**[二分搜索及其扩展](http://www.cnblogs.com/luxiaoxun/archive/2012/10/02/2710161.html)**

**二分搜索**

折半搜索，也称二分查找算法、二分搜索，是一种在有序数组中查找某一特定元素的搜索算法。搜素过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜素过程结束；如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组为空，则代表找不到。这种搜索算法每一次比较都使搜索范围缩小一半。

时间复杂度：二分搜索每次把搜索区域减少一半，很明显时间复杂度为O(logN)。   
空间复杂度：O(1)，虽以递归形式定义，但是尾递归，可改写为循环。

**二分搜索的基本实现**

二分查找法在算法家族大类中属于“分治法”，分治法基本都可以用递归来实现的，二分查找法的递归实现如下：

[复制代码](javascript:void(0);)

int binary\_search(int array[], int low, int high, int target)

{

if (low > high) return -1;

int mid = (low + high)/2;

if (array[mid]> target)

return binary\_search(array, low, mid -1, target);

if (array[mid]< target)

return binary\_search(array, mid+1, high, target);

return mid;

}

[复制代码](javascript:void(0);)

非递归实现：

[复制代码](javascript:void(0);)

int binary\_search(int array[], int low, int high, int target)

{

while(low <= high)

{

int mid = (low + high)/2;

if (array[mid] > target)

high = mid - 1;

else if (array[mid] < target)

low = mid + 1;

else //find the target

return mid;

}

//the array does not contain the target

return -1;

}

[复制代码](javascript:void(0);)

**在轮转后的有序数组上应用二分查找法**

二分法是要应用在有序的数组上，如果是无序的，那么比较和二分就没有意义了。不过还有一种特殊的数组上也同样可以应用，那就是“轮转后的有序数组（Rotated Sorted Array）”。它是有序数组，取期中某一个数为轴，将其之前的所有数都轮转到数组的末尾所得。比如{7, 11, 13, 17, 2, 3, 5}就是一个轮转后的有序数组。非严格意义上讲，有序数组也属于轮转后的有序数组——取首元素作为轴进行轮转。

下边就是二分查找法在轮转后的有序数组上的实现（假设数组中不存在相同的元素）

[复制代码](javascript:void(0);)

int SearchInRotatedSortedArray(int array[], int low, int high, int target)

{

while(low <= high)

{

int mid = (low + high) / 2;

if (target < array[mid])

if (array[mid] < array[high])//the higher part is sorted

high = mid - 1; //the target would only be in lower part

else //the lower part is sorted

if(target < array[low])//the target is less than all elements in low part

low = mid + 1;

else

high = mid - 1;

else if(array[mid] < target)

if (array[low] < array[mid])// the lower part is sorted

low = mid + 1; //the target would only be in higher part

else //the higher part is sorted

if (array[high] < target)//the target is larger than all elements in higher part

high = mid - 1;

else

low = mid + 1;

else //if(array[mid] == target)

return mid;

}

return -1;

}

[复制代码](javascript:void(0);)

对比普通的二分查找法，为了确定目标数会落在二分后的那个部分，需要更多的判定条件。但还是实现了O(log n)的目标。

**找到轮转后的有序数组中第K小的数**

对于普通的有序数组来说，这个问题是非常简单的，因为数组中的第K-1个数（即A[K-1]）就是所要找的数，时间复杂度是O(1)常量。但是对于轮转后的有序数组，在不知道轮转的偏移位置，我们就没有办法快速定位第K个数了。

不过我们还是可以通过二分查找法，在log(n)的时间内找到最小数的在数组中的位置，然后通过偏移来快速定位任意第K个数。当然此处还是假设数组中没有相同的数，原排列顺序是递增排列。

在轮转后的有序数组中查找**最小数**的算法如下：

[复制代码](javascript:void(0);)

//return the index of the min value in the Rotated Sorted Array, whose range is [low, high]

int findIndexOfMinVaule(int A[], int low, int high)

{

if (low > high) return -1;

while (low < high)

