排 序

所谓排序(Sort)是指将一批数据按指定的规则进行排列。常见的排列规则有升序(从小到大)和降序(从大到小)两种。

1、稳定排序和不稳定排序

简单地说就是所有相等的数经过某种排序方法后，仍能保持它们在排序之前的相对次序，我们就说这种排序方法是稳定的。反之，就是不稳定的。

比如：一组数排序前是 a1,a2,a3,a4,a5，其中 a2=a4，经过某种排序后为 a1,a2,a4,a3,a5，则我们说这种排序是稳定的，因为 a2 排序前在 a4 的前面，排序后它还是在 a4 的前面。如果排序后变成 a1,a4,a2,a3,a5 就不是稳定的了。

2、内排序和外排序

在排序过程中，所有需要排序的数都在内存，并在内存中调整它们的存储顺序，称为内排序；

在排序过程中，只有部分数被调入内存，并借助内存调整数在外存中的存放顺序排序方法称为外排序。

3、算法的时间复杂度和空间复杂度

所谓算法的时间复杂度，是指执行算法所需要的计算工作量。

一个算法的空间复杂度，一般是指执行这个算法所需要的内存空间。

一、插入排序

1．直接插入排序

基本思想：

(1) 有 n 个数（存放在数组 a[n]中），开始时，有序里只有一个数 a[0]，第一趟排序将第 2 个数插入到有序区，使有序区增加到两个数。

(2) 第二趟将 3 个数插入到有序区，使有序区增加到 3 个数。

(3) 重复以上排序过程，最多进行 n-1 趟排序后，可以完成 n 个数的排序。

算法演示：

初始序列	75	87	68	92	88	61	77	96	80	72
第 1 趟排序	75	87	68	92	88	61	77	96	80	72
第 2 趟排序	68	75	87	92	88	61	77	96	80	72
第 3 趟排序	68	75	87	92	88	61	77	96	80	72
第 4 趟排序	68	75	87	88	92	61	77	96	80	72
第 5 趟排序	61	68	75	87	88	92	77	96	80	72
第 6 趟排序	61	68	75	77	87	88	92	96	80	72
第 7 趟排序	61	68	75	77	87	88	92	96	80	72
第 8 趟排序	61	68	75	77	80	87	88	92	96	72

第 9 趟排序	61	68	72	75	77	80	87	88	92	96
最后结果	61	68	72	75	77	80	87	88	92	96

程序清单：

```
#include "stdio.h"
int a[10];

void insertsort()          // 直接插入排序算法
{
  int i,j,t;
  for(i=1;i<10;i++)
  {
    t=a[i];
    j=i-1;
    while(j>=0 && t<a[j])  // 元素后移，以便空出一个位置插入 t
    {
      a[j+1]=a[j];
      j--;
    }
    a[j+1]=t;  // 在 j+1 位置插入 t
  }
}

main()
{
  int i;
  for(i=0;i<10;i++)
  {
    a[i]=rand()%100;
    printf("%4d",a[i]);
  }
  printf("\n");
  insertsort();
  for(i=0;i<10;i++) printf("%4d",a[i]);
  getch();
}
```

算法分析：

采用这种排序方法时，要在包含 n 个元素的一维数组 a[0]～a[n-1]中插入一个新元素，关键字的比较次数最少是 1 次(留在原处)，最多是 n 次(插到最前面)；元素的移动次数等于关键字比较次数加 2，总的比较次数和移动次数分别下：

最少比较 n-1 次，最多比较 $\sum_{i=1}^{n-1} i$ 次；最少移动 2(n-1)次，最多移动 $\sum_{i=1}^{n-1}(i+2)$ 次。

