# Merge Sort with Queue



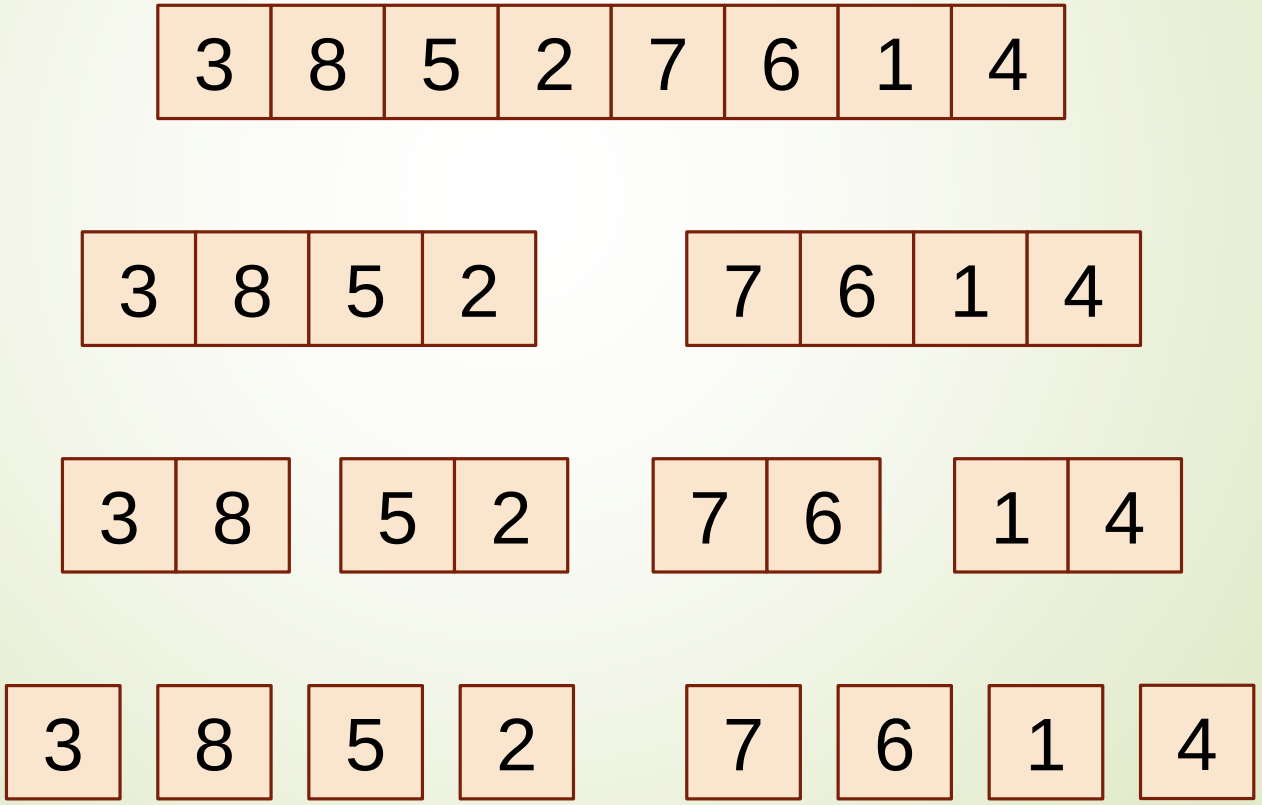
計算機アルゴリズム特論：2015年度

只木進一

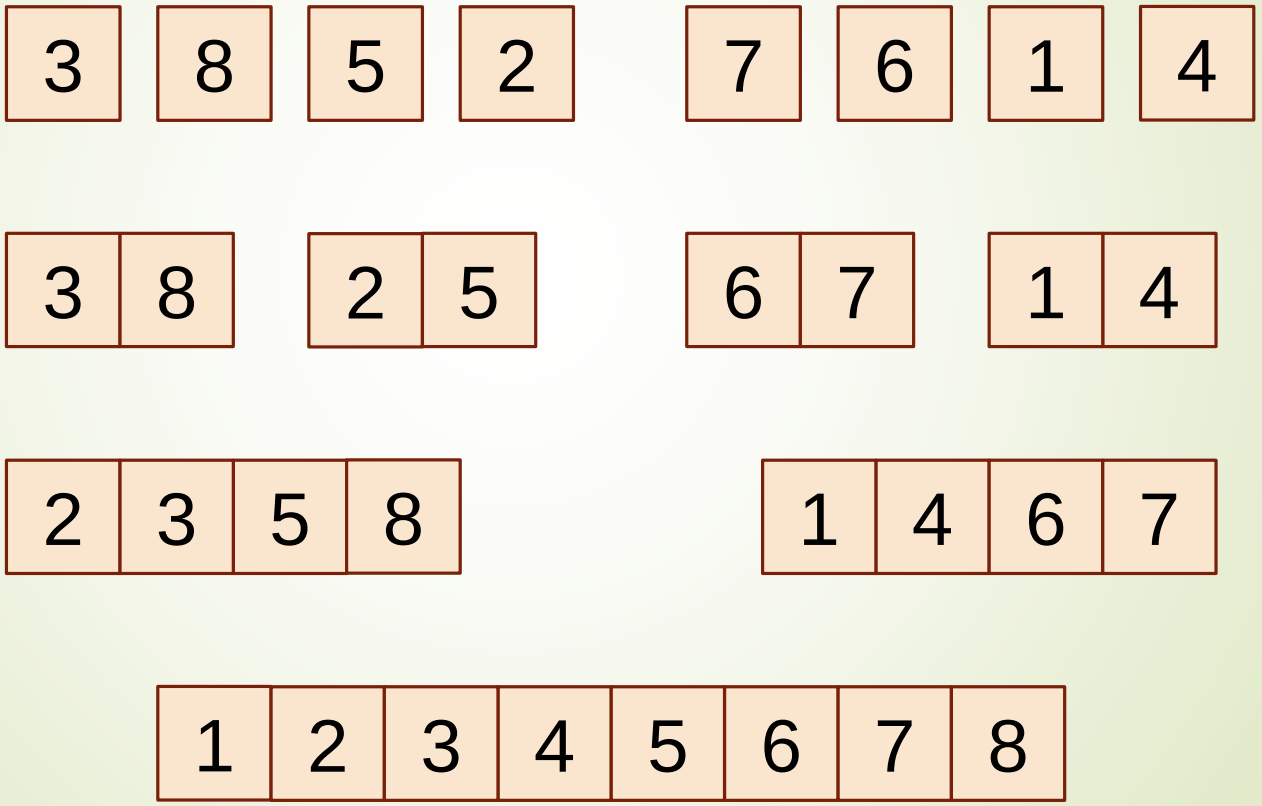
# 待ち行列を使ったMerge Sortの基本的考え方

- 再帰的方法
  - 考え方は単純
  - エラーが発生した場合に、突き止めるのが困難
- 再帰過程を分析して、再帰を使わない方法を探る

# Merge Sort
# リストの分離



# Merge Sort
# リストの結合



- 入力リスト中の順序は、結果に影響しない
  - 長さ１のリストに分割して、二つ毎に整列統合すればよい
  - 二つ毎の整列統合が終わったら、さらにそれらを整列統合する
- 整列統合が終わったリストを、待ち行列で待たせて、後で処理

# Merge Sort
# リストの結合

1. <[3],[8],[5],[2],[7],[6],[1],[4]>
2. <[5],[2],[7],[6],[1],[4],[3,8]>
3. <[7],[6],[1],[4],[3,8],[2,5]>
4. <[1],[4],[3,8],[2,5],[6,7]>
5. <[3,8],[2,5],[6,7],[1,4]>
6. <[6,7],[1,4],[2,3,5,8]>
7. <[2,3,5,8],[1,4,6,7]>
8. <[1,2,3,4,6,7,8]>

# Merge Sortアルゴリズム 再帰での記述

```
List<T> mergeSort(List<T> list){
  if(list.size()==1){return list;}
  Queue<List<T>> q;
  for(T t:list){
    List<T> tl; tl.add(t);  q.add(tl);}
  while(q.size()>1){
    q=mergeListWIthQueue(q);}
  return q.poke();
}
```

```
Queue<List<T>> mergeListWithQueue (Queue<List<T>> q){
  List<T> l1=q.poke();
  List<T> l2=q.poke();
  List<T> l3=mergeList(l1,l2);
  q.add(l3);
  return q;
}
```

# 二つの整列済みリストの結合

```
private List<T> mergeList(List<T> a, List<T> b) {
        List<T> c = Collections.synchronizedList(
                new ArrayList<T>());
        while ((!a.isEmpty()) || (!b.isEmpty())) {
            T t;
            if (!a.isEmpty()) {
                if (!b.isEmpty()) {
                    if (a.get(0).compareTo(b.get(0)) < 0) {
                        t = a.remove(0);
                    } else {t = b.remove(0); }
                } else {t = a.remove(0); }
            } else {t = b.remove(0); }
            c.add(t);
        }
}
```

# 単体テスト

- 一つ毎の関数やメソッドの挙動を調べる方法
- JavaではJUnitで行うことができる
  - テストするクラスを指定して、テストを作成
  - テストコードのテンプレートが生成される