

```

1  /* sort.c */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <malloc.h>
6  #include <math.h>
7  #include "LinkStack.h"
8
9
10 #define LONG_LONG_MIN ((unsigned long long)1 << (sizeof(long long) * 8 - 1))
11 #define qsSwitch 0
12
13 //a, b不能指向同一元素
14 #define SWAP(a, b) (a ^= b, b ^= a, a ^= b)
15
16 void bubbleSort(long long *a, int n)
17 {
18     int i, j;
19     for (i = 0; i < n; i++)
20     {
21         int done = 1;
22         for (j = 1; j < n - i; j++)
23         {
24             if (a[j - 1] > a[j])
25             {
26                 SWAP(a[j - 1], a[j]);
27                 done = 0;
28             }
29         }
30         if (done)
31         {
32             break;
33         }
34     }
35 }
36
37 void selectionSort(long long *a, int n)
38 {
39     int rank;
40     int i, j;
41     for (i = 0; i < n; i++)
42     {
43         rank = 0;
44         for (j = 1; j < n - i; j++)
45         {
46             if (*(a + rank) < *(a + j))
47             {
48                 rank = j;
49             }
50             //带交换的两个变量指向同一个地址时不能调用SWAP宏，会出错
51             if (rank != n - i - 1)
52             {
53                 SWAP(a[rank], a[n - i - 1]);
54             }
55         }
56     }
57 }
58
59 void maxHeapDown(long long *a, int start, int end)
60 {
61     int cur = start; //current node position
62     int left = 2 * cur + 1; //left child position
63     int tmp = a[cur]; //current node value
64     for (; left <= end; cur = left, left = 2 * left + 1)
65     {
66         if (left < end && a[left] < a[left + 1])
67         {
68             left++;
69         }
70         if (tmp >= a[left])
71         {
72             break;
73         }
74         else

```

```

74         {
75             a[cur] = a[left];
76             a[left] = tmp;
77         }
78     }
79 }
80
81 void heapSort(long long *a, int n)
82 {
83     int i;
84     //get maximum heap, Floyd算法--自下而上的下虑, 时间复杂度O(n)
85     for (i = n / 2 - 1; i >= 0; i --)
86     {
87         maxHeapDown(a, i, n - 1);
88     }
89     //sort--put current maximum data in the current last position
90     for (i = n - 1; i > 0; i --)
91     {
92         SWAP(a[0], a[i]);
93         maxHeapDown(a, 0, i - 1);
94     }
95 }
96
97 //合并操作
98 void merge(long long *a, int lo, int mid, int hi)
99 {
100     int len1 = mid - lo + 1;
101     int len2 = hi - mid;
102     long long *left = (long long *)malloc(len1 * sizeof(long long));
103     long long *right = (long long *)malloc(len2 * sizeof(long long));
104     int i, j, k;
105     for (i = 0; i < len1; i ++)
106     {
107         left[i] = a[lo + i];
108     }
109     for (i = 0; i < len2; i ++)
110     {
111         right[i] = a[mid + 1 + i];
112     }
113     i = 0, j = 0, k = lo;
114     while(i < len1 && j < len2)
115     {
116         if (left[i] <= right[j])
117         {
118             a[k ++] = left[i ++];
119         }
120         else
121         {
122             a[k ++] = right[j ++];
123         }
124     }
125     while (i < len1)
126     {
127         a[k ++] = left[i ++];
128     }
129     while (j < len2)
130     {
131         a[k ++] = right[j ++];
132     }
133     free(left);
134     free(right);
135 }
136
137 // [lo, hi]
138 void mergeSort2Way(long long *a, int lo, int hi)
139 {
140     if (lo < hi)
141     {
142         int mid = (lo + hi) >> 1;
143         mergeSort2Way(a, lo, mid);
144         mergeSort2Way(a, mid + 1, hi);
145         merge(a, lo, mid, hi);
146     }

```

```

147 }
148
149 void mergeSort(long long *a, int n)
150 {
151     mergeSort2Way(a, 0, n - 1);
152 }
153
154 void insertionSort(long long *a, int n)
155 {
156     int i, j;
157     long long tmp;
158     for (i = 1; i < n; i++)
159     {
160         tmp = *(a + i);
161         j = i - 1;
162         while (j >= 0 && tmp < *(a + j))
163         {
164             *(a + j + 1) = *(a + j);
165             j--;
166         }
167         *(a + j + 1) = tmp;
168     }
169 }
170
171 //勤于拓展, 懒于交换
172 static int partitionA(long long *a, int lo, int hi)
173 {
174     //任选一个元素和首元素进行交换
175     int tmp = lo + rand() % (hi - lo + 1);
176     //待交换的两个变量指向同一个地址时不能调用SWAP宏, 会出错
177     if (lo != tmp)
178     {
179         SWAP(a[lo], a[tmp]);
180     }
181     long long pivot = a[lo];
182     while (lo < hi)
183     {
184         while (lo < hi && a[hi] >= pivot)
185         {
186             hi--;
187         }
188         a[lo] = a[hi];
189         while (lo < hi && a[lo] <= pivot)
190         {
191             lo++;
192         }
193         a[hi] = a[lo];
194     }
195     a[lo] = pivot;
196     return lo;
197 }
198
199 //懒于拓展, 勤于交换
200 static int partitionB(long long *a, int lo, int hi)
201 {
202     int tmp = lo + rand() % (hi - lo + 1);
203     //待交换的两个变量指向同一个地址时不能调用SWAP宏, 会出错
204     if (lo != tmp)
205     {
206         SWAP(a[lo], a[tmp]);
207     }
208     long long pivot = a[lo];
209     while (lo < hi)
210     {
211         while (lo < hi)
212         {
213             if (pivot < a[hi])
214             {
215                 hi--;
216             }
217             else
218             {
219                 a[lo++] = a[hi];