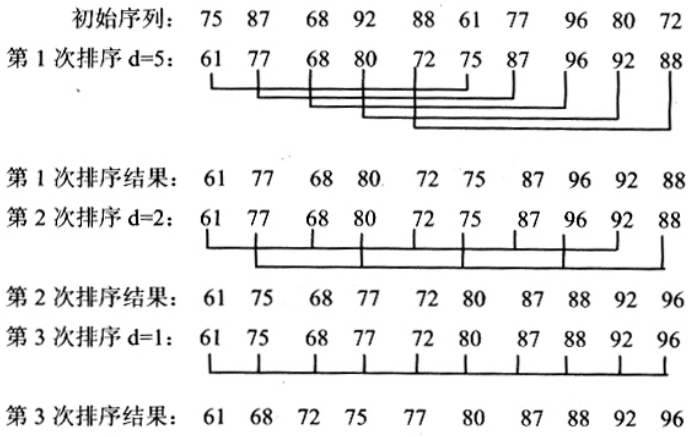
因此，算法的执行时间在最坏的情况下是 $O(n^2)$。直接插入排序是稳定的。

2. 希尔排序

基本思想：

希尔(Shell)排序又称为缩小增量排序方法，其基本思想是：将数据区中所有数据按下标的一定增量分组，对每组数据采用直接插入排序方法进行排序，随着增量逐渐减小，所分成的组包含的数据越来越多，当增量的值减少到 1 时，整个数据合成为一组，构成一组有序序列，表示排序完成。

算法演示：



程序清单：

```
#include "stdio.h"
#define N 10
void shellsort(int r[])
{
  int i,j,temp,gap;
  gap=N/2;                          // 增量置初值
  while(gap>0)
  {
    for(i=gap;i<N;i++)              // 对所有相隔 gap 位置的所有元素组进行排序
    {
      temp=r[i];
      j=i-gap;
      while(j>=0 && temp<r[j])      // 对相隔 gap 位置的元素组进行排序
      { r[j+gap]=r[j]; j=j-gap; }
      r[j+gap]=temp;
    }
    gap=gap/2;                      // 减小增量
  }
}
main()
{
```

```
  int a[N]={75,87,68,92,88,61,77,96,80,72},i;
  shellsort(a);
  for(i=0;i<N;i++) printf("%3d",a[i]);
}
```

算法分析：

希尔排序算法的时间复杂度是 $O(n\log_2 n)$ 。

二、选择排序

基本思想：

(1) 对有 n 个数的序列（存放在数组 a(n)中），从中选出最小的数，与第 1 个数交换位置；

(2) 除第 1 个数外，其余 n-1 个数中选最小的数，与第 2 个数交换位置；

(2) 依次类推，选择了 n-1 次后，这个数列已按升序排列。

算法演示：

初始序列	75	87	68	92	88	61	77	96	80	72
第 1 趟排序	61	87	68	92	88	75	77	96	80	72
第 2 趟排序	61	68	87	92	88	75	77	96	80	72
第 3 趟排序	61	68	72	92	88	75	77	96	80	87
第 4 趟排序	61	68	72	75	88	92	77	96	80	87
第 5 趟排序	61	68	72	75	77	92	88	96	80	87
第 6 趟排序	61	68	72	75	77	80	88	96	92	87
第 7 趟排序	61	68	72	75	77	80	87	96	92	88
第 8 趟排序	61	68	72	75	77	80	87	88	92	96
第 9 趟排序	61	68	72	75	77	80	87	88	92	96
最后结果	61	68	72	75	77	80	87	88	92	96

程序清单：

```
#include "stdio.h"
main()
{
  int i,j,min,t,a[10];
  for(i=0;i<10;i++)
  {
    a[i]=rand()%100;
    printf("%4d",a[i]);
  }
  printf("\n");
  for(i=0;i<9;i++)
  {
    min=i;
    for(j=i+1;j<10;j++)
```

```
        if(a[min]>a[j]) min=j;
      if(i!=min)
      { t=a[i]; a[i]=a[min]; a[min]=t; }
    }
    for(i=0;i<10;i++) printf("%4d",a[i]);
    getch();
  }
```

