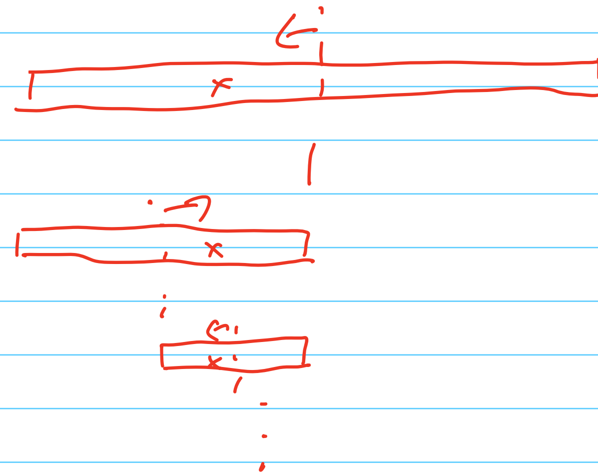
ECS 122A Lecture 2 Jasper Lee

Binary search

Setting: Sorted array, find index of element x if it exists



Sorted array $A[0..n-1]$

```
L := 0
R := n-1

while L ≤ R
    m := L + ⌊(R-L)/2⌋

    if A[m] < x then L := m+1
       A[m] > x then R := m-1
       A[m] == x then return m

return "FAIL"
```

Runtime: Divide by 2 each time, $O(\log n)$ time.

Formally can do induction

(Level of formality depends on what's convincing)

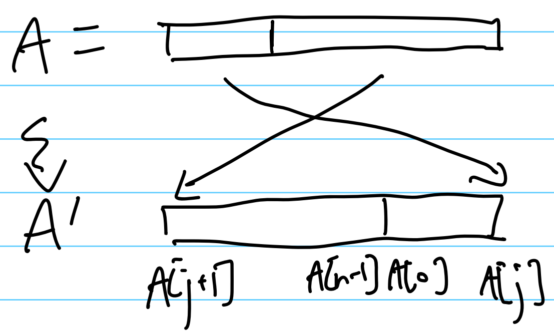
Correctness: Iterative implementation, what to induct on?

No explicit loop counter, but still our induction variable.

Loop invariant: At the end of loop iter i,
if $A[t] == x$ for some t, then $t \in [L, R]$.

Binary search is a much more general idea than searching in a sorted array. More examples today.

Rotated sorted array



sorted

Distinct value

Find the new location t of $A[0]$.

Binary search again!

For written HWs and exams, don't worry about off-by-1 errors. Only worry about that for coding assignments.

Loop invariant: keep $t \in [L, R]$, so $A'[L..R]$ is rotated sorted array

Key idea: Compare $A'[m]$ w/ $A'[R]$

Know $A'[L] > A'[R]$ by invariant.

If $A'[m] > A'[R]$ then $L = m$
$<$ $R = m$

Challenge:

What happens if we allow item duplicates?

Turns out much harder, need $\Omega(n)$ time

Why? (Intuitively, not formal proof yet)

0 1 1 1 1 1 ... 1 , rotate the 0 anywhere.

Find a 0 in an array of 1s. Takes $\Omega(n)$ time.

In fact, all deterministic alg need at least $n-1$ queries!

Single in Pairs.

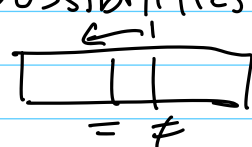
Setting:

- Sorted array, n items, odd $n \geq 3$
- Exactly 1 element appears once
  All other $n-1$ elements each appear twice.

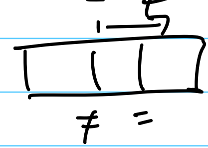
Find the lone unique element.

Idea: Query $A[mid-1]$, $A[mid]$, $A[mid+1]$

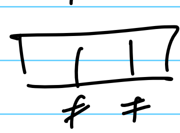
3 possibilities



go left (incl mid has odd # elements)



go right



win!

Runtime clearly still $O(\log n)$

Correctness, invariant:

$A[L..R]$ is an array containing the unique element and all other elements appear in pairs.

The previous examples are basic and kind of silly.

Binary search connects (monotonic) optimization problems to (monotonic) decision problems.

Problem 1

Find max k s.t. $P(k)$ is true

where $P(k) \Rightarrow P(k')$ for $k \geq k'$

also solves

Find max k, compare w/ k'

Problem 2

Fix k', is $P(k)$ true?

solves by binary searching function $\mathbb{1}\{P(k)\}$

Example problem:

Given a stick of integer length k, you can produce 2 sticks of integer lengths k_1, k_2 where $k_1 + k_2 = k$.

The new sticks can be further divided.

Given
- List of n sticks w/ lengths $k_1, \ldots, k_n$
- Integer m

Find max k s.t. we can produce $\geq m$ sticks of length <u>exactly</u> k.

$P(k) \doteq \{$can produce $\geq m$ sticks of length exactly k?$\}$

Why does $P(k) \Rightarrow P(k')$ for $k \geq k'$?

Can just trim off $k - k'$ from each stick!

How to check $P(k)$ for fixed k?

For each stick of length k_i, can produce

$$\left\lfloor \frac{k_i}{k} \right\rfloor \text{ many sticks of length } k$$

$\Rightarrow$ Check if $\sum_i \left\lfloor \frac{k_i}{k} \right\rfloor \geq m$

Bounds? 0 and max k_i

$\Rightarrow$ Binary search on $P(k)$ yields max k possible //

For HW + exam, no need to re-analyse binary search

Runtime?

$O(n \cdot \log L)$ where $L = \max k_i$