ON A MISSION TO MAKE YOU LOVE DSA

CRASH CAMP

BINARY SEARCH

Episode 01

WITH THE GIVEN INSIGHTS YOU CAN EASILY SOLVE 80% TO 90% BINARY SEARCH PROBLEMS ASKED IN

AND MANY MORE...

Index

1. Linear Search (what's problem with it)
2. Use of 'mid' index
3. Diving Search Space (Welcoming Binary Search)
4. Binary Search (The Algorithm)
5. Different ways to calculate 'mid'
6. A trick that saves your time
7. Few more insights
8. Problem list (Type 1, Type 2)
9. Few Type 1 Problems.

Starting few slides are beginner oriented but will definitely give some good insights even if you already know 'Binary Search'

Let's begin the journey

Linear Search

Introduction

- You are a given a sorted array nums[] & an int k,

  nums[] = {1,2,4,6,8,10,15} k = 15
- return true if k exists else false.

Now, how would you approach above problem...

A "linear search" (a loop)

- iterating from i = 0 to i = n-1 (n = size)
- if(nums[i] == k) return true
- if(i == n) return false **K doesn't exist & you checked all values**

Using 'linear search', in worst case you would scan all 'n' (7) elements.

Can we do something better

let's jump to idx 2

```
 0  1 2  3 4  5   6
[1 2 4 6 8 10 15]
```

nums[2] < 15

these value idx[0,1] are also less than 15, so need to check here

now this is our potential search space

By jumping to idx = 2, we avoided 2 elements, so instead of all **7** elements we have only **5** , which is better than linear search

Sorted Search Space

- We saw if search space is sorted, jumping to some idx is better than linear search.
- What should be that idx value.. let's see.

say your search space had 100 sorted values, check if k = 100 exist

1 2 .. 10 .... 99 100

assume you jump to idx 10, which divides array into 2 parts.

left array (10%) [1..10]
right array (90%) [10..100]

although jumping to idx = 10 is better than 'linear search' but still in 'worst case' you have 90% space

can we reduce this search space in worst case even more ?

if we jump to 'mid' value it divides

left array (50%) [1..50]
right array (50%) [50..100]

now in worst case only 50% space is left

now we can conclude, if our search space is 'sorted' we will always jump to 'mid' index

getting something new, leave a like

Dividing Search Space

- You are given a sorted nums[] & an integer k
- return largest value less than k.

i/p

nums[] = {0,1,2,6,7,8,9} k = 6

o/p

2

```
0 1 2   3 4 5 6
[0 1 2  6 7 8 9]
```

this is our ans

If you are at any of these indices you will move to 'right'

values < 6

values >= 6

If you are at any of these indices you will move to 'left'

- Given condition, we can divide search space in 2 parts
- Depending on which part we are, we move 'left' or 'right'.

Let's name these 2 space as

1) Favourable space (F) where your ans may lie
2) Unfavourable space (U) where ans will never lie

We want larget value 'less' than k

so values < K are (F)
values >= K are (U)

ans

```
[0 1 2 6 7 8 9]
[F F F U U U U]
```

If you are at F move to 'right'
else move to 'left'

we conclude, if space is sorted, we can divide search space in F & U, depending on which space we are, we either move 'right' or 'left' .

Few Conclusions

- Till now we concluded, if given space is sorted
  1) jump to 'mid' idx (reducing no. of comparisons)
  2) divide search space in Favorable (F) & Unfavorable (U) space to choose which part of space you would move (right or left)

- You are given a sorted nums[] & an integer k
- return largest value less than k.

i/p

nums[] = {0,1,2,6,7,8,9} k = 6

o/p

2

l = 0 (lower limit)
h = 6 (higher limit)

our search space will lie b/w 'l' & 'h'

[0 1 2 6 7 8 9]
[F F F U U U U]

we calculate mid as

```
mid = (l+h) / 2
```

1

we divide search space with following

```
if(nums[mid] < k) {
    l = mid + 1;
} else {
    h = mid - 1;
}
```

2

we are at 'F', move 'right'
to move 'right' just push l to right of 'mid'

we are at 'U', move 'left'
to move 'left' just push h to left of 'mid'

'l' & 'h' are moving towards each-other, till what point they will chase (initially l < h)...

```
while(l <= h)
```

3

when l becomes > h, it indicates we exhausted our search space

note-> division of space in 'F' & 'U' will always depend on the problem

The Algorithm

- Using 3 points given in prev. slide let's construct an algo

```
int l = 0;
int h = n-1;
while(l <= h) {
  int mid = (l+h) / 2;
  if(nums[mid] < k) {
      l = mid + 1;
  } else {
      h = mid - 1;
  }
}
return h;
```

What is significance of 'return h' ?
will reveal the secret behind it... (my secret trick)

Now this algo is what we call 'Binary Search'

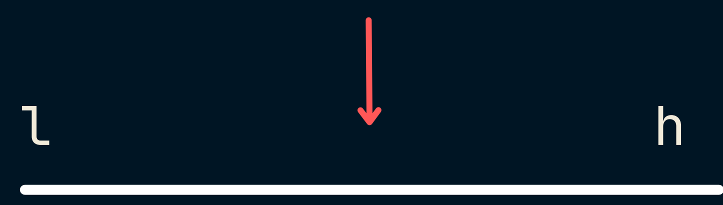
From now on, will you be able to write 'binary search' easily ?

let me know in comments...

