

GeekBand 极客班

互联网人才加油站!

二分搜索和排序数组

www.geekband.com

GeekBand 极客班 互联网人才+加油站！

极客班携手网易云课堂，针对热门IT互联网岗位，联合业内专家大牛，紧贴企业实际需求，量身打造精品实战课程。

专业课程

+

项目碾压

- 顶尖专家技能私授
- 贴合企业实际需求
- 互动交流直播答疑
- 学员混搭线上组队
- 一线项目实战操练
- 业内大牛辅导点评



C++系统工程师



iOS开发工程师



Android开发工程师



PM产品经理

www.geekband.com

二分搜索和排序数组

GeekBand

极客班

Binary Search

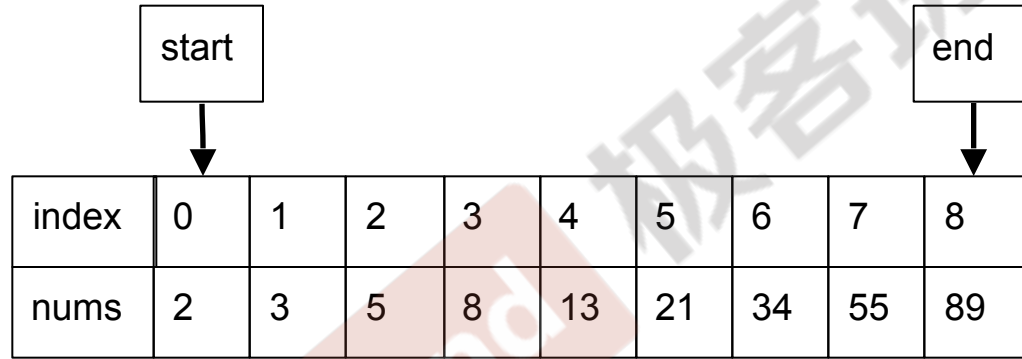
GeekBand 极客班

Classical Binary Search

Given an sorted integer array - nums, and an integer - target. Find the first position of target in nums, return -1 if target doesn't exist.

```
public int binarySearch(int[] nums, int target)
```

How we do binary search?

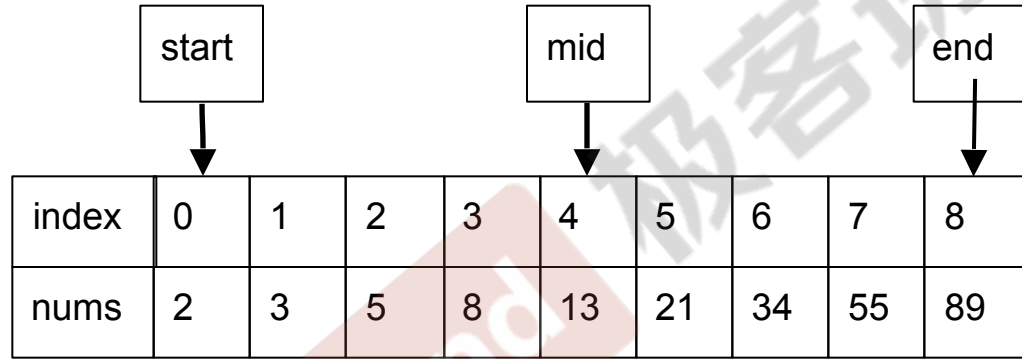


The diagram illustrates the initial state of a binary search. A 'start' box points to index 0 and an 'end' box points to index 8 of a sorted array. The array contains the values [2, 3, 5, 8, 13, 21, 34, 55, 89].

start										
	↓									↓
index	0	1	2	3	4	5	6	7	8	
nums	2	3	5	8	13	21	34	55	89	

1. Find 5

How we do binary search?

