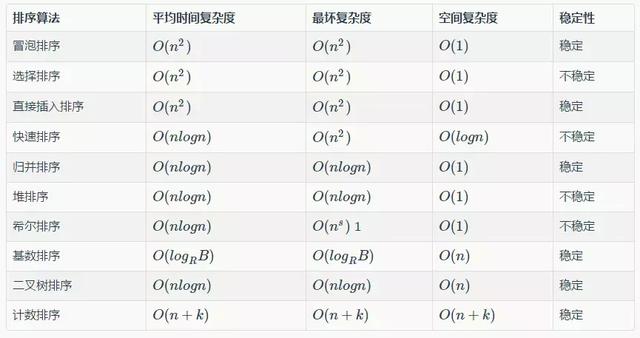
**各种排序算法总结**

排序算法是最基本最常用的算法，不同的排序算法在不同的场景或应用中会有不同的表现，我们需要对各种排序算法熟练才能将它们应用到实际当中，才能更好地发挥它们的优势。今天，来总结下各种排序算法。

下面这个表格总结了各种排序算法的复杂度与稳定性：



各种排序算法复杂度比较.png

# 冒泡排序

冒泡排序可谓是最经典的排序算法了，它是基于比较的排序算法，时间复杂度为O(n^2)，其优点是实现简单，n较小时性能较好。

* 算法原理
* 相邻的数据进行两两比较，小数放在前面，大数放在后面，这样一趟下来，最小的数就被排在了第一位，第二趟也是如此，如此类推，直到所有的数据排序完成
* c++代码实现
* void bubble\_sort(int arr[], int len)  
  {  
   for (int i = 0; i < len - 1; i++)  
   {  
   for (int j = len - 1; j > i; j--)  
   {  
   if (arr[j] < arr[j - 1])  
   {  
   int temp = arr[j];  
   arr[j] = arr[j - 1];  
   arr[j - 1] = temp;  
   }  
   }  
   }  
  }

**选择排序**

* 算法原理
* 先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。
* c++代码实现

void select\_sort(int arr[], int len)  
{  
 for (int i = 0; i < len; i++)  
 {  
 int index = i;  
 for (int j = i + 1; j < len; j++)  
 {  
 if (arr[j] < arr[index])  
 index = j;  
 }  
 if (index != i)  
 {  
 int temp = arr[i];  
 arr[i] = arr[index];  
 arr[index] = temp;   
 }  
 }  
}

**插入排序**

* 算法原理
* 将数据分为两部分，有序部分与无序部分，一开始有序部分包含第1个元素，依次将无序的元素插入到有序部分，直到所有元素有序。插入排序又分为直接插入排序、二分插入排序、链表插入等，这里只讨论直接插入排序。它是**稳定**的排序算法，时间复杂度为O(n^2)
* c++代码实现

void insert\_sort(int arr[], int len)  
{  
 for (int i = 1; i < len; i ++)  
 {  
 int j = i - 1;  
 int k = arr[i];  
 while (j > -1 && k < arr[j] )  
 {  
 arr[j + 1] = arr[j];  
 j --;  
 }  
 arr[j + 1] = k;  
 }  
}

**快速排序**

* 算法原理
* 快速排序是目前在实践中非常高效的一种排序算法，它不是稳定的排序算法，平均时间复杂度为O(nlogn)，最差情况下复杂度为O(n^2)。它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。
* c++代码实现

void quick\_sort(int arr[], int left, int right)  
{  
 if (left < right)  
 {  
 int i = left, j = right, target = arr[left];  
 while (i < j)  
 {  
 while (i < j && arr[j] > target)  
 j--;  
 if (i < j)  
 arr[i++] = arr[j];  
 while (i < j && arr[i] < target)  
 i++;  
 if (i < j)  
 arr[j] = arr[i];  
 }  
 arr[i] = target;  
 quick\_sort(arr, left, i - 1);  
 quick\_sort(arr, i + 1, right);  
 }  
}

