常用的排序算法

一、冒泡排序

冒泡排序（Bubble Sort），是一种较简单的排序算法。

它重复地走访过要排序的元素列，依次比较两个相邻的元素，如果顺序（如从大到小、首字母从Z到A）错误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素列已经排序完成。

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（升序或降序排列），就如同碳酸饮料中二氧化碳的气泡最终会上浮到顶端一样，故名“冒泡排序”。

```
#include<cstdio>
using namespace std;
int n,x[100];
int main()
{
   scanf("%d",&n);
   for (int i=0;i<n;i++)
      scanf("%d",&x[i]);
   for (int t,i=0; i<n-1; i++) /* 外循环为排序趟数，n个数进行n-1趟 */
        for (int j=0; j<n-1-i; j++) { /* 内循环为每趟比较的次数，第i趟比较n-i次 */
            if (x[j] > x[j+1]) { /* 相邻元素比较，若逆序则交换（升序为左大于右，降序反之） */
                t = x[j];
                x[j] = x[j+1];
                x[j+1] = t;
            }
        }
   for (int i=0; i<n; i++)
        printf("%d ", x[i]);
   printf("\n");
   return 0;
}
```

时间复杂度 $O(n^2)$

二、选择排序

选择排序法是一种不稳定的排序算法。它的工作原理是每一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，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，直到全部待排序的数据元素排完。

```
#include<cstdio>
using namespace std;
int n,x[100];
int main()
{
```

```
   scanf("%d",&n);
   for (int i=0;i<n;i++)
      scanf("%d",&x[i]);
   for(int t,i=0;i<n-1;i++)//从首位开始,注意:最后一个数由于已经被动和前面所有数进行了比较,故不需要再主动比较
    {
        int k=i;
        for(int j=i+1;j<n;j++)//依次和后面的数比较找出最小的数
            if(x[j]<x[k])
               k=j;
        if(k != i)//如果最小的数不是首位,则交换
           t=x[k],x[k]=x[i],x[i]=t;
    }
   for (int i=0; i<n; i++)
        printf("%d ", x[i]);
   printf("\n");
   return 0;
}
```

时间复杂度O(n2)O(n2)，选择排序是一个不稳定的排序算法。

三、插入排序

插入排序是指在待排序的元素中，假设前面n-1(其中n>=2)个数已经是排好顺序的，现将第n个数插到前面已经排好的序列中，然后找到合适自己的位置，使得插入第n个数的这个序列也是排好顺序的。

按照此法对所有元素进行插入，直到整个序列排为有序的过程，称为插入排序 。

```
#include<cstdio>
using namespace std;
int n,x[100];
int main()
{
   scanf("%d",&n);
   for (int i=0;i<n;i++)
      scanf("%d",&x[i]);
   for (int pos,cur,i=1; i<n; i++)
      {
         pos = i-1 ;    //有序序列的最后一个元素位置
         cur = x[i];    //保存待排序元素的值
         while (pos >= 0 && x[pos] > cur)
         {
             x[pos + 1] = x[pos];
             pos--;
         }
         x[pos + 1] = cur;    //将待排序元素插入数组中
     }
   for (int i=0; i<n; i++)
        printf("%d ", x[i]);
   printf("\n");
   return 0;
}
```

时间复杂度O(n2)O(n2)

四、希尔排序

```
#include<cstdio>
using namespace std;
int n,x[100];
int main()
{
   scanf("%d",&n);
   for(int i=1;i<=n;i++)
       scanf("%d",&x[i]);
   for(int step=n>>1;step>0;step>>=1) {
       for(int i=step;i<=n;i++) {
           int j=i;
           int temp=x[j];
           while(j-step>=0&&x[j-step]>temp) {
               x[j]=x[j-step];
               j-=step;
           }
           x[j]=temp;
       }
   }
   for(int i=1;i<=n;i++)
       printf("%d ",x[i]);
   return 0;
}
```

时间复杂度O(n2)O(n2)

