浅谈各类排序算法

在哔哩哔哩刷视频时刷到了排序算法图示的视频，见到了许多冷门的排序算法，故突发奇想，于是本文诞生。

灵感来源：计算机专业不得不看的15种排序算法，7分钟动画演示*哔哩哔哩*bilibili,256种排序算法，全网最全的排序算法*哔哩哔哩*bilibili。

对原算法集合的补充：排序算法 – 玩算法 (wansuanfa.com)。

上述资料中部分算法找不到资料，部分为相同思想反复运用，以及混合排序类内排列组合，经过整理如下：

在本篇文章中，你将会见到如下排序算法：

- $O(n^2)$ 时间复杂度的排序：冒泡排序，选择排序，插入排序，地精排序，煎饼排序，圈排序。
- 对 $O(n^2)$ 排序的优化：鸡尾酒排序，梳排序，二分插入排序，希尔排序。
- $O(n\log n)$ 时间复杂度的排序：锦标赛排序，归并排序，堆排序，快速排序。
- 优化：桶排序，多路归并排序，内省排序，tim 排序，模式破坏快速排序 (pdqsort)。
- 不基于比较的排序：珠排序，计数排序，基数排序。
- 便于并行计算的排序：奇偶排序，双调排序。
- 杂项：猴子排序，蠢猴排序，全排列排序，慢速排序，臭皮匠排序，睡眠排序。

注：给出的代码中除睡眠排序以外都是单调不增的。

$O(n^2)$ 时间复杂度排序

这类排序浅显易懂，符合知觉，适合初学者。许多初学的手写排序都在其中，也有很多人曾自己想到某种排序方法。

冒泡排序

最经典且最入门的排序算法，因为其好理解，相信是大家的第一种手写排序。

冒泡排序属于排序网络的一种，即该算法不依赖待排序数组的取值，可以大致理解为只通过 $(x,y)\leftarrow(\min(x,y),\max(x,y))$ 和指针偏移完成排序。