```

```

220         break;
221     }
222 }
223 while (lo < hi)
224 {
225     if (a[lo] < pivot)
226     {
227         lo ++;
228     }
229     else
230     {
231         a[hi --] = a[lo];
232         break;
233     }
234 }
235
236 }
237 a[lo] = pivot;
238 return lo;
239 }
240
241 static int partitionC(long long *a, int lo, int hi)
242 {
243     int tmp = lo + rand() % (hi - lo + 1);
244     //待交换的两个变量指向同一个地址时不能调用SWAP宏，会出错
245     if (hi != tmp)
246     {
247         SWAP(a[hi], a[tmp]);
248     }
249     long long pivot = a[hi];
250     int i = lo - 1, j = lo;
251     for (; j < hi; j++)
252     {
253         if (a[j] <= pivot)
254         {
255             i ++;
256             if (a[i] != a[j])
257             {
258                 SWAP(a[i], a[j]);
259             }
260         }
261     }
262     i ++;
263     if (a[i] != a[hi])
264     {
265         SWAP(a[i], a[hi]);
266     }
267     return i;
268 }
269
270 void quickSortRecur(long long *a, int lo, int hi)
271 {
272     if (lo >= hi)
273     {
274         return ;
275     }
276     int index = partitionC(a, lo, hi);
277     quickSortRecur(a, lo, index - 1);
278     quickSortRecur(a, index + 1, hi);
279 }
280
281 void quickSortNonRecur(long long *a, int lo, int hi)
282 {
283     if (lo >= hi)
284     {
285         return ;
286     }
287     STACK s;
288     StackNew(&s, sizeof(int), NULL);
289     StackPush(&s, &hi);
290     StackPush(&s, &lo);
291     while (!StackEmpty(&s))
292     {

```

```

293     int l, r;
294     long long tmp;
295     StackPop(&s, &l);
296     StackPop(&s, &r);
297     int index = partitionC(a, l, r);
298     if (l < index - 1)
299     {
300         tmp = index - 1;
301         StackPush(&s, &tmp);
302         StackPush(&s, &l);
303     }
304     if (r > index + 1)
305     {
306         tmp = index + 1;
307         StackPush(&s, &r);
308         StackPush(&s, &tmp);
309     }
310 }
311 StackDispose(&s);
312 }
313
314 void quickSort(long long *a, int n)
315 {
316     if (0 == qsSwitch)
317     {
318         quickSortRecur(a, 0, n - 1);
319     }
320     else
321     {
322         quickSortNonRecur(a, 0, n - 1);
323     }
324 }
325
326 //此处要求a[i] >= 0
327 void countSort(long long *a, int n)
328 {
329     long long *b = (long long *)malloc(sizeof(long long) * n);
330     memcpy(b, a, sizeof(long long) * n);
331     long long max = LONG_LONG_MIN;
332     int i = 0;
333     for (; i < n; i++)
334     {
335         if (max < a[i])
336         {
337             max = a[i];
338         }
339     }
340     max++;
341     long long *c = (long long *)malloc(sizeof(long long) * max);
342     memset(c, 0, sizeof(long long) * max);
343     //记录数据在每个桶中的数量
344     for (i = 0; i < n; i++)
345     {
346         c[a[i]]++;
347     }
348     for (i = 1; i < max; i++)
349     {
350         c[i] += c[i - 1];
351     }
352     for (i = n - 1; i >= 0; i--)
353     {
354         a[c[b[i]] - 1] = b[i];
355         c[b[i]]--;
356     }
357     free(b);
358     free(c);
359 }
360
361 //此处要求a[i] >= 0
362 void radixSort(long long *a, int n)
363 {
364     long long max = LONG_LONG_MIN;
365     int i = 0;

```

```

366     for (; i < n; i++)
367     {
368         if (max < a[i])
369         {
370             max = a[i];
371         }
372     }
373     int d = 0;
374     while (max > 0)
375     {
376         d++;
377         max /= 10;
378     }
379     long long c[10];
380     long long *b = (long long *)malloc(sizeof(long long) * n);
381     for (i = 0; i < d; i++)
382     {
383         memset(c, 0, sizeof(long long) * 10);
384         memcpy(b, a, sizeof(long long) * n);
385         //计数排序
386         long long base1 = (long long)pow(10, i);
387         long long base2 = (long long)pow(10, i + 1);
388         int j, rank;
389         for (j = 0; j < n; j++)
390         {
391             rank = (b[j] % base2) / base1;
392             c[rank]++;
393         }
394         for (j = 1; j < 10; j++)
395         {
396             c[j] += c[j - 1];
397         }
398         for (j = n - 1; j >= 0; j--)
399         {
400             rank = (b[j] % base2) / base1;
401             a[c[rank] - 1] = b[j];
402             c[rank]--;
403         }
404     }
405     free(b);
406 }
407
408 int main()
409 {
410     int n;
411     scanf("%d", &n);
412     long long a[n];
413     int i;
414
415     for (i = 0; i < n; i++)
416     {
417         scanf("%lld", &a[i]);
418     }
419
420     //前5个算法的思想是减而治之
421     //bubbleSort(a, n);
422     //insertionSort(a, n);
423     //selectionSort(a, n);
424     //heapSort(a, n);
425     //quickSort(a, n);
426     //归并排序的思想是分而治之
427     //mergeSort(a, n);
428     //countSort(a, n);
429     radixSort(a, n);
430
431     for (i = 0; i < n; i++)
432     {
433         printf("%lld ", a[i]);
434     }
435     printf("\n");
436     return 0;
437 }

```