Recall that in CS:

- a problem is defined precisely by its input + desired output
- a solution is an algorithm that takes any acceptable input and computes the desired output via a sequence of steps that a computer can perform, described in pseudocode that a human can understand.

Algorithms analysis involves

1. analyzing whether an alg. is correct - for all valid inputs, does it output what it should? (later courses)
2. analyzing the algorithm's runtime, so that we can
   - know how fast it is as input size grows
   - compare algs

   (later courses will also deal w/ other forms of computational complexity)

We analyze runtime by:

1. Finding a function that counts the # of primitive operations (arithmetic ops, boolean ops, fetching variable values, etc) in terms of input
2. Using big O notation, bc we only care how alg scales + want to compare to

other algs

Problem: is $x$ in array $A$?

input: $x$, $A$

output: T if $\exists i : A[i]$ equals $x$; F otherwise

ex $x=5$, $A = \langle 4,3,2,10 \rangle$. output: F.

$x=5$, $A = \langle 4,3,5,10,5 \rangle$. output: T bc $A[3] = 5$

Solution 1:

notation to say that the array is indexed 1 to n

**linearSearch**$(A[1 \ldots n], x)$:

**Input:** an array $A[1 \ldots n]$ and an element $x$

**Output:** is $x$ in the (possibly unsorted) array $A$?

```
1 for i := 1 to n:
2     if A[i] = x then
3         return True
4 return False
```

:= is assignment

$O(1)$ $\sum_{i=1}^{n} 1 = n$

But if $A[1] = x$, then linearSearch has a runtime of $O(1)$.

So runtime depends not just on input size, but what the input is!

When runtime depends on the specific input, we could be

1. optimistic — best case
2. pessimistic — worst case
3. neither — avg case

← easier to define
← guarantee for all inputs
$O(\cdot)$ works for all

**Def** worst-case RT of an alg is max T(n), where T(n) is RT of alg on input of size n
over all possible inputs of size n

intuitively: what is the input that makes the runtime as bad as possible?

ex for linear search, worst-case input is $A[1..n]$ where $\nexists\ i \in \{1, 2, \dots, n\}$ s.t. $A[i] = x$.

so worst-case runtime is $O(n)$.

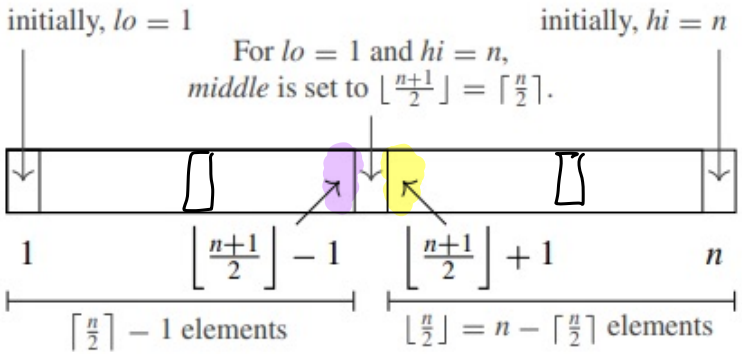
Solution 2:

**binarySearch**$(A[1 \dots n], x)$:
**Input:** a sorted array $A[1 \dots n]$; an element $x$
**Output:** is $x$ in the (sorted) array $A$?

```
1  lo := 1                                  ] O(1)
2  hi := n                                  ]
3  while lo ≤ hi:                           ]
4      middle := ⌊(lo+hi)/2⌋                ] O(1)
5      if A[middle] = x then                ]
6          return True                      ]
7      else if A[middle] > x then           ]
8          hi := middle − 1                 ]
9      else                                 ]
10         lo := middle + 1                 ]
11 return False                             ] O(1)
```

how many times?

It's not obvious how many times the loop executes.




First loop: considering $\{i \in \mathbb{Z} : 1 \le i \le n\}$

Second loop: $\{i : lo \le i \le hi\}$, and its size is halved

⋮

$|\{i : lo \le i \le hi\}|$ gets halved each iteration.

we can only do $\log_2 n$ halvings before the size of the set is 1.