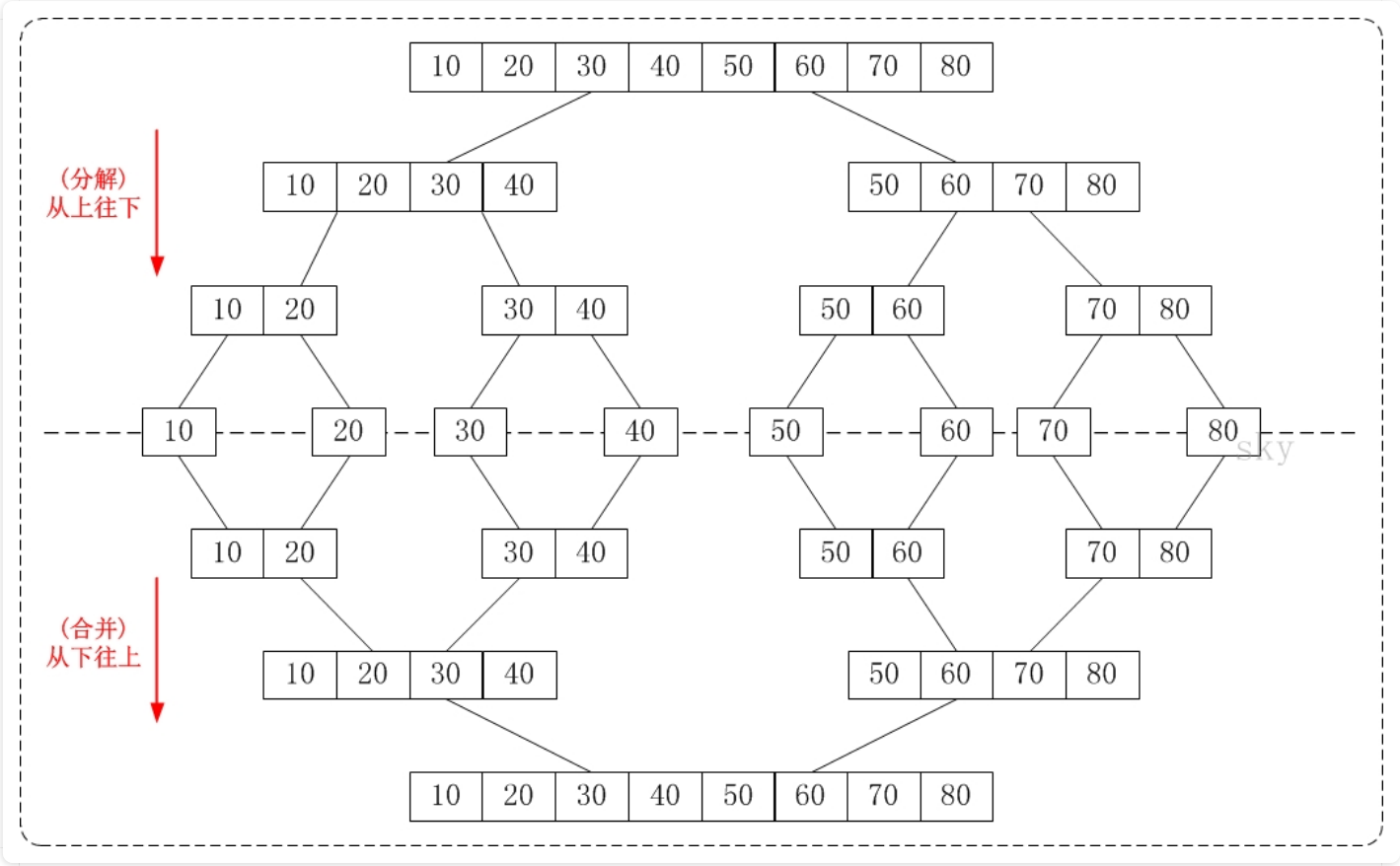
1、归并排序介绍

- 归并排序（MERGE-SORT）是建立在归并操作上的一种有效的排序算法,该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。
- 归并排序(Merge Sort)就是利用归并思想对数列进行排序。根据具体的实现，归并排序包括"从上往下"和"从下往上"2种方式
- 从上往下的归并排序：
  - 分解 – 将当前区间一分为二，即求分裂点 mid = (low + high)/2
  - 求解 – 递归地对两个子区间a[low...mid] 和 a[mid+1...high]进行归并排序。递归的终结条件是子区间长度为1
  - 合并 – 将已排序的两个子区间a[low...mid]和 a[mid+1...high]归并为一个有序的区间a[low...high]



