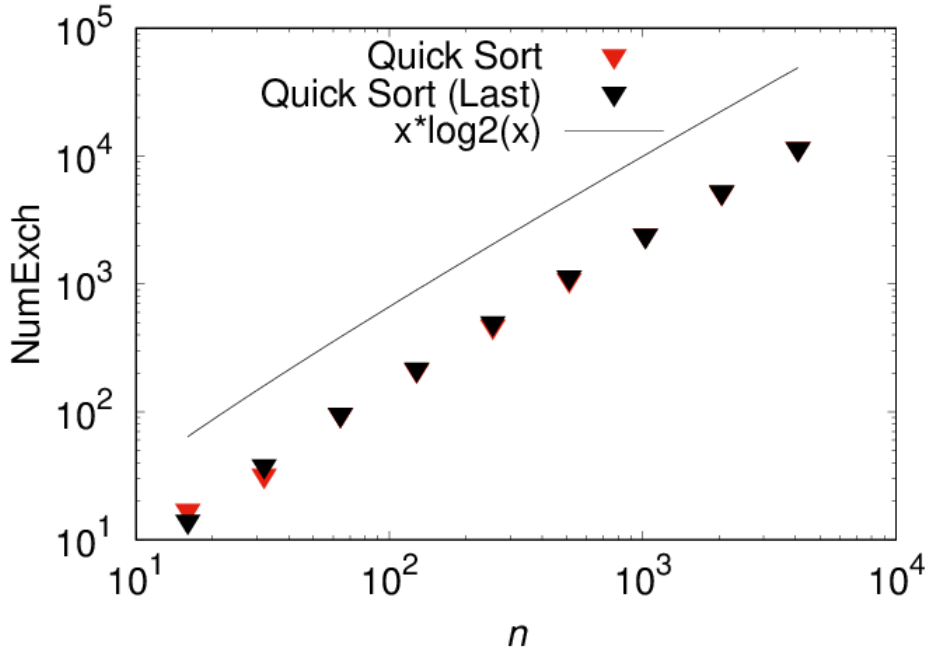
i j

位置iから基準値以上となる最初の要素(30)の位置i
位置jから基準値以下となる最初の要素(11)の位置j
二つの要素を入れ替える

位置iから基準値以上となる最初の要素(30)の位置i
位置jから基準値以下となる最初の要素(15)の位置j
$i \geq j$となったので、iで分割




Quick Sort
Quick Sort (Last)
x*log2(x)
NumComp
n
10^1
10^2
10^3
10^4
10^5


Quick Sort
Quick Sort (Last)
x*log2(x)
NumExch
n
10^1
10^2
10^3
10^4
10^5

QuickSort.java

```java
package sort;

import java.io.IOException;
import java.util.function.BinaryOperator;
import static sort.AbstractSort.testRun;

/**
 *
 * @author tadaki
 * @param <T>
 */
public class QuickSort<T extends Comparable<T>> extends AbstractSort<T> {

    //pivotの選び方
    public static enum PivotType {
        Middle, First, Last, Random;
    }
    public static final boolean debug = false;
    //pivot を選ぶ関数のデフォルト
    private BinaryOperator<Integer> getPivot
            = (begin, end) -> (begin + end) / 2;

    public QuickSort(T[] data) {
        super(data);
    }

    public QuickSort() {
    }

    /**
     * pivotを変更する
     *
     * @param pivotType
     */
    public void setPivotType(PivotType pivotType) {
        switch (pivotType) {
            case First://最初の要素
                getPivot = (begin, end) -> begin;
                break;
            case Last://最後の要素
                getPivot = (begin, end) -> end - 1;
                break;
            case Random://ランダム
                getPivot = (begin, end)
                        -> (int) ((end - begin) * Math.random()) + begin;
                break;
```

```
            default://中央の要素
                getPivot = (begin, end) -> (begin + end) / 2;
                break;
        }
    }

    @Override
    public T[] doSort() {
        sortSub(0, data.length);
        return data;
    }

    protected void sortSub(int begin, int end) {
        if (end <= begin + 1) {
            return;
        }
        int boundary = partition(begin, end);
        sortSub(begin, boundary);
        sortSub(boundary, end);
    }

    /**
     * pivotの値での分割
     *
     * @param begin 配列の開始位置
     * @param end 終了位置：対象の外側であることに注意
     * @return 分割の位置
     */
    protected int partition(int begin, int end) {
        int fromLeft = begin - 1;
        int fromRight = end;
        int m = getPivot.apply(begin, end);
        T v = data[m];
        if (debug) {
            System.out.print("partition (" + begin + "," + end + ") ");
            System.out.println("with pivot " + v + " at " + m);
        }
        for (;;) {
            while (less(data[++fromLeft], v));
            while (less(v, data[--fromRight])) {
                if (fromRight == begin) {
                    break;
                }
            }
            if (fromLeft >= fromRight) {
                break;
```

```
            }
            if (debug) {
                System.out.println(data2Str()
                        + ":swap(" + fromLeft + "," + fromRight + ")");
            }
            exch(fromLeft, fromRight);
        }
        if (debug) {
            System.out.println("end paritioning with " + data2Str());
        }
        return fromLeft;
    }

    /**
     * 配列の文字列化
     *
     * @return
     */
    private String data2Str() {
        StringBuilder sb = new StringBuilder();
        sb.append("[");
        for (int i = 0; i < data.length - 1; i++) {
            sb.append(data[i]).append(",");
        }
        sb.append(data[data.length - 1]).append("]");
        return sb.toString();
    }

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
    static public void main(String args[]) throws IOException {
        if (QuickSort.debug) {
            Integer[] data = {15, 19, 12, 15, 11, 22, 30, 18, 15, 11, 20};
            QuickSort<Integer> qs = new QuickSort<>(data);
            qs.doSort();
        } else {
            int numData = 1000;
            Data[] data = Data.createData(numData);
            testRun(new QuickSort<>(data));
        }
    }

}
```