第十二节课

一维数组应用之简单排序

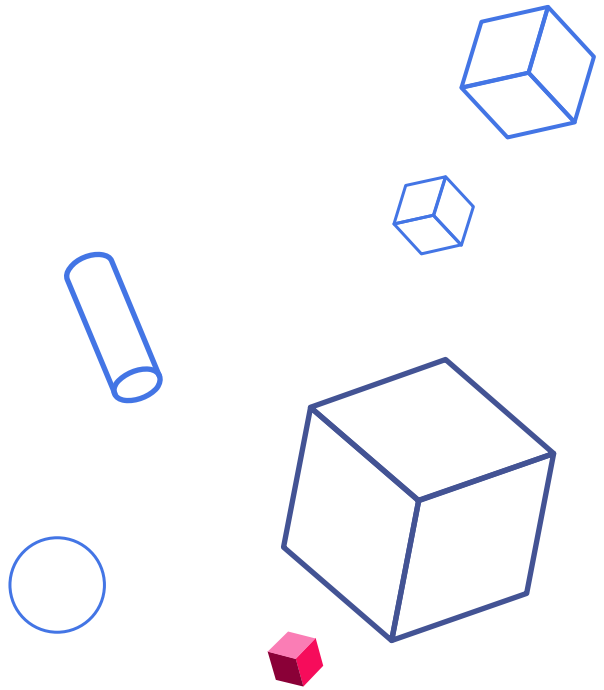
西南大学附属中学校

屈奕池

排序算法的要点

- 排序的范围
- 排序的条件

01.选择排序



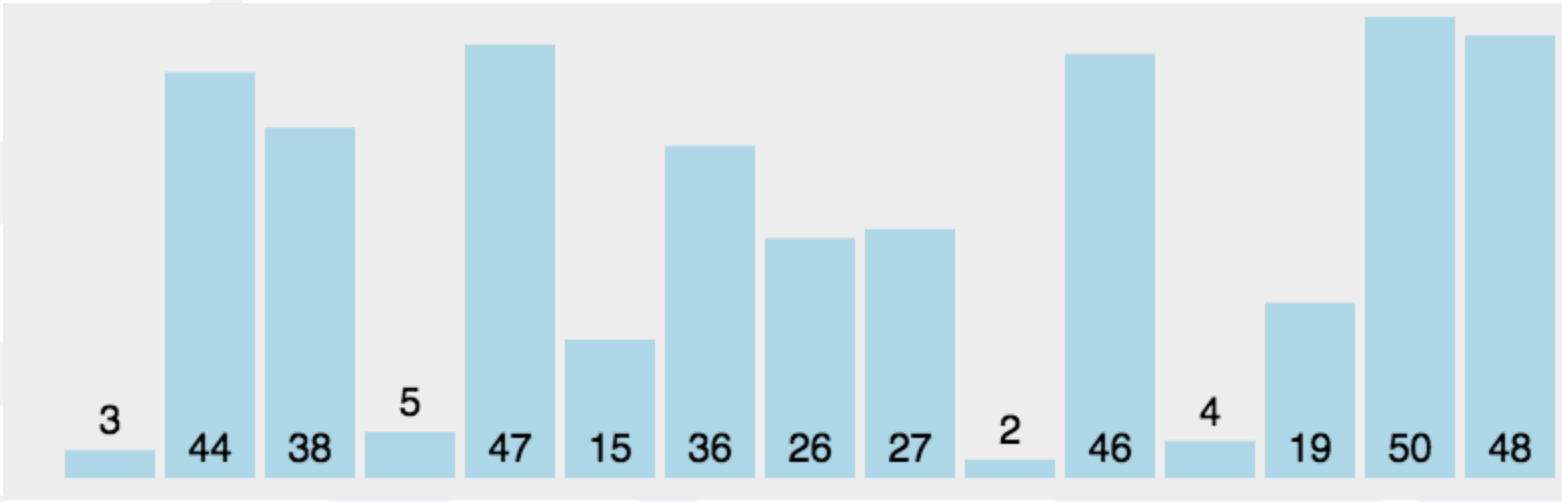
选择排序

算法思想

第一次先从头至尾扫描序列，找出最小的一个元素，和第一个元素交换
接着从剩下的元素中继续这种方式，最终得到一个有序序列。

	3 1 2 5 4 6
第一趟	1 3 2 5 4 6
第二趟	1 2 3 5 4 6
第三趟	1 2 3 5 4 6
第四趟	1 2 3 4 5 6
第五趟	1 2 3 4 5 6
第六趟	1 2 3 4 5 6

