排序的概念

排序：将一组任意排列的数据元素序列，重新排列成一个按关键字有序（升序或降序）的序列。
如果参加排序的数据结点包含多个数据域，那么排序往往针对其中某个域而言

按待排序记录所在位置：

- 内部排序：待排序记录放在内存
- 外部排序：排序过程中需对外存进行访问的排序

按排序依据规则：

- 插入排序：直接插入排序、折半插入排序、希尔排序
- 交换排序冒泡排序、快速排序
- 选择排序：简单选择排序、树型选择排序、堆排序
- 归并排序：2-路归并排序
- 基数排序

待排序的数据类型：

```C++
#define MAXSIZE 20
typedef int KeyType;
typedef struct {
   KeyType key;
   InfoType otherinfo;
} RcdType;
typedef struct {
   RcdType r[MAXSIZE + 1]; // r[0]作为哨兵
   int length;
} SqList;
```

插入排序

基本思想：每步都将一个待排序的对象，按其关键字的大小，插入到前面已经排序好的一组对象的适当位置上，知道对象全部插入为止。

即：边插入边排序，保证子序列中随时都是排好序的

基本操作：有序插入。在有序序列中插入一个元素，保持序列有序，有序长度不断增加。

FxMarkBrown's Blog

直接插入排序

过程：

1. 复制插入元素 x=a[i]
2. 记录后移，查找插入位置
   for(j = i-1;j >= 0 && x < a[j]; --j) a[j+1] = a[j]
3. 插入到正确位置 a[j+1] = x

或者利用哨兵

1. 复制为哨兵：L.r[0] = L.r[i]
2. 记录后移，查找插入位置
   for(j = i-1; L.r[0].key < L.r[j].key; --j) L.r[j+1] = L.r[j]
3. 插入到正确位置 L.r[j+1] = L.r[0]

```C++
void InsertSort(SqList &L) {
    for(i = 2; i < L.length; ++i) {
        if(LT(L.r[i].key, L.r[i-1].key)) {
            L.r[0] = L.r[i];
            L.r[i] = L.r[i-1];
            // 后移
            for(j = i - 2; LT(L.r[0].key, L.r[j].key); --j) {
                L.r[j+1] = L.r[j];
            }
            L.r[j+1] = L.r[0];
        }
    }
}
```

最好的情况（关键字在记录序列中顺序有序）：
比较次数：从第二个元素开始都和前面一个元素比较一次 $n-1 \approx n$
移动次数：0

最坏的情况（关键字在记录序列中逆序有序）：
比较次数：从第二个元素开始都和前面所有元素且和哨兵比较一次：$\sum_{i=1}^{n} i \approx \frac{n^2}{2}$
移动次数：$\sum_{i=2}^{n}(i+1) \approx \frac{n^2}{2}$

平均的情况：
比较次数：$\sum_{i=1}^{n} -1\frac{i+1}{2} \approx \frac{n^2}{2}$
移动次数：$\sum_{i=2}^{n}(\frac{i+1}{2}+1) \approx \frac{n^2}{2}$

时间复杂度$O(n^2)$，空间复杂度$O(1)$
是一种稳定的插入排序

FxMarkBrown's Blog

二分插入排序

```C++
void BInsertSort(SqList &L) {
    for(i = 2; i < L.length; ++i) {
        L.r[0] = L.r[i];
        low = 1; high = i - 1;
        while(low <= high) {
            m = (low + high) / 2;
            if(LT(L.r[0].key, L.r[m].key)) high = m-1;
            else low = m + 1;
        }

        for(j = i - 1; j >= high+ 1; --j) {
            L.r[j+1] = L.r[j];
        }
        L.r[high + 1] = L.r[0];
    }
```

2-路插入排序

为了减少移动次数，增加n个辅助空间（数组d）
空间复杂度：$O(n)$

希尔排序

先将整个待记录序列分割成若干子序列，分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行一次直接插入排序

