



导航

- [首页](#)
- [社区主页](#)
- [当前事件](#)
- [最近更改](#)
- [随机页面](#)
- [使用帮助](#)
- [NOCOW地图](#)
- [新手试练场](#)

搜索

工具箱

- [链入页面](#)
- [链出更改](#)
- [特殊页面](#)
- [可打印版](#)
- [永久链接](#)

- [条目](#)
- [讨论](#)
- [编辑](#)
- [历史](#)

为防止广告，目前nocow只有登录用户能够创建新页面。如要创建页面请先[登录/注册](#)（新用户需要等待1个小时才能正常使用该功能）。

# 排序算法

目录 [\[隐藏\]](#)

- [1 介绍](#)
- [2 基于比较的排序算法](#)
  - [2.1 插入排序\(Insertion Sort\)](#)
  - [2.2 希尔排序\(Shell Sort\)](#)
  - [2.3 冒泡排序\(Bubble Sort\)](#)
  - [2.4 快速排序\(Quick Sort\)](#)
  - [2.5 一个与C++库函数不相上下的QuickSort](#)
  - [2.6 本人觉得直接将template T直接换成int,long之类爽快些!](#)
  - [2.7 快速排序简单实现\(Quick Sort by wssbwssbwssb\)](#)
  - [2.8 堆排序\(Heap Sort\)](#)
  - [2.9 归并排序\(Merge Sort\)](#)
- [3 线性排序算法](#)
  - [3.1 计数排序\(Counting Sort\)](#)
  - [3.2 桶排序\(Bin Sort\)](#)
  - [3.3 基数排序\(Radix Sort\)](#)
- [4 几种排序算法的比较和选择](#)
  - [4.1 稳定性比较](#)
  - [4.2 时间复杂性比较](#)
  - [4.3 辅助空间的比较](#)
  - [4.4 其它比较](#)

## 介绍 [\[编辑\]](#)

将杂乱无章的数据元素，通过一定的方法按关键字顺序排列的过程叫做排序。

## 基于比较的排序算法 [\[编辑\]](#)

### 插入排序(Insertion Sort) [\[编辑\]](#)

插入排序的基本思想:经过i-1遍处理后,L[1..i-1]已排好序。第i遍处理仅将L[i]插入L[1..i-1]的适当位置p，原来p后的元素一一向右移动一个位置,使得L[1..i]又是排好序的序列。对于数据比较大的，通常可以采取二分查找来确定一个数应该加入的位置。

例2:输入序列数据按非减顺序输出。

程序:

```
program InsertionSort;
const
    maxn = 1000000;
var
    n, i, x, p, j : longint;
    a : array[1..maxn] of longint;

    function find(l, r, w: longint): longint;
    var
        mid : longint;
```

```
begin
    if l >= r then exit(1);
    mid := (l + r) div 2;
    if w > a[mid] then find := find(mid+1, r, w)
    else find := find(1, mid, w);
end;

begin
    assign(input, 'num.in');
    reset(input);
    assign(output, 'num.out');
    rewrite(output);

    readln(n);
    a[1] := maxlongint;
    for i := 1 to n do
        begin
            read(x);
            p := find(1, i, x);
            for j := i downto p do a[j+1] := a[j];
            a[p] := x;
        end;

        for i := 1 to n do writeln(a[i]);

        close(input);
        close(output);
    end.
<source lang = "pascal">
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
begin
    write('Enter date:');
    for i:= 1 to n do read(a[i]);
    writeln;
    for i:=2 to n do
        begin
            k:=a[i];j:=i-1;
            while (k<a[j]) and (j>0) do
                begin a[j+1]:=a[j];j:=j-1 end;
            a[j+1]:=k;
        end;
    write('output data:');
    for i:= 1 to n do write(a[i]:6);
    writeln;
end.
```

=== [[选择排序]](Selection Sort) ===  
选择排序的基本思想是：

对待排序的记录序列进行n-1遍的处理，第1遍处理是将L[1..n]中最小者与L[1]交换位置，第2遍处理是将L[2..n]中最小者与L[2]交换位置，.....，第i遍处理是将L[i..n]中最小者与L[i]交换位置。这样，经过i遍处理之后，前i个记录的位置就已经按从小到大的顺序排列好了。

