**数据结构第八次上机实验报告**

学号：161730126 姓名：李双玖

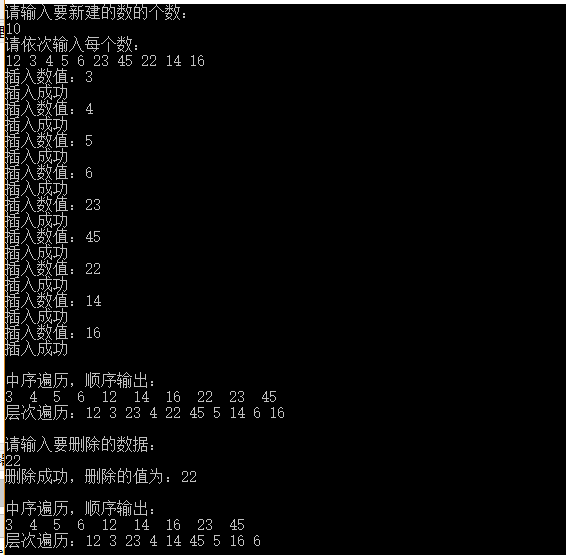
1. 调试成功程序及说明

**1、**

**题目：**实现二叉排序树的插入和删除。

**算法思想：**1）插入：左孩子小右孩子大，用插入数据与根节点比较，小则往左子树递归，大则往右子树递归。直到为子树空，插入到该位置。2）删除分三种情况：a.删除结点无孩子，直接删除；b.删除结点有一个孩子，删除后，将该孩子替代它；c.有两个孩子，删除根节点，找左子树中最右的结点，删除它并把它替代到要删除根节点的位置。

**运行结果：**



**结果分析：**成功。

**源程序：**

1. #include<iostream>
2. #include<cstdlib>
3. #include<queue>
5. **using** **namespace** std;
7. **typedef** **struct** BTNode {
8. **int** data;
9. BTNode \*lchild, \*rchild;
10. }BTNode, \*BiTree;

13. // 插入结点
14. **void** InsertBTree(BiTree &bTree, **int** data) {
15. cout<<"插入数值："<<data<<endl;
16. BTNode \*p = bTree, \*temp;
17. **while** (1) {
18. **if** (data < p->data) {
19. **if** (p->lchild) p = p->lchild;
20. **else** {
21. temp = (BTNode\*)malloc(**sizeof**(BTNode));
22. temp->data = data;
23. temp->lchild = temp->rchild = NULL;
24. p->lchild = temp;
25. **break**;
26. }
27. }
28. **else** **if** (data > p->data) {
29. **if** (p->rchild) p = p->rchild;
30. **else** {
31. temp = (BTNode\*)malloc(**sizeof**(BTNode));
32. temp->data = data;
33. temp->lchild = temp->rchild = NULL;
34. p->rchild = temp;
35. **break**;
36. }
37. }
38. **else** **break**;
39. }
40. cout<<"插入成功\n";
41. }