我们可以把它的每一次循环视作一个泡泡从水下的某个位置冒上去，这也是它名字的来源。

```cpp
#include <bits/stdc++.h>
using namespace std;
int constexpr N=100100;
int a[N],n;
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<=n;i++){
        for(int j=1;j<n;j++)if(a[i]<a[i+1])swap(a[i],a[i+1]);
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

选择排序

同样经典的一种排序方法，选择最大/最小的移动到数组的一端，反复执行直到有序。

也可以同时选择最大最小并分别移到两端，不过会有一些边界问题，感觉实现难度会有影响。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++){
        for(int j=i+1;j<=n;j++){
            if(a[j]>a[i])swap(a[j],a[i]);
        }
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

插入排序

同样经典的一种排序，是笔者的第一种手写排序。

其原理类似于打扑克牌时，新摸到一张牌，把它插入到手牌的正确位置中。

因为它实现逻辑中有类似剪枝的成分，因此时最快的 $O(n^2)$ 排序算法。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++){
        for(int j=i;j>1;j--){
            if(a[j]>a[j-1])swap(a[j],a[j-1]);
            else break;
        }
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

地精排序

逻辑与插入排序几乎完全一样，算是写法上的改变，是唯一只需要一个循环的排序算法。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
```

```
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;){
        if(i>1&&a[i]>a[i-1])swap(a[i],a[i-1]),i--;
        else i++;
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

煎饼排序

每次找出最值，将其翻转到正确位置。

可以用数据结构优化翻转和找值，不过都上数据结构了，为什么不直接用数据结构处理有序数列呢?

该算法算是对选择排序进行一种游戏性的改变，作为排序本身价值不大，不过可以开发出很多题目。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++){
        int pos=i;
        for(int j=i;j<=n;j++)if(a[j]>a[pos])pos=j;
        for(int j=i;j<=(i+pos)/2;j++)swap(a[j],a[pos-j+i]);
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

圈排序

交换次数最少的排序。

该排序算法首先要明白圈的概念。

我们待排数组 $[4,5,3,1,2]$，排好序后为 $[5,4,3,2,1]$。原数组中每个数字应该到达的位置为 $[2,1,3,5,4]$。

我们从前往后看原数组，4 应到达位置 2，但此处为 5，5 应到达位置 1,但此处为 4，而 4 已经遍历过，故 $[4,5]$ 为一个圈。

同理有三个圈 $[4,5],[3],[1,2]$，我们需要对每个圈内的元素排序。

不过我们由已经排好序的数组的得到圈固然不能作为最终的算法。

我们遍历每个数，然后每次再遍历数组计算该数应该在哪个位置，交换并重复该流程。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
```

```
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++){
        int pos=0;
        for(int j=1;j<=n;j++){
            if(a[i]<a[j]||(a[i]==a[j]&&i>=j))pos++;
        }
        if(pos!=i){
            swap(a[pos],a[i]);
            i--;
        }
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

对 $O(n^2)$ 排序的部分优化

鸡尾酒排序

对冒泡排序的常数优化，在指针带着最大值移动到一个端点后，再让它带着最小值走回去做往返跑。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1,k=n;i<k;i++,k--){
        int j=1;
        for(;j<k;j++)if(a[j]<a[j+1])swap(a[j],a[j+1]);
        for(;j>i;j--)if(a[j]>a[j-1])swap(a[j],a[j-1]);
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

梳排序

可以称作缩小增量法冒泡排序，设置了一个增量 s，对下标为 $i+ks$ 的数据分别冒泡排序，再缩小 s 重复，直到 s 为 1。

期望时间复杂度 $O(\frac{n^2}{2^p})$，p 为缩小增量的次数。但其最坏时间复杂度还是 $O(n^2)$。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    int k=sqrt(n);
```

```
    for(;k>=1;k/=1.3){
        for(int i=1;i<=n/k;i++){
            int cnt=0;
            for(int j=1;j+k<=n;j+=k){
                if(a[j]<a[j+k])cnt++,swap(a[j],a[j+k]);
            }
            if(cnt==0)break;
        }
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

二分插入排序

直觉给到的一种优化，可以在 $O(\log n)$ 的时间找到每个数应该插入到的位置，不过由于整体移动的时间仍然为 $O(n)$ 故整体上是一种劣化。

由于感觉这份代码比较唐氏，故不提供。

图书馆排序

参考：[几种有趣的不常见排序-CSDN博客](#).

考虑到二分插入排序在移动空间上产生了大量开销，那么我们不放把数组变的稀疏，在原先的两个数中间加入一些空位，如果二分倒的位置有空位，那么直接插入，否则暴力移动。

感觉太过乱搞，故同不提供代码。

希尔排序

缩小增量插入排序，和梳排序不同的是由于插入排序内带的"剪枝"会提高缩小增量带来的收益。其时间复杂度难以计算，据资料表明在 $O(n^{1.3})$ 和 $O(n^2)$ 之间，中等数据下表现优秀。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int k=n;k>=1;k/=2){
        for(int i=1;i<=n;i++){
            for(int j=i;j-k>0;j-=k){
                if(a[j]>a[j-k])swap(a[j],a[j-k]);
                else break;
            }
        }
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

$O(n\log n)$ 时间复杂度排序

算法名称	最坏时间复杂度	是否原地排序	是否稳定排序
锦标赛排序	$O(n\log n)$	否	是
归并排序	$O(n\log n)$	否	是
堆排序	$O(n\log n)$	是	否
快速排序	$O(n^2)$	是	否

锦标赛排序

基于选择排序，利用了数据结构优化。

我们对原数组建线段树，节点维护区间最大值，因酷似锦标赛的胜负树，故得名锦标赛排序。

我们取出线段树根节点的值加入数组，并将对应叶子节点修改为最小值（终身禁赛），重复 n 次得到有序数组。