例1：输入序列数据按非减顺序输出。

程序如下：

```
<source lang = "pascal">
program xzpx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
begin
    write('Enter data:');
    for i:= 1 to n do read(a[i]);
    writeln;
    for i:=1 to n-1 do
        begin
            k:=i;
            for j:=i+1 to n do
                if a[j]<a[k] then k:=j;
            if k<>i then
                begin t:=a[i];a[i]:=a[k];a[k]:=t;end;
        end;
    write('output data:');
    for i:= 1 to n do write(a[i]:6);
    writeln;
end.
```

折半插入 const max=100; type arr=array[0..max]of integer; var a:arr;

```
i,n:integer;
fin:text;
```

procedure BinSort(var r:arr; rn:integer);

```
var i,j,low,high,mid:integer;
begin
  for i:=2 to rn do
    begin
      r[0]:=r[i];      { 将待插入记录存放到监视哨r[0]里 }
      low:=1;      high:=i-1; { 确定查找的上下界 }
      while(low<=high) do { 寻找插入位置 }
        begin
          mid:=(low+high)div 2; { 记录后移 }
          if r[0]<r[mid] then high:=mid-1 { 左半边查找 }
          else low:=mid+1; { 右半边查找 }
        end;
      for j:=i-1 downto low do r[j+1]:=r[j]; { 记录依次向后移动 }
      r[low]:=r[0]; { 将待插入记录插入到已排序的序列中 }
    end;
  end;
```

begin

```
assign(fin,'input.txt');
reset(fin);
readln(fin,n);
for i:=1 to n do read(fin,a[i]);
write('before sort:');
for i:=1 to n do write(a[i]:5); writeln;
BinSort(a,n);
write('after sort: ');
for i:=1 to n do write(a[i]:5); writeln;
close(fin);
readln;
```

end. </source> 直接插入 const max=100; type arr=array[0..max]of integer; var a:arr;

```
i,n:integer;
fin:text;
```

procedure InsSort(var r:arr; rn:integer); { 直接插入排序算法 }

```
var i,j:integer;
begin
  for i:=2 to rn do
    begin
      r[0]:=r[i];      { 将待插入记录存放到监视哨r[0]里 }
      j:=i-1;
      while(r[0]<r[j]) do { 寻找插入位置 }
        begin
          r[j+1]:=r[j]; { 记录后移 }
          dec(j);
        end;
      r[j+1]:=r[0]; { 将待插入记录插入到已排序的序列中 }
    end;
  end;
```

begin

```
assign(fin,'input.txt');
reset(fin);
readln(fin,n);
for i:=1 to n do read(fin,a[i]);
write('before sort:');
for i:=1 to n do write(a[i]:5); writeln;
InsSort(a,n);
write('after sort: ');
for i:=1 to n do write(a[i]:5); writeln;
close(fin);
readln;
```

end.

希尔排序(Shell Sort)

[编辑]

基本思想：先取一个小于n的整数d1作为第一个增量，把文件的全部记录分成d1个组。所有距离为d1的倍数的记录放在同一个组中。先在各组内进行直接插入排序；然后，取第二个增量d2<d1重复上述的分组和排序，直至所取的增量dt=1(dt<dt-1<...<d2<d1)，即所有记录放在同一组中进行直接插入排序为止。

```
program shell;
const n=7;
var a:array[1..n]of longint;
    i,j,d,x:longint;
begin
  write('Enter data:');
  for i:=1 to n do read(a[i]);
  d:=n div 2;
  while d>=1 do
  begin
    for i:=d+1 to n do
    begin
      x:=a[i];
      j:=i-d;
      while (j>0)and(x<a[j]) do
      begin
        a[j+d]:=a[j];
        j:=j-d;
      end;
      a[j+d]:=x;
    end;
    d:=d div 2;
  end;
  write('output data:');
  for i:=1 to n do write(a[i], ' ');
  writeln;
end.
```

冒泡排序(Bubble Sort)

[编辑]

冒泡排序又称交换排序其基本思想是:对待排序的记录的关键字进行两两比较,如发现两个记录是反序的,则进行交换,直到无反序的记录为止。

例:输入序列数据按非减顺序输出。

程序1:

```
program mppx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
begin
  write('Enter date:');
  for i:= 1 to n do read(a[i]);
  for i:=1 to n-1 do
    for j:=n downto i+1 do
      if a[j-1]<a[j] then
        begin t:=a[j-1];a[j-1]:=a[j];a[j]:=t end;
    write('output data:');
    for i:= 1 to n do write(a[i]:6);
    writeln;
  end.
```