44. // 删除结点
45. **int** DeleteBTree(BiTree &bTree, **int** data) {
46. BTNode \*cur, \*pre;
47. cur = pre = bTree;
48. **while** (1) {
49. // 如果当前指针指向空结点，那么没找到这个值
50. **if** (!cur) **break**;
51. // 如果找到了这个值
52. **if** (cur->data == data) {
53. // 如果删去的结点没有孩子
54. **if** (!cur->lchild && !cur->rchild) {
55. // 如果删的点是根
56. **if** (bTree->data == data) bTree = NULL;
58. **if** (pre->lchild == cur) pre->lchild = NULL;
59. **else** pre->rchild = NULL;
60. }
61. // 如果删去的结点有两个孩子
62. **else** **if** (cur->lchild && cur->rchild) {
63. // 找到以该结点左孩子为根节点的子树中的最大值（即小于该值的数中的最大值）
64. BTNode \*cur2, \*pre2;
65. pre2 = cur;     // per2此处一定要等于cur，不是cur->lchild。这个在Delete时很关键
66. cur2 = cur->lchild;
67. **while** (cur2->rchild) {
68. pre2 = cur2;
69. cur2 = pre2->rchild;
70. }
71. // pre2不能等于cur2，不然如果出现要删的点为根节点的情况时，就不能将上一个结点的该孩子指针置为NULL
72. **int** tempData = DeleteBTree(pre2, cur2->data);
74. // 将找到的最大值结点取代要删除结点的位置
75. BTNode \*temp = (BTNode\*)malloc(**sizeof**(BTNode));
76. temp->data = tempData;
77. temp->lchild = cur->lchild;
78. temp->rchild = cur->rchild;
80. // 特殊情况，要删除的结点是根
81. **if** (bTree->data == data) bTree = temp;
82. **if** (pre->lchild == cur) pre->lchild = temp;
83. **else** pre->rchild = temp;
84. }
85. // 最后一种情况，该结点只有一个孩子
86. **else** {
87. // 找到不为空的孩子
88. BTNode \*temp;
89. **if** (cur->lchild) temp = cur->lchild;
90. **else** **if** (cur->rchild) temp = cur->rchild;
91. // 将该孩子取代这个结点的位置
92. // 要删除的结点为根时
93. **if** (bTree->data == data) {
94. bTree = temp;
95. }
96. **if** (pre->lchild == cur) pre->lchild = temp;
97. **else** pre->rchild = temp;
98. }
99. **break**;
100. }
101. **else** **if** (cur->data > data) {
102. pre = cur;
103. cur = cur->lchild;
104. }
105. **else** **if** (cur->data < data) {
106. pre = cur;
107. cur = cur->rchild;
108. }
109. }
110. **int** temp = cur->data;
111. free(cur);
112. cout<<"删除成功，删除的值为："<<temp<<endl;
113. **return** temp;
114. }

117. // 建二叉排序树
118. **void** CreateBTree(BiTree &bTree) {
119. cout<<"请输入要新建的数的个数：\n";
120. **int** n = 0;
121. cin>>n;
122. cout<<"请依次输入每个数：\n";
123. bTree = (BTNode\*)malloc(**sizeof**(BTNode));
124. cin>>bTree->data;
125. bTree->lchild = NULL;
126. bTree->rchild = NULL;
127. **int** data = 0;
128. **for** (**int** i=0; i<n-1; i++) {
129. cin>>data;
130. InsertBTree(bTree, data);
131. }
132. }

135. // 销毁二叉排序树
136. **void** DestoryBTree(BiTree &bTree) {
137. **if** (bTree) {
138. DestoryBTree(bTree->lchild);
139. DestoryBTree(bTree->rchild);
140. free(bTree);
141. }
142. }

145. // 中序遍历二叉排序树，即顺序输出数据
146. **void** InOrderBTree(BiTree bTree) {
147. **if** (bTree) {
148. InOrderBTree(bTree->lchild);
149. cout<<bTree->data<<"  ";
150. InOrderBTree(bTree->rchild);
151. }
152. }

155. // 层次遍历，非递归
156. **void** LevelOrderTraverse(BiTree bTree) {
157. **if** (!bTree) {
158. cout<<"该树为空！";
159. **return**;
160. }
162. cout<<"\n层次遍历：";
163. queue<BiTree> q;
164. q.push(bTree);
165. BiTree temp;
166. **while** (!q.empty()) {
167. temp = q.front();
168. cout<<temp->data<<" ";
169. **if** (temp->lchild) q.push(temp->lchild);
170. **if** (temp->rchild) q.push(temp->rchild);
171. q.pop();
172. }
173. cout<<endl;
174. }

