**Find if an element exists**

Given a sorted array of length with no duplicate elements, check if exists in the array. If exists, find the position.

use an array? 🡪 what if elements really big big

use a map? 🡪 no we are learning binary search

fun game: guess the number

There is a secret number, and you have to guess it.

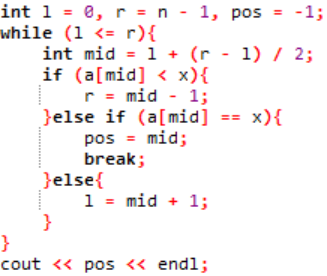
Each time you can query a number, and the other person will say the secret number is smaller or larger or yey if you guessed the correct number.

What strategy?

Naturally, you will pick the middle element (like if 1 to 100 then you would pick 50) because 50% chance smaller and larger, and then you can narrow the range down to 1 to 49 or 51 to 100.

How many times would you need to ask? Every time you narrow the range by half, so you would ask about times if the initial range is 1 to .

Ok back to the question. We can apply the same thing!

Initially set 2 variables l and r to 0 and n – 1 (0 indexed array)

every loop, make a new variable mid.

Check if mid is smaller, larger or equal to x.

if equal, yey! output the position.

else, narrow the range.

if l and r is same and still not equal to x, then we know x does not exist.

time complexity:

wow so fast

Important observation: if the original array was not sorted, and you sort it and binary search for x, the position of x (0-indexed) is the number of elements smaller than x! (assuming no duplicates)

**What if duplicates?**

lower bound: First element that is not < x

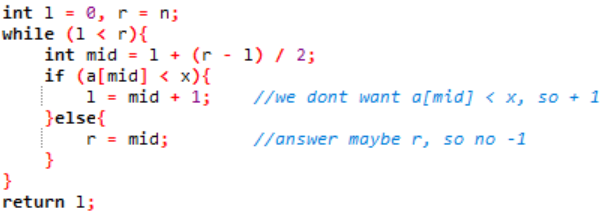
upper bound: First element that is > x

Example x = 3

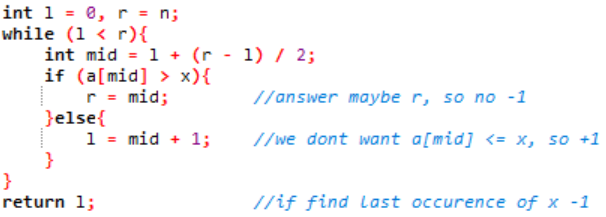
1 2 2 3 3 3 5 6 1 2 2 2 3 3 3 3 1 1 2 4 4 4 5 3 3 6 7

^ ^ ^ ^ ^ ^ ^

Find lower bound in non-descending array:



Find upper bound in non-descending array:



Why r = n and not n – 1? Well as you can see in the second example above, there may not exist first element > x, so we will just put a made-up element (infinity) at the last of the array.

What if non-ascending array?

Just flip everything!

mid = l + (r – l) / 2 🡪 mid = l + (r – l) / 2 + (r – l) % 2

(originally behave like floor, so after flip behave like ceiling)

l = mid + 1 🡪 r = mid – 1

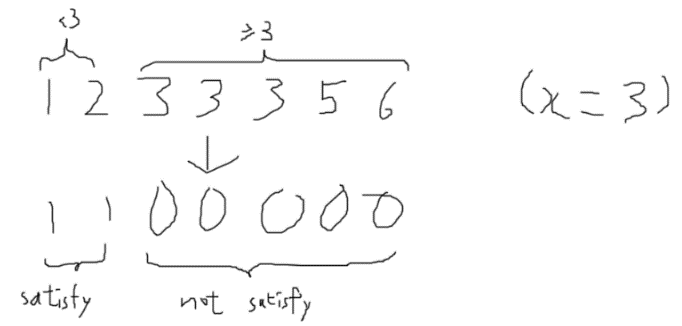
r = mid 🡪 l = mid

**What did we actually do?**

deeper analysis

to find the lower bound and upper bound, we have actually transformed the array into a Boolean array with continuous 0 and continuous 1.

What do I mean?

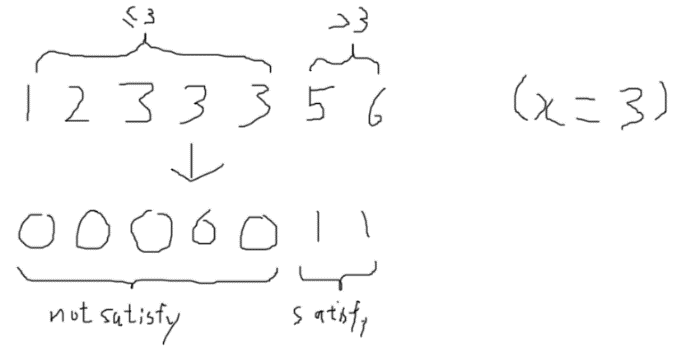


If you remember, lower bound is the first element that is not < x

Therefore, you have transformed the array like this:

1 means satisfy the condition that a[?] < 3, and 0 means otherwise.

You are actually finding the first 0!

Similarly, upper bound is the first element that is > x.

1 means satisfy the condition that a[?] > 3, and 0 means otherwise.

You are finding the first 1!

In a lot of problems, complicated stuff can be transformed to an array with a continuous block of 0 and continuous block of 1, and you just need to binary search to find the first or last 0 or 1.

**Binary search for the answer**

For example, the question may ask “what is the largest number such that some condition is satisfied” or “what is the longest time Bob can do something without blah blah?”

We just need to find the condition, and see if it is ‘sorted’, that it is a continuous block of 0 and a continuous block of 1. If it is, then we can apply binary search!

Some cool problems

Find the peak

Given an array, it is increasing in the first part, and decreasing in the second. Find the position of the ‘peak’.

1 2 5 4 2 🡪 3rd 1 2 3 4 🡪 4th 1 9 8 7 6 🡪 2nd

Can we binary search? What is the condition?

see if 2 adjacent elements are < or >!

the array must be in the form < < < < > > >

so you can do binary search!

Find the rotation

An array is initially sorted without duplicates. Then it is rotate x times. Find x.

4 1 2 3 🡪 1 time 4 5 6 1 2 3 🡪 3 times 2 5 8 1 🡪 3 times

What is the condition?

Check if a[mid] is > a[0] or < a[0]!

if a[mid] is > a[0], that means it is still in the first part where it is increasing from a[0].

If a[mid] is < a[0], that means the array has ‘looped back’ to the smallest element!

Final stuff

binary\_search, lower\_bound, and upper\_bound are actually functions in <algorithm>. But sometimes we still have to write it ourselves as we need to process mid and not directly take a[mid].

Why cut the array into 2? Why not 3? Ternary search actually exists, and it is usually used to find the maximum or the minimum of a function with like only 1 hump.

Binary search is very important as it is crucial to understand it before understand other advanced stuff, and it is frequently used in other algorithms.

Problems:

D801, M1023, NP1521, S144, and a lot more T162