程序2:

```
program mppx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
    bool:boolean;
begin
  write('Enter date:');
  for i:= 1 to n do read(a[i]);
  i:=1;bool:=true;
  while (i<n) and bool do
    begin
      bool:=false;
      for j:=n downto i+1 do
        if a[j-1]<a[j] then
          begin t:=a[j-1];a[j-1]:=a[j];a[j]:=t;bool:=true end;
        i:=i+1;
      end;
    write('output data:');
```

```

    for i:= 1 to n do write(a[i]:6);
    writeln;
end.

```

程序3:

```

program mppx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
begin
    write('Enter date:');
    for i:= 1 to n do read(a[i]);
    writeln;
    k:=n;
    while k>0 do
    begin
        j:=k-1;k:=0;
        for i:=1 to j do
            if a[i]>a[i+1] then
                begin t:=a[i];a[i]:=a[i+1];a[i+1]:=t;k:=i;end;
        end;
        write('output data:');
        for i:= 1 to n do write(a[i]:6);
        writeln;
    end.

```

程序4:

```

var a:array[1..100000] of integer;
    n,i,i,p:integer;
procedure swap(var x,y:integer);
var t:integer;
begin
    t:=x;
    x:=y;
    y:=t;
end;
begin(main);
    readln(n);
    for p:=1 to n do
        read(a[p]);
        for i:=1 to n-1 do
            for j:=i to n do
                if a[i]>a[j] then swap(a[i],a[j]);
        for p:=1 to n do
            writeln(a[p]);
        readln;
    end.

```

双向冒泡 var a:array[1..maxint] of integer;

```

n,i:integer;

```

procedure sort(b:array of integer); var i,bottom,top,t:integer;

```

f:boolean;

```

begin

```

    bottom:=1;
    top:=n;
    f:=true;
    while(f=true) do
    begin
        f:=false;
        for i:=bottom to top-1 do
            if a[i]>a[i+1] then begin t:=a[i];a[i]:=a[i+1];a[i+1]:=t;f:=true;end;
        top:=top-1;
        for i:=top downto bottom+1 do
            if a[i]<a[i-1] then begin t:=a[i];a[i]:=a[i-1];a[i-1]:=t;f:=true end;

```

```
        bottom:=bottom+1;
    end;
```

end;

begin

```
    readln(n);
    for i:=1 to n do
        read(a[i]);
    sort(a);
    for i:=1 to n do
        write(a[i], ' ');
    end;
```

end.

快速排序(Quick Sort) [编辑]

程序1:

program kspv; const n=7; type arr=array[1..n] of integer; var a:arr; i:integer; procedure quicksort(var b:arr; s,t:integer); var i,j,x,t1:integer; begin i:=s;j:=t;x:=b[i]; repeat

```
    while (b[j]>=x) and (j>i) do j:=j-1;
    if j>i then begin t1:=b[i]; b[i]:=b[j];b[j]:=t1;end;
    while (b[i]<=x) and (i<j) do i:=i+1;
    if i<j then begin t1:=b[j];b[j]:=b[i];b[i]:=t1; end
```

until i=j; b[i]:=x; i:=i+1;j:=j-1; if s<j then quicksort(b,s,j); if i<t then quicksort(b,i,t); end; begin write('input data:'); for i:=1 to n do read(a[i]); writeln; quicksort(a,1,n); write('output data:'); for i:=1 to n do write(a[i]:6); writeln; end.

程序2:

program kspv; const n=7; type arr=array[1..n] of integer; var a:arr; i:integer; procedure quicksort(var b:arr; s,t:integer); var i,j,x:integer; begin i:=s;j:=t;x:=b[i]; repeat

```
    while (b[j]>=x) and (j>i) do j:=j-1;
    if j>i then begin b[i]:=b[j];i:=i+1;end;
    while (b[i]<=x) and (i<j) do i:=i+1;
    if i<j then begin b[j]:=b[i];j:=j-1; end
```

until i=j; b[i]:=x; i:=i+1;j:=j-1; if s<j then quicksort(b,s,j); if i<t then quicksort(b,i,t); end; begin write('input data:'); for i:=1 to n do read(a[i]); writeln; quicksort(a,1,n); write('output data:'); for i:=1 to n do write(a[i]:6); writeln; end.

