

## 简单选择排序

- 选择排序
  - 每一趟两两比较大小，找出极值（极大值或极小值）并放置到有序区的位置

□ 初始	1 9 8 5 6 7 4 3 2	
□ 第一趟	9 1 8 5 6 7 4 3 2	选择出此轮最大数9，和索引0数交换
□ 第二趟	9 8 1 5 6 7 4 3 2	选择出此轮最大数8，和索引1数交换
□ 第三趟	9 8 7 5 6 1 4 3 2	以此类推
□ 第四趟	9 8 7 6 5 1 4 3 2	
□ 第五趟	9 8 7 6 5 1 4 3 2	
□ 第六趟	9 8 7 6 5 4 1 3 2	
□ 第七趟	9 8 7 6 5 4 3 1 2	
□ 第八趟	9 8 7 6 5 4 3 2 1	

## 核心算法

- 结果可为升序或降序排列，默认升序排列。以降序为例
- 扩大有序区，减小无序区。图中红色部分就是增大的有序区，反之就是减小的无序区
- 相邻元素依次两两比较，获得每一次比较后的最大值，并记住此值的索引
- 每一趟都从无序区中选择出最大值，然后交换到当前无序区最左端

## 算法实现

```
1 m_list = [  
2     [1, 9, 8, 5, 6, 7, 4, 3, 2],  
3     [1, 2, 3, 4, 5, 6, 7, 8, 9],  
4     [9, 8, 7, 6, 5, 4, 3, 2, 1]  
5 ]  
6 nums = m_list[0]  
7 length = len(nums)  
8 print(nums)  
9  
10 count_iter = 0  
11 count_swap = 0  
12 # 选择排序  
13 for i in range(length-1):  
14     maxindex = i  
15  
16     for j in range(i+1, length):  
17         count_iter += 1  
18         if nums[maxindex] < nums[j]:  
19             maxindex = j  
20  
21     if maxindex != i:  
22         nums[maxindex], nums[i] = nums[i], nums[maxindex]  
23         count_swap += 1
```

```
24
25 print(nums)
26 print(count_iter, count_swap)
```

## 二元选择排序

- 同时选择出每一趟的最大值和最小值，并分别固定到两端的有序区
- 减少迭代的趟数

```
1 m_list = [
2     [1, 9, 8, 5, 6, 7, 4, 3, 2],
3     [1, 2, 3, 4, 5, 6, 7, 8, 9],
4     [9, 8, 7, 6, 5, 4, 3, 2, 1]
5 ]
6 nums = m_list[1]
7 length = len(nums)
8 print(nums)
9
10
11 count_iter = 0
12 count_swap = 0
13
14 for i in range(length//2): # 一次固定2个数，减半
15     maxindex = i # 正索引，假设无序区第一个就是最大数，其索引记作最大
16     minindex = -i-1 # 负索引，假设无序区最后一个就是最小数，其索引记作最小
17
18     for j in range(i+1, length-i): # 每次左边加一个，右边也要减一个，表示无序区两端
        都减少
19         count_iter += 1
20         if nums[maxindex] < nums[j]:
21             maxindex = j
22         if nums[minindex] > nums[-j-1]:
23             minindex = -j -1
24         #print(maxindex, i, "|||", minindex, -i-1)
25
26     if maxindex != i:
27         nums[maxindex], nums[i] = nums[i], nums[maxindex]
28         count_swap += 1
29         # [1, 3, 2]为例，如果i位置上就是最小值，走到这里，说明最大值和最小值交换过了，要
        调整最小值索引为maxindex
30         if i == length + minindex:
31             minindex = maxindex - length
32
33     if minindex != -i-1: # 负索引比较
34         nums[minindex], nums[-i-1] = nums[-i-1], nums[minindex]
35         count_swap += 1
36
37 print(nums)
38 print(count_iter, count_swap)
```