```
#include<bits/stdc++.h>
using namespace std;
constexpr int N=1e5+100;
int mx[N<<2],mp[N<<2];
int n,a[N];
void build(int i,int l,int r){
    if(l==r)return mx[i]=a[l],mp[i]=l,void();
    int mid=(l+r)>>1;
    build(i<<1,l,mid);
    build(i<<1|1,mid+1,r);
    if(mx[i<<1]>=mx[i<<1|1])mp[i]=mp[i<<1];
    else mp[i]=mp[i<<1|1];
    mx[i]=max(mx[i<<1],mx[i<<1|1]);
}
void upd(int i,int l,int r,int p,int x){
    if(l==r)return mx[i]=x,void();
    int mid=(l+r)>>1;
    if(mid>=p)upd(i<<1,l,mid,p,x);
    else upd(i<<1|1,mid+1,r,p,x);
    if(mx[i<<1]>=mx[i<<1|1])mp[i]=mp[i<<1];
    else mp[i]=mp[i<<1|1];
    mx[i]=max(mx[i<<1],mx[i<<1|1]);
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    build(1,1,n);
    for(int i=1;i<=n;i++){
        a[i]=mx[1];
        upd(1,1,n,mp[1],0);
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

归并排序

我们可以借助一个辅助数组 $O(n)$ 地实现两个有序数组的合并，而每个长度为 1 的数组都是有序的。

因此我们不难想到每次把数组不断二分，直到长度为 1。在回溯时将二分的前后数组合并，最终可以得到有序的数组。

时间复杂度 $T(n)=2T(\frac{n}{2})+O(n)$，由主定理得 $T(n)=O(n\log n)$。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N],tmp[N];
void sort(int l,int r){
    if(l==r)return;
    int mid=(l+r)>>1;
    sort(l,mid);
    sort(mid+1,r);
    int i=l,j=mid+1,pos=l;
    for(;i<=mid&&j<=r;){
        if(a[i]>a[j])tmp[pos++]=a[i++];
        else tmp[pos++]=a[j++];
    }
    while(i<=mid)tmp[pos++]=a[i++];
    while(j<=r)tmp[pos++]=a[j++];
    for(int i=l;i<=r;i++)a[i]=tmp[i];
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    sort(1,n);
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

堆排序

不少人认知中得堆排序是把待排序数组扔到一个堆中，再一个一个取出。这是一个很大的误区，它会为堆排序带来 $O(n)$ 的额外空间，使其各方面都变得不强于归并排序。

将数组视作一棵完全二叉树，我们先 $O(n\log n)$ 的将原数组调整到符合堆的形态，把堆顶元素移到数组一端，再 $\log n$ 地修正堆，继续重复，直至数组有序。

这样不消耗额外空间且最坏 $O(n\log n)$ 的时间复杂度，在归省排序中的利用其优化保障了快速排序的时间复杂度。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=1e5+100;
int n;
int a[N];
void heap(int u,int n){
    int tmp=a[u];
    for(int i=u*2;i<=n;i=i*2){
        if(i<n&&a[i]>a[i+1])i++;
```

```
        if(a[i]<tmp){
            a[u]=a[i];
            u=i;
        }else break;
    }
    a[u]=tmp;
}
bool Med;
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=n/2;i>0;i--)heap(i,n);
    for(int i=1;i<=n;i++){
        swap(a[1],a[n-i+1]);
        heap(1,n-i);
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

快速排序

考虑到归并排序中，我们产生额外空间消耗的原因是要合并两个有序数组时需要缓冲数组。那么如果待合并的两个数组的值域满足我们排序的关系，那么我们便不需要费力合并两个数组。