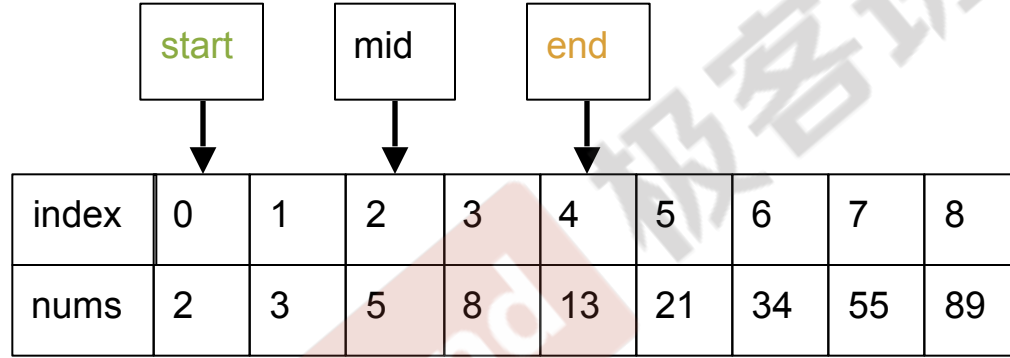


The diagram illustrates the initial state of a binary search on a sorted array. Three boxes labeled 'start', 'mid', and 'end' are positioned above the array. Arrows point from 'start' to index 0, from 'mid' to index 4, and from 'end' to index 8. The array itself is a table with two rows: 'index' and 'nums'. The 'index' row contains values from 0 to 8, and the 'nums' row contains the corresponding sorted values: 2, 3, 5, 8, 13, 21, 34, 55, 89. The cell at index 2 containing the value 5 is highlighted in red.

	start				mid				end
	↓				↓				↓
index	0	1	2	3	4	5	6	7	8
nums	2	3	5	8	13	21	34	55	89

1. Find 5, mid=4

How we do binary search?

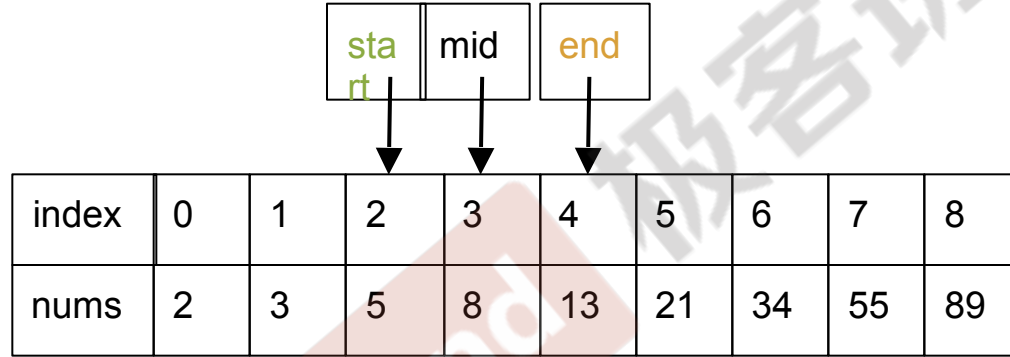


The diagram illustrates the initial state of a binary search on a sorted array. Three boxes labeled 'start', 'mid', and 'end' are positioned above the array. Arrows point from 'start' to index 0, from 'mid' to index 2, and from 'end' to index 4. The array itself is a table with two rows: 'index' and 'nums'. The 'index' row contains values from 0 to 8, and the 'nums' row contains the values 2, 3, 5, 8, 13, 21, 34, 55, and 89. The cell containing '5' at index 2 is highlighted in red.

	start		mid		end				
	↓		↓		↓				
index	0	1	2	3	4	5	6	7	8
nums	2	3	5	8	13	21	34	55	89

1. Find 5, mid=4, 2. Find it!

How we do binary search?



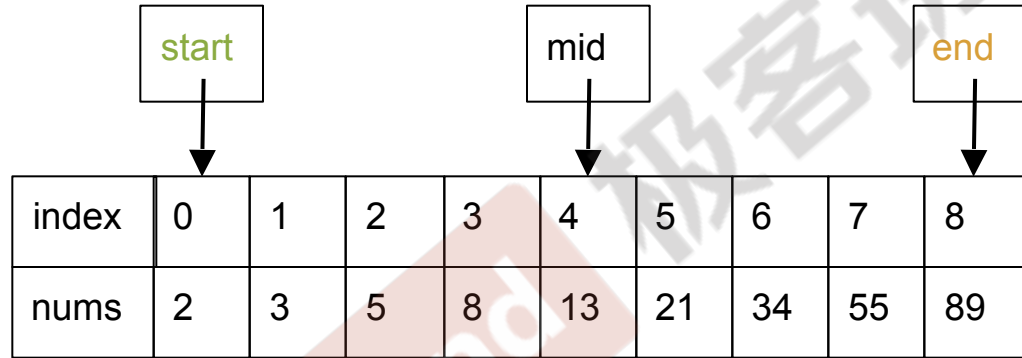
The diagram illustrates the initial state of a binary search on a sorted array. Above the array, three boxes represent the search range: 'start' (green text, split as 'sta' and 'rt'), 'mid' (black text), and 'end' (orange text). Arrows point from these boxes to the array. The 'start' arrow points to index 2, the 'mid' arrow points to index 3, and the 'end' arrow points to index 4. The array itself is a table with two rows: 'index' and 'nums'. The values in the 'nums' row are 2, 3, 5, 8, 13, 21, 34, 55, and 89. The columns for index 2, 3, and 4 are highlighted with a light red background.