2、原理介绍

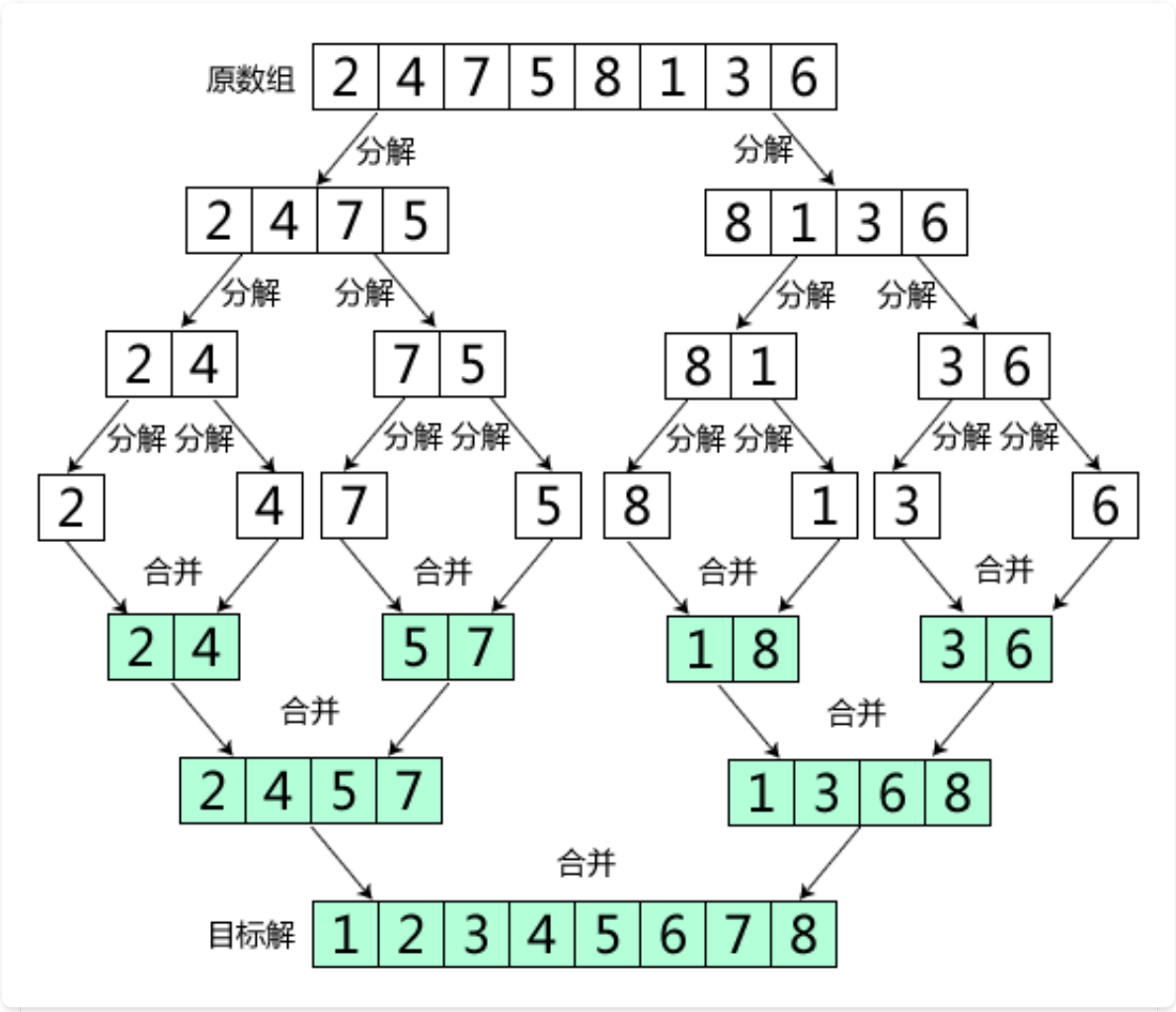
二路归并排序主旨是“分解”与“归并

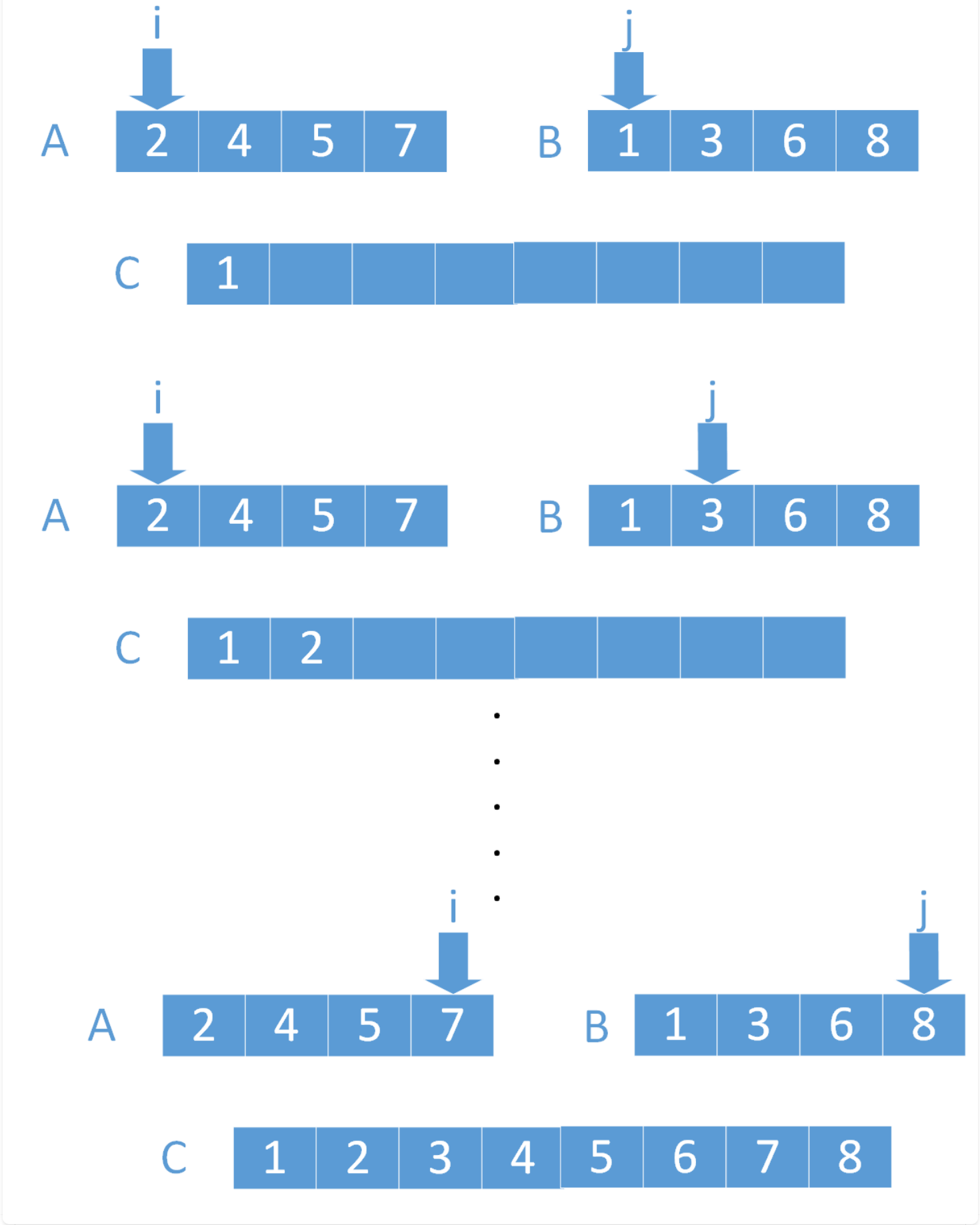
分解

- 1.将一个数组分成两个数组，分别对两个数组进行排序。
- 2.循环第一步，直到划分出来的“小数组”只包含一个元素，只有一个元素的数组默认为已经排好序。

归并

- 1.将两个有序的数组合并到一个大的数组中。
- 2.从最小的只包含一个元素的数组开始两两合并。此时，合并好的数组也是有序的。





- 1.图中原始数组为{2,4,7,5,8,1,3,6}，数组中元素的个数为8个。首先将8个元素的数组二分，每次分解后，

数组中元素的数目为原数组的一般。直到分解为只含有一个元素的数组。

- 2.将小的数组按序合并，每次合并后数组的大小为上层数组的一倍。此时数组中的元素都是按序排列的。
- 3.在合并两个有序数组。如图2
  - 合并时，有两个指针分别指向有序数组A和B的起始元素，比较指针所指元素的大小，如果A[i]较小，则将A[i] 存入数组C中，并且将i后移。循环比较i和j所指的元素。
  - 当一个数组A的元素全部排之后，数组B中的指针就并没有指向B的末尾位置，将B中剩余元素全部存入到C中。