{

int mid = (low + high)/2;

if (A[mid] > A[high])

low = mid +1;

else

high = mid;

}

//at this point, low is equal to high

return low;

}

[复制代码](javascript:void(0);)

接着基于此结果进行偏移，再基于数组长度对偏移后的值取模，就可以找到第K个数在数组中的位置了：

[复制代码](javascript:void(0);)

//return the index of the kth element in the Rotated Sorted Array

int findKthElement(int A[], int m, int k)

{

if (k > m) return -1;

int base = findIndexOfMinVaule(A, 0, m-1);

int index = (base+k-1) % m;

return index;

}

[复制代码](javascript:void(0);)

**整数的求平方根函数**

这个其实也是毕竟常见的面试问题，要求不调用math库，实现对整数的sqrt方法，返回值只需要是整数。  
其实这个问题用数学的表达方式就是：对于非负整数x，找出另一个非负整数n，其中n满足 n^2 <= x < (n+1)2。  
所以最直接的方法就是从0到x遍历过去直到找到满足上述条件的n。这个算法的复杂度自然是O(n)。

仔细想想，其实要找的数是在0和x之间，而他正巧可以视为一个有序的数组。似乎有可以运用二分查找法的可能。再回想二分查找法是要找到满足“与目标数相等”这一条件的数，而这里同样也是要找满足一定条件的数。所以就可以用二分法来解这个问题了，让复杂度降为O(logn)。

为方便起见，假设传入的参数是非负的整数，因此使用unsigned int。

[复制代码](javascript:void(0);)

unsigned int sqrt(unsigned int x)

{

//no value should larger than max\*max, otherwise it would be overflow

unsigned int max = (1 << (sizeof(x)/2\*8))-1; //65535

if (max\*max < x) return max;

unsigned int low = 0;

unsigned int high = max-1;

unsigned int mid = 0;

while (1)

{

mid = (low + high)/2;

if (x < mid \* mid)

high = mid-1;

else if((mid+1)\*(mid+1) <= x)

low = mid+1;

else //if(mid \* mid <= x && x < (mid+1)\*(mid+1))

break;

}

return mid;

}

[复制代码](javascript:void(0);)

题目：有一类数组，例如数组[1,2,3,4,6,8,9,4,8,11,18,19,100] 前半部分是是一个递增数组，后面一个还是递增数组，但整个数组不是递增数组，那么怎么最快的找出其中一个数？  
分析：此题数组不是严格递增的数据，因为有重复的元素。对数组的前半部分和后半部分分别进行二分查找。

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

//二分查找

int binary\_search(int\* a, int low, int high, int goal)

{

while(low <= high)

{

int middle = low + ((high-low)>>1);

if(a[middle] == goal)

return middle;

else if(a[middle] < goal)

low = middle + 1;

else

high = middle - 1;

}

return -1;

}

void getNum(int \*a, int len, int goal)

{

int i, index;

for(i = 0; i < len-1; i++)

{

if(a[i] > a[i+1]) //找到前、后两个数组的分界点

break;

}

if(a[i] >= goal) //对前面数组进行二分查找

{

index = binary\_search(a, 0, i, goal);

printf("%d\n",index);

}

if(a[i+1] <= goal) //对后面数组进行二分查找

{

index = binary\_search(a, i+1, len-1, goal);

printf("%d\n",index);

}

}

int main(void)

{

int a[]={1,2,3,4,6,8,9,4,8,11,18,19,100};

int len = 13, goal = 8;

getNum(a,len,goal);

return 0;

}

[复制代码](javascript:void(0);)

有序数组中找中位数：<http://www.cnblogs.com/luxiaoxun/archive/2012/09/13/2684054.html>

Find the k-th Smallest Element in the Union of Two Sorted Arrays – LeetCode   
<http://www.leetcode.com/2011/01/find-k-th-smallest-element-in-union-of.html>

Median of Two Sorted Arrays –LeetCode   
<http://www.leetcode.com/2011/03/median-of-two-sorted-arrays.html>