而把数组分为两个值域区间我们可以考虑使用双指针，简单维护即可。

类似于归并的递归于回溯操作略过。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
void sort(int l,int r){
    if(l>=r)return;
    int i=l,j=r,tmp=a[l];
    while(i<j){
        while(i<j&&a[j]<=tmp)j--;
        while(i<j&&a[i]>=tmp)i++;
        swap(a[i],a[j]);
    }
    swap(a[i],a[l]);
    sort(l,i-1);
    sort(j+1,r);
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    sort(1,n);
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

对上述排序算法的优化

桶排序

我们把原数组的值域范围分成 k 部分，将每个范围内的数据分别排序，假设使用快速排序且值域均匀，那么时间复杂度 $T(n)=k\times O(\frac{n}{k}\log\frac{n}{k})=O(n+n\log\frac{n}{k})$。

在 $n=k$ 时时间复杂度最优，此时桶排序等于计数排序。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100,B=210;
int n,a[N];
int z[B][N],l[B];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    int mx=a[1],mn=a[1];
    for(int i=1;i<=n;i++)mx=max(a[i],mx),mn=min(a[i],mn);
    int v=(mx-mn)/100+1;
    for(int i=1;i<=n;i++){
        int j=(a[i]-mn)/v;
        z[j][++l[j]]=a[i];
    }
    for(int i=0;i<B;i++)sort(z[i]+1,z[i]+1+l[i]);
    int pos=n;
    for(int i=0;i<B;i++)for(int j=1;j<=l[i];j++)a[pos--]=z[i][j];
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

多路归并排序

既然能够二分，那为什么不多分几段呢?

考虑到找最值的操作，我们可以在 $O(kn)$ 的时间复杂度合并总长度为 n 的 k 个数组。

于是有多路归并排序，时间复杂度 $T(n)=k\times T(\frac{n}{k})+O(kn)=nk\log_k n$

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100,B=4;
int n,a[N],z[N];
void sort(int l,int r){
    if(l==r)return;
    int len=r-l+1;
    int b=min(B,len);
    int k=len/b;
    int ll[b+1],rr[b+1];
    for(int i=1;i<b;i++){
        ll[i]=l+(i-1)*k;
        rr[i]=l+i*k-1;
        sort(ll[i],rr[i]);
    }
```

```
    ll[b]=rr[b-1]+1;
    rr[b]=r;
    sort(ll[b],rr[b]);
    for(int i=l;i<=r;i++){
        int mx=INT_MIN,pos=0;
        for(int j=1;j<=b;j++){
            if(ll[j]<=rr[j]&&a[ll[j]]>mx){
                mx=a[ll[j]];
                pos=j;
            }
        }
        z[i]=mx;
        ll[pos]++;
    }
    for(int i=l;i<=r;i++)a[i]=z[i];
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    sort(1,n);
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

内省排序

对快速排序的优化。

快速排序在数组几乎有序的情况会退化成 $O(n^2)$，于是我们在递归深度达到一定程度后使用堆排序保障其时间复杂度。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N],d;
void heap_sort(int*a,int n){
    auto heap=[&](int u,int n){
        int tmp=a[u];
        for(int i=u*2;i<=n;i=i*2){
            if(i<n&&a[i]>a[i+1])i++;
            if(a[i]<tmp){
                a[u]=a[i];
                u=i;
            }else break;
        }
        a[u]=tmp;
    };
    for(int i=n/2;i>0;i--)heap(i,n);
    for(int i=1;i<=n;i++){
        swap(a[1],a[n-i+1]);
        heap(1,n-i);
    }
}
void sort(int l,int r,int dep){
    if(l>=r)return;
```

