

## 原 C语言实现8种排序

2017年02月24日 22:26:01 J\_小浩子 阅读数 21353 更多

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：[https://blog.csdn.net/change\\_on/article/details/56927267](https://blog.csdn.net/change_on/article/details/56927267)

最近要开始准备春招了，没什么时间学习spring，得忙着刷题。这两天复习排序，综合网上和书上的资料，整理了下面8种排序算法的实现。基于C语言的，java版本很快就出来，具体代码：

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  //冒泡排序
4  void bubbleSort(int data[], int n);
5  //快速排序
6  void quickSort(int data[], int low, int high);
7  int findPos(int data[], int low, int high);
8  //插入排序
9  void bInsertSort(int data[], int n);
10 //希尔排序
11 void shellSort(int data[], int n);
12 //选择排序
13 void selectSort(int data[], int n);
14 //堆排序
15 void heapSort(int data[], int n);
16 void swap(int data[], int i, int j);
17 void heapAdjust(int data[], int i, int n);
18 //归并排序
19 void mergeSort(int data[], int first, int last);
20 void merge(int data[], int low, int mid, int high);
21 //基数排序
22 void radixSort(int data[], int n);
23 int getNumPos(int num, int pos);
24
25
26 int main() {
27     int data[10] = {43, 65, 4, 23, 6, 98, 2, 65, 7, 79};
28     int i;
29     printf("原先数组:");
30     for(i=0;i<10;i++) {
31         printf("%d ", data[i]);
```

```
32     }
33     printf("\n");
34     /*printf("冒泡排序:");
35     bubbleSort(data, 10);
36     for(i=0;i<10;i++) {
37         printf("%d  ", data[i]);
38     }
39     printf("\n");
40     printf("快速排序:");
41     quickSort(data, 0, 9);
42     for(i=0;i<10;i++) {
43         printf("%d  ", data[i]);
44     }
45     printf("\n");
46     printf("插入排序:");
47     bInsertSort(data,10);
48     for(i=0;i<10;i++) {
49         printf("%d  ", data[i]);
50     }
51     printf("\n");
52     printf("希尔排序:");
53     shellSort(data, 10);
54     for(i=0;i<10;i++) {
55         printf("%d  ", data[i]);
56     }
57     printf("\n");
58     printf("选择排序:");
59     selectSort(data, 10);
60     for(i=0;i<10;i++) {
61         printf("%d  ", data[i]);
62     }
63     printf("\n");
64     int data[11] = {-1, 43, 65, 4, 23, 6, 98, 2, 65, 7, 79};
65     int i;
66     printf("原先数组:");
67     int data[11] = {-1, 43, 65, 4, 23, 6, 98, 2, 65, 7, 79};
68     for(i=1;i<11;i++) {
69         printf("%d  ", data[i]);
70     }
71     printf("\n");
72     printf("堆排序:");
73     heapSort(data, 10);
74     for(i=1;i<11;i++) {
75         printf("%d  ", data[i]);
76     }
```



36



5



```
78     }
79     printf("\n");
80     printf("归并排序:");
81     mergeSort(data, 0, 9);
82     for(i=0;i<10;i++) {
83         printf("%d  ", data[i]);
84     }
85     printf("\n");*/
86     printf("基数排序:");
87     radixSort(data, 10);
88     for(i=0;i<10;i++) {
89         printf("%d  ", data[i]);
90     }
91     printf("\n");
92     return 0;
93 }
94 }
95 /*-----冒泡排序-----*/
96 void bubbleSort(int data[], int n) {
97     int i,j,temp;
98     //两个for循环，每次取出一个元素跟数组的其他元素比较
99     //将最大的元素排到最后。
100     for(j=0;j<n-1;j++) {
101         //外循环一次，就排好一个数，并放在后面，
102         //所以比较前面n-j-1个元素即可
103         for(i=0;i<n-j-1;i++) {
104             if(data[i]>data[i+1]) {
105                 temp = data[i];
106                 data[i] = data[i+1];
107                 data[i+1] = temp;
108             }
109         }
110     }
111 }
112 }
113
114 /*-----快速排序-----*/
115 int findPos(int data[], int low, int high) {
116     //将大于t的元素赶到t的左边，大于t的元素赶到t的右边
117     int t = data[low];
118     while(low < high) {
119         while(low < high && data[high] >= t) {
120             high--;
121         }
122         data[low] = data[high];
123         while(low < high && data[low] <= t) {
```



36



5