算法分析：

说明：用这种方法排序，其比较次数与各元素原来的排列顺序无关：第 1 次选择(i=0)比较 n-1 次，第 2 次选择(i=1)比较 n-2 次，……，第 n-1 次选择(i=n-2)比较 1 次，总的比较次数为 $\sum_{i=0}^{n-2}(n-i-1)$ 次，但元素的移动次数和初始排列顺序有关：如果 a[0..n-1]原来就是从小到大排列的，就不需要移动；如果每次选择都要进行交换，移动次数达到最大值，即 3(n-1)次。因此，算法的执行时间为 0(n^2)。直接选择排序是不稳定的。

三、交换排序

1．冒泡法排序

基本思想：(将相邻两个数比较，小的调到前头)

(1) 有 n 个数（存放在数组 a[n]中），第一趟将每相邻两个数比较，小的调到前头，经 n-1 次两两相邻比较后，最大的数已“沉底”，放在最后一个位置，小数上升“浮起”；

(2) 第二趟对余下的 n-1 个数（最大的数已“沉底”）按上法比较，经 n-2 次两两相邻比较后得次大的数；

(3) 依次类推，n 个数共进行 n-1 趟比较，在第 j 趟中要进行 n-j 次两两比较。

算法演示：

初始序列	75	87	68	92	88	61	77	96	80	72
第 1 趟排序	75	68	87	88	61	77	92	80	72	96
第 2 趟排序	68	75	87	61	77	88	80	72	92	96
第 3 趟排序	68	75	61	77	87	80	72	88	92	96
第 4 趟排序	68	61	75	77	80	72	87	88	92	96
第 5 趟排序	61	68	75	77	72	80	87	88	92	96
第 6 趟排序	61	68	75	72	77	80	87	88	92	96
第 7 趟排序	61	68	72	75	77	80	87	88	92	96
第 8 趟排序	61	68	72	75	77	80	87	88	92	96
第 9 趟排序	61	68	72	75	77	80	87	88	92	96
最后结果	61	68	72	75	77	80	87	88	92	96

程序清单：

```
#include "stdio.h"
main()
{
  int i,j,t,a[10];
  for(i=0;i<10;i++)
  {
    a[i]=rand()%100;
    printf("%4d",a[i]);
  }
  printf("\n");
  for(i=0;i<9;i++)
    for(j=0;j<9-i;j++)
      if(a[j]>a[j+1]) { t=a[j];a[j]=a[j+1];a[j+1]=t; }  /* 交换 a[j]和 a[j+1] */
  for(i=0;i<10;i++) printf("%4d",a[i]);
  getch();
}
```

算法分析：

冒泡排序的执行时间与 n 个元素原来的排列顺序有关。排序前，如果 n 个元素已经从小到大排好，则只要进行一趟起泡，关键字比较次数最少(n-1 次)，而且不需要移动元素；如果 n 个元素是从大到小排列的，则需要进行 n-1 趟起泡，关键字的比较次数和元素的移动次数都达到最大值，分别为 $\sum_{i=0}^{n-2}(n-i+1)$ 和 $3\sum_{i=0}^{n-2}(n-i+1)$。因此，冒泡排序的执行时间最坏情况下是 $O(n^2)$。

因为只有在 a[j]>a[j+1]的情况下，才交换 a[j]和 a[j+1]，所以，冒泡排序是稳定的。

2. 快速排序

快速排序是由冒泡排序改进而得的，它的基本思想是：在待排序的 n 个数据中任取一个数据(通常取第一个)，把该数据放入最终位置后，整个数据区间被此数据分割成两个子区间。所有关键字比该数据小的放置在左子区间中，所有比它大的放置在右子区间中，并把该数据排在这两个子区间的中间，这个过程称作一趟快速排序。之后对所有的两个子区间分别重复上述过程，直到每个子区间内只有一个数据为止。简而言之，每趟使表的第一个元素入终位，将数据区间一分为二，对子区间按递归方式继续这种划分，直至划分的子区间长度为 1。