index	0	1	2	3	4	5	6	7	8
nums	2	3	5	8	13	21	34	55	89

1. Find 5, mid=4, 2. Find it!

2. Find 8, mid=4, 2, 3. Find it!

How we do binary search?

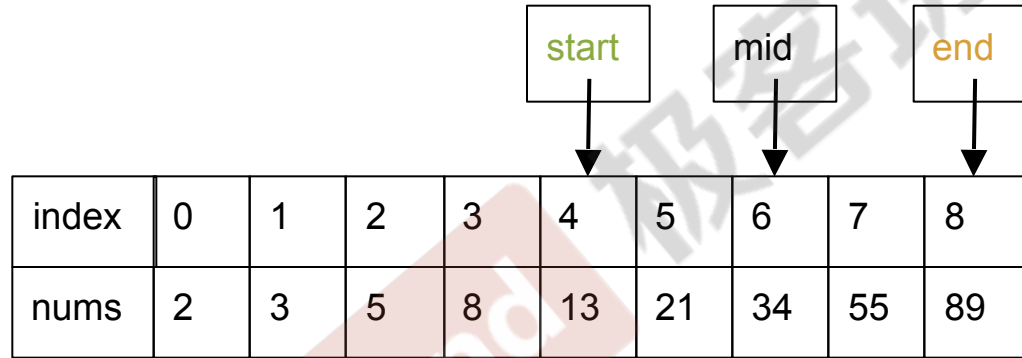


The diagram illustrates the initial state of a binary search on a sorted array. Three boxes labeled 'start', 'mid', and 'end' are positioned above the array. Arrows point from 'start' to index 0, from 'mid' to index 4, and from 'end' to index 8. The array itself is a table with two rows: 'index' and 'nums'.

index	0	1	2	3	4	5	6	7	8
nums	2	3	5	8	13	21	34	55	89

1. Find 5, mid=4, 2. Find it!
2. Find 8, mid=4, 2, 3. Find it!
3. Find 14, mid=4

How we do binary search?



The diagram illustrates the initial state of a binary search on a sorted array. Three boxes labeled 'start', 'mid', and 'end' are positioned above the array. Arrows point from 'start' to index 4, from 'mid' to index 6, and from 'end' to index 8. The array itself is a table with two rows: 'index' and 'nums'.

index	0	1	2	3	4	5	6	7	8
nums	2	3	5	8	13	21	34	55	89

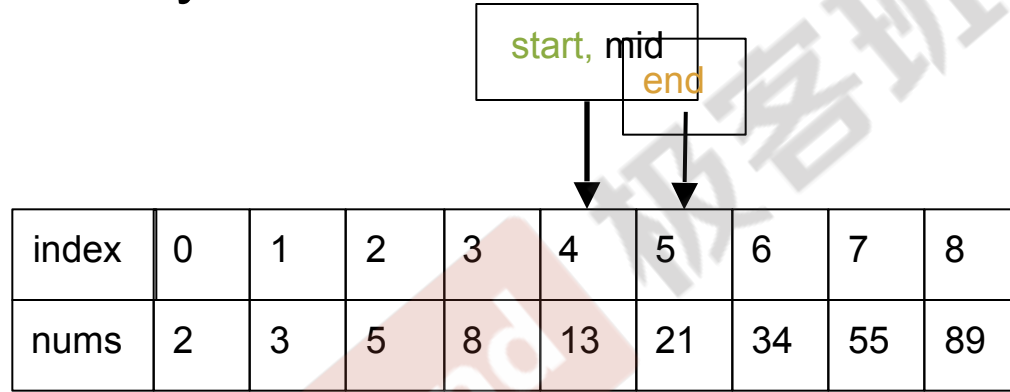
1. Find 5, mid=4, 2. Find it!
2. Find 8, mid=4, 2, 3. Find it!
3. Find 14, mid=4, 6

How we do binary search?