特点：缩小增量，多遍插入排序

FxMarkBrown's Blog

```C++
void ShellSort(SqList &L, int dlta[], int t) {
    // t 排序趟数
    // dlta 是增量序列
    for(k = 0; k < t; ++k) ShellInsert(L, dlta[k]);
}

void ShellInsert(SqList &L, int dk) {
    for(i = dk + 1; i <= L.length; ++i) {
        if (LT(L.r[i].key, L.r[i - dk].key)) {
            L.r[0] = L.r[i];
            for(j = i - dk; j > 0 && LT(L.r[0].key, L.r[j].key); j -= dk) {
                L.r[j + dk] = L.r[j];
            }

            L.r[j + dk] = L.r[0];
        }
    }
}
```

比较次数和移动次数约我$n^{1.}$，属于不稳定排序

快速排序

冒泡排序

```C++
void bubbleSort(SqList &L) {
    for(i = 1; i <= L.length - 1; ++i) {
        flag = false;
        for(j = 1; j <= L.length - i; j++) {
            if(L.r[j].key > L.r[j + 1].key) {
                L.r[j] <-> L.r[j+1];
                flag = true;
            }
        }
        if(!flag) break;
    }
}
```

最好情况（正序）：$O(n)$

- 比较次数：n-1
- 移动次数：0

最坏情况（逆序）：$O(n^2)$

FxMarkBrown's Blog

- 比较次数：$\sum_{i=1}^{n-1}(n-i)=\frac{(n^2-n)}{2}$
- 移动次数：$\sum_{i=1}^{n-i}=\frac{}{2}(n^2-n)$
- 平均情况：$O(n^2)$

快排

基本思想：通过一趟排序，将待排序记录分割成两个独立的两部分，其中一部分记录的关键字均比另一部分的关键字小，则可分别对这两部分记录进行排序，以达到整个序列有序

具体：选定一个中间数作为参考，所有元素与之比较，小的调到其左边，大的调到其右边

（枢轴）中间数：可以是第一个数、最后一个数、最中间一个数、任选一个数

```C++
void QuickSort(SqList &L) {
    QSort(L, 1, L.length);
}

void QSort(SqList &L, int low, int high) {
    if(low < high) {
        pivotloc = Partition(L, low, high);
        QSort(L, low, pivotloc - 1);
        QSort(L, pivotloc + 1, high);
    }
}

int Partition(SqList &L, int low, int high) {
    L.r[0] = L.r[low];
    pivotkey = L.r[low].key;

    int i = low - 1, j = high + 1;
    while(1) {
        do  { ++i; } while (L.r[i] < pivotkey);
        do { ++j; } while (L.r[j] > pivotkey);

        if (i == j) return j;
        else L.r[i] <-> L.r[j];
    }
}
```

平均时间复杂度：$O(n\log_2 n)$，有序情况下最坏，时间复杂度$O(n^2)$
空间：递归过程，需要使用栈，栈长取决于递归调用深度，平均需要$O(\log n)$的栈空间，最坏情况下$O(n)$
不稳定排序，例如551，枢轴量选5

FxMarkBrown's Blog

选择排序

每一趟在$n-i+1(i=1,2,\cdots,n-1)$个记录中选择关键字最小的记录作为有序序列中第i个记录

```
void SelectSort(SqList &L) {
    for(i = 1; i < L.length; ++i) {
        j = SelectMinKey(L, i);
        if (i != j) L.r[i] <-> L.r[j];
    }
}
```

时间复杂度（最好/平均/最坏）：$O(n^2)$
空间复杂度：$O(1)$
不稳定排序

树形选择排序（锦标赛排序）

时间复杂度：$O(n\log_2 n)$
空间复杂度：$O(n)$
不稳定排序

堆排

堆的定义：完全二叉树，任一非叶子结点均小于（或大于）它的孩子结点（小顶堆/大顶堆）

堆排：将无序序列建成一个堆，得到关键字最小（或最大）的记录，输出堆顶的最小（大）值后，使剩余的$n-1$个元素又重建成一个堆，则可得到n个元素的次小值；重读执行，得到一个有序序列，这个过程叫堆排

如何在输出堆顶元素后，调整剩余元素为一个新的堆
小顶堆：

- 输出堆顶元素后，以堆中最后一个元素替代之
- 然后将根结点值与左右子树的根结点值进行比较，并与其中小者进行交换
- 重复上述操作直到叶子结点，得到新的堆，称这个从堆顶至叶子的调整过程为“筛选”

