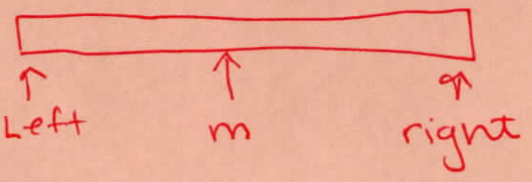
4 Oct 2021

Binary Search

input: A, an array of real-values, sorted
indexed from 1 to n
val $\in \mathbb{R}$, a value we wish to find in A

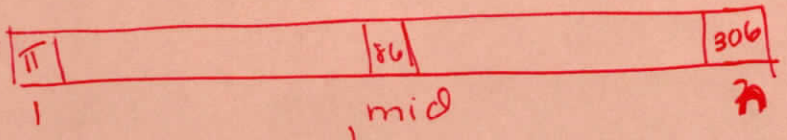
output: TRUE, if val $\in$ A
False, otw




```
1:   left ← 1
2:   right ← n
2.5: assert P
3:   while left ≤ right          <- this is G
3.5:     assert L
4:       mid ← ⌊(left + right)/2⌋
5:       if A[mid] = val
6:       |  return true          } case 1
7:       end if
8:       if A[mid] > val
9:          right ← mid - 1      } case 2
10:      else // A[mid] < val
11:         left ← mid + 1       } case 3
12:      end if/else
13:  end while
13.5: assert Q
14:  return false
```

Why does this work?
→ we know its sorted
→ each time through, we fall into one of the 3 cases + narrowed the search window.

val = 13



need to search here!

Loop Invariants (the theory)

a statement (⇒ must evaluate to true or false) that is true at the beginning & end of every loop & gets us closer to the goal

Statements of Interest

Q = the post-condition, what is true at the end (= what was the loop supposed to do?)

P = the pre-condition, what is true going into the loop for the first time?

G = the loop guard, the condition that must be met to stay in the loop. Hint: use while loops!!

L_i (or sometimes just L) = what is true upon entering the loop or attempting to enter the loop for the i^{th} time?

- must use your variables (implicit or explicit)
- think: what is getting us closer to Q?

How do these work together / what do I need to prove?

Initialization: $P \Rightarrow L_1$

Maintence: $G \wedge L_i \Rightarrow L_{i+1}$ (could break in the middle)

End: $\neg G \wedge L \Rightarrow Q$

Binary Search - How do we know the loop did what it was supposed to do?

Q = If we've returned true, val $\in A$.
and If we haven't yet returned val $\notin A$.

P = left = 1, right = n, and A is a sorted array of length n

G = left $\leq$ right.

L_i = [illegible]
(1) If the loop has returned, then val $\in A$.
and
If val $\in A$, then [illegible]
(2) val $\notin A[1 \ldots left-1]$
(3) val $\notin A[right+1 \ldots n]$

(ie, we're narrowing down the window where val could be located by eliminating choices))

Maint:

know: $G \wedge L_i$ wts: L_{i+1}

By G, we know left ≤ mid ≤ right.

By line 7: If we've encountered val, we've returned true.

In line 8: If the stmt A[mid] > val is true, then val (if in A) cannot be indexed mid or higher. So, I update right & my loop. inv. is still true.

(*) Otherwise, symmetric argument that val cannot be indexed mid or less.

Part (1) of L_{i+1} is true b/c either returned before (in which case L_i tells us val was found) or it returned in line 6.

Parts (2) and (3) follow from the argument above and L_i. In particular either left is the same as before (in which case L_i gives us val ∉ A[1 ... left-1]) or left was updated and (*) tells us val ∉ A[1 ... left-1].

□

End: $\neg G \Rightarrow$ left > right

A[1 ... left+1] ∪ A[right-1 ... n] is the whole array!

$L \Rightarrow$ If loop returned, then val ∈ A.
If loop did not return, then val ∉ A. (4)