BinarySearch & Sorted Array (chapter 2)
学习Binary Search的模板
讲解Search in Rotated Sorted Array等5-7道高频题
学习排序数组的一般处理方法
讲解Median of Two Sorted Array等3-5道高频题

Binary Search
Binary search is a famous question in algorithm. For a given sorted array (ascending order) and a target number, find the first index of this number in O(log n) time complexity. If the target number does not exist in the array, return -1. Example If the array is [1, 2, 3, 3, 4, 5, 10], for given target 3, return 2.
Answer:

```
/*binary search解法模版
T(n) = T(n / 2) + O(1)
T(n)
T(n / 2) T(n / 2)
T(n / 2^2) T(n / 2^2) T(n / 2^2) T(n / 2^2)
...
n / 2^x = 1, thus x = logn
*/
/*
after the while loop, start + 1 = end, that is,
start和end已经相邻，所以在这里判断到底是谁等于target，由此决定输出哪个量来作为结果
consider test case target = 1 1(start) 2(end) 3 4 5 6 7 8
*/
public class Solution {
    public int binarySearch(int[] nums, int target) {
        if (nums.length == 0) {
            return -1;
        }

        int start = 0;
        int end = nums.length - 1;
        int mid;
        while (start + 1 < end) {
            mid = start + (end - start) / 2; // (start + end) / 2 may stackoverflow due to too large value
            if (nums[mid] == target) {
                end = mid; // 相邻即退出循环，解决了相邻重复数取前取后的问题
            } else if (nums[mid] > target) {
                end = mid;
            } else if (nums[mid] < target) {
                start = mid;
            }
        }

        if (nums[end] == target) {
            return end;
        } else if (nums[start] == target) {
            return start;
```

```
        }
        return -1;
    }
}
```

Search for a Range
Given a sorted array of integers, find the starting and ending position of a given target value.
Your algorithm's runtime complexity must be in the order of O(log n).
If the target is not found in the array, return [-1, -1].
For example,
Given [5, 7, 7, 8, 8, 10] and target value 8,
return [3, 4].
Answer:

```
/*
注意最后if和else if的顺序，因为已知有重复数，循环结束start和end指向的肯定是相同值的两数，不依照顺序指根据数值大小输出肯定会报错
*/
public class Solution {
    public int[] searchRange(int[] A, int target) {
        int[] bound = new int[2];
        int start, mid, end;
        start = 0;
        end = A.length - 1;
        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (A[mid] == target) {
                end = mid;
            } else if (A[mid] > target) {
                end = mid;
            } else if (A[mid] < target) {
                start = mid;
            }
        }
        if (A[start] == target) {
            bound[0] = start;
        } else if (A[end] == target) {
            bound[0] = end;
        } else {
            bound[0] = bound[1] = -1;
            return bound;
        }

        start = 0;
        end = A.length - 1;
        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (A[mid] == target) {
                start = mid;
            } else if (A[mid] > target) {
```

```
                end = mid;
            } else if (A[mid] < target) {
                start = mid;
            }
        }
        if (A[end] == target) {
            bound[1] = end;
        } else if (A[start] == target) {
            bound[1] = start;
        } else {
            bound[0] = bound[1] = -1;
            return bound;
        }
        return bound;
    }
}
```

Search Insert Position
Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.
You may assume no duplicates in the array.
Here are few examples.
[1,3,5,6], 5 → 2
[1,3,5,6], 2 → 1
[1,3,5,6], 7 → 4
[1,3,5,6], 0 → 0
Answer:

```
/*
version 1: find the first position >= target
*/
public class Solution {
    public int searchInsert(int[] A, int target) {
        int start, mid, end;
        start = 0;
        end = A.length - 1;
        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (A[mid] == target) {
                return mid;
            } else if (A[mid] > target) {
                end = mid;
            } else if (A[mid] < target) {
                start = mid;
            }
        }
        if (A[start] >= target) {
            return start;
        } else if (A[end] >= target) {
            return end;
```

```
        } else {
            return end + 1;
        }
    }
}
/*
version 2: find the last position < target, return + 1
*/
public class Solution {
    public int searchInsert(int[] A, int target) {
        int start, mid, end;
        start = 0;
        end = A.length - 1;
        if (target < A[0]) {
            return 0;
        }

        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (A[mid] == target) {
                return mid;
            } else if (A[mid] > target) {
                end = mid;
            } else if (A[mid] < target) {
                start = mid;
            }
        }
        if (A[end] == target) {
            return end;
        } else if (A[end] < target) {
            return end + 1;
        } else if (A[start] == target){
            return start;
        } else {
            return start + 1;
        }
    }
}
```

