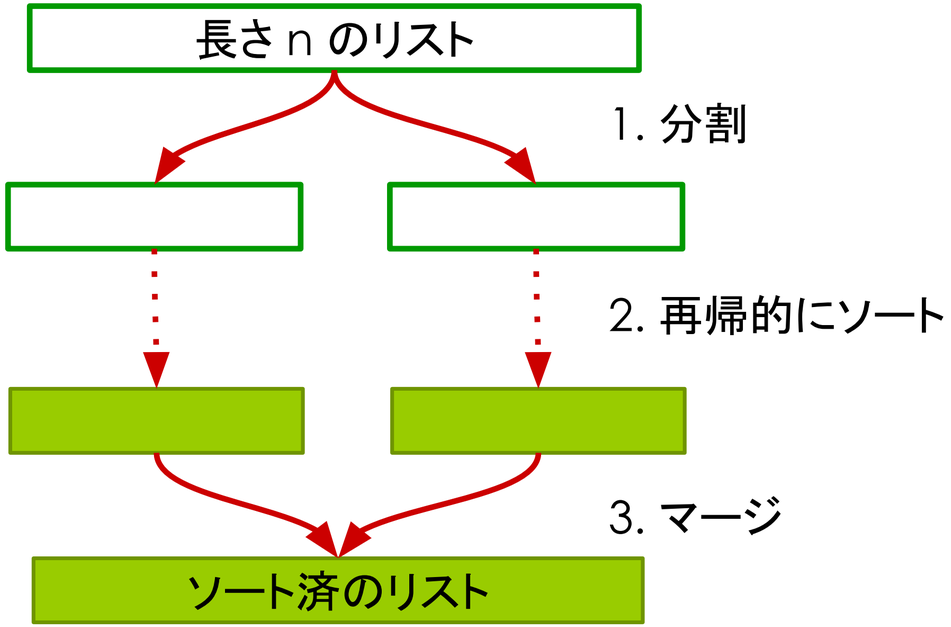
② マージソート

もう1つの早いソートである「マージソート」を学習します。

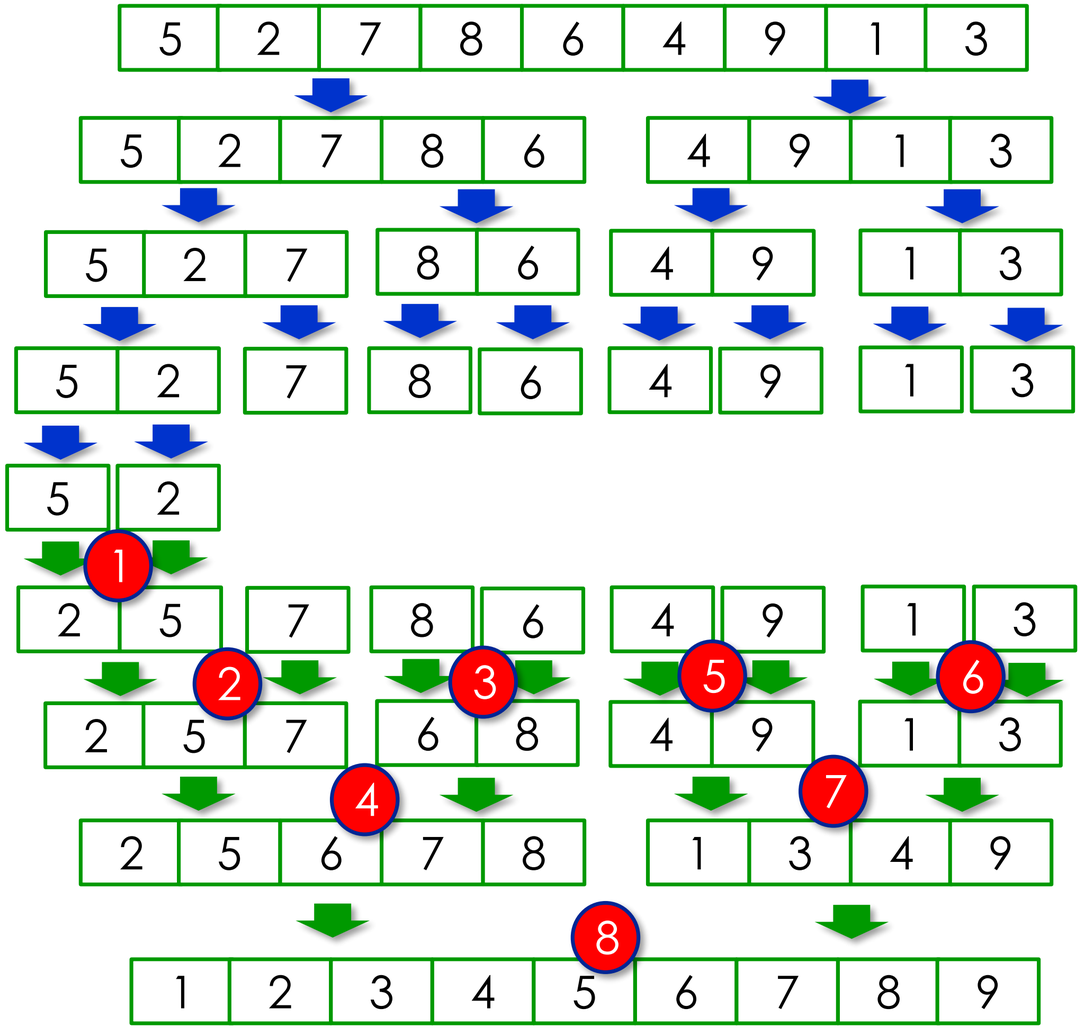
マージソート

- 以下の手順を再帰的に繰返すアルゴリズムです
  1. リストを半分に分割する
  2. 半分になったリストを、それぞれソート対象として再帰的にソートする
  3. ソートされたリストを、順序が保たれるように、1本にマージする