一个与C++库函数不相上下的QuickSort [编辑]

（言过其实了，C++ STL的Sort实现用的是Introsort，是快速排序的变种，主要是递归过深的时候自动转换为堆排或插入排序（是堆排还是插入排序还要视具体实现而定），可以保证最坏情况下还是O(nlogn)，并且充分使用了尾递归优化（快排最后不是两个递归吗？最后一个递归可以不必真的递归，可以像gcd算法一样通过迭代参数来改善运行速度），STL快排可以经受任何实践的考验，而这段代码在最坏情况下还是O(n^2)) -- by 某奋战的Oler

本人觉得直接将template T直接换成int,long之类爽快些! [编辑]

```
<template T>
void sort(T a[],T st,T ed)
{ if(st<ed) //先设一个开关优化，会更快一些
{ T tmp=a[st],i=st,j=ed;
while(i<j)
{ while(a[j]>tmp&&i<j) --j; //C++在判断时，会打开编译开关，把a[j]与tmp放在前比较，这样会更快一些~~
if(i<j) a[i++]=a[j]; //ps:j--,i++(下行)比不了--j,++i快
while(a[i]<tmp&&i<j) ++i; //注意：这里用的不是">="或"<="而是">"<"，事实证明，前者会增加交换的次数，
```

```
做无用功~~~
    if(i<j) a[j--]=a[i];
    } //while
    a[i]=tmp;
    sort(a,st,i-1);
    sort(a,i+1,ed);
    } //if
    //这里不用return语句,会快一些
}
//由于以上的种种,程序在大的排序中 (N>=10e6) 优势越来越大--By LinuxKernel
[ '惭愧惭愧,当时我没拿出BT级别的数据,听你这么一说后试了一下,挂了14:31 2011年6月12日 (LinuxKernel)' ]
```

```
procedure qsort(l,r:longint);
var
    i,j,mid :      longint;
begin
    i:=l;
    j:=r;
    mid:=d[(l+r) div 2];
    repeat
        while d[i]<mid do
            inc(i);
        while d[j]>mid do
            dec(j);
        if i<=j then
            begin
                swap(d[i],d[j]);
                inc(i);
                dec(j);
            end;
        until i>j;
        if i<r then qsort(i,r);
        if l<j then qsort(l,j);
    end;
```

这有一段C++代码,并且用了模版 其中,cmp是比较函数,和stl中qsort中最后那个参数类似。

```
template<typename T>
int cmp(const void *e1, const void *e2) {
    if (*(T*)e1 > *(T*)e2) return 1;
    else if (*(T*)e1 < *(T*)e2) return -1;
    else return 0;
}

template<typename T>
int partition(T a[], int l, int r, int(*cmp)(const void*, const void*)) {
    int i = l, j = r-1;
    while (true) {
        while (i <= j && cmp(a+i, a+r) != 1) ++i;
        while (i <= j && cmp(a+j, a+r) == 1) --j;
        if (i > j) break;
        swap(a[i], a[j]);
    }
    swap(a[r], a[i]);
    return i;
}

template<typename T>
void qSort(T a[], int l, int r, int(*cmp)(const void*, const void*)) {
    if (l < r) {
        int q = partition(a, l, r, cmp);
        qSort(a, l, q-1, cmp);
        qSort(a, q+1, r, cmp);
    }
}
```

快速排序简单实现(Quick Sort by wssbwssbwssb)

[编辑]

```
Procedure Qst(i,j:integer);
Var ii,jj,xx:integer;
begin
    ii:=i;jj:=j;xx:=a[ii];
    repeat
        while (ii<jj) and (a[jj]<=xx) do dec(jj);a[ii]:=a[jj]; //先又指针因为xx已经把a[i]保存了,就可以覆盖
        while (ii<jj) and (a[ii]>=xx) do inc(ii);a[jj]:=a[ii];
    until ii=jj;
    a[ii]:=xx; //这句太重要了 我刚学的时候因为少了这句郁闷了一星期
    if i<ii-1 then qst(i,ii-1);
    if ii+1<j then qst(ii+1,j);
end;
```

```
end;
```

