	算法	最好序列	最坏序列	平均时间	空间	稳定
交换	冒泡	有序 O(n)	逆序 $O(n^2)$	$O(n^2)$	O(1)	✔
	快速(首元素为基准) 每趟找一个基准，low，high 移动	基准元素为均匀划分	有序	$O(n\log_2 n)$	最好/最坏 $O(\log_2 n)$ O(n)	×
插入	直接插入	有序 O(n)	逆序 $O(n^2)$	$O(n^2)$	O(1)	✔
	折半插入	比较次数减少 移动次数不变	比较次数减少 移动次数不变	$O(n^2)$	O(1)	✔
	希尔 给增量使其先部分有序 再全局有序	顺序（取决于增量 d）	逆序（取决于增量 d）	$O(n^{1.3\sim2})$	O(1)	×
选择	简单选择 每一趟找最小的加入序列	有序 移动少,比较次数 (n-1)n / 2	逆序 移动多,比较次数 (n-1)n /2	$O(n^2)$	O(1)	×
	锦标赛	/	/	$O(n\log_2 n)$	O(n)	×
	堆 建最大（增）或最小堆	建堆 O(n) 排序 $O(n\log_2 n)$	建堆比较 4n 次 排序 $O(n\log_2 n)$	$O(n\log_2 n)$	O(1)	×
归并	两路归并	有序,每趟 n 归并趟数 $\log_2 n$	逆序，每趟 n 归并趟数 $\log_2 n$	$O(n\log_2 n)$	O(n)	✔
	递归归并	有序,每趟 n 归并趟数 $\log_2 n$	逆序，每趟 n 归并趟数 $\log_2 n$	$O(n\log_2 n)$	O(n) 递归为 $O(\log_2 n)$	✔
基数	基数排序 (分配+收集)	- 关键字可拆分为 d 元组 - r 较小 - **n 较大**	n 较小，d,r 较大	O(d(n+r)) d 趟,每趟 n 次分配,r 次收集	O(n+2r) n 表示队列中开辟空间，r 表示一个队列的两个指针	✔

基于比较的，至少比较 $\log_2(n!)$向上取整次

希尔排序：

追求元素部分有序再逼近全局有序

d1 = n/2, d2 = d1/2.....

快速排序：

找一个元素作为基准，一趟后找到它的最终位置。Low 和 high 移动，补空位。

时间复杂度：O(n*递归深度)，空间复杂度：O(递归深度)

最深 n，最浅 $\log_2(n+1)$向上取整

基数排序：

1. 不是基于比较的

排序算法选择：

1. 排序规模
2. 关键字分布
3. 稳定性要求

- 规模较小(<50)，直接插入或选择。前者比后者移动次数多，故选择优先
- 规模较大：
    - 关键字分布随机：快速排序、堆（不会出现快排最坏情况）
    - 接近有序：希尔（若不有序，移动次数多）
    - 需要稳定：先用插入排序得到<50 的有序段再归并。
- 基本有序：插入类或冒泡
- 规模很大但关键字位数少，基数排序

表 8-2 各种方法解决冲突时散列表的平均查找长度

解决冲突的方法	查找成功的平均查找长度 S_n	查找不成功的平均查找长度 U_n
线性探测法	$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$	$\frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$
二次探测法	$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)}\right)$	$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)}\right)$
双散列法	$-\frac{1}{2}\log_e(1-\alpha)$	$\frac{1}{(1-\alpha)}$
链地址法	$1+\frac{\alpha}{2}$	$\alpha+e^{-\alpha}$

习 题 八

又达到平衡。结果如图 8-15（c）所示。

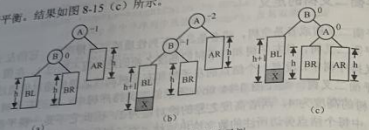

图 8-15 LL 平衡旋转的示例

2. RR 平衡旋转

RR 旋转和 LL 旋转对称。如因为在 A 的右孩子 B 的右子树上插入新结点，使 A 的平衡因子由 1 变成 2，则需进行 RR 平衡旋转。图 8-16 是 RR 平衡旋转的例子。为使树恢复平衡，从 A 沿刚才的插入路径连续取两个结点 A 和 B，以结点 B 为旋转轴，将结点 A 逆时针向下旋转成为 B 的左孩子，结点 B 代替原来结点 A 的位置，结点 B 原来的左孩子成为结点 A 的右孩子，从而使二叉排序树又达到平衡。