					<table><tr><td>start</td><td></td><td>mid</td><td></td><td>end</td></tr><tr><td>↓</td><td></td><td>↓</td><td></td><td>↓</td></tr></table>			start		mid		end	↓		↓		↓					
start		mid		end																		
↓		↓		↓																		
index	0	1	2	3	4	5	6	7	8													
nums	2	3	5	8	13	21	34	55	89													

1. Find 5, mid=4, 2. Find it!
2. Find 8, mid=4, 2, 3. Find it!
3. Find 14, mid=4, 6, 5

How we do binary search?



The diagram illustrates the initial state of a binary search. A box at the top contains the labels 'start, mid' in green and 'end' in orange. Two arrows point from this box to the array below: one from 'start, mid' pointing to index 4, and one from 'end' pointing to index 5. Below the arrows is a table representing a sorted array of 10 elements.

index	0	1	2	3	4	5	6	7	8
nums	2	3	5	8	13	21	34	55	89

1. Find 5, mid=4, 2. Find it!
2. Find 8, mid=4, 2, 3. Find it!
3. Find 14, mid=4, 6, 5, 4. Return -1

Recursion or While-Loop?

对于有序线性容器的搜索，二分查找或其变种基本上是解题的最佳方法

binary search template in recursion

```
int binarySearch(int *array, int left, int right, int value) {  
    if (left > right) {  
        // value not found  
        return -1;  
    }  
  
    int mid = left + (right - left) / 2;  
    if (array[mid] == value) {  
        return mid;  
    } else if (array[mid] < value) {  
        return binarySearch(array, mid + 1, right, value);  
    } else {  
        return binarySearch(array, left, mid - 1, value);  
    }  
}
```

Keypoints:

1. $start + 1 < end$
2. $left + (right - left) / 2$
3. $A[mid] ==, <, >$
4. $A[start/end] == target$

A generic binary search template

```
int binary_search(const int a[], const int size, const int val) {  
    int lower = 0;  
    int upper = size-1;  
  
    /* invariant: if a[i]==val for any i, then lower <= i <= upper */  
    while (lower <= upper) {  
        int i = lower + (upper-lower)>>1;  
        if (val == a[i]) {  
            return i;  
        } else if (val < a[i]) {  
            upper = i-1;  
        } else { /* val > a[i] */  
            lower = i+1;  
        }  
    }  
    return -1;  
}
```


Find i in a given array that $arr[i] == i$.

在此例中， $A[3] = 3$ 。同时，不难发现一个规律： $A[3]$ 左侧的数据满足 $value < index$ ， $A[3]$ 右侧的数据满足 $value > index$ 。

Index	0	1	2	3	4	5	6	7	8
Value	- 7	-2	0	3	7	9	10	12	13

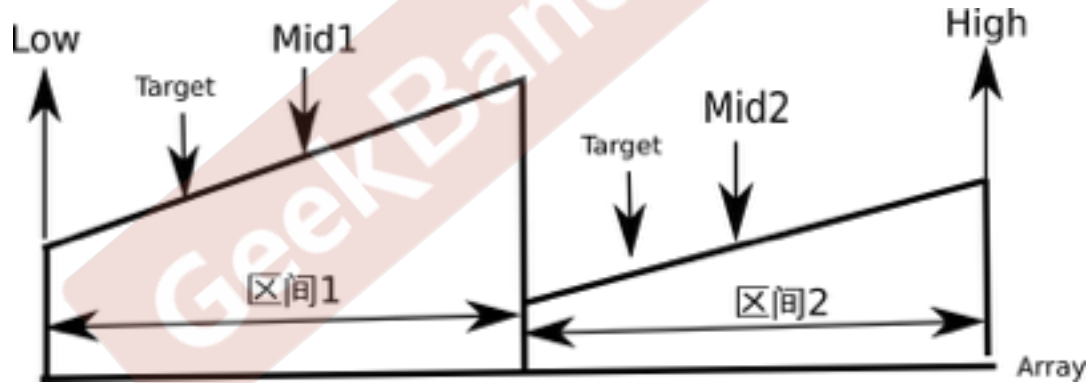
Search Insert Position

<http://oj.leetcode.com/problems/search-insert-position/>

GeekBand

Search in Rotated Sorted Array

An array is sorted without duplicates. However, someone mysteriously shifted all the elements in this array (e.g. 1,2,3,4,5 -> 5,1,2,3,4). Implement a function to find an element in such array (return -1 if no such element).



