快速排序 – 执行流程

① 从序列中选择一个轴点元素（pivot）

✓ 假设每次选择 0 位置的元素为轴点元素

② 利用 pivot 将序列分割成 2 个子序列

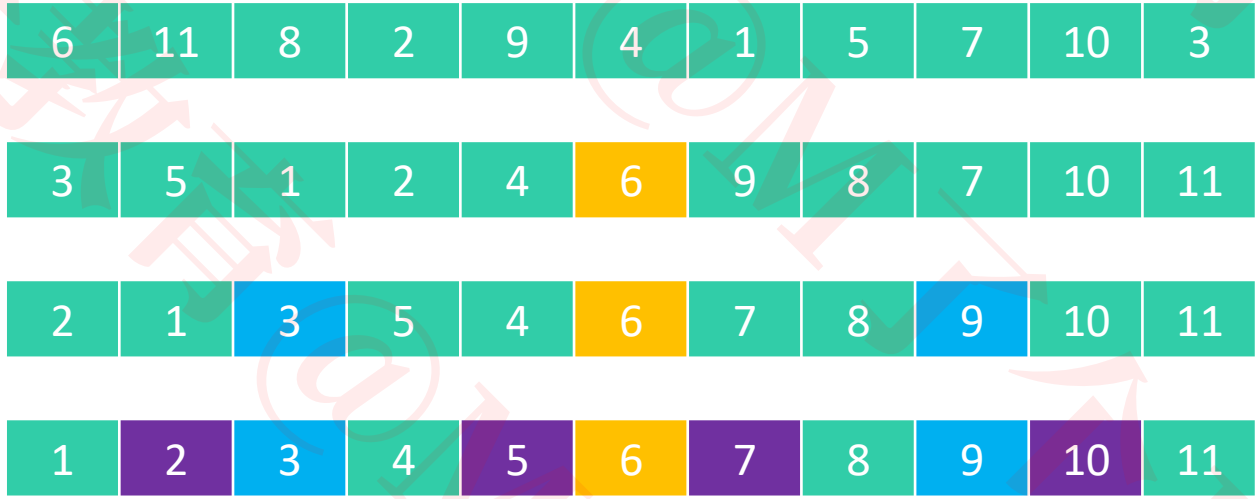
✓ 将小于 pivot 的元素放在pivot前面（左侧）

✓ 将大于 pivot 的元素放在pivot后面（右侧）

✓ 等于pivot的元素放哪边都可以

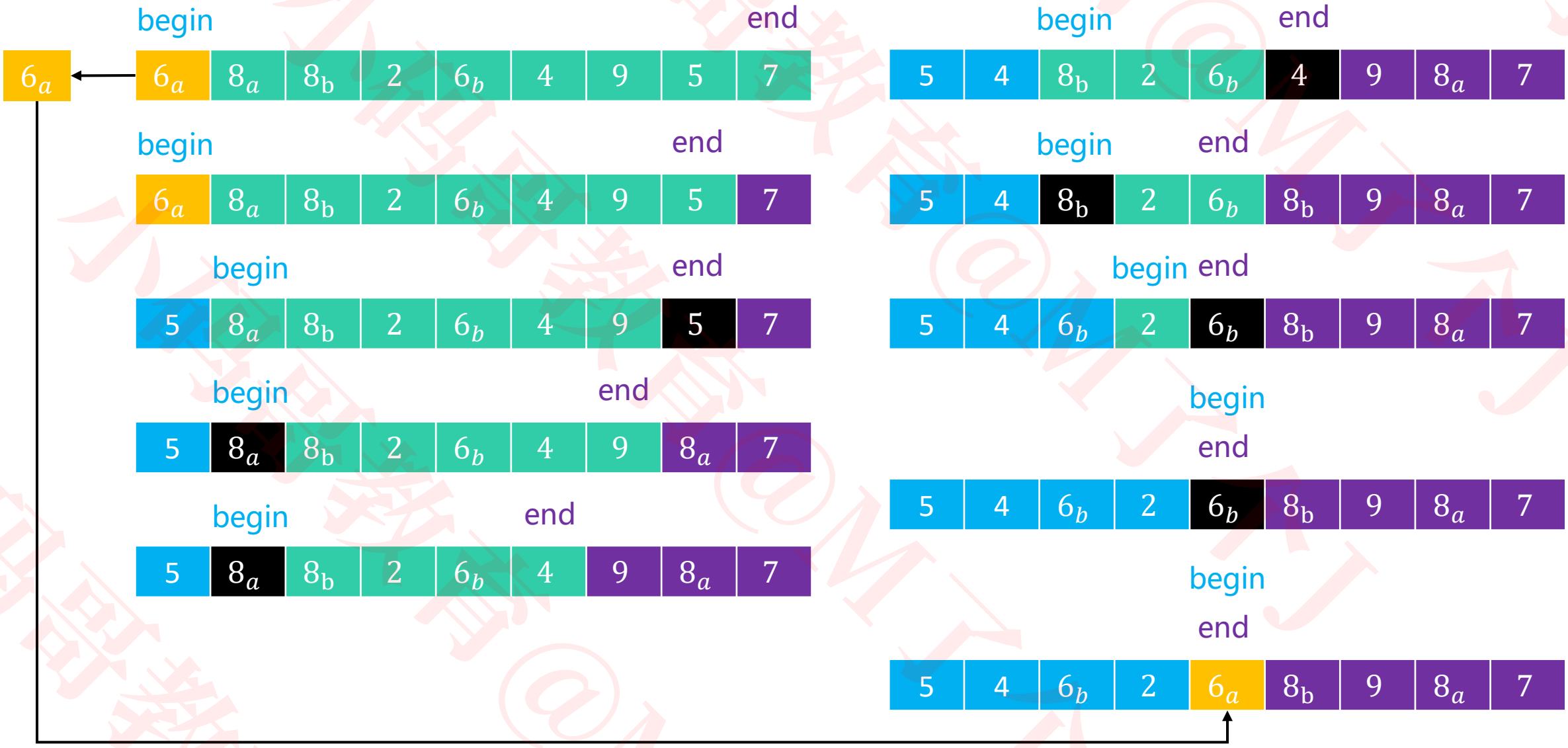
③ 对子序列进行 ① ② 操作

✓ 直到不能再分割（子序列中只剩下1个元素）



- 快速排序的本质
- 逐渐将每一个元素都转换成轴点元素

快速排序 – 轴点构造


begin
end
6_a 8_a 8_b 2 6_b 4 9 5 7
6_a
begin
end
6_a 8_a 8_b 2 6_b 4 9 5 7
begin
end
5 8_a 8_b 2 6_b 4 9 5 7
begin
end
5 8_a 8_b 2 6_b 4 9 8_a 7
begin
end
5 8_a 8_b 2 6_b 4 9 8_a 7
begin
end
5 4 8_b 2 6_b 4 9 8_a 7
begin
end
5 4 8_b 2 6_b 8_b 9 8_a 7
begin
end
5 4 6_b 2 6_b 8_b 9 8_a 7
begin
end
5 4 6_b 2 6_b 8_b 9 8_a 7
begin
end
5 4 6_b 2 6_a 8_b 9 8_a 7