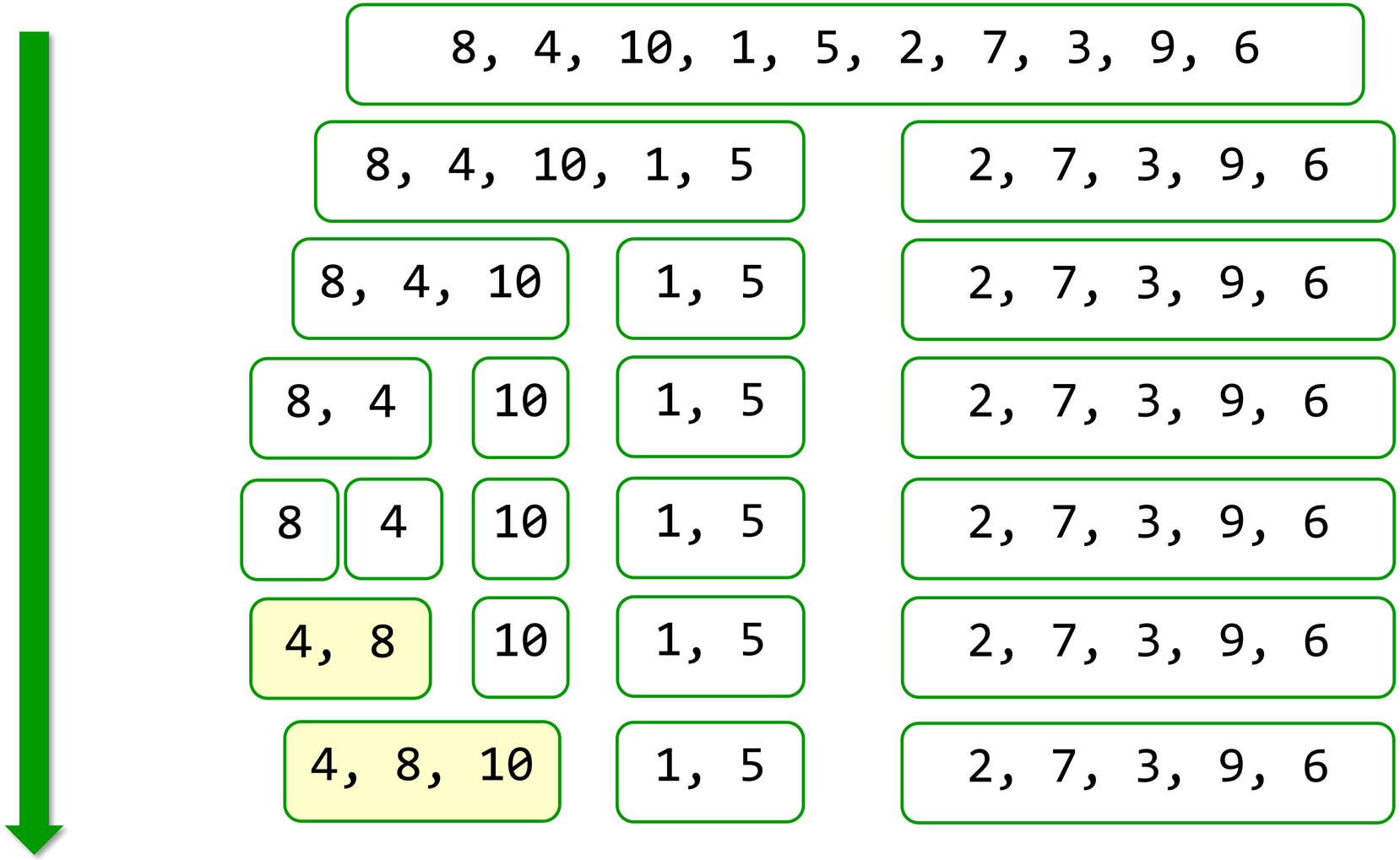


マージソート手順

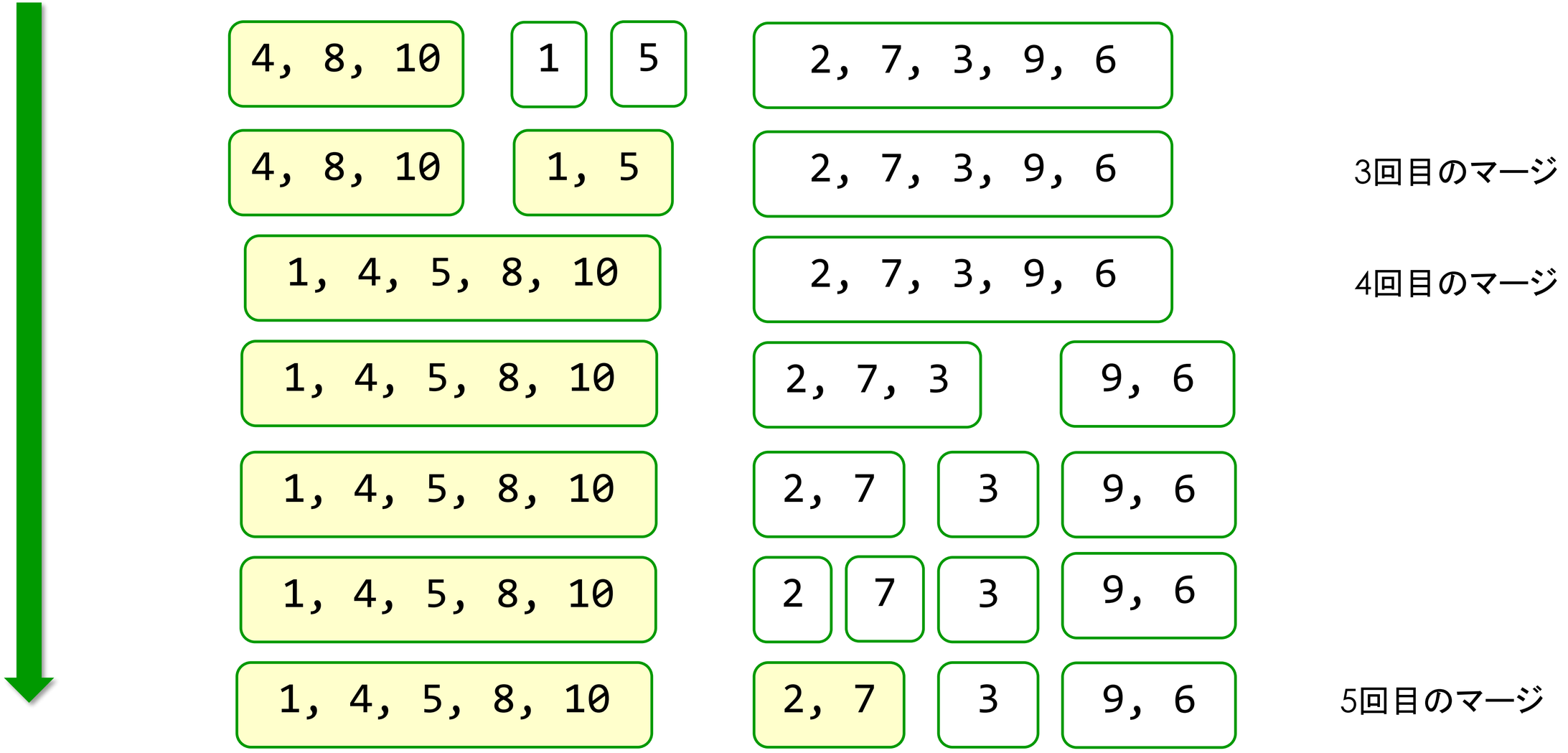
- 次のような手順となります
  1. リストの長さが1の場合は、ソート済みのため終了
  2. リストを半分に分割する（青）
  3. 半分に分割したリストを、それぞれ再帰的にソートする
  4. 前半のリストと後半のリストを、1本のリストにマージする（緑）
- 前半をソートしてから、後半をソートすることに注意！
  - マージ順は、右の番号の通り