**归并排序**

* 算法原理
* 归并排序具体工作原理如下（假设序列共有n个元素）：
* 将序列每相邻两个数字进行归并操作（merge)，形成floor(n/2)个序列，排序后每个序列包含两个元素
* 将上述序列再次归并，形成floor(n/4)个序列，每个序列包含四个元素
* 重复步骤2，直到所有元素排序完毕
* 归并排序是稳定的排序算法，其时间复杂度为O(nlogn)，如果是使用链表的实现的话，空间复杂度可以达到O(1)，但如果是使用数组来存储数据的话，在归并的过程中，需要临时空间来存储归并好的数据，所以空间复杂度为O(n)
* c++代码实现

void merge(int arr[], int temp\_arr[], int start\_index, int mid\_index, int end\_index)  
{  
 int i = start\_index, j = mid\_index + 1;  
 int k = 0;  
 while (i < mid\_index + 1 && j < end\_index + 1)  
 {  
 if (arr[i] > arr[j])  
 temp\_arr[k++] = arr[j++];  
 else  
 temp\_arr[k++] = arr[i++];  
 }  
 while (i < mid\_index + 1)  
 {  
 temp\_arr[k++] = arr[i++];  
 }  
 while (j < end\_index + 1)  
 temp\_arr[k++] = arr[j++];  
 for (i = 0, j = start\_index; j < end\_index + 1; i ++, j ++)  
 arr[j] = temp\_arr[i];  
}  
void merge\_sort(int arr[], int temp\_arr[], int start\_index, int end\_index)  
{  
 if (start\_index < end\_index)  
 {  
 int mid\_index = (start\_index + end\_index) / 2;  
 merge\_sort(arr, temp\_arr, start\_index, mid\_index);  
 merge\_sort(arr, temp\_arr, mid\_index + 1, end\_index);  
 merge(arr, temp\_arr, start\_index, mid\_index, end\_index);  
 }  
}

**堆排序**

二叉堆

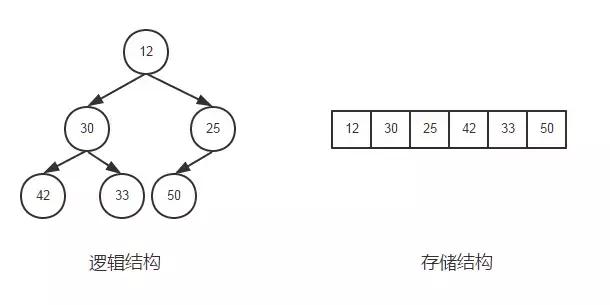
二叉堆是完全二叉树或者近似完全二叉树，满足两个特性

* 父结点的键值总是大于或等于(小于或等于)任何一个子节点的键值
* 每个结点的左子树和右子树都是一个二叉堆

当父结点的键值总是大于或等于任何一个子节点的键值时为最大堆。当父结点的键值总是小于或等于任何一个子节点的键值时为最小堆。一般二叉树简称为堆。

堆的存储

一般都是数组来存储堆，i结点的父结点下标就为(i – 1) / 2。它的左右子结点下标分别为2 \* i + 1和2 \* i + 2。如第0个结点左右子结点下标分别为1和2。存储结构如图所示：



堆结构.png

堆排序原理

堆排序的时间复杂度为O(nlogn)

* 算法原理（以最大堆为例）
* 先将初始数据R[1..n]建成一个最大堆，此堆为初始的无序区
* 再将关键字最大的记录R[1]（即堆顶）和无序区的最后一个记录R[n]交换，由此得到新的无序区R[1..n-1]和有序区R[n]，且满足R[1..n-1].keys≤R[n].key
* 由于交换后新的根R[1]可能违反堆性质，故应将当前无序区R[1..n-1]调整为堆。
* 重复2、3步骤，直到无序区只有一个元素为止。
* c++代码实现

/\*\*  
 \* 将数组arr构建大根堆  
 \* @param arr 待调整的数组  
 \* @param i 待调整的数组元素的下标  
 \* @param len 数组的长度  
 \*/  
void heap\_adjust(int arr[], int i, int len)  
{  
 int child;  
 int temp;  
 for (; 2 \* i + 1 < len; i = child)  
 {  
 child = 2 \* i + 1; // 子结点的位置 = 2 \* 父结点的位置 + 1  
 // 得到子结点中键值较大的结点  
 if (child < len - 1 && arr[child + 1] > arr[child])  
 child ++;  
 // 如果较大的子结点大于父结点那么把较大的子结点往上移动，替换它的父结点  
 if (arr[i] < arr[child])  
 {  
 temp = arr[i];  
 arr[i] = arr[child];  
 arr[child] = temp;  
 }  
 else  
 break;  
 }  
}  
/\*\*  
 \* 堆排序算法  
 \*/  
void heap\_sort(int arr[], int len)  
{  
 int i;  
 // 调整序列的前半部分元素，调整完之后第一个元素是序列的最大的元素  
 for (int i = len / 2 - 1; i >= 0; i--)  
 {  
 heap\_adjust(arr, i, len);  
 }  
 for (i = len - 1; i > 0; i--)  
 {  
 // 将第1个元素与当前最后一个元素交换，保证当前的最后一个位置的元素都是现在的这个序列中最大的  
 int temp = arr[0];  
 arr[0] = arr[i];  
 arr[i] = temp;  
 // 不断缩小调整heap的范围，每一次调整完毕保证第一个元素是当前序列的最大值  
 heap\_adjust(arr, 0, i);  
 }  
}

