 2018年1月3日*16:47:04*[7](https://cuijiahua.com/blog/2018/01/algorithm_7.html#comments) 4,866 °C



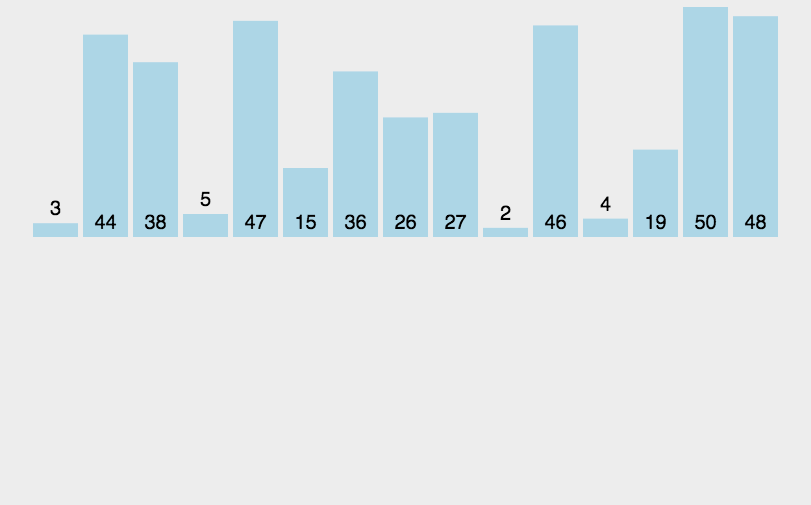
**一、前言**

归并排序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。

**二、算法思想**

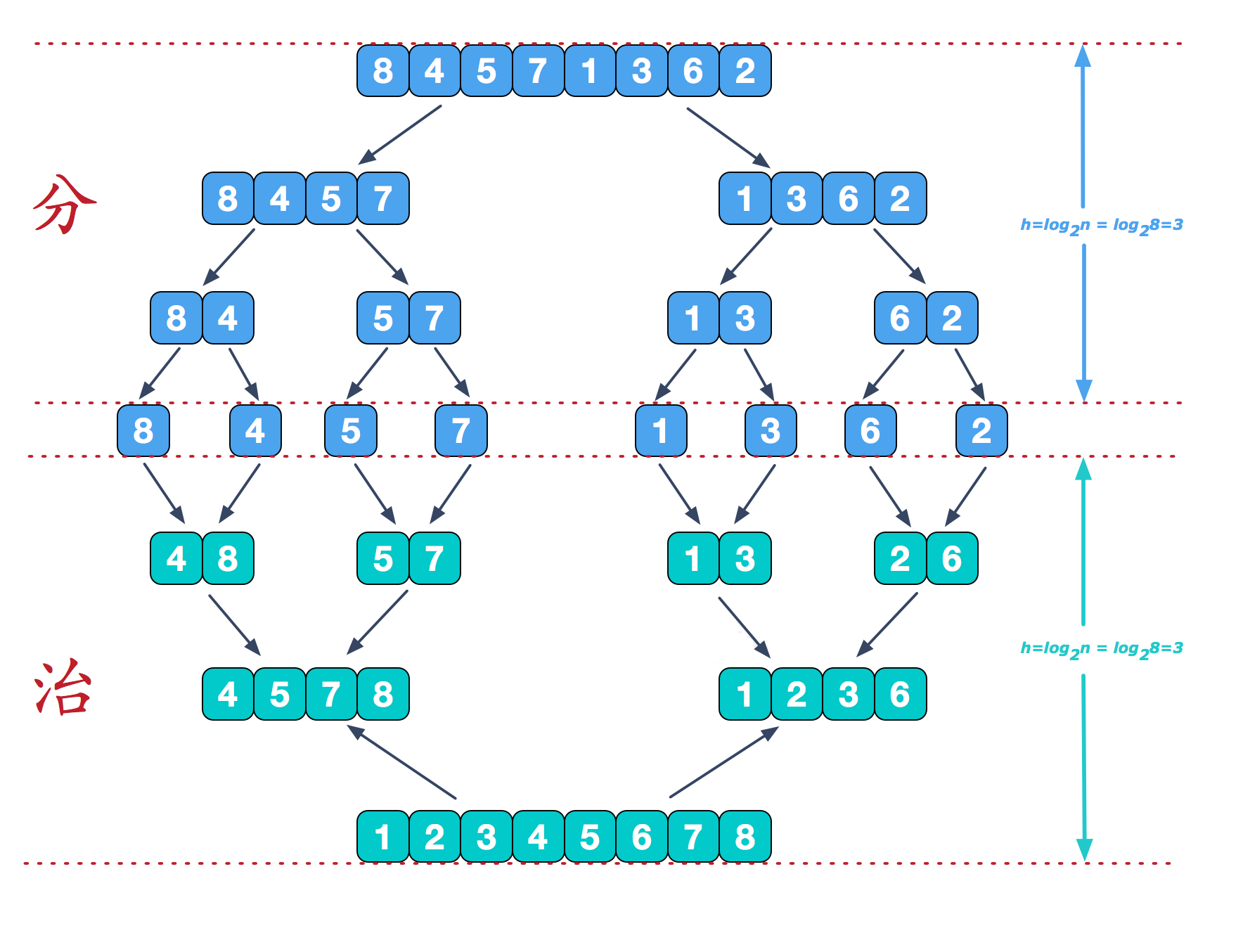
该算法采用经典的**分治**（divide-and-conquer）策略（分治法将问题**分**(divide)成一些小的问题然后递归求解，而**治(conquer)**的阶段则将分的阶段得到的各答案"修补"在一起，即分而治之)。