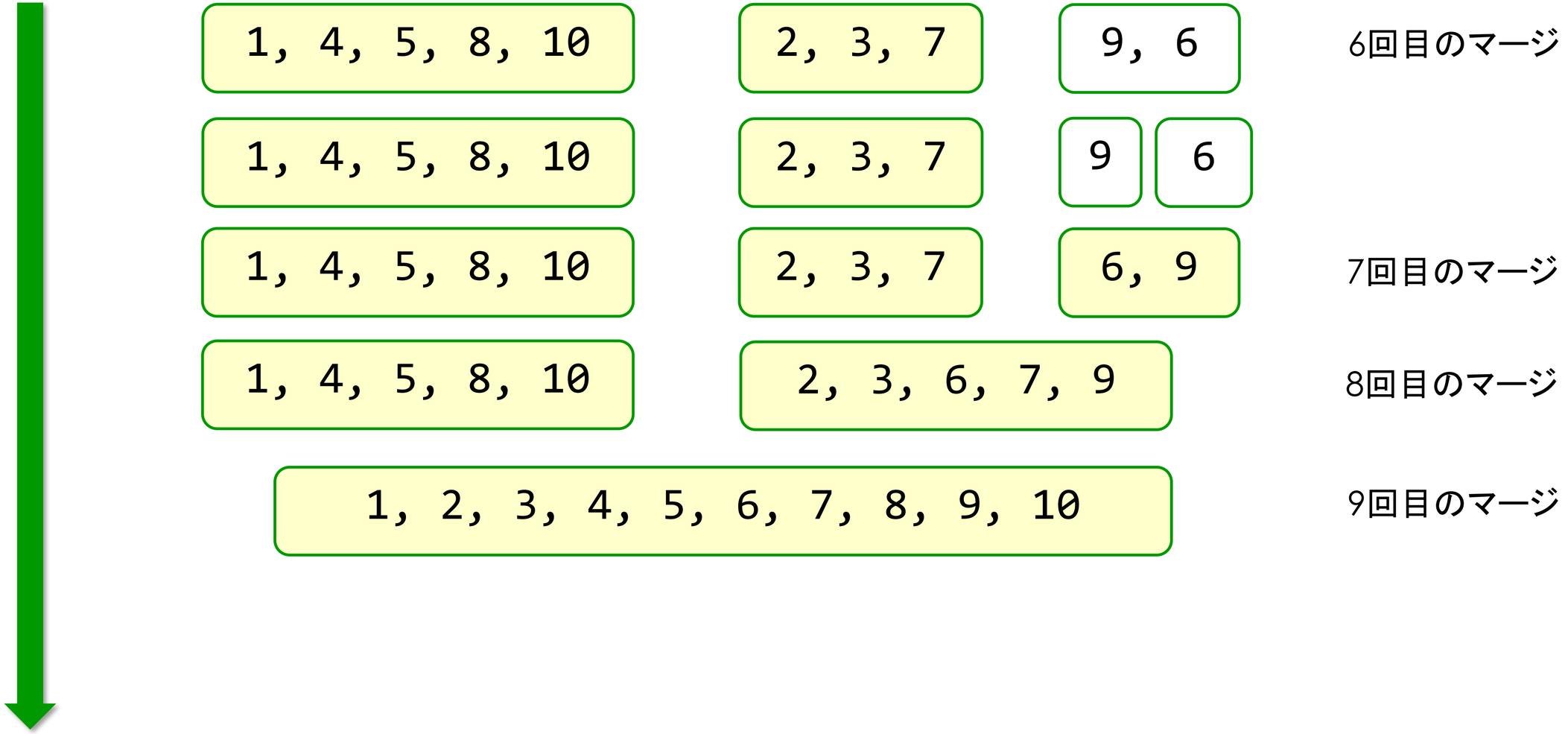
マージソート：具体例


8, 4, 10, 1, 5, 2, 7, 3, 9, 6
8, 4, 10, 1, 5
2, 7, 3, 9, 6
8, 4, 10
1, 5
2, 7, 3, 9, 6
8, 4
10
1, 5
2, 7, 3, 9, 6
8
4
10
1, 5
2, 7, 3, 9, 6
4, 8
10
1, 5
2, 7, 3, 9, 6
4, 8, 10
1, 5
2, 7, 3, 9, 6
1回目のマージ
2回目のマージ

マージソート：具体例


4, 8, 10
1
5
2, 7, 3, 9, 6
4, 8, 10
1, 5
2, 7, 3, 9, 6
3回目のマージ
1, 4, 5, 8, 10
2, 7, 3, 9, 6
4回目のマージ
1, 4, 5, 8, 10
2, 7, 3
9, 6
1, 4, 5, 8, 10
2, 7
3
9, 6
1, 4, 5, 8, 10
2
7
3
9, 6
1, 4, 5, 8, 10
2, 7
3
9, 6
5回目のマージ

マージソート：具体例


1, 4, 5, 8, 10
2, 3, 7
9, 6
6回目のマージ
1, 4, 5, 8, 10
2, 3, 7
9
6
1, 4, 5, 8, 10
2, 3, 7
6, 9
7回目のマージ
1, 4, 5, 8, 10
2, 3, 6, 7, 9
8回目のマージ
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
9回目のマージ

Python で定義してみると…

- まず、マージソート全体の手順は以下のように記述できます

```
def msort(lst):
    if len(lst) <= 1:
        return lst
    lst1 = lst[0: len(lst) // 2]
    lst2 = lst[len(lst) // 2:]

    sorted_lst1 = msort(lst1)
    sorted_lst2 = msort(lst2)

    return merge(sorted_lst1, sorted_lst2)
```