Find the Square Root

```
def sqrt(n):  
    start = 0  
    end = n  
    m = 0  
    min_range = 0.0000000001;  
  
    while end - start >  
min_range:  
        m = (start + end) /  
2.0;  
  
        pow2 = m * m  
        if abs(pow2 - n) <= min_range:  
            return m  
        elif pow2 < n:  
            start = m  
        else:  
            end = m  
  
    return m
```

矩阵搜索

Check if an element is in a $M \times N$ matrix, each row and column of which is sorted.

我们可以构造一个矩阵：

1	5	10	20
2	6	11	30
7	9	12	40
8	15	31	41

如果要在上述矩阵中找到9，应该如何计算？最简单的方法显然是遍历每行每列，这样的时间复杂度是 $O(n^2)$ ，而且完全没有利用到矩阵已经部分有序的特性。

Range Search

Given a sorted array of integers with duplicates. Implement a function to get the start and end position of a given value.

GeekBand

Search in a 2D Matrix

28	29	31	36	36	43	49	54	58	64	66	68	68	77	84	89	97	97	97	105
33	38	39	48	53	58	61	62	62	69	70	71	71	77	90	91	103	105	108	110
33	44	51	56	62	63	67	69	74	83	86	91	96	98	104	110	110	110	118	125
41	44	52	59	71	75	83	92	100	107	114	122	122	124	132	133	140	141	145	146
43	49	54	59	72	79	83	101	104	115	122	124	124	124	140	140	140	142	151	154
45	52	55	62	72	80	92	104	111	122	130	132	132	136	149	156	160	165	171	179
51	60	68	73	81	86	94	110	115	129	138	141	146	149	157	163	169	178	186	191
51	61	74	83	92	95	95	111	120	129	138	143	146	155	159	171	180	188	196	201
56	67	81	89	94	98	98	119	123	133	147	149	149	164	167	174	183	188	205	211
62	69	85	95	97	101	105	119	131	139	152	152	160	165	171	178	190	195	212	215
71	72	90	95	106	111	116	121	131	144	155	160	163	173	180	186	197	199	220	221
79	87	99	104	108	115	123	130	140	144	161	166	174	177	189	192	205	209	225	230
84	94	103	106	117	120	129	133	145	153	165	173	175	178	196	199	209	209	231	235
91	96	105	108	120	128	137	141	151	154	169	182	187	192	201	209	214	220	233	237
95	100	109	110	123	137	139	141	159	161	174	184	188	201	205	213	218	228	233	241
101	107	115	119	130	146	155	155	168	173	178	187	190	202	209	221	226	230	237	249
109	109	115	125	131	148	156	164	173	180	180	196	204	212	217	222	232	240	249	253
113	117	124	126	138	151	157	167	181	183	184	204	213	219	223	231	236	242	250	253
115	123	131	138	142	152	160	167	184	190	197	210	215	224	225	238	244	246	256	258
118	126	131	138	149	161	170	176	184	193	206	217	217	225	234	240	249	257	265	267

Search in a 2D Matrix

28	29	31	36	36	43	49	54	58	64	66	68	68	77	84	89	97	97	97	105
33	38	39	48	53	58	61	62	62	69	70	71	71	77	90	91	103	105	108	110
33	44	51	56	62	63	67	69	74	83	86	91	96	98	104	110	110	110	118	125
41	44	52	59	71	75	83	92	100	107	114	122	122	124	132	133	140	141	145	146
43	49	54	59	72	79	83	101	104	115	122	124	124	124	140	140	140	142	151	154
45	52	55	62	72	80	92	104	111	122	130	132	132	136	149	156	160	165	171	179
51	60	68	73	81	86	94	110	115	129	138	141	146	149	157	163	169	178	186	191
51	61	74	83	92	95	95	111	120	129	138	143	146	155	159	171	180	188	196	201
56	67	81	89	94	98	98	119	123	133	147	149	149	164	167	174	183	188	205	211
62	69	85	95	97	101	105	119	131	139	152	152	160	165	171	178	190	195	212	215
71	72	90	95	106	111	116	121	131	144	155	160	163	173	180	186	197	199	220	221
79	87	99	104	108	115	123	130	140	144	161	166	174	177	189	192	205	209	225	230
84	94	103	106	117	120	129	133	145	153	165	173	175	178	196	199	209	209	231	235
91	96	105	108	120	128	137	141	151	154	169	182	187	192	201	209	214	220	233	237
95	100	109	110	123	137	139	141	159	161	174	184	188	201	205	213	218	228	233	241
101	107	115	119	130	146	155	155	168	173	178	187	190	202	209	221	226	230	237	249
109	109	115	125	131	148	156	164	173	180	180	196	204	212	217	222	232	240	249	253
113	117	124	126	138	151	157	167	181	183	184	204	213	219	223	231	236	242	250	253
115	123	131	138	142	152	160	167	184	190	197	210	215	224	225	238	244	246	256	258
118	126	131	138	149	161	170	176	184	193	206	217	217	225	234	240	249	257	265	267