Ways to calulate mid

- There are many ways-

1) mid = (l+h) / 2

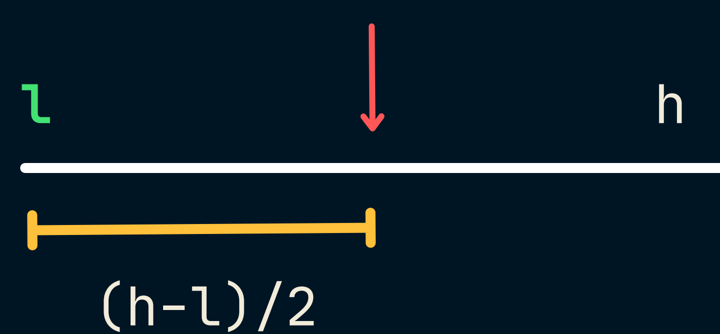


many lang. have varibale limits c/c++ has 2147483647 (int)

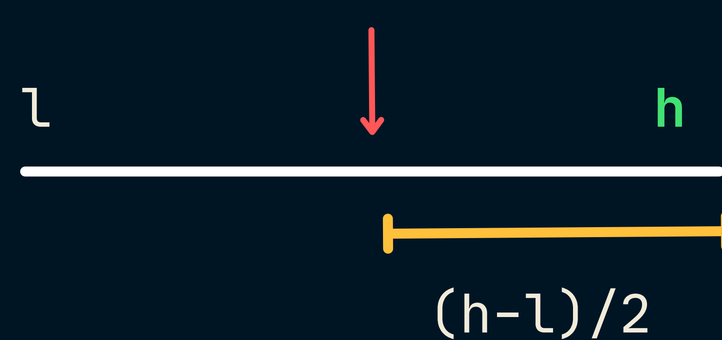
now, if both l & h are INT_MAX so l+h will cause 'overflow', so only use this way to get mid, if constraint are small.

we have other ways which also take care of 'overflow'

2) mid = l + (h-l)/2



3) mid = h - (h-l)/2



4) mid = (l + h) >> 1

>> is a right shift operator which is equivalent of (divide by 2)

Secret Trick of l & h

- Earlier we decided if search space is sorted we will divide it in 2 parts 'F' & 'U'

  Favorable (F)
  Unfavorable (U)

- Assume for some problem 1st part is 'F' & 2nd is 'U' (trick will work even if it's vice-versa).

```
[F F F F U U U U U]
 ↑               ↑
 l               h
```

- all 'F's form 1 space & all 'U's form another

- When we start 'Binary Search'
  - l is at 1st value of 1st space
  - h is at last value of 2nd space

- 'l' will always move towards 2nd space (till it doesn't crosses 1st space)
- 'h' will always move towards 1st space (till it doesn't crosses 2nd space)

```
[F F F F U U U U U]
       ↑ ↑
       h l
```

here l > h,so Binary Search terminates

- When 'Binary Search' ends (while loop terminates)
  - l is at 1st value of 2nd space
  - h is at last value of 1st space

more than 90% times our answer will be given by either 'l' or 'h' when 'Binary Search ends'

getting some hint why we only wrote 'return h' someslides back ?

Problem Types

This will be our generic template & more than 90% problems will be solved just with minor tweaks in it.

```
int l = 0;
int h = n-1;
while(l <= h) {
  int mid = (l+h) / 2;
  if( nums[mid] < k ) {
      l = mid + 1;
  } else {
      h = mid - 1;
  }
}
return h;
```

This 'if()' decides whether we are in 'F' or 'U' & accordingly move to 'left' or 'right'

(refer point 2 in slide 5)

Now depending on what goes inside that 'if()' we will categorize Binary Search in 2 types

- Type 1
- Type 2

Type 1

simple values will decide whether to enter if or else.
ex- nums[mid] > k, mid*mid < k etc...

Type 2

Inside if() we will call a function whose result will evaluate to either 'true' or 'false'

Type 2 is sometimes referred to as Binary Search on answer

More Insights

When to use Binary Search ?

If your search space is sorted & you can apply a linear search this is the intuition to go for Binary Search

How to use Binary Search ?

1) Divide the search space into 2 parts 'F' & 'U'
   ->To divide search space you need to figure out what goes inside that if()
2) Figure out who gives you answer 'l' or 'h'

I bet almost 90% of Binary Search problems will be solved using above 2 steps + that basic template

now let's solve some problems ...

Problems

Problem List

1. UpperBound
2. LowerBound
3. Sqrt(x)
4. Valid Perfect Square
5. Find the smallest letter greater than target
6. Search Insert Position
7. Valid Triangle Number
8. Arranging Coins
9. Capacity to ship packages within D days
10. Koko eating bananas
11. Allocate minimum number of pages
12. Aggressive Cows
13. Nth magical number
14. Minimum Time to Complete Trips

1st 6 are type 1 problems which don't require much observations while remaining are type 2, so they are a bit challenging

UpperBound

Description

Given a sorted array nums & an integer k, return index of smallest value greater than k

i/p	o/p
nums = [1,2,3,3,4,5]	4
k = 3	

Why to use Binary Search ?