長さが1以下のリストはソートする必要はありません

リストを前半と後半に分割します

再帰的にソートします

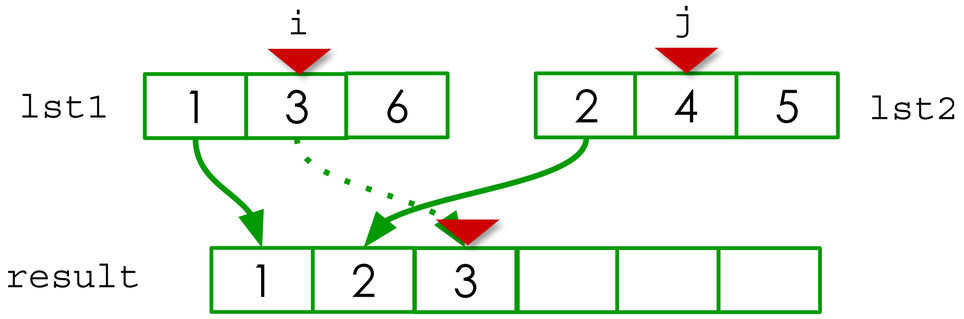
ソートの結果を、1本にマージします

Python で定義してみると…

- 最後の「マージ」の部分は、以下のような手順になります

```
def merge(lst1, lst2):
    result = []
    i = 0
    j = 0
    while (i < len(lst1)) and (j < len(lst2)):
        if lst1[i] <= lst2[j]:
            result.append(lst1[i])
            i += 1
        else:
            result.append(lst2[j])
            j += 1
    while i < len(lst1):
        result.append(lst1[i])
        i += 1
    while j < len(lst2):
        result.append(lst2[j])
        j += 1

    return result
```



2つのリストを先頭から見ていき、小さいものから順にresultに追加していきます

lst1に残りがあれば、最後にそれを追加します

lst2に残りがあれば、最後にそれを追加します

マージソートの時間を計測する

- クイックソートと同様の手順で、リストの長さを、 100 から 1000 まで 100 刻みで増やして、時間を計測してみましょう

```
for i in range(1, 11):
    target = random.sample(range(1000000), i * 100)
    %timeit msort(target)
```

マージソートの時間を計測する

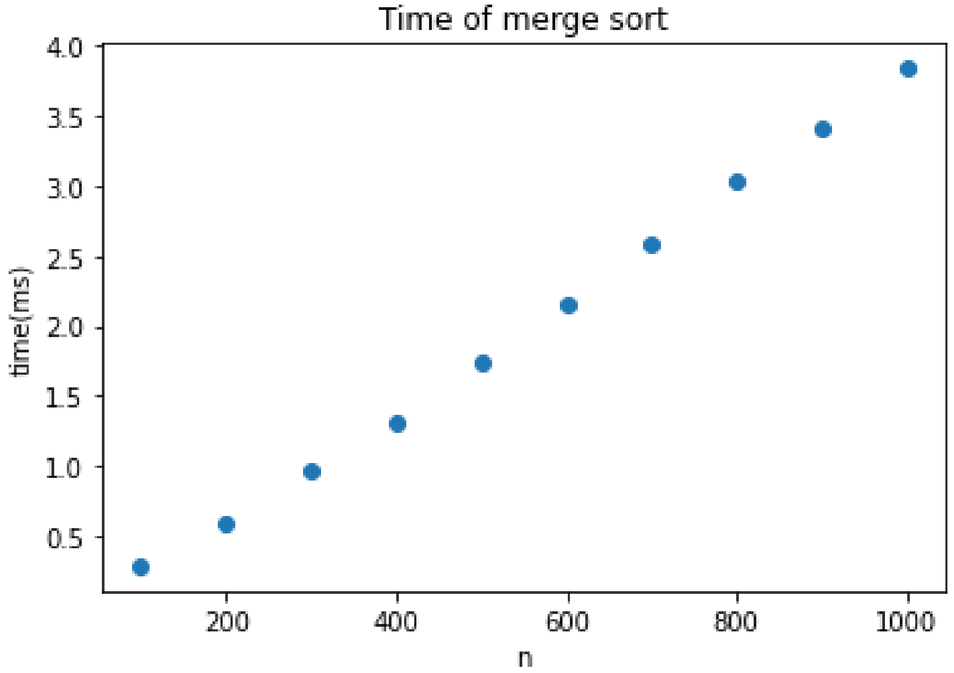
- その結果を matplotplib を使ってプロットしてみます
  - 計測時間は、自分の計測結果を入れてください

```
xs = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
ys = [0.285, 0.597, 0.965, 1.3, 1.74, 2.16, 2.58, 3.03, 3.42, 3.84]

plt.plot(xs, ys, 'o')
plt.xlabel('n')
plt.ylabel('time(ms)')
plt.title('Time of merge sort')
plt.show()
```

