本文内容是对归并排序的梳理和总结，本文内容包括：

排序算法十大经典方法

1. 冒泡排序：比较相邻元素并交换，每一轮将最大的元素移动到最后。
2. 选择排序：每一轮选出未排序部分中最小的元素，并将其放在已排序部分的末尾。
3. 插入排序：将未排序部分的第一个元素插入到已排序部分的合适位置
4. 希尔排序：改进的插入排序，将数据分组排序，最终合并排序
5. **归并排序：将序列拆分成子序列，分别排序后合并，递归完成**
6. 快速排序：选定一个基准值，将小于基准值的元素放在左边，大于基准值的元素放在右边，递归排序
7. 堆排序：将序列构建成一个堆，然后一次将堆顶元素取出并调整堆
8. 计数排序：统计每个元素出现的次数，再按照元素大小输出
9. 桶排序：将数据分到一个或多个桶中，对每个桶进行排序，最后输出
10. 基数排序：按照元素的位数从低到高进行排序，每次排序只考虑一位

1 归并排序(Merge Sort)

- 归并排序（MERGE-SORT）是利用归并的思想实现的排序方法，该算法的核心是分而治之

1.1 算法步骤

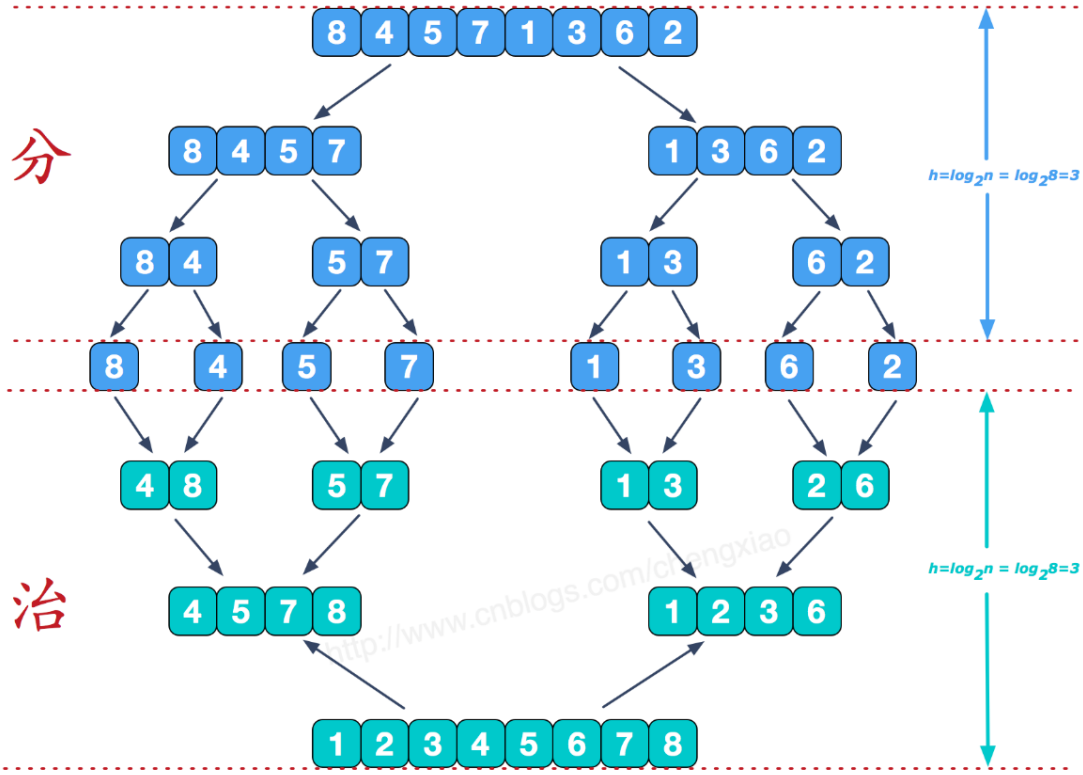
1. 把长度为n的序列分成两个长度为n/2的子序列
2. 对这两个子序列分别采用归并排序