堆的建立：
显然：

- 单结点的二叉树是堆
- 在完全二叉树中所有以叶子结点（序号$i>\frac{n}{2}$）为根的子树是堆
- 这样，我们只需要依次将以序号$\frac{n}{2},\frac{n}{2}-1,\cdots,1$的结点为根的子树调整为堆即可

方法：

FxMarkBrown's Blog

- 从无序序列的第$floor(\frac{n}{2})$个元素（即此无序序列对应的完全二叉树的最后一个非终端结点）起，至第一个元素止，反复筛选

```C++
void HeapSort(HeapType &H) {
    for(int i = H.length / 2; i > 0; --i) {
        HeapAdjust(H, i, H.length);
    }

    for(i = H.length; i > 1; --i) {
        H.r[1] <-> H.r[i];
        HeadpAdjust(H, 1, i -1);
    }
}

typedef SqList HeapType;
void HeapAdjust(HeapType &H, int s, int m) {
    rc = H.r[s];
    for (j = 2*s; j <= m; j *= 2) {
        if (j < m && LT(H.r[j].key, H.r[j + 1].key)) ++j;
        if (!LT(rc.key, H.r[j].key)) break;
        H.r[s] = H.r[j]; s = j;
    }
    H.r[s] = rc;
}
```

时间复杂度：$O(n\log n)$
空间复杂度：$O(1)$

归并排序

将两个或两个以上的有序表组合成一个新的有序表

FxMarkBrown's Blog

```C++
void Merge(RcdType SR[], RcdType &TR[], int i, int m, int n) {
    for(j = m + 1, k = i; i <= m && j <= n; ++k) {
        if (LQ(SR[i].key, SR[j].key)) TR[k] = SR[i++];
        else TR[k] = SR[j++];
    }

    if(i <= m) TR[k..n] = SR[i..m];
    if(j <= n) TR[k..n] = SR[j..n];
}

void MSort(RcType SR[], RcType &TR1[], int s, int t) {
    if (s == t) TR1[s] = SR[s];
    else
        m = (s+t)/2;
        MSort(SR, TR2, s, m);
        MSort(SR, TR2, m + 1, t);
        Merge(TR2, TR1, s, m ,t);
    }

void MergeSort(SqList &L) {
    MSort(L.r, L.r, 1, L.length);
}
```

时间复杂度：$O(n \log n)$

各种排序方法的综合比较

时间性能

1. 按平均的时间性能来分，有三类排序方法
   - 时间复杂度为O(nlogn)的方法有：
     快速排序、堆排序和归并排序，其中以快速排序为最好
   - 时间复杂度为O (n2)的有：
     直接插入排序，冒泡排序和简单选择排序，其中以直接插入为最好，特别是对那些关键字近似有序的序列尤为如此
   - 时间复杂度为O(n)的排序方法只有：基数排序
2. 当待排序列按关键字顺序有序时，
   直接插入排序和冒泡排序能达到O(n)的时间复杂度
   对于快速排序而言，此时时间性能退化到O(n2)
3. 简单选择排序、堆排序和归并排序的时间性能不能随序列中关键字的分布而改变
   空间性能
4. 所有的简单排序方法（包括：直接插入、冒泡和简单选择）和希尔排序、堆排序的空间复杂度为O(1)

FxMarkBrown's Blog

5. 快速排序为O(logn)，为栈所需的辅助空间
6. 归并排序所需辅助空间最多，其空间复杂度为O(n)
7. 链式基数排序需附设队列首尾指针，空间复杂度为O(rd)
   稳定性能
8. 稳定的排序方法是指：对于两个关键字相等的记录，它们在序列中的相对位置，在排序之前和经过排序之后，没有改变
9. 直接插入排序、冒泡排序、归并排序和基数排序是稳定的
10. 快速排序、堆排序和希尔排序等时间性能比较好的排序方法都是不稳定的
11. 一般来说，排序过程中的“比较”是在“相邻的两个记录关键字”间进行的排序方法是稳定的。
12. 对于不稳定的排序方法，只要能举出一个实例说明即可

FxMarkBrown’s Blog