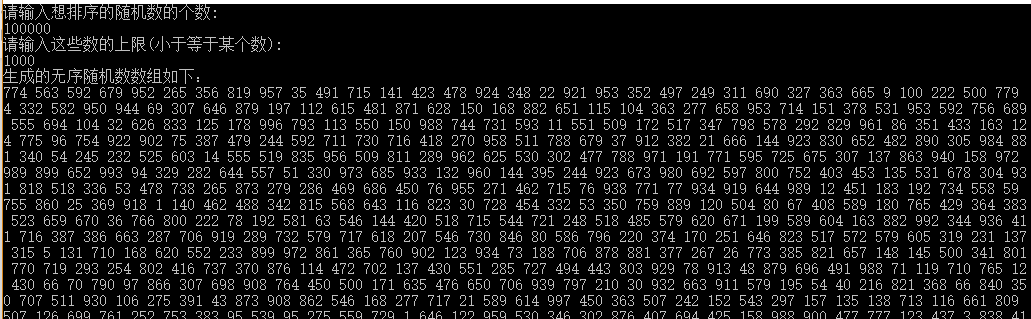
177. **int** main() {
178. BiTree bTree;
179. CreateBTree(bTree);
180. cout<<"\n中序遍历，顺序输出：\n";
181. InOrderBTree(bTree);
182. LevelOrderTraverse(bTree);
183. cout<<"\n请输入要删除的数据：\n";
184. **int** data;
185. cin>>data;
186. DeleteBTree(bTree, data);
187. //if (bTree) cout<<"删根失败"<<bTree<<"\n";
188. cout<<"\n中序遍历，顺序输出：\n";
189. InOrderBTree(bTree);
190. LevelOrderTraverse(bTree);
191. DestoryBTree(bTree);
192. }