3. 将两个排序好的子序列合并成一个最终的排序序列

1.2 举例说明

- **需求说明**：有数组如下，使用归并排序该数组

- **基本思想**：归并排序的核心思想是先分后治，看下面的图解说明

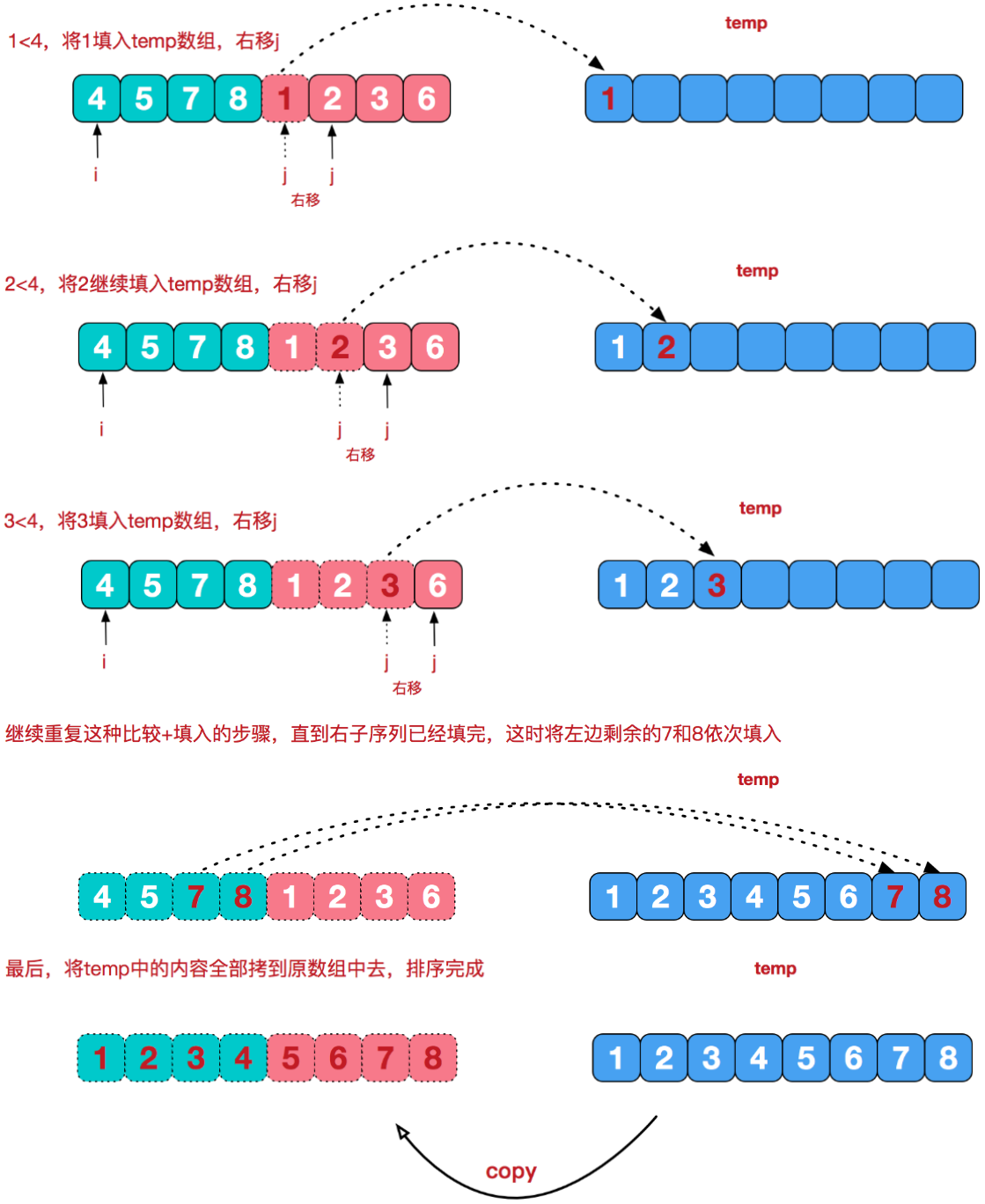


- **分的阶段**

  分阶段可以理解为就是递归拆分子序列的过程，递归深度为log2n。

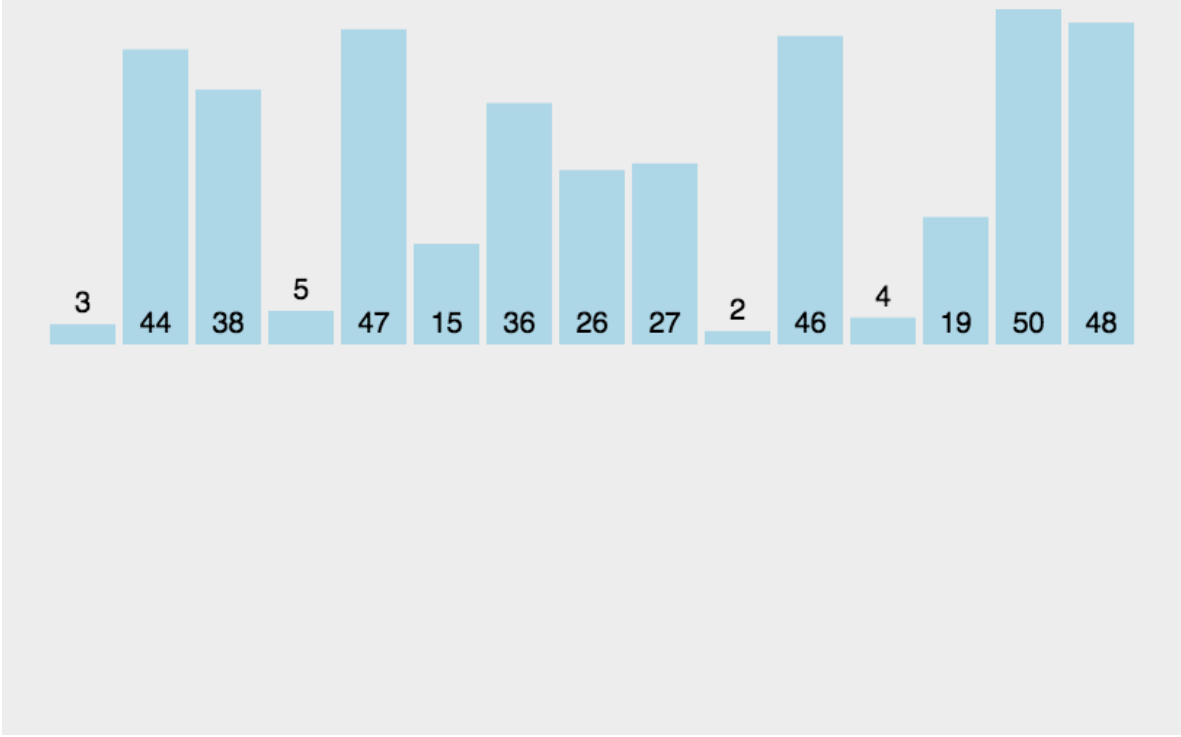
- **治的阶段**

  治阶段是将两个已经有序的子序列合并成一个有序序列。比如上图中的最后一次合并，要将[4,5,7,8]和[1,2,3,6]两个已经有序的子序列，合并为最终序列[1,2,3,4,5,6,7,8]，步骤如下。



1.3 图解算法

如果下图不动，点击这里查看在线的图解



1.4 代码实现

```java
public static void main(String[] args) {
    int[] arr = {8, 4, 5, 7, 1, 3, 6, 2};
    sort(arr);
    System.out.println(Arrays.toString(arr));
}

public static void sort(int[] arr) {
    //在排序前，先建好一个长度等于原数组长度的临时数组，避免递归中频繁开辟空间
    int[] temp = new int[arr.length];
    sort(arr, 0, arr.length - 1, temp);
}

private static void sort(int[] arr, int left, int right, int[] temp) {
    if (left < right) {
        int mid = (left + right) / 2;

        //左边归并排序，使得左子序列有序
        sort(arr, left, mid, temp);

        //右边归并排序，使得右子序列有序
        sort(arr, mid + 1, right, temp);
```

```
22
23          //将两个有序子数组合并操作
24          merge(arr, left, mid, right, temp);
25      }
26  }
27
28  private static void merge(int[] arr, int left, int mid, int
    right, int[] temp) {
29      int i = left;//左序列指针
30      int j = mid + 1;//右序列指针
31      int t = 0;//临时数组指针
32      while (i <= mid && j <= right) {
33          if (arr[i] <= arr[j]) {
34              temp[t++] = arr[i++];
35          } else {
36              temp[t++] = arr[j++];
37          }
38      }
39      while (i <= mid) {//将左边剩余元素填充进temp中
40          temp[t++] = arr[i++];
41      }
42      while (j <= right) {//将右序列剩余元素填充进temp中
43          temp[t++] = arr[j++];
44      }
45      t = 0;
46      //将temp中的元素全部拷贝到原数组中
47      while (left <= right) {
48          arr[left++] = temp[t++];
49      }
50  }
```

1.5 算法分析

- **稳定性**： 稳定
- **时间复杂度**： 最佳：O(nlogn)， 最差：O(nlogn)，平均：O(nlogn)
- **空间复杂度**： O(n)