排序

排序是将文件中的记录进行按关键字有序排序的处理过程，是数据处理种常用的操作。

排序的基本操作：比较关键字大小，存储位置移动。

稳定的排序：记录中两个或以上关键字相等的记录，在排序前和排序后的顺序一致，则该排序方法稳定。

排序类型：内部排序（记录不多，都可以在内存储排序），外部排序（记录过多，需要内、外存进行数据交换）。

排序算法的优劣取决于：执行时间（时间复杂度），所需辅助空间（空间复杂度），算法稳定性。

各个排序算法的性能：

方法	平均时间	最坏所需时间	辅助空间	稳定性
直接插入	O(n^2)	O(n^2)	O(1)	稳定
Shell排序	O(n^1.3)		O(1)	不稳定
直接选择	O(n^2)	O(n^2)	O(1)	不稳定
堆排序	O(nlog2n)	O(nlog2n)	O(1)	不稳定
冒泡排序	O(n^2)	O(n^2)	O(1)	稳定
快速排序	O(nlog2n)	O(n^2)	O(log2n)	不稳定
归并排序	O(nlog2n)	O(nlog2n)	O(n)	稳定
基数排序	O(d(n+r))	O(d(n+r))	O(n+r)	稳定

1. 快、希、选、堆 不稳定；
2. 快归堆 nlogn；
3. 快排 log2n，归并 o(n)；
4. 借助于顺序结构：折半插入、希尔、归并、堆、快排；
5. 每趟排序均有元素排好：交换类、选择类；
6. 排序趟数与数据初始状态有关：冒泡、快排；
7. 比较次数与数据初态无关：简单选择、折半插入、基数排序；
8. 时间复杂度与初态无关：简单选择、基数排序、堆排序、归并排序；

插入类排序

将待插入元素Ri插入已排好的记录中的适当位置。

直接插入排序

将未排好序的无序部分逐个插入进已排好序的部分。

最好情况：待排序记录已经按指定次序排序。元素月有序，直接插入排序越快。

平均时间复杂度是O(n^2)。

折半插入排序

在将未排好序的无序部分逐个插入进已排好序的部分时，可以转变为折半插入排序，减少比较次数，使得性能效率更高。
并且折半插入排序只能用顺序存储。

由于仅仅减少关键字比较次数，没有减少移动次数，所以时间复杂度O(n^2)。

希尔排序

按照一定步长跨越地选择对应元素，对这些对应元素并在步长内采用直接插入排序，随后减少步长再排序，从而逐渐让整体逐渐趋向于有序。

交换类排序

不断交换反序的偶对，直到不再有反序偶对。

冒泡排序

依次比较相邻元素，通过两两交换，（例如按从大到小排序）将大的元素前置、小的元素后置——通过多趟这样的处理后，得到有序序列。

时间复杂度：O(n^2)；
空间复杂度：O(1)。

快速排序【⋆⋆⋆⋆⋆算法，最经常考】

以一个记录为参照R[s]，一趟排序后，R[s]的左半边均会小于（大于）右半边。
下述步骤以递增排序为例来描述：

1. 在记录中任取一个记录作为参照R[s]（一般选择第一个记录），从末尾开始往前比较；
2. 找到第一个小于R[s]的元素R[1]插入到R[s]的位置；
3. 随后从插入位置出发继续比较：
    1. 若是小于R[s]，则继续往后比较；
    2. 若是大于R[s]的元素R[2]，则插入到R[1]的位置，再继续往前比较；
    3. 直到找到小于R[s]的元素，插入到R[2]的位置；
4. 重复2、3的操作，直到让序列分为两个部分——这样一来，左半边的元素会小于右半边；
5. 随后对这两个部分内部再依此方式排序、分割——直到整个序列有序。

```
/*一趟快速排序*/
int quickOnePass(sqList *L, int low, int high){
    int i = low, j = high;
    L->R[0] = L->R[i];  //R[0]作为临时单元和哨兵
    do{
        while(L->R[0].key < L->R[j].key && (j > i))
            j--;
        if(j > i){
            L->R[i] = L->R[j];
```

```
            i++;
        }
        while(L->R[i] = L->R[0].key && (j > 1))
            i++;
        if(j > i){
            L->R[j] = L->R[i];
            j--;
        }
    }while(i != j);     //i==j时退出扫描
    L->R[i] = L->R[0];
    return i;
}

/*递归*/
void quickSort(sqList *L, int low , int high){
    int k;
    if(low < high){
        /*序列分为两个部分后，分别对每个子序排序*/
        k = quickOnePass(L, low, high);
        quickSort(L, low, k-1);
        quickSort(L, k+1, high);
    }
}
```