选择排序



Q：在排序运行过程中，你觉得需要记录什么信息？

交换位置的下标以及他们的值

核心代码（版本一）

3 1 2 5 4 6

1 3 2 5 4 6

1 2 3 5 4 6

1 2 3 5 4 6

1 2 3 4 5 6

1 2 3 4 5 6

1 2 3 4 5 6

思路：找到并记录下最小值的下标，并交换

```
int i,j,t,Min,flag;
for( i=1; i<n; i++ ){ //遍历的次数(访问每一个排序的位置)
      Min=a[i];
      for( j=i+1 ;j<=n ;j++  ){  //从这个位置往后找到比它更小的数，交换
            if(Min>a[j]){
                flag=j;  //记录最小值的下标
            }
      }
      //交换
      t=a[i];
      a[i]=a[flag];
      a[flag]=t;
}
```

时间复杂度：$O(n^2)$

核心代码（版本一修改）

3 1 2 5 4 6

1 3 2 5 4 6

1 2 3 5 4 6

1 2 3 5 4 6

1 2 3 4 5 6

1 2 3 4 5 6

1 2 3 4 5 6

思路：找到并记录下最小值的下标，并交换

```
int i,j,t,Min,flag;
for( i=1; i<n; i++ ){ //遍历的次数(访问每一个排序的位置)
      Min=a[i],flag=0;
      for( j=i+1 ;j<=n ;j++  ){  //从这个位置往后找到比它更小的数，交换
              if(Min>a[j]){
                  flag=j;  //记录最小值的下标
              }
      }
      //交换
      if(flag!=0){
        t=a[i];
        a[i]=a[flag];
        a[flag]=t;
      }
}
```

时间复杂度：$O(n^2)$

核心代码（版本二）

3 1 2 5 4 6

1 3 2 5 4 6

1 2 3 5 4 6

1 2 3 5 4 6

1 2 3 4 5 6

1 2 3 4 5 6

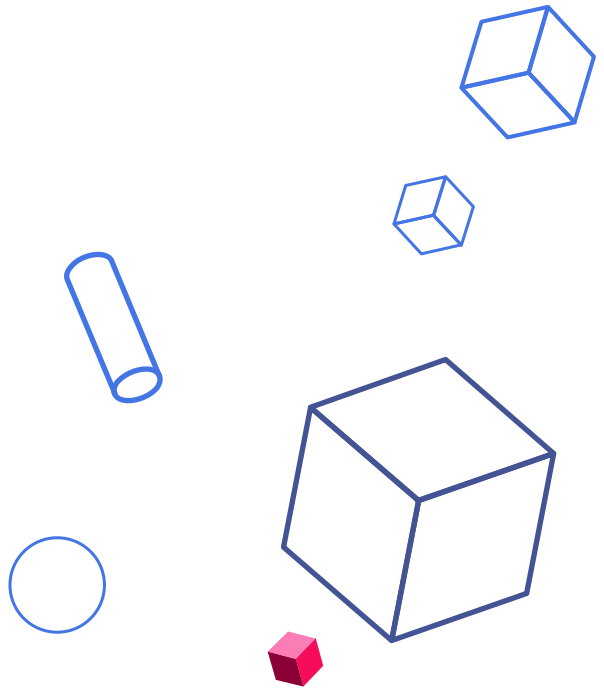
1 2 3 4 5 6

思路：遇到比当前位置小的数字就交换

```
int i,j,t;
for( i=1 ; i<n ; i++ ){ //遍历的次数(访问每一个排序的位置)
    for( j=i+1 ; j<=n ; j++ ){  //从这个位置往后找到比它更小的数，交换
            if(a[i]>a[j]){
                t=a[i];
                a[i]=a[j];
                a[j]=t;
            }
        }
    }
```

