**1 STL之upper\_bound算法**

算法思路：

1）输入排序数组a[left,…，right]以及key，现在需要找出数组中的一个大于key的数；

2）折半思想，逼近寻找a[l] <= key < a[r]，l < r；

3）计算m = (l + r) >> 1，

如果a[m] > key，则r = m；

如果a[m] <= key，则l = m + 1；

4）由于循环条件l<r，算法的每次操作都可以保证r=m>=1或者l<=r，这样此算法必终止于l=r处，即数组中只有一个元素的情况，如果a[l] > key，则l即为所求，否则未查找到返回-1；

**2 STL之lower\_bound算法**

算法思路：

1）输入排序数组a[left,…，right]以及key，现在需要找出数组中的一个不小于key的数；

2）折半思想，逼近寻找a[l] < key <= a[r]，l < r；

3）计算m = (l + r) >> 1，

如果a[m] >= key，则r = m；

如果a[m] < key，则l = m + 1；

4）由于循环条件l<r，算法的每次操作都可以保证r=m>=l或者l<=r，这样此算法必终止于l=r处，即数组中只有一个元素的情况，如果a[l] >= key，则l即为所求，否则未查找到返回-1；

**3 旋转数组查找**

Suppose a sorted array is rotated at some pivot unknown to you beforehand. (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

**1 You are given a target value to search. If found in the array return its index, otherwise return -1.**

You may assume no duplicate exists in the array.

**2 there are duplicate in the array.**

**3 find the minumn number.**

**1 算法思路**

1）计算m=（left+right）/ 2；

2）如果a[m] == key，则返回m；

3）否则，

若a[left] <= a[m]，说明数组left…m部分有序，

如果a[left] <= key < a[m]，说明必须搜索left…m部分，因为m…right部分大于或者小于key，此时即转化为二分搜索，right=m-1；

否则搜索m…right部分，left = m+1；

若a[left] > a[m]，说明m…right部分有序，

如果a[m] <= key < a[right]，说明必须搜索m…right部分，因为left…m部分小于或者大于key，此时即转化为二分搜索，left=m+1；

否则搜索left…m部分，right = m – 1；

**代码如下：**

int binary\_search\_nodup(int a[], int left, int right, int key)

{

while(left <= right)

{

int m = (left + right) >> 1;

if(a[m] == key) return m;

else if(a[left] <= a[m]) //when m=left, a[l] == a[m]

{

if(a[left] <= key && key < a[m])

right = m - 1;

else

left = m + 1;

}

else //right part is sorted

{

if(a[m] < key && key <= a[right])

left = m + 1;

else

right = m - 1;

}

}

return -1;

}

此算法适用于数组中无重复元素的情况，分支条件a[left]<=a[m]中的等号适用于left=m的情况；

如果key位于有序部分，则后续查找都会进入a[left] <= a[m]分支；

如果key一直位于无序部分，则操作会二分至只有一个元素的情况；

**2 算法思路**

如果a[left] < a[m]，必定left…m有序；

如果a[left] > a[m]，必定m…right有序；

区别在于a[left] == a[m]的情况发生时，left不一定等于m，此时

如果a[m] ==a[right]，那么无法确定哪部分有序，此时需要搜索整个数组，left++排除一个元素；比如 6 **7** 3 **6** 6 6 6 或者 6 6 6 **6** **7** 4 5 6

如果a[m] != a[right]，此时left…m部分的元素值均相同，继续搜索m+1…right部分；

int binary\_search\_withdup(int a[], int left, int right, int key)

{

while(left <= right)

{

int m = (left + right) >> 1;

if(a[m] == key) return m;

else if(a[left] < a[m])

{

if(a[left] <= key && key < a[m])

right = m - 1;

else

left = m + 1;

}

else if(a[left] > a[m])//right part is sorted

{

if(a[m] < key && key <= a[right])

left = m + 1;

else

right = m - 1;

}

else

{

if(a[m] != a[right])

{

left = m + 1;

}

else

{

left++;

}

}

}

return -1;

}

**3 算法思路**

先考虑数组无重复元素的情况：

1）计算m =（left+right）/ 2；

2）如果a[m] > a[right]，则最小值在m…right部分，left = m + 1；

3）如果a[m] < a[right]，则最小值在left…m部分，right = m；

int findMin\_nodup(int a[], int left, int right)

{

while(left < right)

{

int m = (left + right) >> 1;

if(a[m] > a[right])

left = m + 1;

else if(a[m] < a[right])

right = m;

}

return left;

}

对于有重复元素的情况，由于循环条件为left<right，因此m < right，如果a[m] = a[right]，我们可以去掉a[right]这个元素，因为此时如果a[right]为数组中的最小值，a[m]也为数组中的最小值；

int findMin\_withdup(int a[], int left, int right)

{

while(left < right)

{

int m = (left + right) >> 1;

if(a[m] > a[right])

left = m + 1;

else if(a[m] < a[right])

right = m;

else

**right--;**

}

return left;

}