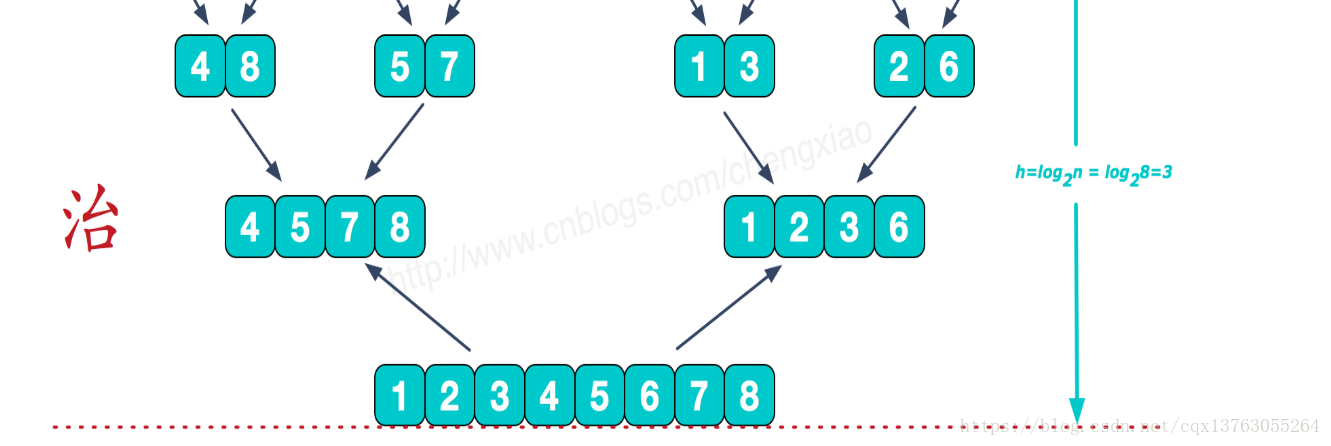
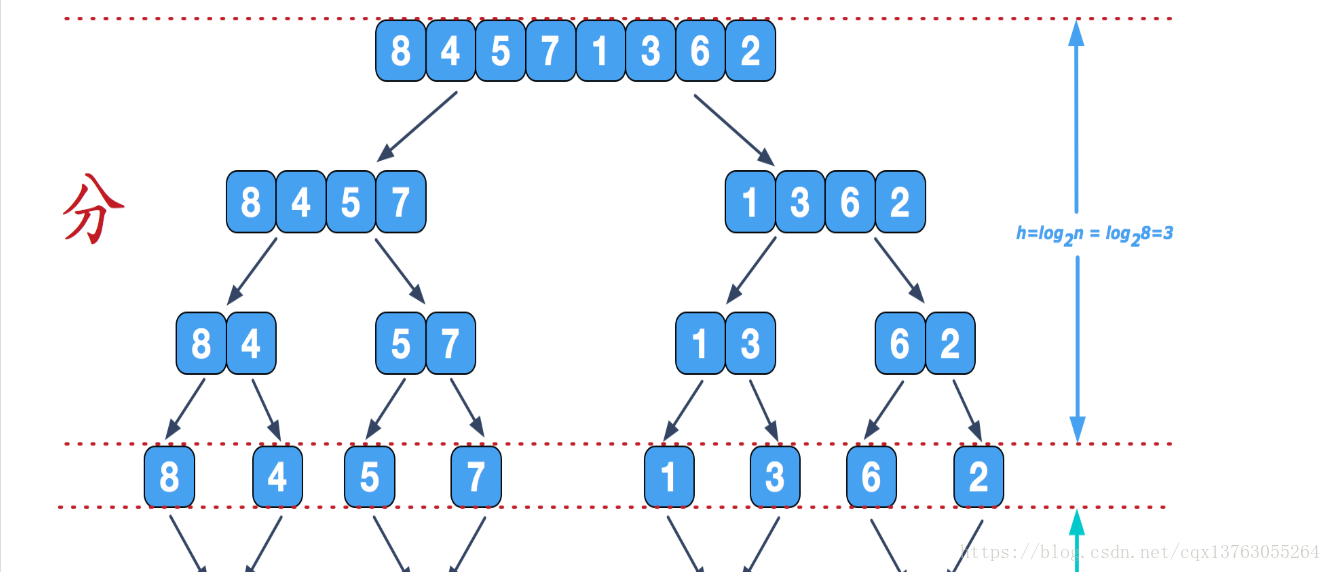
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 排序算法 | 平均时间复杂度 | 最坏时间复杂度 | 空间复杂度 | 是否稳定 |
| 冒泡排序 | O（n2） | O（n2） | O（1） | 是 |
| 选择排序 | O（n2） | O（n2） | O（1） | 不是 |
| 插入排序 | O（n2） | O（n2） | O（1） | 是 |
| 归并排序 | O(nlogn) | O(nlogn) | O（n） | 是 |
| 快速排序 | O(nlogn) | O（n2） | O（logn） | 不是 |
| 堆排序 | O(nlogn) | O(nlogn) | O（1） | 不是 |
| 希尔排序 | O(nlogn) | O（ns） | O（1） | 不是 |
| 计数排序 | O(n+k) | O(n+k) | O(n+k) | 是 |
| 基数排序 | O(2\*N∗M) | O(2\*N∗M) | O(N+M) | 是 |