```
124         low++;
125     }
126     data[high] = data[low];
127 }
128 data[low] = t;
129 //返回此时t在数组中的位置
130 return low;
131 }
132 //在数组中找一个元素，对大于该元素和小于该元素的两个数组进行再排序
133 //再对两个数组分为4个数组，再排序，直到最后每组只剩下一个元素为止
134 void quickSort(int data[], int low, int high) {
135     if(low > high) {
136         return;
137     }
138     int pos = findPos(data, low, high);
139     quickSort(data, low, pos-1);
140     quickSort(data, pos+1, high);
141 }
142
143 /*-----插入排序-----*/
144 void bInsertSort(int data[], int n) {
145     int low,high,mid;
146     int temp,i,j;
147     for(i=1;i<n;i++) {
148         low = 0;
149         //把data[i]元素插入到它的前面data[0-(i-1)]中
150         temp =data[i];
151         high = i-1;
152         //该while是折半，缩小data[i]的范围(优化手段)
153         while(low <= high) {
154             mid = (low+high)/2;
155             if(data[mid] > temp) {
156                 high = mid-1;
157             }
158             else {
159                 low = mid+1;
160             }
161         }
162         int j = i;
163         //让data与已经排序好的数组的各个元素比较，小的放前面
164         while((j > low) && data[j-1] > temp) {
165             data[j] = data[j-1];
166             --j;
167         }
168         data[low] = temp;
169     }
```



36



5



```
170     }
171 }
172
173 /*-----希尔排序-----*/
174 void shellSort(int * data, int n) {
175     int step,i,j,key;
176     //将数组按照step分组，不断二分到每组只剩下一个元素
177     for(step=n/2;step>0;step/=2) {
178         //将每组中的元素排序，小的在前
179         for(i=step;i<n;i++) {
180             key = data[i];
181             for(j=i-step;j>=0 && key<data[j];j-=step) {
182                 data[j+step] = data[j];
183             }
184             //和上面的for循环一起，将组中小的元素换到数组的前面
185             data[j+step] = key;
186         }
187     }
188 }
189
190 /*-----选择排序-----*/
191 void selectSort(int data[], int n) {
192     int i,j,mix,temp;
193     //每次循环数组，找出最小的元素，放在前面，前面的即为排序好的
194     for(i=0;i<n-1;i++) {
195         //假设最小元素的下标
196         int mix = i;
197         //将上面假设的最小元素与数组比较，交换出最小的元素的下标
198         for(j=i+1;j<n;j++) {
199             if(data[j] < data[mix]) {
200                 mix = j;
201             }
202         }
203     }
204     //若数组中真的有比假设的元素还小，就交换
205     if(i != mix) {
206         temp = data[i];
207         data[i] = data[mix];
208         data[mix] = temp;
209     }
210 }
211 }
212
213 /*-----堆排序-----*/
214 //堆排序将数组先组成二叉树，默认从数组的data[1]开始排，data[0]是
215 //无效数据
```



36



5



```
216 void heapSort(int data[], int n) {
217     int i;
218     //先将数组组成一棵完全二叉树
219     //从2/n开始,就是从倒数第二排结点往前开始
220     for(i=n/2;i>0;i--) {
221         heapAdjust(data, i, n);
222     }
223     //循环每个结点,将大的结点交换到堆顶
224     for(i=n;i>1;i--) {
225         swap(data, 1, i);
226         //每次交换完都要调整二叉树,将剩下的最大的结点交换到堆顶
227         heapAdjust(data, 1, i-1);
228     }
229 }
230 //交换函数
231 void swap(int data[], int i, int j) {
232     int temp;
233     temp = data[i];
234     data[i] = data[j];
235     data[j] = temp;
236 }
237 void heapAdjust(int data[], int i, int n) {
238     int j, temp;
239     //假设第一个结点的元素是最大的
240     temp = data[i];
241     //结点: 2*i是i结点的左结点, 2*i+1是i结点的右结点
242     //把结点元素大的交换到前面
243     for(j=2*i;j<=n;j*=2) {
244         if(j < n && data[j] < data[j+1]) {
245             j++;
246         }
247         if(temp >= data[j]) {
248             break;
249         }
250     }
251     data[i] = data[j];
252     i = j;
253 }
254 data[i] = temp;
255 }
256
257 /*-----归并排序-----*/
258 void mergeSort(int data[], int first, int last) {
259     int mid = 0;
260     //将数组不停的二分分组再组合,直到每组只剩一个元素
261     if(first < last) {
```



36



5



```
262     mid = (first+last)/2;
263     mergeSort(data, first, mid);
264     mergeSort(data, mid+1, last);
265     merge(data, first, mid, last);
266 }
267 return;
268 }
269 void merge(int data[], int low, int mid, int high) {
270     int i, k;
271     //定义一个临时数组存放传进来的无序数组排好序之后的数组
272     int *temp = (int *)malloc((high-low+1)*sizeof(int));
273     //将无序数组分成两个序列
274     int left_low = low;
275     int left_high = mid;
276     int right_low = mid+1;
277     int right_high = high;
278     //将两个序列比较排序, 小的排前
279     for(k=0; left_low<=left_high && right_low<=right_high; k++) {
280         if(data[left_low]<=data[right_low]) {
281             temp[k] = data[left_low++];
282         }
283         else{
284             temp[k] = data[right_low++];
285         }
286     }
287     //左序列如果有剩下元素未排序, 加到临时数组的末尾
288     if(left_low <= left_high) {
289         for(i=left_low; i<=left_high; i++) {
290             temp[k++] = data[i];
291         }
292     }
293     //右序列如果有剩下元素未排序, 加到临时数组的末尾
294     if(right_low <= right_high) {
295         for(i=right_low; i<=right_high; i++) {
296             temp[k++] = data[i];
297         }
298     }
299     //将排好序的小分组转移到原数组中
300     for(i=0; i<high-low+1; i++) {
301         data[low+i] = temp[i];
302     }
303     free(temp);
304     return;
305 }
306 }
307 /*-----基数排序-----*/
```



36



5