**动态效果示意图：**

[](https://cuijiahua.com/wp-content/uploads/2018/01/algorithm_7_0.gif)

**分而治之：**

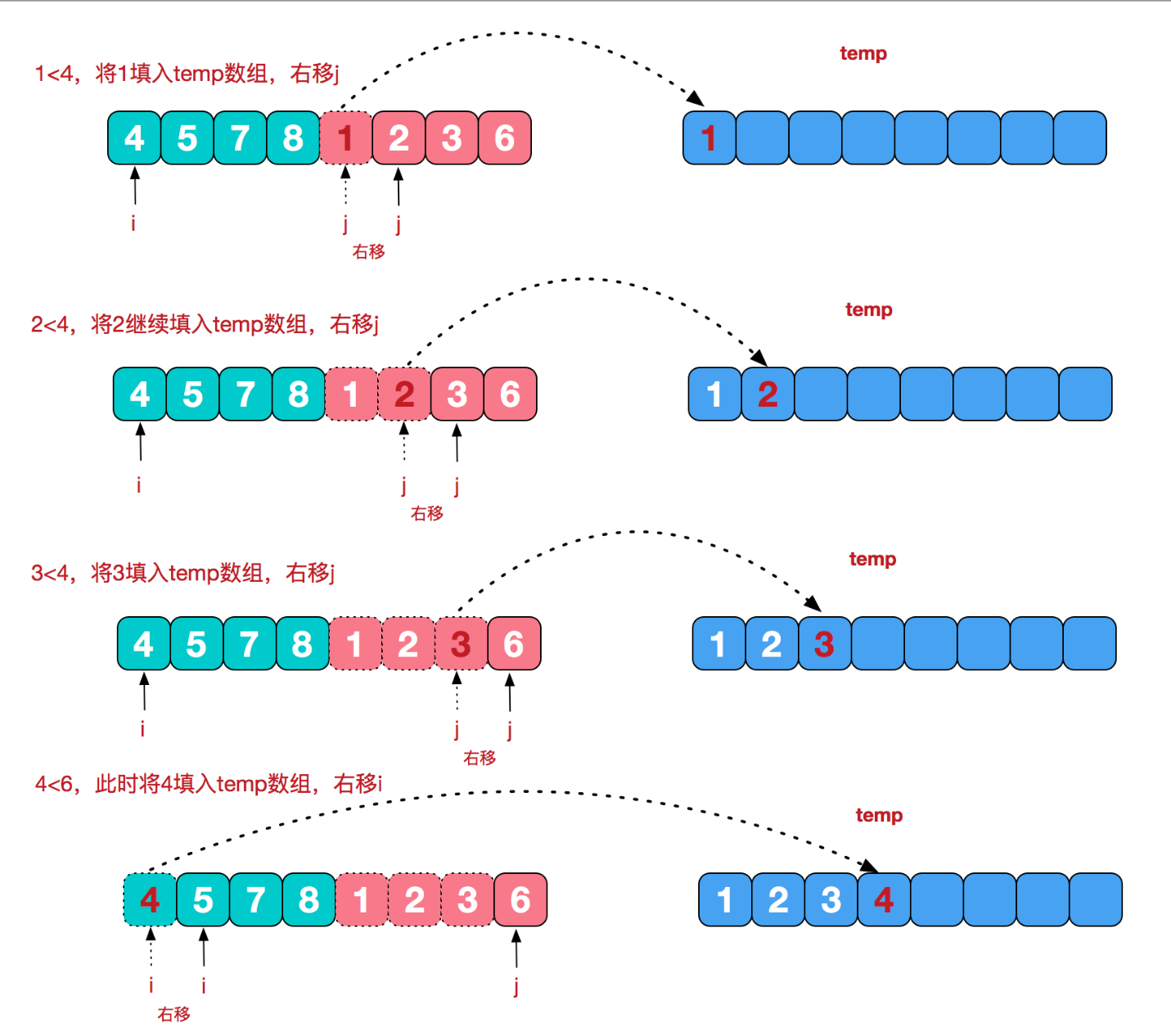
**1、分阶段**

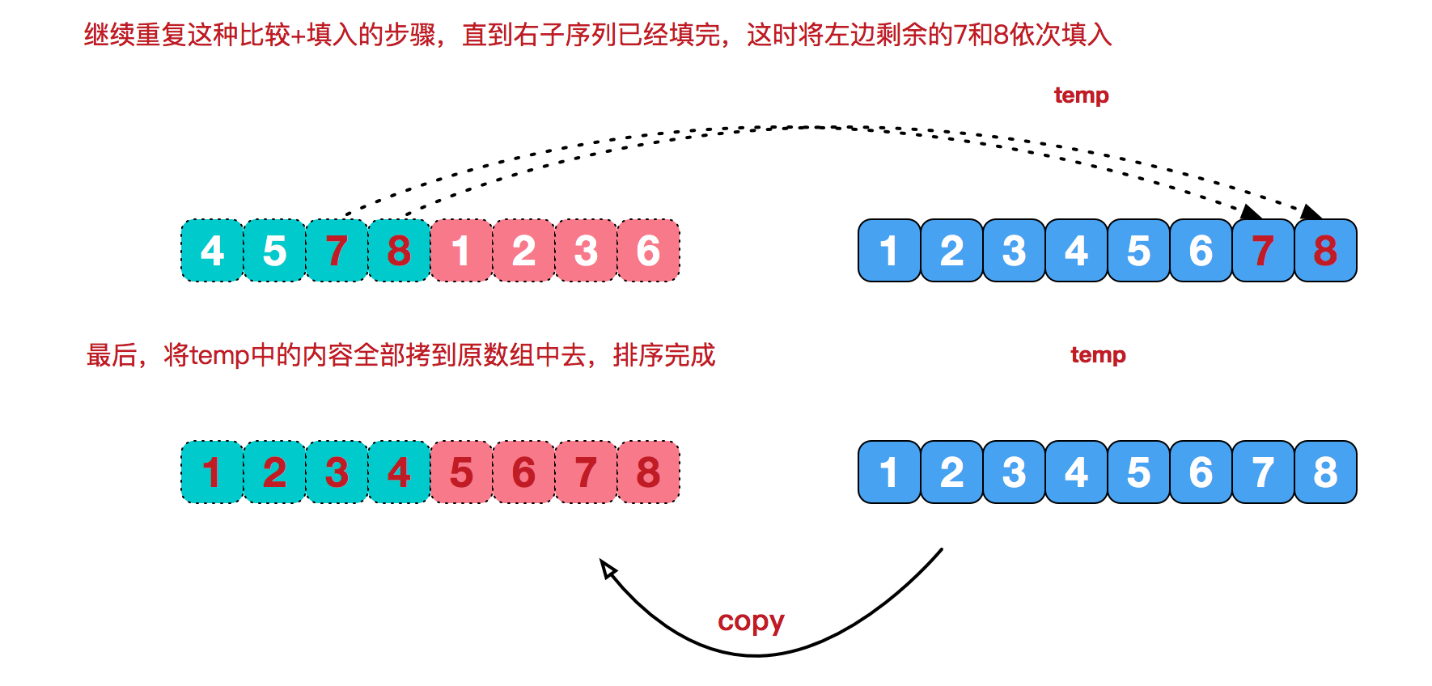
[](https://cuijiahua.com/wp-content/uploads/2018/01/algorithm_7_1.png)

可以看到这种结构很像一棵完全二叉树，本文的归并排序我们采用递归去实现（也可采用迭代的方式去实现）。**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logn。

**2、治阶段**

再来看看**治**阶段，我们需要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]，来看下实现步骤。

[](https://cuijiahua.com/wp-content/uploads/2018/01/algorithm_7_2.png)

[](https://cuijiahua.com/wp-content/uploads/2018/01/algorithm_7_3.png)