一趟快速排序采用从两头向中间扫描的办法，同时交换与基准数据逆序的数据。具体做法是：设两个指示器 i 和 j，它们的初值分别为指向无序区中第一个和最后一个数据。假设无序区中的数据为 R[s..t]，则 i 的初值为 s，j 的初值为 t，首先将 R[s]移至临时变量 temp 中作为基准，令 j 自 t 起向左扫描直至 R[j]<temp 时，将 R[j]移至 i 所指的位置上，然后令 i 自 i+1 起向右扫描直至 R[i]>temp 时，将 R[i]移至 j 所指的位置上，依次重复直至 i=j，此时所有 R[k](k=s, s+1, …, j-1)都小于 temp，而所有 R[k](k=j+1,

j+2, …, t)必大于 temp,则可将 temp 中的记录移至所指位置 R[i],它将无序区中的数据分割成 R[s..j-1]和 R[j+1..t]，以便分别进行排序。

快速排序过程如图所示，蓝色标注的元素为本次比较的元素，天蓝标注的为分成的子区间，带阴影的部分为当前排序区间。

算法演示：

初始序列	75	87	68	92	88	61	77	96	80	72
第 1 趟排序	72	61	68	75	88	92	77	96	80	87
第 2 趟排序	68	61	72	75	88	92	77	96	80	87
第 3 趟排序	61	68	72	75	88	92	77	96	80	87
第 4 趟排序	61	68	72	75	87	80	77	88	96	92
第 5 趟排序	61	68	72	75	77	80	87	88	96	92
第 6 趟排序	61	68	72	75	77	80	87	88	96	92
第 7 趟排序	61	68	72	75	77	80	87	88	92	96
最后结果	61	68	72	75	77	80	87	88	92	96

程序清单：

```
#include "stdio.h"
#define N 10
void quicksort(int r[],int s,int t)  /* 对 r[s]至 r[t]区间的元素进行快速排序 */
{
  int i=s,j=t,temp;
  if(s<t)                          /* 区间内至少存在一个元素的情况 */
  {
    temp=r[s];                     /* 用区间的第一个元素作为基准 */
    while(i!=j)                    /* 从区间两端交替向中间扫描,直至 i==j 为至 */
    {
      while(j>i && r[j]>temp) j--;  /* 从右向左扫描，找第 1 个小于 temp 的 r[j] */
      if(i<j) { r[i]=r[j]; i++; }   /* 找到这样的 r[j]时，则 r[i]和 r[j]交换 */
      while(i<j && r[i]<temp) i++;  /* 从左向右扫描，找第 1 个小于 temp 的 r[i] */
      if(i<j) { r[j]=r[i]; j--; }   /* 找到这样的 r[i]时，则 r[i]和 r[j]交换 */
    }
    r[i]=temp;
    quicksort(r,s,i-1);            /* 对左区间递归排序 */
    quicksort(r,i+1,t);            /* 对右区间递归排序 */
  }
}
main()
{
  int a[N]={75,87,68,92,88,61,77,96,80,72},i;
  quicksort(a,0,N-1);
```

```
    for(i=0;i<N;i++) printf("%3d",a[i]);
}
```

算法分析：

快速排序算法的时间复杂度是O(nlog2n)，而且快速排序是不稳定的。

四、排序法的效率比较

排序法	最差时间复杂度	平均时间复杂度	稳定性	空间复杂度
插入排序	$O(n^2)$	$O(n^2)$	稳定	0(1)
希尔排序	$O(n\log_2 n)$	$O(n\log_2 n)$	不稳定	0(1)
直接选择排序	$O(n^2)$	$O(n^2)$	稳定	0(1)
冒泡排序	$O(n^2)$	$O(n^2)$	稳定	0(1)
快速排序	$O(n^2)$	$O(n\log_2 n)$	不稳定	$O(\log_2 n)\sim O(n)$