时间复杂度：$O(n^2)$

02.插入排序



插入排序

基本思想：不断的从无序的序列中取出第一个数，放到有序的序列中

一个数本身是有序的

3 1 2 5 4 6

第一趟 1 3 2 5 4 6

第二趟 1 2 3 5 4 6

第三趟 1 2 3 5 4 6

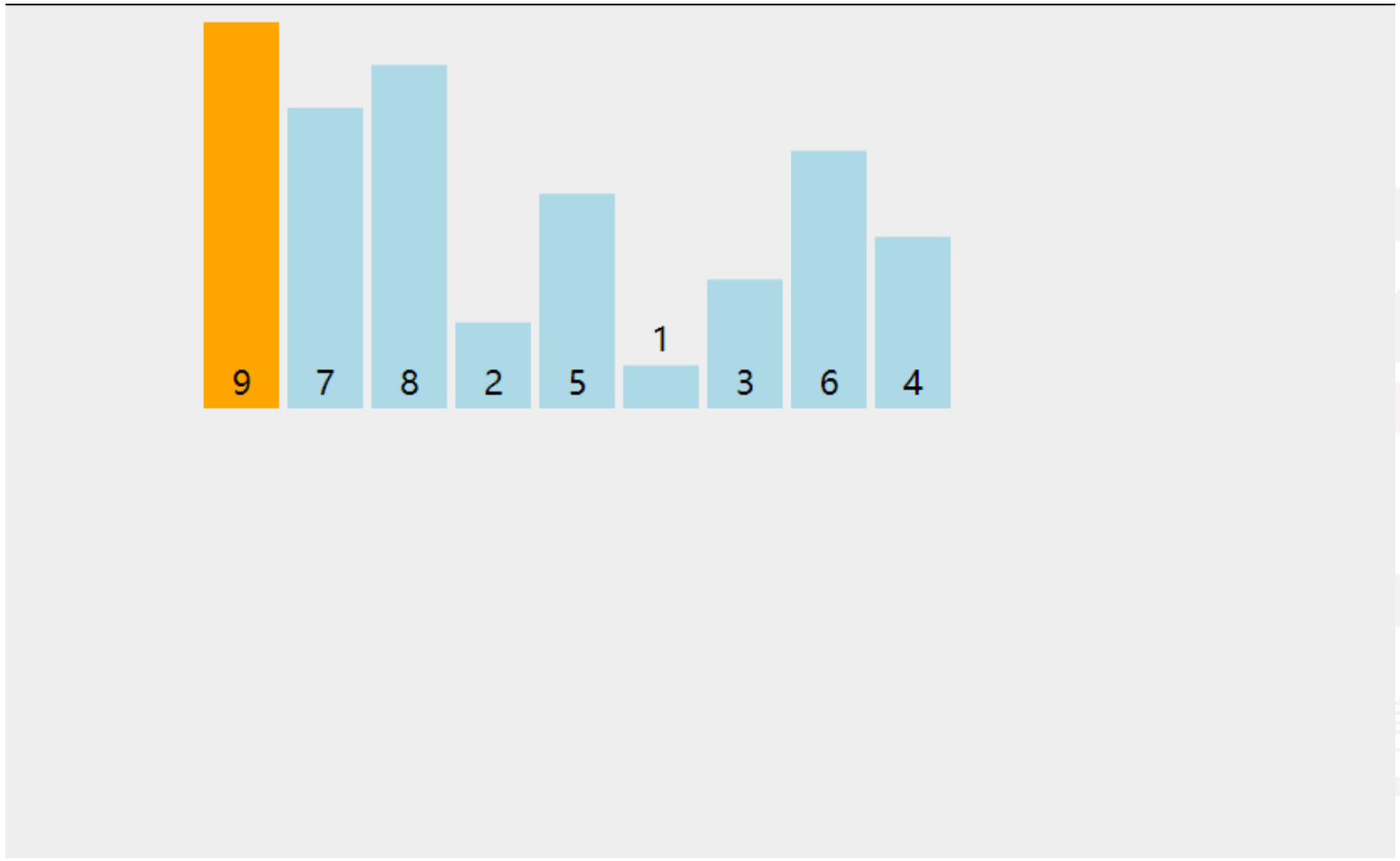
第四趟 1 2 3 4 5 6

第五趟 1 2 3 4 5 6

第六趟 1 2 3 4 5 6

非常像人们排序一手扑克牌一样

插入排序



Q：在排序运行过程中，
你觉得需要记录什么信息？

要插入到有序区的数
有序区的变化的长度

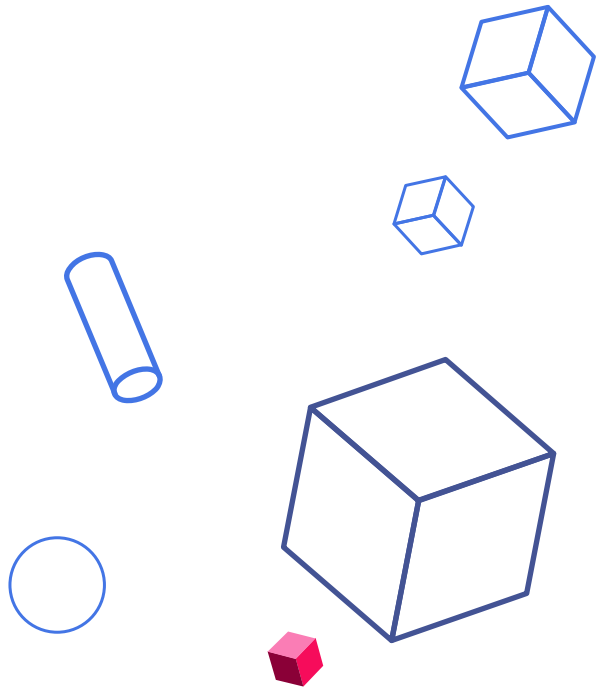
核心代码


```
int i,j,n,key;
for (  i=2  ; i<=n  ;  i++ ) { //n是数组的长度
    key = a[i];  //取出无序序列的第一个数
    j = i-1;     //有序区的最后一个位置