类似C库函数的Pascal快排，可以排序任意数组

```
type
  FCompair=Function (const cmpx,cmpy:pointer):boolean;
  PInt32=^LongInt;

procedure qsort(var first;count,dsizelongword;great:FCompair);
var
  tmp,cmp:pointer;

procedure qsortin(l,r:pointer);
var
  i,j:pointer;
begin
  i:=l;j:=r;move(l^,cmp^,dsizel);
  while i<=j do
  begin
    while great(j,cmp) do dec(j,dsizel);
    while great(cmp,i) do inc(i,dsizel);
    if i<=j then
    begin
      move(i^,tmp^,dsizel);move(j^,i^,dsizel);move(tmp^,j^,dsizel);
      inc(i,dsizel);dec(j,dsizel);
    end;
  end;
  if l<j then qsortin(l,j);
  if i<r then qsortin(i,r);
end;

begin
  getmem(tmp,dsizel);
  getmem(cmp,dsizel);
  qsortin(@first,@first+(count-1)*dsizel);
  freemem(tmp,dsizel);
  freemem(cmp,dsizel);
end;

function Int32cmp(const cmpx,cmpy:pointer):boolean;
begin
  exit(PInt32(cmpx)^>PInt32(cmpy)^);
end;

var
  a:array[1..15] of longint;
  i:longint;
begin
  randomize;
  for i:=1 to 15 do
    a[i]:=random(200);
  qsort(a[1],15,sizeof(longint),@Int32cmp);//一定要有@
  for i:=1 to 15 do
    write(a[i], ' ');
  writeln(a[15]);
end.
```

堆排序(Heap Sort)

[编辑]

堆：设有数据元素的集合 (R1,R2,R3,...Rn)它们是一棵顺序二叉树的结点且有

$$R_i \leq R_{2i} \text{ 和 } R_i \leq R_{2i+1} \text{ (或 } \geq \text{)}$$

堆的性质：堆的根结点上的元素是堆中的最小元素，且堆的每一条路径上的元素都是有序的。

堆排序的思想是：

- 1) 建初始堆（将结点[n/2],[ n/2]-1,...3,2,1分别调成堆）
- 2) 当未排序完时

输出堆顶元素，删除堆顶元素，将剩余的元素重新建堆。

程序如下：

```
const max=100;
```



```

type arrtype=array[1..max] of integer;
var a:arrtype;
    i,n,temp:integer;

procedure heap(var r:arrtype;nn,ii:integer);
var x,i,j:integer;
begin
    i:=ii;
    x:=r[i];
    j:=2*i;
    while j<=nn do
        begin
            if (j<nn) and (r[j]<r[j+1]) then j:=j+1;
            if x<r[j] then
                begin
                    r[i]:=r[j];i:=j;j:=2*i
                end
            else j:=nn+1;
        end;
    r[i]:=x;
end;
begin
    readln(n);

    for i:=1 to n do
        read(a[i]);
    writeln;
    for i:=n div 2 downto 1 do
        heap(a,n,i);
    for i:=n downto 2 do
        begin
            temp:=a[1];
            a[1]:=a[i];
            a[i]:=temp;
            heap(a,i-1,1);
        end;

    for i:=1 to n do
        write(a[i]:4);
    writeln;readln;
end.

```

这是一段C++程序： 其中，heap\_size和length作为全局变量，也可以作为参数传到函数中。

```

// heapsort

int heap_size ;
int length ;

inline int PARENT(int i) { return (i+1)/2-1 ; }
inline int LEFT(int i) { return 2*(i+1)-1 ; }
inline int RIGHT(int i) { return 2*(i+1) ; }

void build_max_heap(double *a)
{
    int i ;
    heap_size = length ;
    for (i = length/2-1; i>=0; --i)
    {
        max_heapify(a, i) ;
    }
}

void max_heapify(double *a, int i)
{
    int l = LEFT(i) ;
    int r = RIGHT(i) ;
    int largest ;
    if ((l<heap_size)&&(a[l]>a[i]))
    {
        largest = l ;
    }
    else
    {
        largest = i ;
    }
    if ((r<heap_size)&&(a[r]>a[largest]))
    {
        largest = r ;
    }
    if (i != largest)
    {
        swap(a[i], a[largest]) ;
        max_heapify(a, largest) ;
    }
}