Search in Rotated Sorted Array
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). 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.
Answer:
画图解题

```
public class Solution {
    public int search(int[] A, int target) {
        int start = 0;
        int end = A.length - 1;
        int mid;
```

```
        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (A[mid] == target) {
                return mid;
            }
            if (A[start] < A[mid]) {
                // situation 1, red line
                if (A[start] <= target && target <= A[mid]) {
                    end = mid;
                } else {
                    start = mid;
                }
            } else {
                // situation 2, green line
                if (A[mid] <= target && target <= A[end]) {
                    start = mid;
                } else {
                    end = mid;
                }
            }
        } // while

        if (A[start] == target) {
            return start;
        }
        if (A[end] == target) {
            return end;
        }
        return -1;
    }
}
```

Search in Rotated Sorted Array II
Follow up for "Search in Rotated Sorted Array": What if duplicates are allowed? Would this affect the run-time complexity? How and why? Write a function to determine if a given target is in the array.
Answer:

```
/*
考点：不能用二分法，基于比较的排序，最快nlogn
黑盒测试，访问A数组，内容未知，很可能前面很多重复的无关元素，而target在最后一个，没有查询到所有元素之前都无法确定
*/
public class Solution {
    public boolean search(int[] A, int target) {
        for (int i = 0; i < A.length; i ++) {
            if (A[i] == target) {
                return true;
            }
```

```
        }
        return false;
    }
}
```

Search a 2D Matrix
Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:
Integers in each row are sorted from left to right.
The first integer of each row is greater than the last integer of the previous row.
For example,
Consider the following matrix:

```
[
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
```

Given target = 3, return true.
Answer:

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if (matrix == null || matrix.length == 0) {
            return false;
        }
        if (matrix[0] == null || matrix[0].length == 0) {
            return false;
        }

        int rows = matrix.length;
        int cols = matrix[0].length;
        // find the row index, the last number <= target
        int start = 0;
        int end = rows - 1;
        while (start + 1 < end) {
            int mid = start + (end - start) / 2;
            if (matrix[mid][0] == target) {
                return true;
            } else if (matrix[mid][0] < target) {
                start = mid;
            } else {
                end = mid;
            }
        }
        //should notice the order of if and else if
        if (matrix[end][0] <= target) {
            rows = end;
        } else if (matrix[start][0] <= target) {
            rows = start;
        } else {
```

```
            return false;
        }

        // find the column index, the number equal to target
        start = 0;
        end = cols - 1;
        while (start + 1 < end) {
            int mid = start + (end - start) / 2;
            if (matrix[rows][mid] == target) {
                return true;
            } else if (matrix[rows][mid] < target) {
                start = mid;
            } else {
                end = mid;
            }
        }
        if (matrix[rows][start] == target) {
            return true;
        } else if (matrix[rows][end] == target) {
            return true;
        } else {
            return false;
        }
    }
}
```

First Bad Vision
The code base version is an integer and start from 1 to n. One day, someone commit a bad version in the code case, so it caused itself and the following versions are all failed in the unit tests.
You can determine whether a version is bad by the following interface:
Java:

```
public VersionControl {
    boolean isBadVersion(int version);
}
```

C++:

```
class VersionControl {
public:
    bool isBadVersion(int version);
};
```

Python:

```
class VersionControl:
    def isBadVersion(version)
```

Find the first bad version.
Note
You should call isBadVersion as few as possible.
Please read the annotation in code area to get the correct way to call isBadVersion in different language. For example, Java is VersionControl.isBadVersion.
Example

Given n=5
Call isBadVersion(3), get false
Call isBadVersion(5), get true
Call isBadVersion(4), get true
return 4 is the first bad version
Challenge
Do not call isBadVersion exceed O(logn) times.
Answer:

```
/**
* public class VersionControl {
* public static boolean isBadVersion(int k);
*}
*  you can use VersionControl.isBadVersion(k) to judge wether
*   the kth code version is bad or not.
*/
class Solution {
/**
* @param n: An integers.
*  @return: An integer which is the first bad version.
*  /
    public int findFirstBadVersion(int n) {
        int start = 1, end = n;
        while (start + 1 < end) {
            int mid = start + (end - start) / 2;
            if (VersionControl.isBadVersion(mid)) {
                end = mid;
            } else {
                start = mid + 1;
            }
        }
        if (VersionControl.isBadVersion(start)) {
            return start;
        }
        return end;
    }
}
```

Find Peak Element
A peak element is an element that is greater than its neighbors.
Given an input array where num[i] ≠ num[i+1], find a peak element and return its index.
The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.
You may imagine that num[-1] = num[n] = -∞.
For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2.
Note:
Your solution should be in logarithmic complexity.
Answer:

```
/*
```

mid是峰，mid是谷，mid是上升的中点，mid是下降的中点
基本上就是三个条件：A: 我自己是峰，B，左边有一个峰，C，右边有一个峰
只要发现mid比mid-1或者mid+1的任何一边的小，就往那边走就行
*/
/*
5316 start = 5 end = 6 mid = 3 -> start = 5 end = 3 5 > 3 thus return 5
1239 start = 1 end = 9 mid = 2 -> start = 2 end = 9 mid = 3 -> start = 3 end = 9 3 < 9 thus return 9
*/

```
public class Solution {
    public int findPeakElement(int[] num) {
        if (num == null) {
            return -1;
        }
        if (num.length == 1) {
            return 0;
        }

        int start, mid, end;
        start = 0;
        end = num.length - 1;
        while (start + 1 < end) {
            mid  = start + (end - start) / 2;
            if (num[mid] > num[mid - 1] && num[mid] > num[mid + 1]) {
                return mid;
            } else if (num[mid] > num[mid + 1]) {
                end = mid;
            } else {
                start = mid;
            }
        }

        if (num[start] > num[start + 1]) {
            return start;
        }
        if (num[end] > num[end - 1]) {
            return end;
        }
        return -1;
    }
}
```

Sorted Array
Remove Duplicates from Sorted Array
Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.
Do not allocate extra space for another array, you must do this in place with constant memory.
For example,
Given input array A = [1,1,2],

Your function should return length = 2, and A is now [1,2].
Answer:

```
public class Solution {
    public int removeDuplicates(int[] A) {
        if (A == null || A.length == 0) {
            return 0;
        }

        int len = 1;
        for (int i = 1; i < A.length; i++) {
            if (A[i] != A[i - 1]) {
                A[len] = A[i];
                len++;
            }
        }
        return len;
    }
}
```

Remove Duplicates from Sorted Array II
Follow up for "Remove Duplicates":
What if duplicates are allowed at most twice?
For example,
Given sorted array A = [1,1,1,2,2,3],
Your function should return length = 5, and A is now [1,1,2,2,3].
Answer:

```
public class Solution {
    public int removeDuplicates(int[] A) {
        if (A == null) {
            return 0;
        }
        if (A.length <= 1) {
            return A.length;
        }

        // 拷贝2次后就不再拷贝
        boolean canCopy = true;
        int len = 1;
        for (int i = 1; i < A.length; i++) {
            if (A[i] == A[i - 1]) {
                if (!canCopy) {
                    continue;
                }
                canCopy = false;
            } else {
                canCopy = true;
            }
            A[len] = A[i];
            len++;
```

```
        }
        return len;
    }

    /*
    public static void main(String[] strs) {
        int[] A = {1, 1, 1, 2, 2, 3};
        removeDuplicates(A);

        for (int i : A) {
            System.out.print(i + " ");
        }
    }
    */
}
```

Merge Sorted Array
Given two sorted integer arrays A and B, merge B into A as one sorted array.
Note:
You may assume that A has enough space (size that is greater or equal to m + n) to hold additional elements from B. The number of elements initialized in A and B are m and n respectively.
Answer:

```
public class Solution {
    public void merge(int A[], int m, int B[], int n) {
        int cur = m + n - 1;
        int pA = m - 1; // 指向A的尾部
        int pB = n - 1; // 指向B的尾部

        while (cur >= 0) {
            if (pA < 0 || pB < 0) {
                break;
            }
            // 从尾部往前比较
            if (A[pA] > B[pB]) {
                A[cur--] = A[pA--];
            } else {
                A[cur--] = B[pB--];
            }
        }

        // copy the left over elements in B to A.
        while (pB >= 0) {
            A[cur--] = B[pB--];
        }
    }
}
```

Median of Two Sorted Arrays

There are two sorted arrays A and B of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).
Answer:

```
/*
1. 我们借用findKthNumber的思想。先实现findKthNumber，如果是偶数个，则把中间2个加起来平均，奇数就用中间的
2. 为了达到LOG级的复杂度
每次在A，B取前k/2个元素
1). A的元素不够k/2. 则我们可以丢弃B前k/2.
举个栗子：
A: 6 7 8
B: 1 2 3 4 5
k = 8
2). A[mid] < B[mid] (mid是k/2 -1索引处的元素)，这种情况下，我们可以丢弃A前k/2。
举个栗子：
A: 1 2
B: 4 5 6 7 8
k = 4，我们就可以首先排除1，2.
*/
/*
drop the left side of B.
drop the left side of A.
drop the left side of B.
drop the left side of A.
当2者相等，有2种情况：
1. 当k为偶数，则kth存在于任何一个结尾处，其实也是可以丢弃一半的。
2. 当k为奇数，则kth不存在于A,B的left side。也是可以丢弃任意一半。
A: 1 3 4 6 8
B: 1 2 3 6 7
k = 8 mid = 8 / 2 - 1 = 3
k = 9 mid = 9 / 2 - 1 = 3
*/
public class Solution {
    public double findMedianSortedArrays(int A[], int B[]) {
        if (A == null || B == null) {
            return 0;
        }