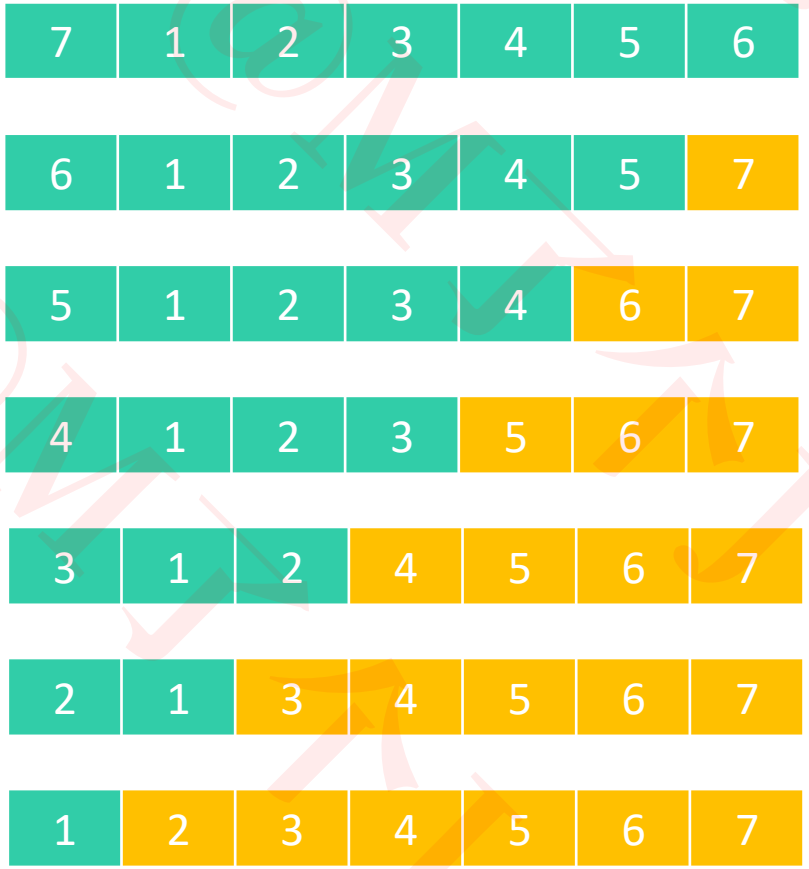
快速排序 – 时间复杂度

- 在轴点左右元素数量比较均匀的情况下，同时也是最好的情况
  - $T(n) = 2 * T(n/2) + O(n) = O(nlogn)$

- 如果轴点左右元素数量极度不均匀，最坏情况
  - $T(n) = T(n-1) + O(n) = O(n^2)$

- 为了降低最坏情况的出现概率，一般采取的做法是
  - 随机选择轴点元素

- 最好、平均时间复杂度：$O(nlogn)$
- 最坏时间复杂度：$O(n^2)$
- 由于递归调用的缘故，空间复杂度：$O(logn)$
- 属于不稳定排序



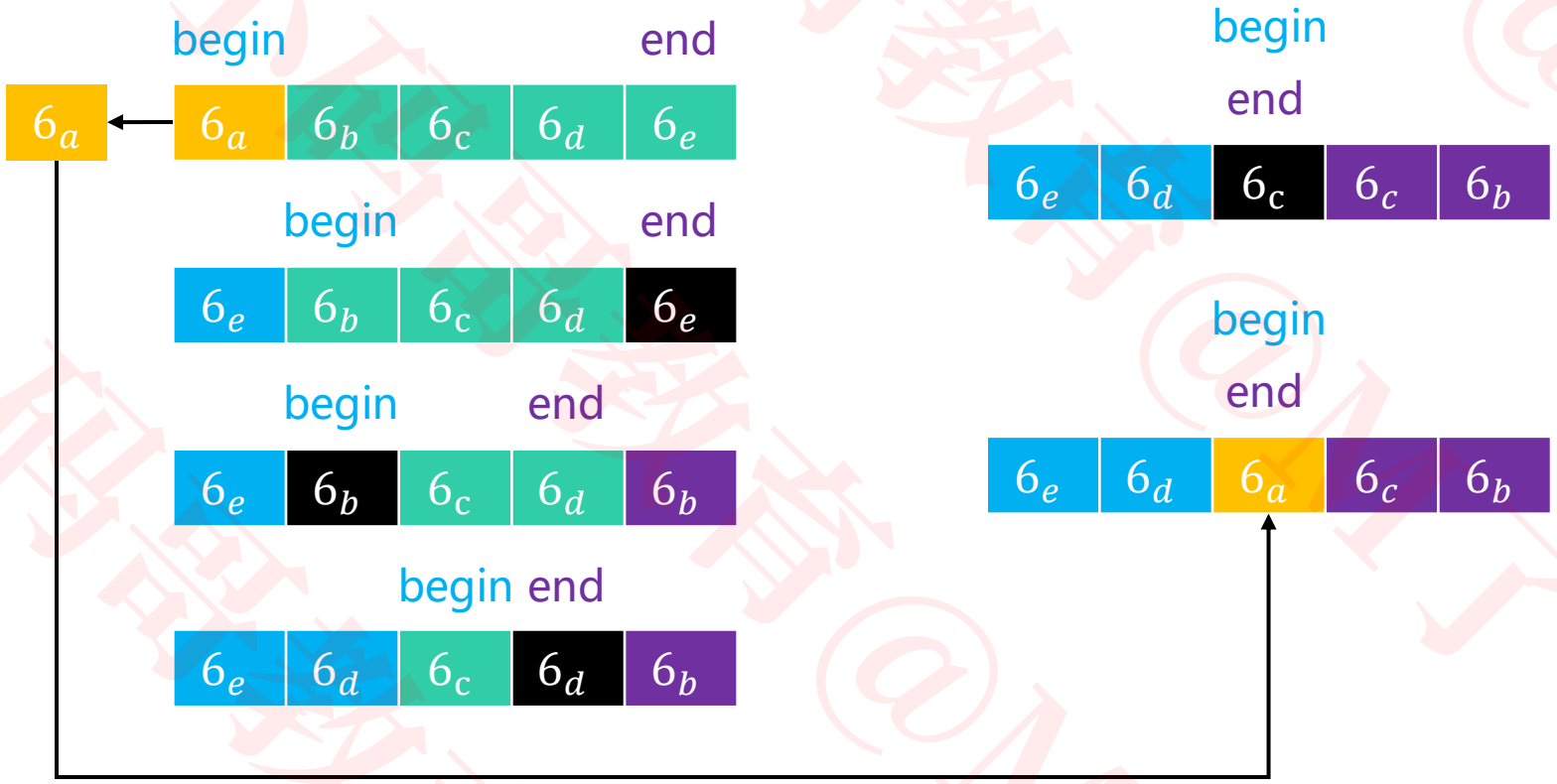
快速排序 - 实现

```java
/* [begin, end) */
private void sort(int begin, int end) {
    // 至少要有2个元素
    if (end - begin < 2) return;

    int middle = pivotIndex(begin, end);
    sort(begin, middle);
    sort(middle + 1, end);
}
```

```java
private int pivotIndex(int begin, int end) {
    // 随机交换begin位置的元素
    swap(begin, begin + (int)(Math.random() * (end - begin)));
    T pivot = array[begin];
    end--; // end指向最后1个元素
    while (begin < end) {
        while (begin < end) {
            if (cmp(pivot, array[end]) < 0) {
                end--;
            } else {
                array[begin++] = array[end];
                break;
            }
        }
        while (begin < end) {
            if (cmp(pivot, array[begin]) > 0) {
                begin++;
            } else {
                array[end--] = array[begin];
                break;
            }
        }
    }
    array[begin] = pivot;
    return begin;
}
```