排序后，每一趟所选的R[s]可以构成一颗二叉树：树高==排序趟数，每一层的元素即为每一趟的基准元素。
若序列在排序前有序，那么会构成一棵单支树。

所以对于快速排序，元素越有序，速度越慢，反之速度越快。
元素**乱序**：时间复杂度为O(nlogn)，空间复杂度O(logn)；
元素**有序**：时间复杂度为O(n^2)，空间复杂度O(n)。

选择类排序

选择当前序列中最小值，放置到最后（或第一位），依此类推分成有序和无序两个部分，直到整个记录有序。

简单选择排序

若序列中有n个元素，通过n-1次关键字比较，从n-i+1个记录中选取关键字最小的记录，然后和第i个记录进行交换。

时间复杂度：O(n^2)；
空间复杂度：O(1)。

```
void simpleSelectionSort(sqList *L){
    int m, n, k;
    for(m=1; m<L->length; m++){
        k = m;
        for(n=m+1; n<=L->length; n++)
            if(LT(L->R[n].key, L->R[k].key)) k = n;
        if(k != m){
            /*交换记录*/
```

```
                L->R[0] = L->R[m];
                L->R[m] = L->R[k];
                L->R[k] = L->R[0];
            }

        }
    }
```

堆排序

n个元素的序列，满足
$ki \leq k2i, ki \leq k(2i+1)$ （小根堆）
或
$ki \geq k2i, ki \geq k(2i+1)$ （大根堆）
所得到一个以k1为根且从上到下、从左到右编号的完全二叉树，得到的序列则是将二叉树以顺序结构存储，堆的结构和该序列结构一致。

每次输出一个堆顶元素，将从最底层的叶子节点补充，再重新调整。

时间复杂度$O(nlogn)$； 空间复杂度$O(1)$。

堆排序的构思：

1. 堆一组待排序的记录，按堆的定义建立堆；
2. 将堆顶记录和最后一个记录交换位置，则得到前n-1个记录是无序的，最后一个记录是有序的；
3. 堆顶记录交换后，前n-1个记录不再是堆，需要重新组织成一个堆，然后堆顶记录和倒数第二个记录交换位置，将整个序列中次小关键字的记录调整出无序区；
4. 重复上述步骤，直到记录有序。

排序过程：

1. 建立初始堆：
    1. 按照完全二叉树的层次逐个插入；
    2. 按照从下往上、从右往左，先兄弟、后双亲的顺序来比较；
    3. （此处以小根堆为例子）按照如上顺序，将较小值置上；
2. 让堆顶于最后一个元素交换；
3. 重新调整，从上往下调。

归并类排序

将序列中的每个元素都视为单个序列，相邻的序列合成一个序列并比较排序，依此类推，直到最终合成一个有序序列。
遵循——双指针、不回溯、尾插法，谁小谁尾插、谁小谁后移、相等即删除。

时间复杂度：$O(nlog2n)$；
空间复杂度：$O(n)$。

基数类排序

将序列中的单个元素拆成多个关键字（每个元素按位拆成子序列），再将子序列中对应位进行比较，按个位、十位、百位依次在将各个子序排序（低位优先），使得整个序列逐渐有序。

在实际操作中，将会构建0~9的桶，先按元素的个位数字分配进对应的桶——桶内按队列（FIFO），桶间从小到大——从而得到一个排序好一趟的序列。随后再将元素的十位、百位等依此类推进行排序，直到整个序列有序。