        int len = A.length + B.length;
        if (len % 2 == 0) {
            return (double)(dfs(A, B, 0, 0, len / 2) + dfs(A, B, 0, 0, len / 2 + 1)) / 2.0;
        } else {
            return dfs(A, B, 0, 0, len / 2 + 1);
        }
    }

    public double dfs(int A[], int B[], int aStart, int bStart, int k) {
        if (aStart >= A.length) {
            return B[bStart + k - 1];
        } else if (bStart >= B.length) {
```

```
            return A[aStart + k - 1];
        }

        if (k == 1) {
            return Math.min(A[aStart], B[bStart]);
        }

        int mid = k / 2 - 1;

        if (aStart + mid >= A.length) {
            return dfs(A, B, aStart, bStart + k / 2, k - k / 2);
        } else if (bStart + mid >= B.length) {
            return dfs(A, B, aStart + k / 2, bStart, k - k / 2);
        } else if (A[aStart + mid] > B[bStart + mid]) {
            return dfs(A, B, aStart, bStart + k / 2, k - k / 2);
        } else {
            return dfs(A, B, aStart + k / 2, bStart, k - k / 2);
        }
    }
}
```

Recover Rotated Sorted Array
Given a rotated sorted array, recover it to sorted array in-place. Example [4, 5, 1, 2, 3] -> [1, 2, 3, 4, 5]
Answer:

```
/*
Class ArrayList
.set(index, object): Replaces the element at the specified position in this list with the specified element.
*/
public class Solution {
/**
* @param nums: The rotated sorted array
* @return: The recovered sorted array
*/
    public void recoverRotatedSortedArray(ArrayList<Integer> nums) {
        for (int i = 0; i < nums.size() - 1; i++) {
            if (nums.get(i) > nums.get(i + 1)) {
                reverseArray(nums, 0, i);
                reverseArray(nums, i + 1, nums.size() - 1);
                reverseArray(nums, 0, nums.size() - 1);
                return;
            }
        }
    }

    public void reverseArray(ArrayList<Integer> nums, int start, int end) {
        for (int i = start, j = end; i < j; i++, j--) {
            int tmp = nums.get(i);
```

```
            nums.set(i, nums.get(j));
            nums.set(j, tmp);
        }
    }
}
```

Reverse Words in a String
Given an input string, reverse the string word by word. For example, Given s = "the sky is blue", return "blue is sky the".
Answer:

```
/*
Class String
.trim(): Returns a string whose value is this string, with any leading and trailing whitespace removed.
.split(String regex): Splits this string around matches of the given regular expression. Return String[]
Regular Expression
\\s+: 第一个"\"是用来转义的, "\s+"正则表达式表示至少出现一个空格
*/
public class Solution {
    public String reverseWords(String s) {
        if (s == null || s.length() == 0) {
            return "";
        }

        StringBuilder sb = new StringBuilder();
        String sTrim = s.trim();
        String[] str = sTrim.split("\\s+");
        for (int i = str.length - 1; i >= 0; i--) {
            sb.append(str[i]);
            if (i != 0) {
                sb.append(" ");
            }
        }
        return sb.toString();
    }

    /*
    public static void main {
        reverseWords("I love umass");
    }
    */
}
```

Find Minimum in Rotated Sorted Array
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).
Find the minimum element.
You may assume no duplicate exists in the array.

Answer:
画图

```
public class Solution {
    public int findMin(int[] num) {
        if (num == null || num.length == 0) {
            return -1;
        }

        int start = 0;
        int end = num.length - 1;
        int mid;
        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (num[mid] >= num[end]) {
                start = mid;
            } else {
                end = mid;
            }
        }

        if (num[start] < num[end]) {
            return num[start];
        } else {
            return num[end];
        }
    }
}
```

Find Minimum in Rotated Sorted Array II
Follow up for "Find Minimum in Rotated Sorted Array":
What if duplicates are allowed?
Would this affect the run-time complexity? How and why?
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).
Find the minimum element.
The array may contain duplicates.
Answer:

```
public class Solution {
    public int findMin(int[] num) {
        int tmp = Integer.MAX_VALUE;
        for (int i = 0; i < num.length; i++) {
            if (num[i] < tmp) {
                tmp = num[i];
            }
        }
        return tmp;
    }
}
```