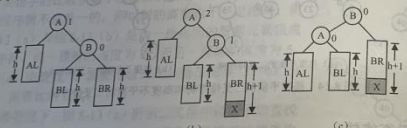

图 8-16 RR 平衡旋转的示例

3. LR 平衡旋转

如果是因为在 A 的左孩子 B 的右子树上插入新结点，使 A 的平衡因子由-1 变成-2，则需要进行 LR 平衡旋转，如图 8-17 所示是 LR 平衡旋转的例子。为使二叉排序树恢复平衡，则需要进行先逆时针后顺时针的平衡旋转，即先将 A 的左孩子 B 的右孩子 C 向逆

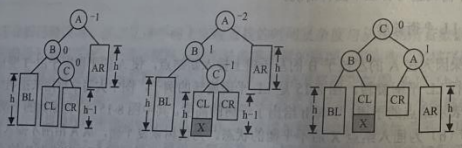

图 8-17 LR 平衡旋转的示例

时针方向旋转代替 B 的位置，再以结点 C 为旋转轴，将结点 A 向顺时针方向旋转成为 C 的右孩子，结点 C 代替原来结点 A 的位置，结点 C 原来的左孩子转为结点 B 的右孩子，结点 C 原来的右孩子转为结点 A 的左孩子，从而使二叉排序树又达到平衡。

4. RL 平衡旋转

如果是因为在 A 的右孩子 B 的左子树上插入新结点，使 A 的平衡因子由 1 变成 2，则需要进行 RL 平衡旋转，图 8-18 是 RL 平衡旋转的例子。为使树恢复平衡，则需要进行先顺时针后逆时针的平衡旋转，即先将 A 的右孩子 B 的左孩子 C 向顺时针方向旋转代替 B 的位置，再以结点 C 为旋转轴，将结点 A 向逆时针方向旋转成为 C 的左孩子，结点 C 代替原来结点 A 的位置，结点 C 原来的左孩子转为结点 A 的右孩子，结点 C 原来的右孩子转为结点 B 的左孩子，从而使二叉排序树又达到平衡。

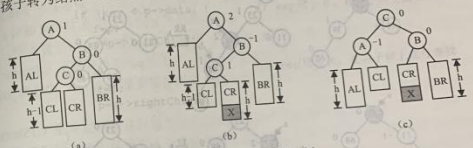

图 8-18 RL 平衡旋转的示例

8.5.3 平衡二叉树中插入结点

在平衡二叉树中插入一个新结点后，如果二叉排序树中某个结点的平衡因子的绝对值|bf|>1，则出现了不平衡，这时需要根据平衡旋转的类型立即进行平衡化处理，使得二叉排序树中各结点重新平衡。平衡二叉树插入结点的算法思想如下。

1）按二叉排序树的性质插入结点。

2）如果插入结点之后出现不平衡的结点，则转步骤 3）；否则插入完成。

3）找到失去平衡的最小子树。

4）判断平衡旋转的类型作相应平衡化处理。

在此算法中，步骤 1）是比较简单的。但如何发现插入后出现不平衡的结点呢？由平衡二叉树的定义可知，在插入之后如果树上出现平衡因子绝对值大于 1 的结点，则说明二叉排序树已失去平衡。这时失去平衡的最小子树的根结点必为离插入结点最近，而且插入之前平衡因子绝对值为 1 的结点。为此，要解决上述三个问题可以作如下处理。

1）在查找结点 x 的插入位置的过程中，记下从根结点到插入位置的路径上离插入位置最近的且平衡因子绝对值为 1 的结点，并令指针 a 指向该结点；如果此路径上所有结点的平衡因子绝对值为 1 的结点，则指针 a 指向根结点。

2）对于从 a 结点到 x 结点的路径上的每一个结点（不包括结点 x），根据结