1. 快速排序
2. #include <iostream>
3. #include <vector>
5. **using** **namespace** std;
7. **void** swap(vector<**int**> &nums, **int** i, **int** j) {
8. **int** temp = nums[i];
9. nums[i] = nums[j];
10. nums[j] = temp;
11. }
13. **int** mypartition(vector<**int**> &nums, **int** p, **int** r) {
14. **int** x = nums[r];
15. **int** i = p - 1;
16. **for** (**int** j = p;j < r;j++) {
17. **if** (nums[j] <= x) {
18. i ++;
19. swap(nums,i,j);
20. }
21. }
22. i += 1;
23. swap(nums,i,r);
24. **return** i;
25. }
27. **void** quicksort(vector<**int**> &nums, **int** p, **int** r) {
28. **if** (p < r) {
29. **int** q = mypartition(nums,p,r);
30. quicksort(nums,p,q-1);
31. quicksort(nums,q+1,r);
32. }
33. }
35. **int** main() {
36. **int** a[5] = {2,1,3,5,4};
37. vector<**int**> test(a,a+5);
38. quicksort(test,0,test.size()-1);
39. **for** (**int** i = 0;i < test.size();i++) {
40. cout << test[i] << " ";
41. }
42. **return** 0;
43. }
44. 堆排序
45. #include <iostream>
46. #include <vector>
48. #define left(x) 2\*x+1;//获得左节点在数组中的下标
49. #define right(x) 2\*(x+1);//获得右节点在数组中的下标
51. **using** **namespace** std;
53. **void** MaxHeapify(**int**\* a,**int** i,**int** low,**int** high)//输入为要被排序的数组和根节点，数组a当中被维护的那一部分的下标low，high
54. {
55. **int** l = left(i);//计算下标为i的节点的左子节点
56. **int** r = right(i);//计算下标为i的节点的右子节点
57. **int** largest;//保存i,l,r(即i和它的左右子节点)之间的最大数的下标
58. **int** temp;//交互数组中的数所使用的临时变量
59. //找到三个数当中最大的那个数，将最大的那个数和i进行互换
60. **if** (l<=high && a[l]>a[i])
61. {
62. largest = l;
63. }
64. **else**{
65. largest = i;
66. }
68. **if** (r<=high && a[r]>a[largest])
69. {
70. largest = r;
71. }
72. **if** (largest != i)
73. {
74. temp = a[i];
75. a[i] = a[largest];
76. a[largest] = temp;
77. MaxHeapify(a, largest,low,high);//交换有可能破坏子树的最大堆性质，所以对所交换的那个子节点进行一次维护，而未交换的那个子节点，根据我们的假设，是保持着最大堆性质的。
78. }
79. }
81. **void** BuildMaxHeap(**int**\* a,**int** length)
82. {
83. **for** (**int** i = length / 2-1; i >= 0; i--)
84. {
85. MaxHeapify(a, i, 0, length - 1);
86. }
87. }
89. **void** HeapSort(**int** a[],**int** length)
90. {
91. **int** temp;
92. BuildMaxHeap(a,length);
93. **for** (**int** i = length - 1; i >= 1; i--)
94. {
95. //交换根节点和数组的最后一个节点
96. temp = a[i];
97. a[i] = a[0];
98. a[0] = temp;
99. MaxHeapify(a, 0, 0, i-1);//维护从下标为i-1到0的子数组
100. }
101. }
103. **void** HeapDelete(**int** a[],**int** length,**int** target) {
104. **int** index = 0;
105. **for** (**int** i = 0;i < length;i++) {
106. **if** (a[i] == target) {
107. length --;
108. a[i] = a[length];
109. index = i;
110. **break**;
111. }
112. }
113. BuildMaxHeap(a,length);
114. }
116. **int** main() {
117. **int** a[5] = {2,1,3,5,4};
118. //HeapSort(a,4);
119. BuildMaxHeap(a,5);
120. **for** (**int** i = 0;i < 5;i++) {
121. cout << a[i] << " ";
122. }
123. HeapDelete(a,5,2);
124. **for** (**int** i = 0;i < 4;i++) {
125. cout << a[i] << " ";
126. }
127. **return** 0;
128. }
129. 归并排序
130. ///两路
131. #include <iostream>
133. **using** **namespace** std;
135. **void** merge(**int** \*data,**int** start,**int** end,**int** \*result)
136. {
137. **int** left\_length = (end - start + 1) / 2 + 1;//左部分区间的数据元素的个数
138. **int** left\_index = start;
139. **int** right\_index = start + left\_length;
140. **int** result\_index = start;
141. **while**(left\_index < start + left\_length && right\_index < end+1)
142. {
143. //对分别已经排好序的左区间和右区间进行合并
144. **if**(data[left\_index] <= data[right\_index])
145. result[result\_index++] = data[left\_index++];
146. **else**
147. result[result\_index++] = data[right\_index++];
148. }
149. **while**(left\_index < start + left\_length)
150. result[result\_index++] = data[left\_index++];
151. **while**(right\_index < end+1)
152. result[result\_index++] = data[right\_index++];
153. }
155. **void** merge\_sort(**int** \*data, **int** start, **int** end, **int** \*result)
156. {
157. **if**(1 == end - start)//如果区间中只有两个元素，则对这两个元素进行排序
158. {
159. **if**(data[start] > data[end])
160. {
161. **int** temp  = data[start];
162. data[start] = data[end];
163. data[end] = temp;
164. }
165. **return**;
166. }
167. **else** **if**(0 == end - start)//如果只有一个元素，则不用排序
168. **return**;
169. **else**
170. {
171. //继续划分子区间，分别对左右子区间进行排序
172. merge\_sort(data,start,(end-start+1)/2+start,result);
173. merge\_sort(data,(end-start+1)/2+start+1,end,result);
174. //开始归并已经排好序的start到end之间的数据
175. merge(data,start,end,result);
176. //把排序后的区间数据复制到原始数据中去
177. **for**(**int** i = start;i <= end;++i)
178. data[i] = result[i];
179. }
180. }
181. ///多路