```

```
}

void heapsort(double *a)
{
    int i ;
    build_max_heap(a) ;
    for (i = length-1; i>0; --i)
    {
        swap(a[0], a[i]) ;
        --heap_size ;
        max_heapify(a, 0) ;
    }
}
```

归并排序(Merge Sort)

[编辑]

归并就是将多个有序的数列合成一个有序的数列。将两个有序序列合并为一个有序序列叫二路归并(merge).归并排序就是n个长度为1的子序列,两两归并最后变为有序的序列。

1.二路归并

例1:将有序表L1=(1,5,7),L2=(2, 3,4,6,8, 9,10)合并一个有序表 L.

```
program gb;
const m=3;n=7;
type arrl1=array[1..m] of integer;
arrl2=array[1..n] of integer;
arrl=array[1..m+n] of integer;
var a:arrl1;
b:arrl2;
c:arrl;
i,j,k,r:integer;
begin
write('Enter l1(sorted):');
for i:=1 to m do read(a[i]);
write('Enter l2(sorted):');
for i:=1 to n do read(b[i]);
i:=1;j:=1;k:=0;
while (i<=m) and (j<=n) do
begin
k:=k+1;
if a[i]<=b[j] then begin c[k]:=a[i];i:=i+1; end
else begin c[k]:=b[j];j:=j+1;end;
end;
if i<=m then for r:=i to m do c[n+r]:=a[r];
if j<=n then for r:=j to n do c[m+r]:=b[r];
writeln('Output data:');
for i:=1 to m+n do write(c[i]:5);
writeln;
end.
```

2.归并排序

通常采取直接的区间操作（原版是用另外的一个数组来存储二分的数据，这样的话，数组开不了1000以上）

```
program MergeSort;
type
    arr1 = array[1..10000] of longint;
var
    n, i : longint;
    a, ans, temp: arr1;

    procedure merge(l, mid, r : longint; var c : arr1);
    var
        i, j, k, p: longint;
    begin
        i := l;
        j := mid + 1;
        k := l - 1;
        temp := c;
        while (i <= mid) and (j <= r) do
            begin
                inc(k);
                if temp[i] < temp[j] then
                    begin
                        c[k] := temp[i];
                        inc(i);
                    end
                else
                    begin
                        c[k] := temp[j];
                        inc(j);
                    end;
            end;
        if i <= mid then
            begin
                for p := i to mid do
                    begin
                        inc(k);
                        c[k] := temp[p];
                    end;
            end;
        if j <= r then
            begin
                for p := j to r do
                    begin
                        inc(k);
                        c[k] := temp[p];
                    end;
            end;
        end;

    procedure mergesort(var b: arr1; l, r : longint);
    var
        mid : longint;
    begin
        if l = r then
            begin
                b[l] := a[l];
                exit;
            end;
        mid := (l + r) div 2;
        mergesort(b, l, mid);
        mergesort(b, mid+1, r);
        merge(l, mid, r, b);
    end;

begin
    readln(n);
    for i := 1 to n do read(a[i]);
    mergesort(ans, 1, n);

    for i := 1 to n do writeln(ans[i]);
end.
```

3、推广：统计逆序对个数

只需将归并排序程序中加一句即可

```
if temp[i]<temp[j] then
begin
    c[k]:=temp[i];
    inc(i);
end
```

```
else
begin
  c[k]:=temp[j];
  inc(j);
end;
inc(t,mid-i+1); //这里加一句,最后输出t,即为逆序对的个数
```

线性排序算法

[编辑]

计数排序(Counting Sort)

[编辑]

基本思想是对于序列中的每一元素x,确定序列中小于x的元素的个数。

例：n个整数序列且每个值在[1,m],排序之。

```
program jspx;
const m=6;n=8;
var i,j:integer;
    a,b:array[1..n] of integer;
    c:array[1..m] of integer;
begin
  writeln('Enter data:');
  for i:=1 to n do read(a[i]);
  for i:=1 to m do c[i]:=0;
  for i:=1 to n do c[a[i]]:=c[a[i]]+1;
  for i:=2 to m do c[i]:=c[i]+c[i-1];
  for i:=n downto 1 do
    begin
      b[c[a[i]]]:=a[i];
      c[a[i]]:=c[a[i]]-1;
    end;
  writeln('Sorted data:');
  for i:= 1 to n do
    write(b[i]:6);
  end.
```