    while (a[j]>key && j>=1 ) {   //给key腾位置
       a[j+1] = a[j];
       j--;
     }
     a[j+1] = key;  //插入key
}
```


时间复杂度：$O(n^2)$

FZOI
西大附中信息学竞赛
High School Affiliated to Southwest University

西南大学附属中学
High School Affiliated to Southwest University



03.冒泡排序

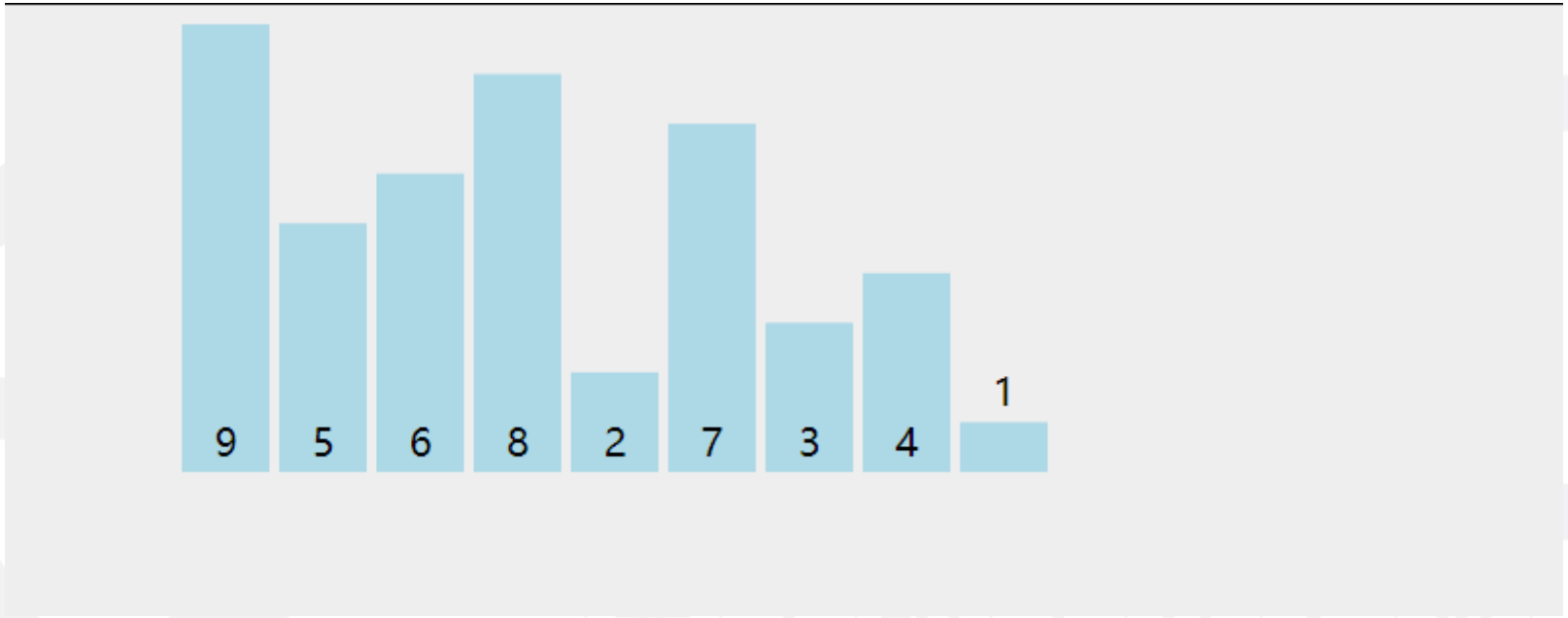
冒泡排序

基本思想：重复地走访过要排序的元素列，一次比较两个相邻的元素，
如果他们的顺序错误就把他们交换过来

	6 3 2 5 4 1
第一趟	3 2 5 4 1 6
第二趟	2 3 4 1 5 6
第三趟	2 3 1 4 5 6
第四趟	2 1 3 4 5 6
第五趟	1 2 3 4 5 6