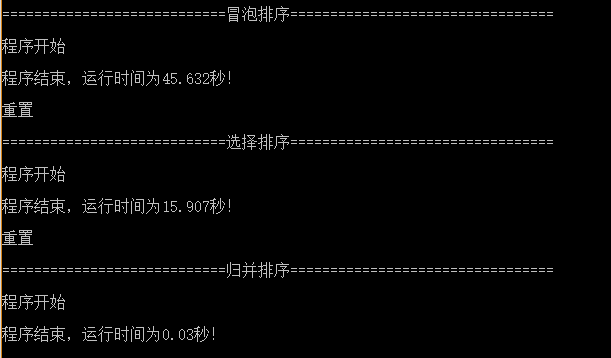
**2、**

**题目：**实现交换、选择、归并等简单排序算法；

**算法思想：**冒泡、选择都是简单基础排序。归并排序：对于一段数组先分两半，各自排序，再将两半部分内容有序地合并起来，采用递归实现。

**运行结果：**





**结果分析：**成功。

**源程序：**

1. /\*
2. 实现交换、选择、归并等简单排序算法；
3. \*/
4. #include<iostream>
5. #include<cstdlib>
6. #include<ctime>
8. **using** **namespace** std;
10. **clock\_t** start,finish;   // 开始时间，结束时间
11. **double** totalTime;       // 总的时间
13. **void** timeStart() {
14. cout<<"\n程序开始\n";
15. start = clock();
16. }
18. **void** timeFinish() {
19. finish = clock();
20. totalTime = (**double**)(finish-start)/CLOCKS\_PER\_SEC;
21. cout<<"\n程序结束，运行时间为"<<totalTime<<"秒！"<<endl;
22. }
24. // 重置测试数组
25. **void** Reset(**int** \*testNum, **int** \*num, **int** n) {
26. cout<<"\n重置\n";
27. **for** (**int** i=0; i<n; i++) {
28. testNum[i] = num[i];
29. //cout<<testNum[i]<<" ";
30. }
31. //cout<<endl;
32. }
34. // 遍历数组
35. **void** Traverse(**int** \*num, **int** n) {
36. cout<<"\n遍历数组：\n";
37. **for** (**int** i=0; i<n; i++) {
38. cout<<num[i]<<" ";
39. }
40. cout<<endl;
41. }
43. // 冒泡排序
44. **void** BubbleSort(**int** \*num, **int** n) {
45. **for** (**int** i=0; i<n; i++) {
46. **bool** allSort = **true**;
47. **for** (**int** j=1; j<n-i; j++) {
48. **if** (num[j] < num[j-1]) {
49. **int** temp = num[j];
50. num[j] = num[j-1];
51. num[j-1] = temp;
52. allSort = **false**;
53. }
54. }
55. **if** (allSort) **break**;
56. }
57. }
59. // 选择排序
60. **void** SelectSort(**int** \*num, **int** n) {
61. **for** (**int** i=0; i<n; i++) {
62. **int** min = num[i], pos;
63. **for** (**int** j=i+1; j<n; j++) {
64. **if** (num[j] < min) {
65. min = num[j];
66. pos = j;
67. }
68. }
69. num[pos] = num[i];
70. num[i] = min;
71. }
72. }
74. // 归并排序
75. **void** MergerSort(**int** \*num, **int** n) {
76. // 递归终止条件
77. **if** (n == 1) **return**;
79. **int** mid = n/2;
80. // 对前半部分归并排序
81. MergerSort(num, mid);
82. // 对后半部分归并排序
83. MergerSort(num+mid, n-mid);
85. // 将排好序两部分合并到temp数组
86. **int** \*temp = (**int**\*)malloc(**sizeof**(**int**) \* n);
87. **int** i = 0, j = mid, k = 0;
88. **while** (i<mid && j<n) {
89. **if** (num[i] <= num[j]) {
90. temp[k] = num[i];
91. i++;
92. } **else** {
93. temp[k] = num[j];
94. j++;
95. }
96. k++;
97. }
98. // i指针未到中间
99. **while** (i<mid) {
100. temp[k] = num[i];
101. k++;
102. i++;
103. }
104. // j指针未到结尾
105. **while** (j<n) {
106. temp[k] = num[j];
107. k++;
108. j++;
109. }
111. // 将排序好的temp数组赋给num
112. **for** (**int** i=0; i<n; i++) num[i] = temp[i];
113. free(temp);
114. }
116. // 快速排序
117. **void** QuickSort(**int** \*num, **int** n) {
118. // 递归终止条件
119. **if** (n <= 1) **return**;
121. **int** i = 0, j = n-1;
122. **int** temp = num[i];  // 将i位置的数先存起来，i位置现在相当于“空”
123. **while** (i<j) {
124. **while** (num[j] >= temp && j>i) j--;
125. // 此处直接将j位置不合格的数放在i位置就可以了，因为i位置本来就是“空的”
126. num[i] = num[j];
127. **while** (num[i] <= temp && i<j) i++;
128. num[j] = num[i];
129. }
130. num[i] = temp;
132. QuickSort(num, i);
133. QuickSort(num+i+1, n-i-1);  // 此处不用再把pos位置的数传进去排序了！！
134. }
136. // 堆排序中的筛选算法
137. **void** HeapAdjust(**int** \*num, **int** n, **int** pos) {
138. **int** lchild = 2\*pos+1, rchild = 2\*pos+2;
139. **int** maxPos;
140. **while** (pos<n && lchild<n) {
141. **if** (num[lchild] >= num[rchild]) maxPos = lchild;
142. **else** **if** (rchild < n) maxPos = rchild;
144. **if** (num[pos] < num[maxPos]) {
145. **int** temp = num[maxPos];
146. num[maxPos] = num[pos];
147. num[pos] = temp;
149. pos = maxPos;
150. lchild = 2\*pos+1, rchild = 2\*pos+2;
151. }
152. **else** **break**;
153. }
154. }
156. // 堆排序（完全二叉树排序）
157. **void** HeapSort(**int** \*num, **int** n) {
158. **for** (**int** i=n-1; i>=0; i--) {
159. HeapAdjust(num, n, i);
160. }
162. **for** (**int** i=n-1; i>0; i--) {
163. **int** temp = num[0];
164. num[0] = num[i];
165. num[i] = temp;
166. HeapAdjust(num, i, 0);
167. }
168. }
170. **int** main() {
171. **int** count = 0, limit = 0;
172. cout<<"请输入想排序的随机数的个数:\n";
173. cin>>count;
174. cout<<"请输入这些数的上限(小于等于某个数):\n";
175. cin>>limit;
177. cout<<"生成的无序随机数数组如下：\n";
178. **int** \*num = (**int**\*)malloc(**sizeof**(**int**) \* count);
179. **int** \*testNum = (**int**\*)malloc(**sizeof**(**int**) \* count);
180. srand(time(0));
181. **for** (**int** i=0; i<count; i++) {
182. num[i] = rand()%limit;
183. testNum[i] = num[i];
184. cout<<num[i]<<" ";
185. }
186. cout<<endl;