```
    if(dep>=d){
        heap_sort(a+l-1,r-l+1);
        return;
    }
    int i=l,j=r,tmp=a[l];
    for(;i<j;){
        while(i<j&&a[j]<=tmp)j--;
        while(i<j&&a[i]>=tmp)i++;
        swap(a[i],a[j]);
    }
    swap(a[i],a[l]);
    sort(l,i-1,dep+1);
    sort(i+1,r,dep+1);
}
signed main(){
    cin >> n;
    d=__lg(n);
    for(int i=1;i<=n;i++)cin >> a[i];
    sort(1,n,0);
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

tim 排序

内省排序速度快，但不是稳定排序，考虑到归并排序具有稳定性，我们做一个基于归并的混合排序。

混合的思路大同小异，在深度大的时候换用其他小数据优秀的算法。因为同为稳定排序的锦标赛排序在小数据下表现不够优秀，因此我们选择 $O(n^2)$ 中最快的插入排序，它同样是稳定的排序。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100,D=16;
int n,a[N],len,z[N];
void sort(int l,int r,int dep){
    if(l==r)return;
    if(dep==D){
        for(int i=l;i<=r;i++){
            for(int j=i;j>l;j--){
                if(a[j]>a[j-1])swap(a[j],a[j-1]);
                else break;
            }
        }
        return;
    }
    int mid=(l+r)>>1;
    sort(l,mid,dep+1);sort(mid+1,r,dep+1);
    if(r-mid<=len){
        for(int i=l;i<=r;i++){
            for(int j=i;j>l;j--){
                if(a[j]>a[j-1])swap(a[j],a[j-1]);
                else break;
            }
        }
    }else{
```

```
        int pos=l,i,j;
        for(i=l,j=mid+1;i<=mid&&j<=r;){
            if(a[i]>a[j])z[pos++]=a[i++];
            else z[pos++]=a[j++];
        }
        while(i<=mid)z[pos++]=a[i++];
        while(j<=r)z[pos++]=a[j++];
        for(int i=l;i<=r;i++)a[i]=z[i];
    }
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    len=__lg(n);
    sort(1,n,0);
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

模式破坏快速排序 (pdqsort)

参考自：字节跳动最佳实践：打造 Go 语言最快的排序算法 - 环信 (easemob.com),【算法】pdqsort - 简书 (jianshu.com).

我们考虑三种比较优秀的原地排序：

	最好	平均	最坏
插入排序	$O(n)$	$O(n^2)$	$O(n^2)$
堆排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
快速排序	$O(n\log n)$	$O(n\log n)$	$O(n^2)$

pdqsort 将三种排序结合（以下决策优先级为罗列顺序）：

1. 在序列足够短时，使用插入排序；
2. 在快速排序效果不佳时使用堆排序；
3. 正常情况使用一定改进的快速排序。

再考虑到，快速排序的效率与其基准值有关，若基准值接近序列最值，则效率低下，所以我们考虑改进基准值选择。

一般在长度较小的数组选取三个数，取中位数。在较长的数组中选取九个数取中位数。

同时，若我们选取的数有单调关系，那么我们进行有限次数的插入排序。若有限次数后未能满足条件，那么再按选定的基准快排。

可靠的 pdqsort 可以参考洛谷排序模板最优解代码，这里给出笔者的简单实现。

受笔者理解程度和代码能力影响，本代码实际效率弱于内省排序。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
```