3、归并排序的时间复杂度和稳定性

- 归并排序时间复杂度
  - 归并排序的时间复杂度是O(N*lgN)。
  - 假设被排序的数列中有N个数。遍历一趟的时间复杂度是O(N)，需要遍历多少次呢?
  - 归并排序的形式就是一棵二叉树，它需要遍历的次数就是二叉树的深度，而根据完全二叉树的可以得出它的时间复杂度是O(N*lgN)。
- 归并排序稳定性
  - 归并排序是稳定的算法，它满足稳定算法的定义。
  - 算法稳定性 – 假设在数列中存在a[i]=a[j]，若在排序之前，a[i]在a[j]前面；并且排序之后，a[i]仍然在a[j]前面。则这个排序算法是稳定的!

4、代码实践

```
/*
 * 将一个数组中的两个相邻有序区间合并成一个
 *
 * 参数说明:
 *     a -- 包含两个有序区间的数组
 *     start -- 第1个有序区间的起始地址。
 *     mid   -- 第1个有序区间的结束地址。也是第2个有序区间的起始地址。
 *     end   -- 第2个有序区间的结束地址。
 */
void merge(int* a, int start, int mid, int end)
{
    int *tmp = new int[end-start+1];    // tmp是汇总2个有序区的临时区域
    int i = start;              // 第1个有序区的索引
    int j = mid + 1;          // 第2个有序区的索引
    int k = 0;                  // 临时区域的索引

    while(i <= mid && j <= end)
```

```
    {
        if (a[i] <= a[j])
            tmp[k++] = a[i++];
        else
            tmp[k++] = a[j++];
    }

    while(i <= mid)
        tmp[k++] = a[i++];

    while(j <= end)
        tmp[k++] = a[j++];

    // 将排序后的元素，全部都整合到数组a中。
    for (i = 0; i < k; i++)
        a[start + i] = tmp[i];

    delete[] tmp;
}

/*
 * 归并排序(从上往下)
 *
 * 参数说明：
 *     a -- 待排序的数组
 *     start -- 数组的起始地址
 *     endi -- 数组的结束地址
 */
void mergeSortUp2Down(int* a, int start, int end)
{
    if(a==NULL || start >= end)
        return ;

    int mid = (end + start)/2;
    mergeSortUp2Down(a, start, mid); // 递归排序a[start...mid]
    mergeSortUp2Down(a, mid+1, end); // 递归排序a[mid+1...end]

    // a[start...mid] 和 a[mid...end]是两个有序空间，
    // 将它们排序成一个有序空间a[start...end]
    merge(a, start, mid, end);
}



/*
 * 对数组a做若干次合并：数组a的总长度为len，将它分为若干个长度为gap的子数组；
 *             将"每2个相邻的子数组" 进行合并排序。
 *
 * 参数说明：
 *     a -- 待排序的数组
 *     len -- 数组的长度
 *     gap -- 子数组的长度
 */
void mergeGroups(int* a, int len, int gap)
{
```

```
    int i;
    int twolen = 2 * gap;    // 两个相邻的子数组的长度

    // 将"每2个相邻的子数组" 进行合并排序。
    for(i = 0; i+2*gap-1 < len; i+=(2*gap))
    {
        merge(a, i, i+gap-1, i+2*gap-1);
    }

    // 若 i+gap-1 < len-1，则剩余一个子数组没有配对。
    // 将该子数组合并到已排序的数组中。
    if ( i+gap-1 < len-1)
    {
        merge(a, i, i + gap - 1, len - 1);
    }
}

/*
 * 归并排序(从下往上)
 *
 * 参数说明：
 *     a -- 待排序的数组
 *     len -- 数组的长度
 */
void mergeSortDown2Up(int* a, int len)
{
    int n;

    if (a==NULL || len<=0)
        return ;

    for(n = 1; n < len; n*=2)
        mergeGroups(a, len, n);
}

int main()
{
    int i;
    int a[] = {80,30,60,40,20,10,50,70};
    int ilen = (sizeof(a)) / (sizeof(a[0]));

    cout << "before sort:";
    for (i=0; i<ilen; i++)
        cout << a[i] << " ";
    cout << endl;

    mergeSortUp2Down(a, 0, ilen-1);        // 归并排序(从上往下)
    //mergeSortDown2Up(a, ilen);            // 归并排序(从下往上)

    cout << "after  sort:";
    for (i=0; i<ilen; i++)
        cout << a[i] << " ";
    cout << endl;
```

```
    return 0;
}
```

5、推荐阅读

归并排序0
归并排序1
归并排序2