189. cout<<"\n============================冒泡排序=================================\n";
190. timeStart();
191. BubbleSort(testNum, count);
192. timeFinish();
193. //Traverse(testNum, count);

196. // 重置测试数组
197. Reset(testNum, num, count);
198. cout<<"\n============================选择排序=================================\n";
199. timeStart();
200. SelectSort(testNum, count);
201. timeFinish();
202. //Traverse(testNum, count);

205. // 重置测试数组
206. Reset(testNum, num, count);
207. cout<<"\n============================归并排序=================================\n";
208. timeStart();
209. MergerSort(testNum, count);
210. timeFinish();
211. //Traverse(testNum, count);

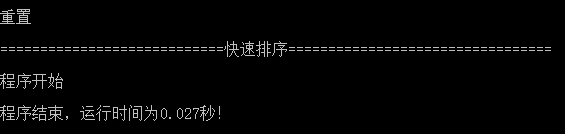
214. // 重置测试数组
215. Reset(testNum, num, count);
216. cout<<"\n============================快速排序=================================\n";
217. timeStart();
218. QuickSort(testNum, count);
219. timeFinish();
220. //Traverse(testNum, count);
222. // 重置测试数组
223. Reset(testNum, num, count);
224. //  int Num[10] = {2, 8, 3, 3, 5, 1, 9, 8, 0, 6};
225. //  count = 10;
226. //  Traverse(Num, count);
227. cout<<"\n=============================堆排序=================================\n";
228. timeStart();
229. HeapSort(testNum, count);
230. timeFinish();
231. //Traverse(testNum, count);
233. free(num);
234. free(testNum);
235. }