```
308 //该函数的作用是找出num的pos位数的数字(比如：23的个位数数字是3)
309 int getNumPos(int num, int pos) {
310     int i;
311     int temp = 1;
312     for(i=0;i<pos-1;i++) {
313         temp *= 10;
314     }
315     return (num / temp) % 10;
316 }
317 void radixSort(int data[], int n) {
318     int i,j,k,pos,num,index;
319     //这几句话是创建一个从0-9(行)× (n+1)(列)的网格，第一列从上往下是0-9,
320     //第二列是该行包含的元素个数，默认为0个
321     int *radixArrays[10];
322     for(i=0;i<10;i++) {
323         radixArrays[i] = (int *)malloc(sizeof(int) * (n+1));
324         radixArrays[i][0] = 0;
325     }
326     //pos最大为31为数，计算机能承受的最大范围了
327     for(pos=1;pos<=31;pos++) {
328         //该for循环是将数组的元素按照位数(pos)的值放进网格内
329         for(i=0;i<n;i++) {
330             num = getNumPos(data[i], pos);
331             index = ++radixArrays[num][0];
332             radixArrays[num][index] = data[i];
333         }
334         //该for循环是将上面的for循环已经按照某个位数(pos)排列好的元素存入数组
335         for(i=0,j=0;i<10;i++) {
336             for(k=1;k<=radixArrays[i][0];k++) {
337                 data[j++] = radixArrays[i][k];
338             }
339             //清空网格，以便给下个位数排列
340             radixArrays[i][0] = 0;
341         }
342     }
343 }
```

以上排序算法的优劣（时间复杂度和空间复杂度对比）：



36



5





## 数组排序算法

算法	时间复杂度			空间复杂度
	最佳	平均	最差	最差
<a href="#">Quicksort</a>	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(\log(n))$
<a href="#">Mergesort</a>	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$
<a href="#">Timsort</a>	$O(n)$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$
<a href="#">Heapsort</a>	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(1)$
<a href="#">Bubble Sort</a>	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
<a href="#">Insertion Sort</a>	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
<a href="#">Selection Sort</a>	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
<a href="#">Shell Sort</a>	$O(n)$	$O((n \log(n))^2)$	$O((n \log(n))^2)$	$O(1)$
<a href="#">Bucket Sort</a>	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$
<a href="#">Radix Sort</a>	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$

[http://blog.csdn.net/change\\_on](http://blog.csdn.net/change_on)

测试：

```
原先数组:43 65 4 23 6 98 2 65 7 79
堆排序:2 4 6 7 23 43 65 65 79 98
```

```
原先数组:43 65 4 23 6 98 2 65 7 79
归并排序:2 4 6 7 23 43 65 65 79 98
```

36

5

5

<

>

<

>

<

>

<

>

<

>

<

>

<

>

<

>

<

>

<

>

```
原先数组:43 65 4 23 6 98 2 65 7 79
基数排序:2 4 6 7 23 43 65 65 79 98
```

```
原先数组:43 65 4 23 6 98 2 65 7 79
希尔排序:2 4 6 7 23 43 65 65 79 98
```

```
原先数组:43 65 4 23 6 98 2 65 7 79
冒泡排序:2 4 6 7 23 43 65 65 79 98
```

```
原先数组:43 65 4 23 6 98 2 65 7 79
快速排序:2 4 6 7 23 43 65 65 79 98
插入排序:希尔排序:[wayne@wayne demo]$ vim sort.c
[wayne@wayne demo]$ g++ sort.c -o sort
[wayne@wayne demo]$ ./sort
原先数组:43 65 4 23 6 98 2 65 7 79
插入排序:2 65 4 23 6 43 65 65 79 98
```

```
原先数组:43 65 4 23 6 98 2 65 7 79
选择排序:2 4 6 7 23 43 65 65 79 98
```

## c语言排序算法（一）

阅读数 8189

排序算法，是算法之中相对基础的，也是各门语言的必学的算法。本篇文章用C语言为大家介绍排序... 博文 来自：[liu659\\_的博客](#)



想对作者说点什么



IT民工\_zhh：nice（8个月前 #4楼）



じょいくてい：问一下62-69行为什么数组里加一个-1（8个月前 #3楼） [查看回复\(1\)](#)



涂山北枫：谢谢，推荐收藏评论三联（11个月前 #2楼）

[查看 5 条热评](#)

## 十大经典排序算法（C语言实现）

阅读数 1万+

原文链接：<https://www.cnblogs.com/onepixel/articles/7674659.html>1、冒泡排序（BubbleSort）... 博文 来自：[小码的嵌入式...](#)



36



5