184. **int** main() {
185. **int** a[5] = {2,1,3,5,4};
186. **int** result[5];
187. merge\_sort(a,0,5,result);
188. **for** (**int** i = 0;i < 5;i++) {
189. cout << a[i] << " ";
190. }
191. **return** 0;
192. }
193. 二分查找
194. #include <iostream>
195. #include <vector>
197. **using** **namespace** std;
199. **int** binary\_find(vector<**int**> &nums, **int** start, **int** end, **int** target) {
200. **if** (start > end) **return** -1;
201. **int** mid = (start + end)/2;
202. **if** (nums[mid] == target) {
203. **return** mid;
204. }
205. **else** **if** (nums[mid] < target) {
206. binary\_find(nums,mid+1,end,target);
207. }
208. **else** **if** (nums[mid] > target){
209. binary\_find(nums,start,mid-1,target);
210. }
211. }
213. **int** main() {
214. **int** a[5] = {1,2,3,4,6};
215. vector<**int**> test(a,a+5);
216. cout << binary\_find(test,0,test.size()-1,6) << endl;
217. **return** 0;
218. }

5.回溯生成全排列

1. #include <iostream>
2. #include <string>
3. #include <vector>
4. #include <algorithm>
6. **using** **namespace** std;
8. //next\_permutation
9. //prev\_permutation
11. **void** generate (vector<vector<**int**> > &result, vector<**int**> &temp, **int** index, **int** n) {
12. **if** (index == n){
13. result.push\_back(temp);
14. }
15. **else** {
16. **for** (**int** i = 0;i < n;i++) {
17. vector<**int**>::iterator it = find(temp.begin(),temp.end(),i+1);
18. **if** (it == temp.end()) {
19. temp.push\_back(i+1);
20. generate(result,temp,index+1,n);
21. temp.pop\_back();
22. }
23. }
24. }
25. }
27. **int** main() {
28. vector<vector<**int**> > result;
29. vector<**int**> temp;
30. generate(result,temp,0,10);
31. **for** (**int** i = 0;i < result.size();i++) {
32. **for** (**int** j = 0;j < result[i].size();j++) {
33. cout << result[i][j] << " ";
34. }
35. cout << endl;
36. }
37. **return** 0;
38. }