桶排序(Bin Sort)

[编辑]

桶排序的思想是若待排序的记录的关键字在一个明显有限范围内(整型)时,可设计有限个有序桶,每个桶装入一个值,顺序输出各桶的值,将得到有序的序列。

例：输入n个0到100之间的整数，由小到大排序输出。

```
program Bin_sort;
const n=7;
var b:array[0..100] of integer;
    k:0..100;
    i:integer;
begin
  write('Enter date:(0-100)');
  for i:=0 to 100 do b[i]:=0;
  for i:= 1 to n do
    begin
      read(k);
      b[k]:=b[k]+1;
    end;
  writeln('Output data:');
  for i:=0 to 100 do
    while b[i]>0 do begin write(i:6);b[i]:=b[i]-1 end;
  writeln;
end.
```

基数排序(Radix Sort)

[编辑]

基本思想是对n个元素依次按k,k-1,...1位上的数字进行桶排序。

```
program jspx;
const n=8;
type link=^node;
    node=record
      data:integer;
      next:link;
    end;
var i,j,l,m,k:integer;
    a:array[1..n] of integer;
```

```
s:string;
q,head:array[0..9] of link;
p,pl:link;
begin
writeln('Enter data:');
for i:=1 to n do read(a[i]);
for i:=5 downto 1 do
begin
for j:=0 to 9 do
begin
new(head[j]);
head[j].next:=nil;
q[j]:=head[j]
end;
for j:=1 to n do
begin
str(a[j],s);
for k:=1 to 5-length(s) do
s:='0'+s;
m:=ord(s[i])-48;
new(p);
p.data:=a[j];
p.next:=nil;
q[m].next:=p;
q[m]:=p;
end;
l:=0;
for j:=0 to 9 do
begin
p:=head[j];
while p.next<>nil do
begin
l:=l+1;p1:=p;p:=p.next;dispose(p1);a[l]:=p.data;
end;
end;
end;
writeln('Sorted data:');
for i:= 1 to n do
write(a[i]:6);
end.
```

几种排序算法的比较和选择 [\[编辑\]](#)

稳定性比较 [\[编辑\]](#)

插入排序、冒泡排序、二叉树排序、二路归并排序及其他线形排序是稳定的。  
选择排序、希尔排序、快速排序、堆排序是不稳定的。

时间复杂性比较 [\[编辑\]](#)

插入排序、冒泡排序最优为 $O(n)$ ,最坏为 $O(n^2)$ ,平均 $O(n^2)$  ;  
快速排序最优为 $O(n\log n)$ ,最坏为 $O(n^2)$ ,平均 $O(n\log n)$ ;  
堆排序最优为 $O(n\log n)$ ,最坏为 $O(n\log n)$ ,平均 $O(n\log n)$ ;  
线形排序的时间复杂性为 $O(n)$ 。

辅助空间的比较 [\[编辑\]](#)

线形排序、归并排序的辅助空间为 $O(n)$ ,快速排序的辅助空间为 $O(\log n)$ ,其它排序的辅助空间为 $O(1)$ 。

其它比较 [\[编辑\]](#)

插入、冒泡排序的速度较慢，但参加排序的序列局部或整体有序时，这种排序能达到较快的速度。  
反而在这种情况下，快速排序慢了。 当 $n$ 较小时，对稳定性不作要求时宜用选择排序，对稳定性有要求时宜用插入或冒泡排序。  
若待排序的记录的关键字在一个明显有限范围内时,且空间允许时用桶排序。  
当 $n$ 较大时，关键字元素比较随机，对稳定性没要求宜用快速排序。  
当 $n$ 较大时，关键字元素可能出现本身是有序的，对稳定性有要求时，空间允许的情况下宜用归并排序。  
当 $n$ 较大时，关键字元素可能出现本身是有序的，对稳定性没有要求时宜用堆排序。



此页面已被浏览过56,441次。 本页面由NOCOW用户Samzhu19于2012年8月29日 (星期三) 17:31做出最后修改。 在cosechy@gmail.com、NOCOW用户785045613和Magic-lucifer、NOCOW匿名用户61.135.172.68和其他的工人基础上。 本站全部文字内容使用GNU Free Documentation License 1.2授权。 [隐私权政策](#) [关于NOCOW](#) [免责声明](#) [陕ICP备09005692号](#)