**3、代码**

C++：

C++



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | #include <iostream>  #include <vector>    using namespace std;    void Merge(vector<int> &input, int left, int mid, int right, vector<int> temp){      int i = left;                // i是第一段序列的下标      int j = mid + 1;            // j是第二段序列的下标      int k = 0;                    // k是临时存放合并序列的下标        // 扫描第一段和第二段序列，直到有一个扫描结束      while (i <= mid && j <= right){          // 判断第一段和第二段取出的数哪个更小，将其存入合并序列，并继续向下扫描          if (input[i] <= input[j]){              temp[k++] = input[i++];          }          else{              temp[k++] = input[j++];          }      }      // 若第一段序列还没扫描完，将其全部复制到合并序列      while (i <= mid){          temp[k++] = input[i++];      }        // 若第二段序列还没扫描完，将其全部复制到合并序列      while (j <= right){          temp[k++] = input[j++];      }        k = 0;      // 将合并序列复制到原始序列中      while (left <= right){          input[left++] = temp[k++];      }  }    void MergeSort(vector<int> &input, int left, int right, vector<int> temp){      if (left < right){          int mid = (right + left) >> 1;          MergeSort(input, left, mid, temp);          MergeSort(input, mid + 1, right, temp);          Merge(input, left, mid, right, temp);      }  }    void mergesort(vector<int> &input){      // 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      vector<int> temp(input.size());      MergeSort(input, 0, input.size() - 1, temp);  }    void main(){      int arr[] = { 6, 4, 8, 9, 2, 3, 1};      vector<int> test(arr, arr + sizeof(arr) / sizeof(arr[0]));      cout << "排序前:";      for (int i = 0; i < test.size(); i++){          cout << test[i] << " ";      }      cout << endl;        vector<int> result = test;      mergesort(result);      cout << "排序后:";      for (int i = 0; i < result.size(); i++){          cout << result[i] << " ";      }      cout << endl;      system("pause");  } |

运行结果如下图所示：

[排序（7）：归并排序](https://cuijiahua.com/wp-content/uploads/2018/01/algorithm_7_4.png)

可以看到已经可以看到归并排序算法顺利执行了。

Python：

Python

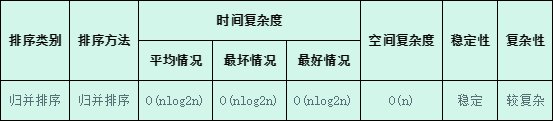


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | # -\*- coding:utf-8 -\*-    def MergeSort(input\_list):      '''      函数说明:归并排序（升序）      Author:          www.cuijiahua.com      Parameters:          input\_list - 待排序列表      Returns:          sorted\_list - 升序排序好的列表      '''      def merge(input\_list, left, mid, right, temp):          '''          函数说明:合并函数          Author:              www.cuijiahua.com          Parameters:              input\_list - 待合并列表              left - 左指针              right - 右指针              temp - 临时列表          Returns:              无          '''          i = left          j = mid + 1          k = 0            while i <= mid and j <= right:              if input\_list[i] <= input\_list[j]:                  temp[k] = input\_list[i]                  i += 1              else:                  temp[k] = input\_list[j]                  j += 1              k += 1            while i <= mid:              temp[k] = input\_list[i]              i += 1              k += 1          while j <= right:              temp[k] = input\_list[j]              j += 1              k += 1            k = 0          while left <= right:              input\_list[left] = temp[k]              left += 1              k += 1        def merge\_sort(input\_list, left, right, temp):          if left >= right:              return;          mid = (right + left) // 2          merge\_sort(input\_list, left, mid, temp)          merge\_sort(input\_list, mid + 1, right, temp)            merge(input\_list, left, mid, right, temp)        if len(input\_list) == 0:          return []      sorted\_list = input\_list      temp = [0] \* len(sorted\_list)      merge\_sort(sorted\_list, 0, len(sorted\_list) - 1, temp)      return sorted\_list    if \_\_name\_\_ == '\_\_main\_\_':      input\_list = [6, 4, 8, 9, 2, 3, 1]      print('排序前:', input\_list)      sorted\_list = MergeSort(input\_list)      print('排序后:', sorted\_list) |

运行效果同上。

**三、算法分析**

**1、归并排序算法的性能**

[](https://cuijiahua.com/wp-content/uploads/2018/01/algorithm_7_5.png)

其中，**log2n**为以2为底，n的对数。

**2、时间复杂度**

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\*log2n)**。

**3、空间复杂度**

由前面的算法说明可知，算法处理过程中，需要一个大小为**n**的临时存储空间用以保存合并序列。

**4、算法稳定性**

在归并排序中，相等的元素的顺序不会改变，所以它是**稳定的**算法。

**5、归并排序和堆排序、快速排序的比较**

若从空间复杂度来考虑：首选堆排序，其次是快速排序，最后是归并排序。

若从稳定性来考虑，应选取归并排序，因为堆排序和快速排序都是不稳定的。

若从平均情况下的排序速度考虑，应该选择快速排序。