k是整数的范围

1. 归并排序

核心思想：分而治之，先把问题分解成小问题，再递归求解。

（1）稳定性  
　 归并排序是一种**稳定**的排序。  
（2）存储结构要求  
　可用**顺序存储结构。也易于在链表上实现。**（3）时间复杂度  
　对长度为n的文件，需进行趟二路归并，每趟归并的时间为O(n)，故其时间复杂度无论是在最好情况下还是在最坏情况下**均是O(nlgn)。**  
（4）空间复杂度  
　 需要一个辅助向量来暂存两有序子文件归并的结果，故其辅助空**间复杂度为O(n)**，显然它不是就地排序。

public static void sort(Comparable[] a) {  
 int N=a.length;  
 Comparable[] temp = new Comparable[N];  
 //自顶向下的归并方法  
// sort(a, 0, a.length - 1, temp);  
 //自底向上的归并方法

//每2，4，8进行归并  
 for (int i = 1; i < N; i\*=2) {

//1-2,2-4,4-6,进行归并  
 for (int j = 0; j < N-i; j+=2\*i) {

//归并  
 *merge*(a,j,j+i-1,Math.*min*(N-1,j+2\*i-1),temp);  
 }  
 }  
 }

public static void sort(Comparable[] a, int left, int right, Comparable[] temp) {  
 if (right <= left) {  
 return;  
 }  
 int mid = left+(right -left) / 2;  
 *sort*(a, left, mid, temp);//分解左边  
 *sort*(a, mid + 1, right, temp);//分解右边  
 *merge*(a, left, mid, right, temp);//合并  
}

public static void merge(Comparable[] a, int left, int mid, int right, Comparable[] temp) {  
 //双指针，i为左边数组的指针，j为右边数组的指针，t为被填充的临时数组  
 int i = left;  
 int j = mid+1;  
 int t = 0;  
 //1.先把左右两边按顺序填充到临时数组，直到有一边被处理完毕  
 //i<=mid说明此时左边的数组索引还未走到中间，左边还没有拷贝完毕，j<right说明右边数组还没有走到右边，右边还没有拷贝完毕  
 while (i <= mid && j <= right) {  
 if (*less*(a[i],a[j])) {//如果左边元素小于右边元素，则把左边元素拷贝进临时数组  
 temp[t++] = a[i++];  
 } else {//否则说明右边元素小于左边元素，则把右边元素拷贝进临时数组  
 temp[t++] = a[j++];  
 }  
 }  
 //2.将剩下的（左或者右）拷贝进临时数组，此时全部拷贝完毕  
 while (i <= mid) {  
 //说明左边元素还没有全部填充，此时把左边元素填充进去  
 temp[t++] = a[i++];  
 }  
 while (j <= right) {  
 //说明右边元素还没有全部填充，此时把右边元素填充进去  
 temp[t++] = a[j++];  
 }  
 //3.将临时数组temp[]拷贝回a[]  
 t = 0;  
 while (left <= right) {  
 a[left++] = temp[t++];  
 }  
}

1. **计数排序**

核心思想：

1. 找出序列中最大值和最小值，开辟Max-Min+1的辅助空间
2. 最小的数对应下标为0的位置，遇到一个数就给对应下标处的值+1,。
3. 遍历一遍辅助空间，就可以得到有序的一组序列

public static void countSort(int[] arr){  
 int min = Integer.*MAX\_VALUE*;  
 int max = Integer.*MIN\_VALUE*;  
 for (int value : arr) {  
 if (value > max) {  
 max = value;  
 }  
 if (value < min) {  
 min = value;  
 }  
 }  
 int[] temp = new int[max - min + 1];  
 for (int value : arr) {  
 temp[value - min]++;  
 }  
 int idx = 0;  
 for (int i = 0; i < temp.length; i++) {  
 while (temp[i]-- != 0) {  
 arr[idx++] = i + min;  
 }  
 }  
}