**涨姿势，图文带你了解 8 大排序算法**

程序员圣经 2019-07-22 20:00:00

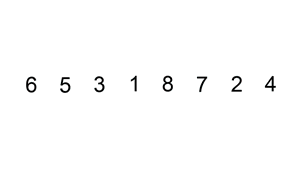
作者：Jay13  
来源：www.cricode.com/3212.html

排序算法可以分为内部排序和外部排序，内部排序是数据记录在内存中进行排序，而外部排序是因排序的数据很大，一次不能容纳全部的排序记录，在排序过程中需要访问外存。

常见的内部排序算法有：插入排序、希尔排序、选择排序、冒泡排序、归并排序、快速排序、堆排序、基数排序等。

本文将依次介绍上述八大排序算法。

算法一：插入排序



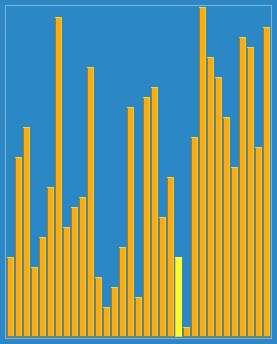
插入排序是一种最简单直观的排序算法，它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

**算法步骤**：

1）将第一待排序序列第一个元素看做一个有序序列，把第二个元素到最后一个元素当成是未排序序列。

2）从头到尾依次扫描未排序序列，将扫描到的每个元素插入有序序列的适当位置。（如果待插入的元素与有序序列中的某个元素相等，则将待插入元素插入到相等元素的后面。）

**算法二：希尔排序**



希尔排序，也称递减增量排序算法，是插入排序的一种更高效的改进版本。但希尔排序是非稳定排序算法。

希尔排序是基于插入排序的以下两点性质而提出改进方法的：

* 插入排序在对几乎已经排好序的数据操作时， 效率高， 即可以达到线性排序的效率
* 但插入排序一般来说是低效的， 因为插入排序每次只能将数据移动一位

希尔排序的基本思想是：先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行依次直接插入排序。

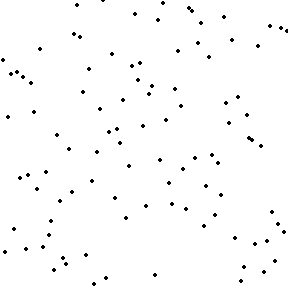
**算法步骤**：

1）选择一个增量序列t1，t2，…，tk，其中ti>tj，tk=1；

2）按增量序列个数k，对序列进行k 趟排序；

3）每趟排序，根据对应的增量ti，将待排序列分割成若干长度为m 的子序列，分别对各子表进行直接插入排序。仅增量因子为1 时，整个序列作为一个表来处理，表长度即为整个序列的长度。

算法三：选择排序



选择排序(Selection sort)也是一种简单直观的排序算法。

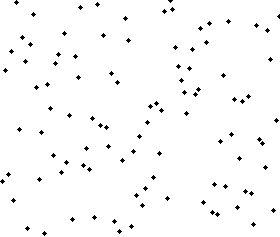
**算法步骤**：

1）首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置

2）再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。

3）重复第二步，直到所有元素均排序完毕。

**算法四：冒泡排序**



冒泡排序（Bubble Sort）也是一种简单直观的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。

**算法步骤**：

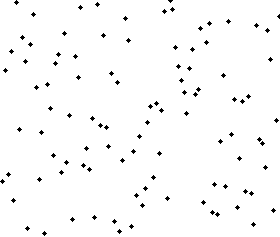
1）比较相邻的元素。如果第一个比第二个大，就交换他们两个。

2）对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。

3）针对所有的元素重复以上的步骤，除了最后一个。

4）持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

算法五：归并排序



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。

**算法步骤**：

1. 申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列

2. 设定两个指针，最初位置分别为两个已经排序序列的起始位置

3. 比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置

4. 重复步骤3直到某一指针达到序列尾

5. 将另一序列剩下的所有元素直接复制到合并序列尾

算法六：快速排序



快速排序是由东尼·霍尔所发展的一种排序算法。在平均状况下，排序 n 个项目要Ο(n log n)次比较。在最坏状况下则需要Ο(n2)次比较，但这种状况并不常见。事实上，快速排序通常明显比其他Ο(n log n) 算法更快，因为它的内部循环（inner loop）可以在大部分的架构上很有效率地被实现出来。