**3、**

**题目：**实现快速排序算法；

**算法思想：**用数组中第一个数作为比较值，将数组中小于这个数的数放在数组前半部分，大于这个数的数放在数组后半部分。然后再将前半部分和后半部分递归该操作。

**运行结果：**



**结果分析：**成功。

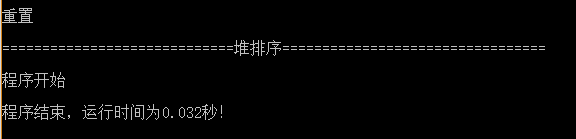
**源程序：**同2。

**4、**

**题目：**实现堆排序算法；

**算法思想：**1）建堆：从最后一个往第一个，逐个进行“筛选”，即建成最大顶堆；2）排序：将堆顶取出，与最后一个进行交换，然后再对第一个点进行“筛选”，此次筛选数组长度减一。

**运行结果：**



**结果分析：**成功。

**源程序：**同2。

**5、**

**题目：**

问题描述

　　给定n个整数，请统计出每个整数出现的次数，按出现次数从多到少的顺序输出。

输入格式

　　输入的第一行包含一个整数n，表示给定数字的个数。

　　第二行包含n个整数，相邻的整数之间用一个空格分隔，表示所给定的整数。

输出格式

输出多行，每行包含两个整数，分别表示一个给定的整数和它出现的次数。按出现次数递减的顺序输出。如果两个整数出现的次数一样多，则先输出值较小的，然后输出值较大的。

样例输入

12

5 2 3 3 1 3 4 2 5 2 3 5

样例输出

3 4

2 3

5 3

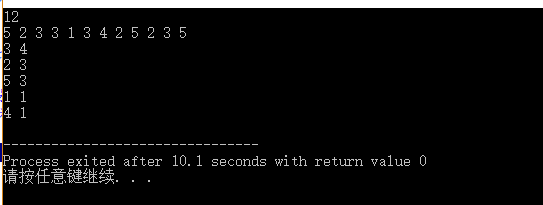
1 1

4 1

问题分析：该题目可用一个数组，以下标作为数，数组内容存储该数出现次数来实现（这就相当于直接映射，是机试题目里面常用的一种解题法，很多看似非线性的题型最后其实都可以采取哈希或者映射的方法来巧解，尤其是一些看似是树形结构的模拟类题目，要好好体会哈希思想在机试题目中的巧用）。

**算法思想：**

**运行结果：**



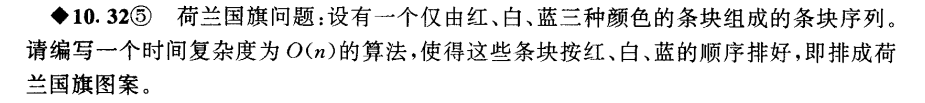
**结果分析：**成功。

**源程序：**

1. #include<iostream>
2. #include<vector>
4. **using** **namespace** std;
6. **int** main() {
7. **int** n;
8. scanf("%d", &n);
9. **int** a[n];
10. **for** (**int** i=0; i<n; i++) {
11. scanf("%d", &a[i]);
12. }
13. **int** num[1005];
14. **for** (**int** i=0; i<=1000; i++) num[i] = 0;
16. vector<**int**> p;
17. **for** (**int** i=0; i<n; i++) {
18. **if** (!num[a[i]]) p.push\_back(a[i]);
19. num[a[i]]++;
20. }
22. **for** (**int** i=0; i<p.size(); i++) {
23. **int** max = p[i];
24. **int** maxPos = i;
25. **for** (**int** j=i; j<p.size(); j++) {
26. **if** (num[p[j]] > num[max] || (num[p[j]] == num[max] && p[maxPos] > p[j])) {
27. max = p[j];
28. maxPos = j;
29. }
30. }
31. p[maxPos] = p[i];
32. p[i] = max;
33. }
35. **for** (**int** i=0; i<p.size(); i++) {
36. cout<<p[i]<<" "<<num[p[i]]<<endl;
37. }
38. }