Space is sorted + you can apply 'linear search'

To use 'Binary Search' we need to divide the search space in 2 parts 'F' & 'U'

We are asked smallest value greater than k

so nums[i] > k -> 'F'
   nums[i] <= k -> 'U'

```
     0 1 2 3 4 5
    [1 2 3 3 4 5]
l -> U U U U F F <- h
             ^
             this is our answer
```

1 Our ans is 1st element of 2nd space, so whenever your mid is at 'U' move 'right' & at 'F' move 'left'.

2 When Binary Search ends who points to 1st element of 2nd space ... l   (refer slide 9)

UpperBound

```
int upperBound(vector<int>& nums, int k) {

    int l = 0;
    int h = nums.size() - 1;

    while(l <= h) {

        int mid = l + (h-l) / 2;

        if(nums[mid] > k) {                    we are in 'F' so
            h = mid - 1;                       move left
        } else {
            l = mid + 1;                       we are in 'U' so
        }                                      move right
    }

    return l;          return l

}
```

Getting the idea...

we just took care of 2 things

1) Which is of 'Favorable' or 'Unfavorable' space.
2) Who gives us ans 'l' or 'h'

LowerBound

Description

Given a sorted array nums & an integer k, return index of first element not less than k

i/p	o/p
nums = [1,2,3,3,4,5] k = 3	2

You know why to use Binary Search... right ?

Let's divide search space in 2 parts 'F' & 'U'

We are asked 1st value not less than or greater than equal to k

```
so nums[i] >= k -> 'F'
   nums[i] <  k -> 'U'
```

```
        0 1  2 3 4 5
       [1 2  3 3 4 5]
l       U U  F F F F      h
```

this is our answer

1 ans is 1st ele of 2nd space,so if mid is at 'U' move 'right' (l = mid+1) & at 'F' move 'left' (h = mid - 1)

2 When Binary Search ends who points to 1st element of 2nd space ... l (refer slide 9)

LowerBound

```
int lowerBound(vector<int>& nums, int k) {

    int l = 0;
    int h = nums.size() - 1;

    while(l <= h) {

        int mid = l + (h-l) / 2;

        if(nums[mid] >= k) {
            h = mid - 1;        we are in 'F' so
                                move left
        } else {
            l = mid + 1;        we are in 'U' so
                                move right
        }
    }

    return l;    return l

}
```

Getting the idea...

we just took care of 2 things

1) Which is of 'Favorable' or 'Unfavorable' space.
2) Who gives us ans 'l' or 'h'

Sqrt(x)

Description

Given a non-negative integer x, compute and return the square root of x.(return only integer part)

i/p	o/p
x = 4	2
x = 8	2 it should be 2.82 but only int part, so 2

note-> your are not allowed to use any inbuilt method for calculating power or sqrt

Brute Force

consider below number line-

x = 15
o/p = 3

0 1 2 3 4 5 6 7 8 ... 15

- iterate from i = 0 till i*i <= x
- keep updating variable ans
- return ans

i	i*i	comment	ans	x
0	0*0=0	0<15,i++	0	15
1	1*1=1	1<15,i++	1	15
2	2*2=4	4<15,i++	2	15
3	3*3=9	9<15,i++	3	15
4	4*4=16	16>15,stop & return ans = 3		

can you see, we are iterating over the number line & the number line is sorted

sorted space + linear search,
go for Binary Search

Sqrt(x)

x = 15
o/p = 3

0 1 2 3 4 5 6 7 8 ... 15

- We concluded to go for Binary Search, but we need to divide
  1) Search space in 2 parts ('F' & 'U')
  2) Decide whether 'l' or 'h' gives ans.

Dividing 'Search Space'

- we were iterating till i*i <= k
  so all i for which i*i <= k are 'F'
  i*i > k are 'U'

x = 15
o/p = 3

l											h
	0	1	2	3	4	5	6	7	8 ...	15	
	F	F	F	F	U	U	U	U	U...	U	

this is
our ans

1 • ans is last value of 1st space, so when mid is at 'F' move 'right' (l=mid+1) else move 'left' (h=mid-1)

2 • when 'Binary Search' ends who points to last value of 1st space... h (refer slide 9)

Sqrt(x)

```
int mySqrt(int x) {

        long long l = 0;
        long long h = x;

        while(l <= h)
        {
            long long mid = l + (h-l)/2;

             if(mid*mid > x) {
                h = mid-1;
             } else {
                l = mid+1;
             }
        }
        return h;
    }
```

Leave a Like

Comment if you love posts like this, will motivate me to make posts like these

Share, with friends

will be continuing this series, see it takes hell lot of efforts & these are Slides so there might be some 'Typos' or I would have missed something so try to help me make it correct & avoid negatively criticizing things.