五、快速排序

快速排序(Quick Sort)使用分治法策略。

它的基本思想是：选择一个基准数，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。

快速排序流程：

(1) 从数列中挑出一个基准值。

(2) 将所有比基准值小的摆放在基准前面，所有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。

(3) 递归地把"基准值前面的子数列"和"基准值后面的子数列"进行排序。

```
#include<cstdio>
using namespace std;
int n,x[100];
void qsort(int L,int R) {
   int i=L,j=R,mid=x[(i+j)/2],t;
   while (i<j) {
        while (x[i]<mid) i++;
        while (x[j]>mid) j--;
        if (i<=j) {
               t=x[i],x[i]=x[j],x[j]=t;i++;j--;
        }
   }
   if (i<R) qsort(i,R);
   if (L<j) qsort(L,j);
}
```

```
int main()
{
   scanf("%d",&n);
   for (int i=0;i<n;i++)
      scanf("%d",&x[i]);
   qsort(0,n-1);
   for (int i=0; i<n; i++)
        printf("%d ", x[i]);
   printf("\n");
   return 0;
}
```

快速排序时间复杂度

快速排序的时间复杂度在最坏情况下是O(n2)O(n2)，平均的时间复杂度是O(nlogn)O(nlogn)。

假设被排序的数列中有n个数。遍历一次的时间复杂度是O(n)，需要遍历多少次呢？至少log(n+1)次，最多n次。

1.为什么最少是log(n+1)次？快速排序是采用的分治法进行遍历的，我们将它看作一棵二叉树，它需要遍历的次数就是二叉树的深度，而根据完全二叉树的定义，它的深度至少是log(n+1)。因此，快速排序的遍历次数最少是log(n+1)次。

2.为什么最多是n次？这个应该非常简单，还是将快速排序看作一棵二叉树，它的深度最大是N。因此，快读排序的遍历次数最多是n次。

六、归并排序

```
#include<cstdio>
using namespace std;
int n,x[1000],z[1000];
void merge_sort(int L,int R)
{
   if (L==R) return;
   int mid=(L+R)/2;
   merge_sort(L,mid);merge_sort(mid+1,R);
   int i=L,j=mid+1,k=L;
   while (i<=mid && j<=R)
       if (x[i]<x[j])
          z[k++]=x[i++];
       else
          z[k++]=x[j++];
   while (i<=mid) z[k++]=x[i++];
   while (j<=R) z[k++]=x[j++];
   for (int i=L;i<=R;i++) x[i]=z[i];
}
int main()
{
   scanf("%d",&n);
   for (int i=1;i<=n;i++) scanf("%d",&x[i]);
   merge_sort(1,n);
   for (int i=1;i<=n;i++)
      printf("%d ",x[i]);
   printf("\n");
   return 0;
}
```

时间复杂度：O(nlogn)。

空间复杂度：O(n)，归并排序需要一个与原数组相同长度的数组做辅助来排序。

稳定性：在数据量大且数据递增或递减连续性好的情况下，效率比较高，且是O(nlogn)复杂度下唯一一个稳定的排序。

七、堆排序

```cpp
#include<cstdio>
using namespace std;
int n,x[100];
void check(int v,int nmax){
    int k=2*v,t;
    if (k>nmax) return;
    if (k+1>nmax) {
         if (x[v]>x[k])
              t=x[k],x[k]=x[v],x[v]=t;
         return;
    }
    int j=k;if (x[k]>x[k+1]) j=k+1;
    if (x[v]>x[j]) {
          t=x[j],x[j]=x[v],x[v]=t;
          check(j,nmax);
    }
}
int main()
{
   scanf("%d",&n);
   for (int i=1;i<=n;i++) scanf("%d",&x[i]);
   for (int i=n/2;i>=1;i--)
       check(i,n);
   int m=n;
   for (int i=1;i<=m;i++) {
      printf("%d ",x[1]);
      x[1]=x[n];n--;check(1,n);
   }
   printf("\n");
   return 0;
}
```

八、基数排序