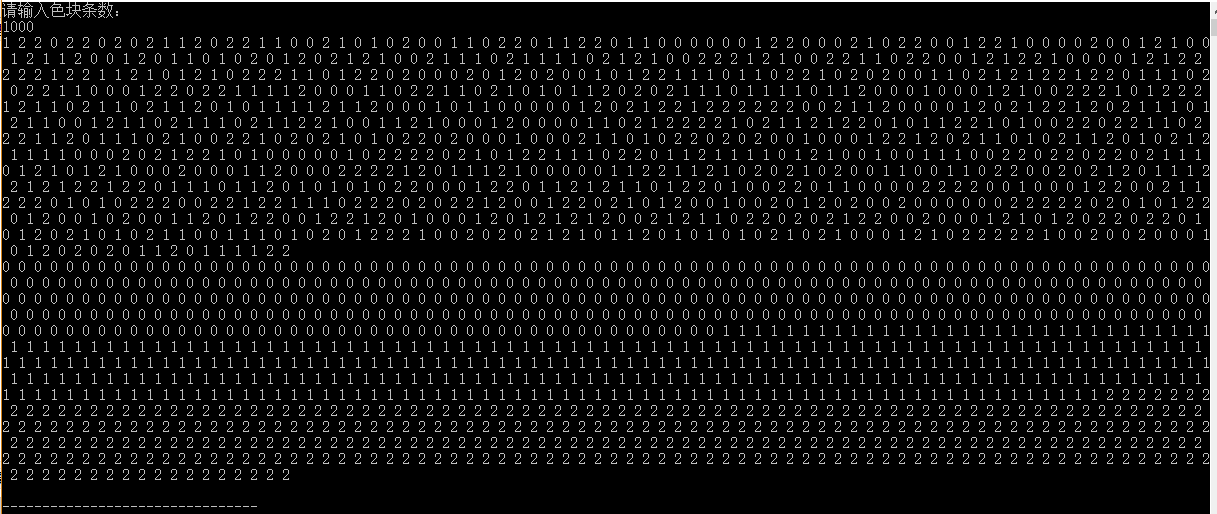
**6、**

**题目：**



**算法思想：**i从头往后,ki在i之后；j从尾往前,kj在j之前。红0，白1，蓝2。i位置之前都为0，i-ki位置内都为1，若ki遇到0则送到i位置，遇到2则送到j位置。j与kj同理。具体思想见代码注释。

**运行结果：**



**结果分析：**成功。

**源程序：**

1. /\*
2. 荷兰国旗
3. \*/
4. #include<iostream>
5. #include<cstdlib>
6. #include<ctime>
8. **using** **namespace** std;
10. **int** main() {
11. srand(time(0));
12. **int** n;
13. cout<<"请输入色块条数：\n";
14. cin>>n;
15. **int** a[n];
16. **for** (**int** i=0; i<n; i++) {
17. a[i] = rand()%3;
18. }
20. **for** (**int** i=0; i<n; i++) {
21. cout<<a[i]<<" ";
22. }
23. cout<<endl;
25. **int** i = 0, j = n-1, k;
26. **int** temp, t;
28. // 当i位置为0，前进
29. **while** (a[i] == 0) i++;
30. // 当k位置为0或2时，前进
31. k = i;
32. **while** (a[k] != 1) {
33. // 如果k位置为0，与i位置（i位置必为2）交换，i前进
34. **if** (a[k] == 0) {
35. t = a[k];
36. a[k] = a[i];
37. a[i] = t;
38. i++;
39. }
40. // k前进
41. k++;
42. }
43. // 拿到1存起来
44. temp = a[k];
45. a[k] = a[i];
47. **int** ki = i, kj = j;
48. **while** (1) {
49. // j位置往前走，直到不为2
50. **while** (a[j] == 2) j--;
51. // 若kj在j之后，则将kj=j
52. **if** (kj > j) kj = j;
53. // 当kj位置为1或2时，kj--
54. **while** (a[kj] != 0 && kj > ki) {
55. // 当kj位置为2时，与j位置（j位置必为1）交换，j--
56. **if** (a[kj] == 2) {
57. t = a[kj];
58. a[kj] = a[j];
59. a[j] = t;
60. j--;
61. }
62. kj--;
63. }
64. // 如果kj与ki相遇，则结束
65. **if** (kj == ki) **break**;
66. // 当kj位置为0时，放到i位置（i位置原来为空），同时将j位置的1放到kj位置，j位置为空
67. a[i] = a[kj];
68. a[kj] = a[j];
70. // i位置同理
71. **while** (a[i] == 0) i++;
72. **if** (ki < i) ki = i;
73. **while** (a[ki] != 2 && ki < j) {
74. **if** (a[ki] == 0) {
75. t = a[ki];
76. a[ki] = a[i];
77. a[i] = t;
78. i++;
79. }
80. ki++;
81. }
82. **if** (ki == kj) **break**;
83. a[j] = a[ki];
84. a[ki] = a[i];
85. }
86. a[i] = temp;
88. **for** (**int** i=0; i<n; i++) {
89. cout<<a[i]<<" ";
90. }
91. cout<<endl;
92. }