快速排序使用分治法（Divide and conquer）策略来把一个串行（list）分为两个子串行（sub-lists）。

**算法步骤**：

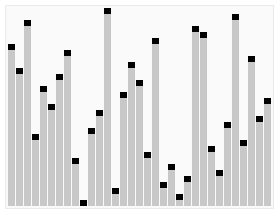
1 从数列中挑出一个元素，称为 “基准”（pivot），

2 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。

3 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

递归的最底部情形，是数列的大小是零或一，也就是永远都已经被排序好了。虽然一直递归下去，但是这个算法总会退出，因为在每次的迭代（iteration）中，它至少会把一个元素摆到它最后的位置去。

**算法七：堆排序**



堆排序（Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆积是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

堆排序的平均时间复杂度为Ο(nlogn) 。

**算法步骤**：

1）创建一个堆H[0..n-1]

2）把堆首（最大值）和堆尾互换

3）把堆的尺寸缩小1，并调用shift\_down(0)，目的是把新的数组顶端数据调整到相应位置

4） 重复步骤2，直到堆的尺寸为1

**算法八：基数排序**

基数排序是一种非比较型整数排序算法，其原理是将整数按位数切割成不同的数字，然后按每个位数分别比较。由于整数也可以表达字符串（比如名字或日期）和特定格式的浮点数，所以基数排序也不是只能使用于整数。

说基数排序之前，我们简单介绍桶排序：

算法思想：是将阵列分到有限数量的桶子里。每个桶子再个别排序（有可能再使用别的排序算法或是以递回方式继续使用桶排序进行排序）。桶排序是鸽巢排序的一种归纳结果。当要被排序的阵列内的数值是均匀分配的时候，桶排序使用线性时间（Θ（n））。但桶排序并不是 比较排序，他不受到 O(n log n) 下限的影响。

简单来说，就是把数据分组，放在一个个的桶中，然后对每个桶里面的在进行排序。

例如要对大小为[1..1000]范围内的n个整数A[1..n]排序

首先，可以把桶设为大小为10的范围，具体而言，设集合B[1]存储[1..10]的整数，集合B[2]存储 (10..20]的整数，……集合B[i]存储( (i-1)\*10， i\*10]的整数，i = 1，2，..100。总共有 100个桶。

然后，对A[1..n]从头到尾扫描一遍，把每个A[i]放入对应的桶B[j]中。 再对这100个桶中每个桶里的数字排序，这时可用冒泡，选择，乃至快排，一般来说任 何排序法都可以。

最后，依次输出每个桶里面的数字，且每个桶中的数字从小到大输出，这 样就得到所有数字排好序的一个序列了。

假设有n个数字，有m个桶，如果数字是平均分布的，则每个桶里面平均有n/m个数字。如果

对每个桶中的数字采用快速排序，那么整个算法的复杂度是

O(n + m \* n/m\*log(n/m)) = O(n + nlogn – nlogm)

从上式看出，当m接近n的时候，桶排序复杂度接近O(n)

当然，以上复杂度的计算是基于输入的n个数字是平均分布这个假设的。这个假设是很强的 ，实际应用中效果并没有这么好。如果所有的数字都落在同一个桶中，那就退化成一般的排序了。

前面说的几大排序算法 ，大部分时间复杂度都是O（n2），也有部分排序算法时间复杂度是O(nlogn)。而桶式排序却能实现O（n）的时间复杂度。但桶排序的缺点是：

1）首先是空间复杂度比较高，需要的额外开销大。排序有两个数组的空间开销，一个存放待排序数组，一个就是所谓的桶，比如待排序值是从0到m-1，那就需要m个桶，这个桶数组就要至少m个空间。

2）其次待排序的元素都要在一定的范围内等等。

总结

各种排序的稳定性，时间复杂度、空间复杂度、稳定性总结如下图：



**关于时间复杂度：**

(1)平方阶(O(n2))排序

各类简单排序：直接插入、直接选择和冒泡排序；

(2)线性对数阶(O(nlog2n))排序

快速排序、堆排序和归并排序；

(3)O(n1+§))排序，§是介于0和1之间的常数。

希尔排序

(4)线性阶(O(n))排序

基数排序，此外还有桶、箱排序。

关于稳定性：

稳定的排序算法：冒泡排序、插入排序、归并排序和基数排序

不是稳定的排序算法：选择排序、快速排序、希尔排序、堆排序