快速排序 – 与轴点相等的元素

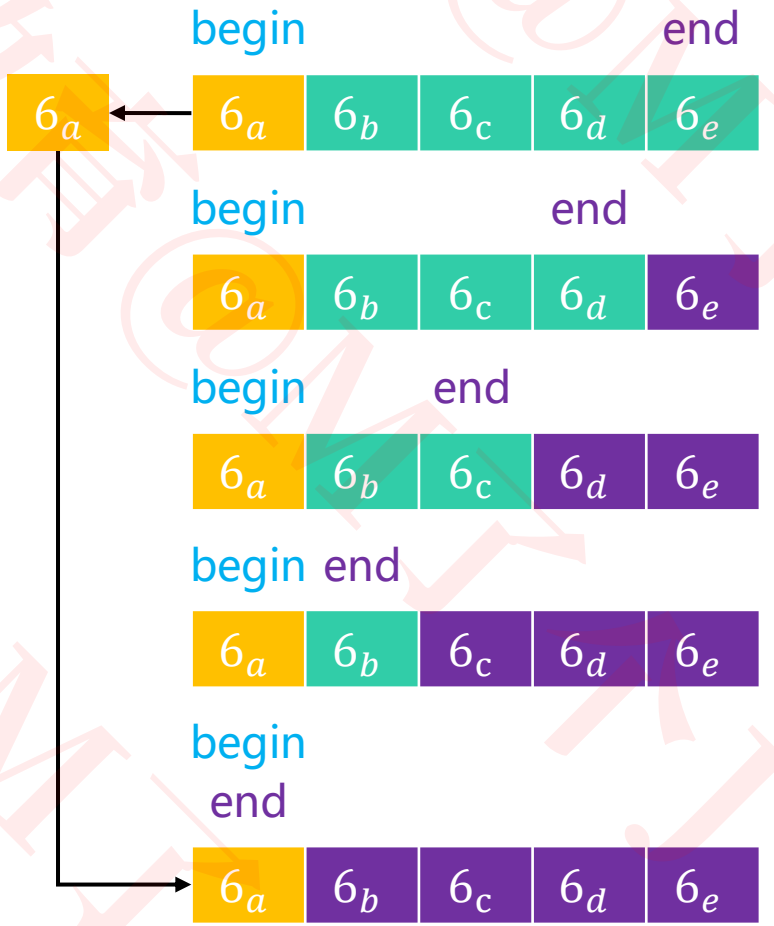


- 如果序列中的所有元素都与轴点元素相等，利用目前的算法实现，轴点元素可以将序列分割成 2 个均匀的子序列

快速排序 – 与轴点相等的元素

- 思考：cmp 位置的判断分别改为 ≤、≥ 会起到什么效果?

```
while (begin < end) {
    if (cmp(pivot, array[end]) <= 0) {
        end--;
    } else {
        array[begin++] = array[end];
        break;
    }
}
while (begin < end) {
    if (cmp(pivot, array[begin]) >= 0) {
        begin++;
    } else {
        array[end--] = array[begin];
        break;
    }
}
```



- 轴点元素分割出来的子序列极度不均匀
- 导致出现最坏时间复杂度 $O(n^2)$