~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

题目描述如下哦：

一个循环有序数组，例如：7,8,9,0,1,2,3,4,5,6 不知道其最小值的位置，要查找任一数值的位置。算法时间复杂度是log2(n)。

---------

分下如下哦：

时间复杂度是log2(n),这么小一定就是二分查找了，但是二分查找技术的要求是相当高的，要求是啥呢？嘿嘿，它要求

1、是顺序存储的线性表

2、记录必须按关键码有序

由题目描述可知：符合顺序存储的线性表，但是呢又不完全是按关键码有序的，是循环有序的，肯定是用二分查找，关键是怎么用二分查找。可以这样考虑：从二分查找的要求条件考虑，必须是有序的，可以有两种有序的方式：整个数组有序或者部分有序。既然整体有序是不可能的了，那就从部分有序来着手吧。下面就是考虑怎么变成部分有序。

用s代表首号元素，m代表中间元素，e代表末尾元素，那么根据中间元素就可以把整个数组分成两个子数组：s1,s2；（这里我有一个纠结的地方，就是当数组个数为偶数时，中间元素m到底分到哪个子数组中去，其实这里有的时候并不是关键因素，即分到哪边去都一样，具体要看下一步是怎么操作的）

这样的话，因为是循环有序的数组，两个子数组s1,s2中肯定有一个是有序的，而另一个是无序的，那么如果要查找的数字在有序的这个子数组里，那就可以直接用二分查找了，如果不是的话就再继续这样分，直到他在有序的那一边。这样就是从部分有序着手解决问题了。

由于最小值的位置不定，所以该数组有三种情况排列：

（1）最小值的位置在（s,m】之间时

 则s,m,e满足以下关系：

m<=e<=s;  ----其中m--e之间是有序的子数组

（2）最小值的位置在（m,e]之间时

则s,m,e满足以下关系：

e<=s<=m; ----其中s--m之间是有序的子数组

 (3) s就是整个数组中最小的元素

则s,m,e满足以下关系

s<=m<=e; ---此时两个子数组都是有序的子数组，直接可以用二分查找了。

当然分析最小值的位置时是这样分析的，编程的时候是要逆向编代码的（通常都是这样的吧，分析的时候是正向思维，而编写代码的时候是逆向思维）

只能通过已知的整体数组知道s,m,e的具体值，进而比较它们的大小关系进而确定最小值的范围，则通过比较待查值跟s,m,e,比较大小即可知道待查值在具体的哪个子数组中喽。。。

算法思路就是如此，那么下面看具体代码喽：

**[c-sharp]** [view plaincopy](http://blog.csdn.net/aixiaolin/article/details/6408679)

1. #include <stdlib.h>
2. #include <stdio.h>
3. **int** binarySearch(**int** target, **int** array[], **int** start, **int** end)
4. {
5. **int** mid = -1;
6. **int** l = start - 1;
7. **int** r = end + 1;
8. **while**(l+1 != r) {
9. mid = l + (r - l) / 2;
10. **if**(array[mid] < target) {
11. l = mid;
12. } **else** {
13. r = mid;
14. }
15. }
16. **if**(r <= end && array[r] == target) {
17. **return** r;
18. } **else** {
19. **return** -1;
20. }
21. }
22. **int** search(**int** target, **int** array[], **int** start, **int** end)
23. {
24. **if**(start > end){
25. **return** -1;
26. }
27. **int** mid = start + (end - start) / 2;
28. **int** s = array[start];
29. **int** m = array[mid];
30. **int** e = array[end];
31. **if**(s <= m) {
32. **if**(s <= e) { //mini value is at start
33. **return** binarySearch(target, array, start, end);
34. } **else** { //mini value is in range (mid, end]
35. **if**(target >=s && target <= m) {
36. **return** binarySearch(target, array, start, mid);
37. } **else** {
38. **return** search(target, array, mid+1, end);
39. }
40. }
41. } **else** { //mini value is in range (start, mid]
42. **if**(target > m && target <= e) {
43. **return** binarySearch(target, array, mid + 1, end);
44. } **else** {
45. **return** search(target, array, start, mid);
46. }
47. }
48. }
49. **int** main(**int** argc, **char**\* argv[])
50. {
51. **int** t1[] = {7,8,9,0,1,2,3,4,5,6};
52. **int** i = -1;
53. **int** index = -1;
54. **for**(; i< 11; i++) {
55. index = search(i, t1, 0, 9);
56. printf("Find %d at index %d!/n", i, index);
57. }
58. **return** 0;
59. }

当具体查找在哪个有序子数组里时 ，先确定最小值的位置，然后再比较待查值跟s,m,e的大小关系进而确定。