**Binary Search**

Table of Contents

[LEVEL 1: **Amateur** 1](#_Toc199917695)

[1. Unique length 3 palindromic subsequences 2](#_Toc199917696)

[LEVEL 2: **Pro** 3](#_Toc199917697)

[LEVEL 3: **Legend** 4](#_Toc199917698)

[**SOLUTIONS:** 5](#_Toc199917699)

[**LEVEL 1:** 5](#_Toc199917700)

[**LEVEL 2:** 6](#_Toc199917701)

# LEVEL 1: **Amateur**

### Binary Search

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

### Lower Bound, upper bound

Link: <https://www.geeksforgeeks.org/problems/implement-lower-bound/1>

Link: <https://www.geeksforgeeks.org/problems/implement-upper-bound/1>

### First and last occurence

Link: <https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/>

# LEVEL 2: **Pro**

# LEVEL 3: **Legend**

# **SOLUTIONS:**

## **LEVEL 1:**

**1. Binary Search**

Iterative approach

class Solution:

    def search(self, nums: List[int], target: int) -> int:

        low, high = 0, len(nums)-1

        while low<=high:   #break condition

            mid = (low+high)//2        #find mid every time

            if nums[mid]==target:

                return mid

            if target>nums[mid]:

                low = mid+1

            else:

                high=mid-1

        return -1

Recursive approach

class Solution:

    def search(self, nums: List[int], target: int) -> int:

        def helper(low,high): #break condition

            if low>high:

                return -1

            mid = (low+high)//2

            if nums[mid]==target:

                return mid

            if target>nums[mid]:

                return helper(mid+1,high)

            return helper(low,mid-1)

        return helper(0,len(nums)-1)

**2. Lower Bound**

**Lower Bound**

The lower bound refers to the index of the first element in a sorted list that is **greater than or equal** to a given target value.

If the target value is present in the list, the lower bound will be the index of its first occurrence. If the target value is not present, the lower bound will be the index where the target value should be inserted to maintain the sorted order.

# just iterate till you can, and low will reach to lower bound

class Solution:

    def lowerBound(self, arr, target):

        low, high = 0, len(arr)-1

        while low<=high:

            mid = (low+high)//2

            if target>arr[mid]:

                low = mid+1

            else:

                high = mid-1

        return low

#Can use python bisect library

import bisect

class Solution:

    def lowerBound(self, arr, target):

        #code here

        return bisect.bisect\_left(arr,target)

**Upper Bound**

The upper bound refers to the index of the first element in a sorted list that is **strictly greater** than a given target value.

If the target value is present in the list, the upper bound will be the index after the last occurrence of the target value. If the target value is not present, the upper bound will be the same as the lower bound, indicating the insertion point.

#using bisect library

import  bisect

class Solution:

    def upperBound(self, arr, target):

        return bisect.bisect\_right(arr,target)

class Solution:

    def upperBound(self, arr, target):

        n = len(arr)

        ans=n               #if it never goes inside else, len(arr) is ans

        low, high = 0 , n-1

        while low<=high:

            mid = (low+high)//2

            if target>=arr[mid]:  #greater or equal

                low = mid+1

            else:

                ans = mid

                high = mid-1

        return ans

**3. First and Last occurrence of element**

class Solution:  #using basic binary search concept, and tracking target index

    def leftOccurrence(self,nums,target):

        low, high = 0, len(nums)-1

        ans = -1

        while low<=high:

            #another way to do (low +high) may go out of range so use this

            mid = low + (high-low)//2

            #even if we get value, at left we can get lower index

            if target==nums[mid]:

                ans = mid

                high = mid-1

            elif target<nums[mid]:

                high = mid-1

            else:

                low = mid+1

        return ans

    def rightOccurrence(self,nums,target):

        low, high = 0, len(nums)-1

        ans = -1

        while low<=high:

            mid = low + (high-low)//2

            #even if we get value, at righ we can get higher index value

            if target==nums[mid]:

                ans = mid

                low = mid+1

            elif target<nums[mid]:

                high = mid-1

            else:

                low = mid+1

        return ans

    def searchRange(self, nums: List[int], target: int) -> List[int]:

        lower = self.leftOccurrence(nums,target)

        upper = self.rightOccurrence(nums,target)

        return [lower,upper]

Using upper bound and lower bound concept

import bisect

class Solution:

    def searchRange(self, nums: List[int], target: int) -> List[int]:

        #lower bound give first occurance

        low = bisect.bisect\_left(nums,target)

        #upper bound give num greater than target (so -1 for last occurance)

        high = bisect.bisect\_right(nums,target) - 1

        #if num is not present in array

        #can also have low > nums[len] if target>max(arr)

        if(low==len(nums) or nums[low]!=target):

            return [-1,-1]

        return [low,high]

## **LEVEL 2:**