```
void heap_sort(int*a,int n){
    auto heap=[&](int u,int n){
        int tmp=a[u];
        for(int i=u*2;i<=n;i=i*2){
            if(i<n&&a[i]>a[i+1])i++;
            if(a[i]<tmp){
                a[u]=a[i];
                u=i;
            }else break;
        }
        a[u]=tmp;
    };
    for(int i=n/2;i>0;i--)heap(i,n);
    for(int i=1;i<=n;i++){
        swap(a[1],a[n-i+1]);
        heap(1,n-i);
    }
}
int inrt(int*a,int n,int cnt=-1){
    int c=0;
    for(int i=1;i<=n;i++){
        for(int j=i;j>1;j--){
            if(c==cnt)return -1;
            if(a[j]>a[j-1])swap(a[j-1],a[j]),c++;
            else break;
        }
    }
    return c;
}
void sort(int l,int r,int lmt=0){
    if(r-l+1<=10){
        inrt(a+l-1,r-l+1);
        return;
    }
    if(lmt>8){
        heap_sort(a+l-1,r-l+1);
        return;
    }
    int k=(r-l+1)>=50?9:3;
    int p[k];
    for(int i=0;i<k;i++)p[i]=a[rand()%(r-l+1)+l];
    int cnt=inrt(p-1,k);
    if(cnt==0){
        cnt=inrt(a+l-1,r-l+1,(r-l+1)/8);
        cnt=(cnt==-1);
    }
    if(cnt){
        for(int i=1;i<=(r-l+1)/8;i++)swap(a[rand()%(r-l+1)+l],a[rand()%(r-
l+1)+l]);
        int tmp=p[k/2];
        for(int i=l;i<=r;i++)if(a[i]==tmp){swap(a[l],a[i]);break;}
        int i=l,j=r;
        for(;i<j;){
            while(i<j&&a[j]<=tmp)j--;
            while(i<j&&a[i]>=tmp)i++;
            swap(a[i],a[j]);
```

```
        }
        swap(a[l],a[i]);
        int del=min(r-j+1,i-l+1);
        if(del<=(r-l+1)/8)lmt++;
        sort(l,i-1,lmt);
        sort(i+1,r,lmt);
    }
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    sort(1,n);
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

不基于比较的排序

珠排序

现在有 V 个杆子，我们对于每个元素 a_i，我们给前 a_i 个杆子上分别加一个珠子。

让珠子随重力落下，那么最底层的珠子数为最大值，次底层为次大值。

时间复杂度 $O(nV)$。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100,V=200500;
int n,a[N];
int d[V];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++){
        for(int j=1;j<=a[i];j++){
            d[j]++;
        }
    }
    for(int i=1;i<=n;i++){
        a[i]=0;
        for(int j=1;j<V;j++){
            if(d[j]==0)continue;
            a[i]=j;
            d[j]--;
        }
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

计数排序

我们用值域数组统计每个数字的部分信息（如出现次数），最后按数量把他们加入原数组即可。

时空复杂度均为 $O(V)$，其中 V 为值域。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100,V=500100;
int n,a[N],v[V];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    int mn=a[1],mx=a[1],pos=n;
    for(int i=1;i<=n;i++)mn=min(mn,a[i]),mx=max(mx,a[i]);
    for(int i=1;i<=n;i++)v[a[i]-mn]++;
    for(int i=0;i<=mx-mn;i++)while(v[i])a[pos--]=i+mn,v[i]--;
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

基数排序

我们利用桶排序和计数排序的思想。

类比高精度如何比较两数大小，先给位数小的前面补满 0，然后由高位到低位地比较，第一个不同的位数的大小关系即为两数的大小关系。

于是我们考虑以某一位的大小为比较函数排序，然后对每一位都执行这一过程。

因为“一位”的值域比较小，因此我们可以计数排序。

由于排序造成的影响是覆盖前一次的影响的，所以我们应该由低位到高位做上述过程，以保证高位的影响最大。

时间复杂度 $O(n\log_k V)$，其中 k 为进制数，V 为值域大小。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100,B=16;
int n,a[N];
int v[B][N],l[B],mx;
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++)mx=max(a[i],mx);
    for(int b=1;b<mx;b*=B){
        for(int i=0;i<B;i++)l[i]=0;
        for(int i=1;i<=n;i++){
            int j=a[i]/b%B;
            v[j][++l[j]]=a[i];
        }
        int pos=0;
        for(int i=B-1;i>=0;i--){
            for(int j=1;j<=l[i];j++){
                a[++pos]=v[i][j];
```

```
            }
        }
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

便于并行计算的排序

这类排序都属于排序网络。

奇偶排序

进行 n 轮操作，一轮修改所有下标为 $2k+1, 2k+2$ 的对，再一轮修改 $2k, 2k+1$，交替往复。

对冒泡排序的改进，减少了一定的操作次数，时间复杂度仍然为 $O(n^2)$。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++){
        for(int j=1;j+(i&1)<=n;j+=2){
            if(a[j+(i&1)]<a[j+(i&1)+1])swap(a[j+(i&1)],a[j+(i&1)+1]);
        }
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

双调排序

参考自[算法 - 并行排序算法之双调排序（Bitonic Sort）](#).