以下のようになりましたか？

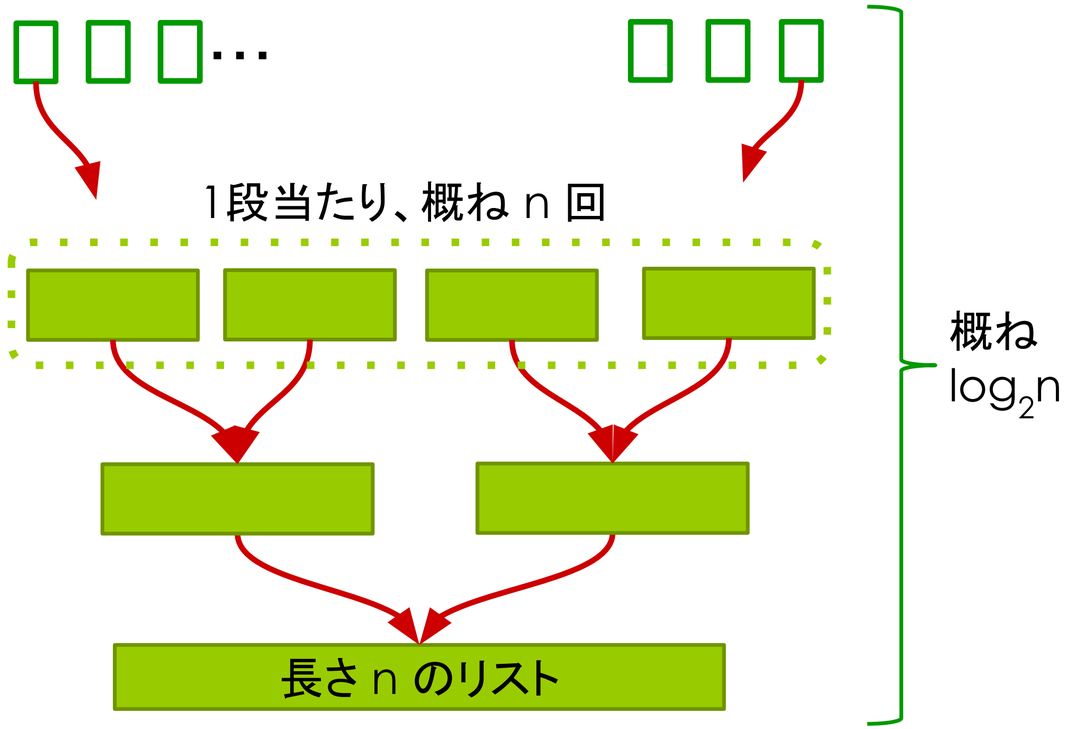
- Pythonのソートやクイックソートの場合と同様に、「ほぼ」直線的なグラフになるはずです



こちらも一見直線的なグラフに見えますが、理論的には少し違います

マージソートの計算量

- リストの長さを n とした時、マージソートの計算量は？
  - ※この場合は、平均計算量も最悪計算量も変わりません
- 厳密な証明は省きますが、直感的には以下のイメージです
  - 1回の分割で、リストの長さは半分になります
  - そのため、リストの長さを n とすれば、分割の段数は、約 $\log_2 n$ になります
  - 1段のマージ処理の計算回数は、概ね n に比例します
  - そのため…



➡ $O(n \log n)$