以上代码还有没有优化的可能？

如果一趟比较后，极大值、极小值的值相等，说明什么？

说明，剩余比较的数将全部相等，那么排序可以立即停止。

```
1 m_list = [  
2     [1, 9, 8, 5, 6, 7, 4, 3, 2],  
3     [1, 2, 3, 4, 5, 6, 7, 8, 9],  
4     [9, 8, 7, 6, 5, 4, 3, 2, 1]  
5 ]  
6 nums = m_list[1]  
7 length = len(nums)  
8 print(nums)  
9  
10  
11 count_iter = 0  
12 count_swap = 0  
13  
14 for i in range(length//2): # 一次固定2个数，减半  
15     maxindex = i # 正索引，假设无序区第一个就是最大数，其索引记作最大  
16     minindex = -i-1 # 负索引，假设无序区最后一个就是最小数，其索引记作最小  
17  
18     for j in range(i+1, length-i): # 每次左边加一个，右边也要减一个，表示无序区两端  
        都减少  
19         count_iter += 1  
20         if nums[maxindex] < nums[j]:  
21             maxindex = j  
22         if nums[minindex] > nums[-j-1]:  
23             minindex = -j -1  
24         #print(maxindex, i, "|||", minindex, -i-1)  
25         if nums[maxindex] == nums[minindex]: # 元素全相同  
26             break  
27  
28         if maxindex != i:  
29             nums[maxindex], nums[i] = nums[i], nums[maxindex]  
30             count_swap += 1  
31         # [1, 3, 2]为例，如果i位置上就是最小值，走到这里，说明最大值和最小值交换过了，要  
        调整最小值索引为maxindex  
32         if i == length + minindex:  
33             minindex = maxindex - length  
34  
35         if minindex != -i-1: # 负索引比较  
36             nums[minindex], nums[-i-1] = nums[-i-1], nums[minindex]  
37             count_swap += 1  
38  
39 print(nums)  
40 print(count_iter, count_swap)
```

考虑一种特殊情况

[1, 1, 1, 1, 1, 1, 1, 1, 2] 这种情况，找到的最小值索引是-2，最大值索引8，上面的代码会交换2次，最小值两个1交换是无用功，所以，增加一个判断

```
1 m_list = [  
2     [1, 9, 8, 5, 6, 7, 4, 3, 2],  
3     [1, 2, 3, 4, 5, 6, 7, 8, 9],  
4     [9, 8, 7, 6, 5, 4, 3, 2, 1],  
5     [1, 1, 1, 1, 1, 1, 1, 1, 1],
```

```

6     [1, 1, 1, 2]
7 ]
8 nums = m_list[4]
9 length = len(nums)
10 print(nums)
11
12
13 count_iter = 0
14 count_swap = 0
15
16 for i in range(length//2): # 一次固定2个数，减半
17     maxindex = i # 正索引，假设无序区第一个就是最大数，其索引记作最大
18     minindex = -i-1 # 负索引，假设无序区最后一个就是最小数，其索引记作最小
19
20     for j in range(i+1, length-i): # 每次左边加一个，右边也要减一个，表示无序区两端
        都减少
21         count_iter += 1
22         if nums[maxindex] < nums[j]:
23             maxindex = j
24         if nums[minindex] > nums[-j-1]:
25             minindex = -j -1
26         #print(maxindex, i, "|||", minindex, -i-1)
27         if nums[maxindex] == nums[minindex]: # 元素全相同
28             break
29
30         if maxindex != i:
31             nums[maxindex], nums[i] = nums[i], nums[maxindex]
32             count_swap += 1
33         # [1, 3, 2]为例，如果i位置上就是最小值，走到这里，说明最大值和最小值交换过了，要
        调整最小值索引为maxindex
34         if i == length + minindex:
35             minindex = maxindex - length
36
37         if minindex != -i-1 and nums[minindex] != nums[-i-1]: # 负索引比较，值不一
        样再交换
38             nums[minindex], nums[-i-1] = nums[-i-1], nums[minindex]
39             count_swap += 1
40
41 print(nums)
42 print(count_iter, count_swap)

```

## 总结

- 简单选择排序需要数据一趟趟比较，并在每一趟中发现极值
- 没有办法知道当前这一趟是否已经达到排序要求，但是可以知道极值是否在目标索引位置上
- 遍历次数 $1, \dots, n-1$ 之和 $n(n-1)/2$
- 时间复杂度 $O(n^2)$
- 减少了交换次数，提高了效率，性能略好于冒泡法