我们将两个单调性不同的序列直接拼接形成的序列称作双调序列，特殊地，所有单调序列都是双调序列。

> Batcher 定理：
>
> 将任意一个长度为 n 的双调序列 A 分为等长的两半 X、Y，将 X 中的元素与 Y 中的元素一一按照原序进行比较，也就是 a_i 与 $a_{i+\frac{n}{2}}(i \le \frac{n}{2})$ 进行比较，将较大者放入 MAX 序列中，较小者放入 MIN 序列，得到的 MAX 和 MAX 序列仍然是双调序列，并且 MAX 序列中任意一个元素是不小于 MIN 序列中任意一个元素的。

将上述过程记作`bitonicMerge`。

那么我们可以当我们有一个两半等长的双调序列时，我们调用`bitonicMerge`，得到两个新的序列，然后以类似快速排序的方式递归，即可得到有序数组。

然后我们需要知道如何将初始数组变成双调序列。

首先长度不大于 3 的序列一定是双调的。那么我们将相邻长度为 2 的子序列先分别按升序降序交错排序，那么它们两两成为长度为 4 的双调序列，即可再次升序降序交错排序，如此重复即可。

时间复杂度 $T(n)=2T(\frac{n}{2})+D(n),D(n)=2D(\frac{n}{2})+O(\frac{n}{2})$，即 $T(n)=O(n\log^2 n)$。

```
#include<bits/stdc++.h>
using namespace std;
constexpr int N=2e5+100;
int n,a[N];
void merge(int l,int r,bool rule){
    if(l^r){
        int mid=(l+r)>>1;
        int del=mid-l+1;
        for(int i=l;i<=mid;i++){
            if((a[i]>a[i+del])==rule)swap(a[i],a[i+del]);
        }
        merge(l,mid,rule);
        merge(mid+1,r,rule);
    }
}
void sort(int l,int r,bool rule){
    if(l^r){
        int mid=(l+r)>>1;
        sort(l,mid,!rule);
        sort(mid+1,r,rule);
        merge(l,r,rule);
    }
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    int m=1;
    while(m<n)m<<=1;
    swap(n,m);
    sort(1,n,false);
    for(int i=1;i<=m;i++)cout << a[i] << ' ';
    return 0;
}
```

杂项

意识模糊间，你突然变作排序宇宙的猴子，接下来，你将经历神奇的冒险。

猴子排序

> 无限猴子定律：
>
> 在无穷长的时间后，即使是随机打字的猴子也可以打出一些有意义的单词，比如，cat, dog。因此，可以类推，会有一个足够幸运的猴子或连续或不连续地打出一本书，即使其几率比连续抓到一百次同花顺还要低。但在足够长的时间（长到你数不清它的秒数有多少位）后，其发生是必定的。

有一只猴子，它随机的拿起序列中的两个数字并交换位置。那么只要它交换的次数足够多，则最终总得到一个有序的序列。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
```

```
bool check(){
    for(int i=2;i<=n;i++)if(a[i]>a[i-1])return 1;
    return 0;
}
mt19937 rd(time(0));
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    while(check()){
        int x=rd()%n+1;
        int y=rd()%n+1;
        swap(a[x],a[y]);
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