Search in a 2D Matrix

28	29	31	36	36	43	49	54	58	64	66	68	68	77	84	89	97	97	97	105
33	38	39	48	53	58	61	62	62	69	70	71	71	77	90	91	103	105	108	110
33	44	51	56	62	63	67	69	74	83	86	91	96	98	104	110	110	110	118	125
41	44	52	59	71	75	83	92	100	107	114	122	122	124	132	133	140	141	145	146
43	49	54	59	72	79	83	101	104	115	122	124	124	124	140	140	140	142	151	154
45	52	55	62	72	80	92	104	111	122	130	132	132	136	149	156	160	165	171	179
51	60	68	73	81	86	94	110	115	129	138	141	146	149	157	163	169	178	186	191
51	61	74	83	92	95	95	111	120	129	138	143	146	155	159	171	180	188	196	201
56	67	81	89	94	98	98	119	123	133	147	149	149	164	167	174	183	188	205	211
62	69	85	95	97	101	105	119	131	139	152	152	160	165	171	178	190	195	212	215
71	72	90	95	106	111	116	121	131	144	155	160	163	173	180	186	197	199	220	221
79	87	99	104	108	115	123	130	140	144	161	166	174	177	189	192	205	209	225	230
84	94	103	106	117	120	129	133	145	153	165	173	175	178	196	199	209	209	231	235
91	96	105	108	120	128	137	141	151	154	169	182	187	192	201	209	214	220	233	237
95	100	109	110	123	137	139	141	159	161	174	184	188	201	205	213	218	228	233	241
101	107	115	119	130	146	155	155	168	173	178	187	190	202	209	221	226	230	237	249
109	109	115	125	131	148	156	164	173	180	180	196	204	212	217	222	232	240	249	253
113	117	124	126	138	151	157	167	181	183	184	204	213	219	223	231	236	242	250	253
115	123	131	138	142	152	160	167	184	190	197	210	215	224	225	238	244	246	256	258
118	126	131	138	149	161	170	176	184	193	206	217	217	225	234	240	249	257	265	267

Search a 2D Matrix II

Write an efficient algorithm that searches for a value in an $m \times n$ 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.

Find a peak

There is an array which we can assume the numbers in adjacent positions are different. and $A[0] < A[1]$ && $A[A.length - 2] > A[A.length - 1]$. We consider a position P is a peak if $A[P] > A[P-1]$ && $A[P] > A[P+1]$. Find a peak in this array.

GeekBand

Sorted Array

GeekBand 极客班

Intersection of 2 sorted array

array1: [2 3 4 6]

array2: [3 6 9 10]

return [3,6]

GeekBand

极客班

Remove Duplicates from Sorted Array I

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 `nums = [1,1,2]`,

Your function should return `length = 2`, with the first two elements of `nums` being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.

Remove Duplicates from Sorted Array II

Follow up for "Remove Duplicates":

What if duplicates are allowed at most twice?

For example,

Given sorted array `nums = [1,1,1,2,2,3]`,

Your function should return `length = 5`, with the first five elements of `nums` being 1, 1, 2, 2 and 3. It doesn't matter what you leave beyond the new length.

Merge Sorted Array

1. Merge Two Sorted Array into a new Array
2. Merge Two Sorted Array A and B into A, assume A has enough space.

GeekBand

Merge 2 Sorted Array

GeekBand

极客班

Merge K Sorted List

Merge k sorted linked lists to be one sorted list.

GeekBand

极客班

Related Questions

1. Rotate String: abcdefg, offset=3 -> efgabcd
2. Rotate Words List: I love you -> you love I

GeekBand

极客班

Conclusion

Binary Search

- Exclude half every time

Sorted Array

- If array is sorted, try binary search
- If array is not sorted, try sort it first

Homework: Find the First Bad Version

The code base version is an integer and start from 0 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 unittests. You can determine whether a version is bad by the following interface:

```
boolean isBadVersion(int version);
```