Day1

December 16, 2023 5:43 PM

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.
You must write an algorithm with O(log n) runtime complexity.

Example 1:
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
Example 2:
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1

From <https://leetcode.com/problems/binary-search/>

Basic thought:
Divide the array into two parts, and to see if it's in the first range of array, if not, switch to the second array.
Repeat the process.

Version 1 : [left, right]
Left = 0
Right = numSize -1
While(left <= right):
    Middle = (left + right) /2
    If(nums[middle] > target):
        Right = middle -1
    Else if (nums[middle] < target):
        Left = middle + 1
    Else return middle
Return -1

Version 2: [left, right)
Left = 0
Right = numSize -1
While(left < right):
    Middle = (left + right) /2
    If(nums[middle] > target):
        Right = middle
    Else if (nums[middle] < target):
        Left = middle + 1
    Else return middle
Return -1

Ordered Tree
A tree is ordered if there is a meaningful linear order among the children (Left to Right)

Tree ADT
Accessor methods:

Position root()	Position parent(p)	Iterable children(p)	Integer numChildren(p)

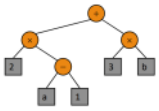
Query Methods:

Boolean	isInternal(p)	Boolean isExternal(p)	Boolean isRoot(p)

Binary Tree
A binary tree is a tree with the following properties:
Each internal node has at most two children
Proper binary trees (or full): each node has zero or two children
The children of a node are an ordered pair

Alternative recursive definition : A binary tree is either:
A tree consisting of a single node, or
A tree whose root has an ordered pair of children, each of which is a binary tree.

Binary Tree associated with an arithmetic expression
Internal nodes: operators
External nodes: operands
Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



Binary Tree associated with a decision process
Internal nodes: questions with yes/no answer
External nodes: decisions
Example: dining decision

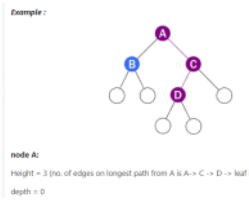
In a nonempty proper binary tree T, with n_E external nodes and n_I internal nodes, we have $n_E = n_I + 1$.

Binary Tree ADT
Methods:

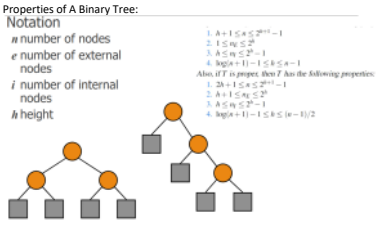
Position left(p)	Position right(p)	Position sibling(p)
addLeft(p,e)	addRight(p,e)	set(p,e)
Attach(p,T1,T2)		

remove(p): Removes the node at position p, replacing it with its child (if any), and returns the element that had been stored at p; an error occurs if p has two children.

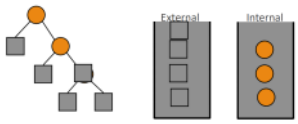
Height and Depth



Properties of A Binary Tree:



In a nonempty proper binary tree T, with n_E external nodes and n_I internal nodes, we have $n_E = n_I + 1$.



Given an integer array nums and an integer val, remove all occurrences of val in nums **in-place**. The order of the elements may be changed. Then return *the number of elements in* nums *which are not equal to* val.
Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.
- Return k.

Custom Judge:
The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with correct length.
                            // It is sorted with no values equaling val.
int k = removeElement(nums, val); // Calls your implementation
assert k == expectedNums.length;
sort(nums, 0, k); // Sort the first k elements of nums
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Example 1:
Input: nums = [3,2,2,3], val = 3
Output: 2, nums = [2,2,_,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 2.
It does not matter what you leave beyond the returned k (hence they are underscores).
Example 2:
Input: nums = [0,1,2,2,3,0,4,2], val = 2
Output: 5, nums = [0,1,4,0,3,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4.
Note that the five elements can be returned in any order.
It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

- 0 <= nums.length <= 100
- 0 <= nums[i] <= 50
- 0 <= val <= 100

From <https://leetcode.com/problems/remove-element/>

Thought:
To delete an element, we have to put the elements behind it forward one memory location.
We cannot delete it directly.

Use two pointers, one fast and one slow.
Fast pointer is used to find the val (the target element) in the array.
Slow pointer is the index of the array after deleting the elements.

```
Slow = 0
For ( fast = 0; fast < nums.size ; fast ++){
    If(nums[fast] != val){
        Nums[slow] = nums[fast]
        Slow ++
    }
}
Return slow
```

Given an integer array nums sorted in **non-decreasing** order, return *an array of **the squares of each number** sorted in non-decreasing order.*

Example 1:
Input: nums = [-4, -1, 0, 3, 10]
Output: [0, 1, 9, 16, 100]
Explanation: After squaring, the array becomes [16, 1, 0, 9, 100].
After sorting, it becomes [0, 1, 9, 16, 100].
Example 2:
Input: nums = [-7, -3, 2, 3, 11]
Output: [4, 9, 9, 49, 121]

Constraints:

- 1 <= nums.length <= 10^4
- -10^4 <= nums[i] <= 10^4
- nums is sorted in **non-decreasing** order.

From <https://leetcode.com/problems/squares-of-a-sorted-array/>

Simple solution: Make square of every number and sort the array

```
For(int i = 0; i < nums.size ; i++):
    Nums[i] = nums[i] * nums[i]
Sort(nums)
Return nums
```

Better solution:
Two pointers method. Two pointers start from both side and compare the value of them

```
Int squares[]
K = nums.size -1
For ( I = 0, j = nums.size - 1; i<= j; i++,j --):
    If(nums[i]^2 < nums[j]^2):
        Squares[k--] = nums[j]^2
    Else:
        Squares[k--] = nums[i]^2x
Return squares
```