蠢猴排序

一只愚蠢的猴王想学会插入排序，但他脑子实在不够，他在插入一个元素时会执行猴子排序的操作。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
mt19937 rd(time(0));
void sort(int n){
    auto check=[&](){
        for(int i=2;i<=n;i++)if(a[i]>a[i-1])return 1;
        return 0;
    };
    while(check()){
        int x=rd()%n+1;
        int y=rd()%n+1;
        swap(a[x],a[y]);
    }
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++){
        sort(i);
    }
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

全排列排序

一只聪明的猴子知道了自己要干什么，但他的脑容量甚至不够写出上述的任何排序算法，于是它枚举数组的每种排列，并判断是否有序。

```
#include <bits/stdc++.h>
using namespace std;
```

```
constexpr int N=100100;
int n,a[N],p[N],b[N];
bool check(){
    for(int i=1;i<=n;i++)b[p[i]]=a[i];
    for(int i=2;i<=n;i++)if(b[i]>b[i-1])return 0;
    return 1;
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    for(int i=1;i<=n;i++)p[i]=i;
    do{
        if(check())break;
    }while(next_permutation(p+1,p+1+n));
    for(int i=1;i<=n;i++)cout << b[i] << ' ';
    return 0;
}
```

慢速排序

拥有智慧的猴子首领们看不下去了。

猴子首领们学过选择排序。大首领要找到序列的最值，但他懒得遍历序列，于是安排两个小首领将前 $\frac{n}{2}$ 和后 $\frac{n}{2}$ 的最值找出来，自己将其放到序列尾。而小首领同样不想遍历序列，于是层层外包……

算法流程：

- 递归排序前 $\frac{n}{2}$ 和后 $\frac{n}{2}$，取更优的最值移到序列尾。
- 排序前 $n-1$ 个元素。

时间复杂度 $T(n)=2T(\frac{n}{2})+T(n-1)+O(1)$，难以计算，但可以估计是指数级算法。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
void sort(int l,int r){
    if(l==r)return;
    int mid=(l+r)>>1;
    sort(l,mid);sort(mid+1,r);
    if(a[mid]<a[r])swap(a[mid],a[r]);
    sort(l,r-1);
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    sort(1,n);
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

臭皮匠排序

路过这里的三个臭皮匠嘲笑着猴子的懒惰，然后进行了自己的排序方法。

臭皮匠们喜欢互帮互助。

一个臭皮匠将序列首尾项对比并交换，然后请另外两人分别将前 $\frac{2}{3}$ 和后 $\frac{2}{3}$ 排好序，自己再对前 $\frac{2}{3}$ 排一遍序。

时间复杂度 $T(n)=3T(\frac{2}{3}n)+O(1)=O(n^{log_{1.5}3})\approx O(n^{2.7})$。

```
#include <bits/stdc++.h>
using namespace std;
constexpr int N=100100;
int n,a[N];
int sort(int l,int r){
    if(a[l]<a[r])swap(a[l],a[r]);
    if(r-l<=1)return;
    int k=(r-l+1)/3;
    sort(l,r-k);
    sort(l+k,r);
    sort(l,r-k);
}
signed main(){
    cin >> n;
    for(int i=1;i<=n;i++)cin >> a[i];
    sort(1,n);
    for(int i=1;i<=n;i++)cout << a[i] << ' ';
    return 0;
}
```

睡眠排序

排序了许久，猴子们都累了。

睡觉前，猴王把序列的每个数分配给一只猴子，并把这只猴子的闹钟时间改为这个数字，计划第二天按猴子起床统计排序后序列。

```
#include<bits/stdc++.h>
#include<windows.h>
using namespace std;
int n,a[10000];
void* p(void *ai){
    int a = *((int*)ai);
    Sleep(a*n);
    cout << a << " ";
    return ai;
}
pthread_t pt[10000];
int main(){
    cin >> n;
    for(int i=1;i<=n;i++){
        cin >> a[i];
        pthread_create(&pt[i], NULL, p, (void*)&a[i]);
    }
```

```
    pthread_exit(NULL);
    return 0;
}
```