大数往后，小数往前
这个过程很想气泡上升的情况
所以叫冒泡排序

冒泡排序



核心代码


```
int i,j,t,n;
for (i=1;i<=n-1;i++){   //一共进行n-1次遍历
 for (j=1;j<=n-i+1;j++){   //每次进行完成以后，还有n-i个数字需要比较，需要比较n-i+1次
    if(a[j]>a[j+1]) {
      t=a[j];
      a[j]=a[j+1];
      a[j+1]=t;
     }
   }
}
```

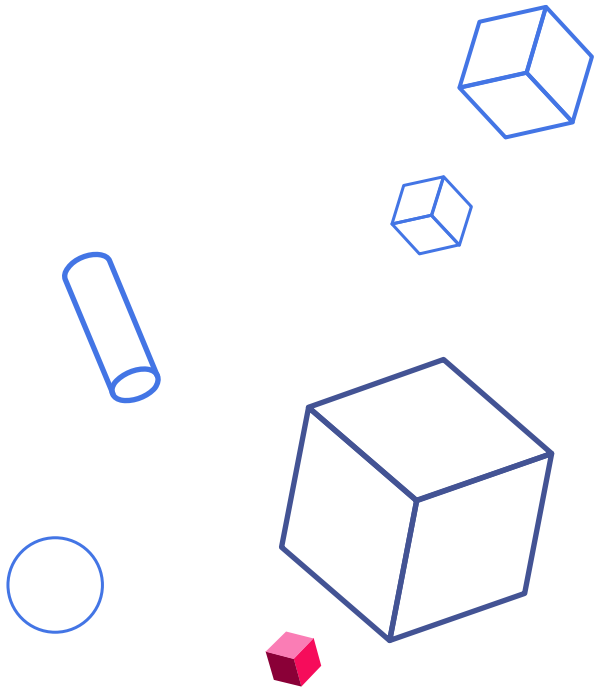
小结

上述排序是基于数据之间的比较来进行的排序

还有某些排序是不需要比较就可以实现排序

比较排序

04.计数排序



已知有20个大小在10以内的数
待排序列：9 3 5 4 9 1 2 7 8 1 3 6 5 3 4 0 10 9 7 9
按从小到大顺序输出这个序列

Q：能否思考一个方法在O(n)的时间复杂度内完成?

遍历、比较这样的过程是比较费时间的

通过标记数组的使用，我们可以发现，数组下标也可以表示对应的数值

已知有20个大小在10以内的数
待排序列：9 3 5 4 9 1 2 7 8 1 3 6 5 3 4 0 10 9 7 9
按从小到大顺序输出这个序列

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
0	0	0	0	0	0	0	0	0	0	0

```
a[9]++;
a[3]++;
a[5]++;
a[4]++;
a[9]++;
...
```

遍历整个数组
若a[i]!=0，输出a[i]个下标i

计数排序

已知有20个大小在10以内的数
待排序列：9 3 5 4 9 1 2 7 8 1 3 6 5 3 4 0 10 9 7 9
按从小到大顺序输出这个序列

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
0	0	0	0	0	0	0	0	0	0	0

```
a[9]++;
a[3]++;
a[5]++;
a[4]++;
a[9]++;
...
```

遍历整个数组
若a[i]!=0，输出下标i

排序+去重的效果

核心代码


```
int n,i,j,a[100]={0},x;
for (i=1;i<=n;i++){  //n个数
      cin>>x;
      a[x]++;
      Max=max(Max,x); //保存这些数中的最大值
}
for(i=1;i<=Max;i++){
  if(a[i]!=0){
    for(j=1;j<=a[i];j++){
       cout<<i<<“ ”;
     }
  }
}
```

排序算法的要点

- 排序的范围
- 排序的条件

05.时间复杂度对比

各种排序比较

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定

Thanks

For Your Watching