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: $\frac{n^2}{2} = O(n^2)$

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

ex for i=1 to i=n do
for j=1 to j=n do
for k=1 to k=n
sum = sum+1 O(1) ] $\sum_{k=1}^{n} 1 = n$

$f(n) = n^3 = O(n^3)$

ex while n>1 do
n=n-1 ] O(1) ] n times do O(1).

O(n)

ex while n>1 do
n=n/2 ] O(1) ] $\log_2 n$ times do O(1)

suppose $n = 32 = 2^5$. $\log_2 32 = 5$

| | | |
|---|---|---|
| 1st loop | n = 16 | $\log_2 n$ |
| 2nd | n = 8 | |
| 3rd | n = 4 | |
| 4th | n = 2 | |
| 5th | n = 1 | |

O(log n)

Problem: is x in array A?
input: x, A
output: T if ∃i : A[i] equals x; F otherwise

ex x=5, A = <4,3,2,10> F

x=5, A = <4,3,2,10,5> T bc A[5]=5
1 2 3 4 5

indexed from 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
```

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

if $A[i] = x$, 1 time do $O(1)$ prim. ops.

if $\nexists i : A[i] = x$, $n$ times do $O(1)$ prim. ops.

runtime depends on not just $n$, but what the input is!

when runtime depends on input, we could be:

1. optimistic — best case
2. pessimistic — worst case ← easier to define; guarantee for all inputs
3. neither — average case

$O(\cdot)$ works for all

**Def** Worst-case runtime of an alg. is

$\max$ $T(n)$, where $T(n)$ is RT of alg on input of size $n$.

over all possible input of size $n$

Solution #2:

```
binarySearch(A[1...n], x):
Input: a sorted array A[1...n]; an element x
Output: is x in the (sorted) array A?
1  lo := 1
2  hi := n
3  while lo ≤ hi:
4      middle := ⌊(lo+hi)/2⌋
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)

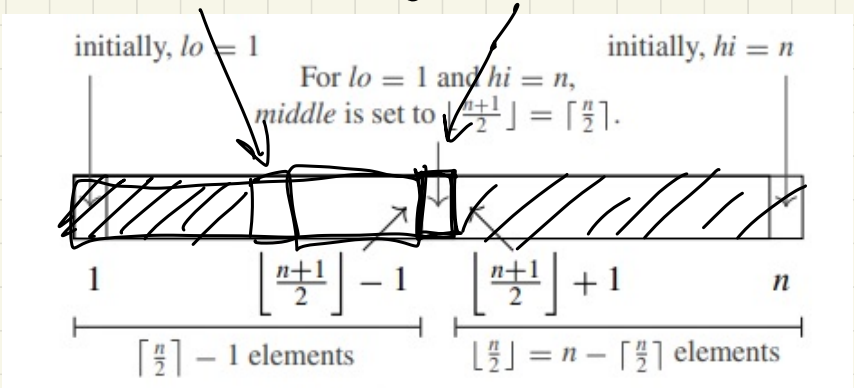
O(1)

do $\log_2 n$ loops

① give f expressing # of prim. ops in worst case for A[1...n].

worst case input: $\exists i : A[i] = x$

5 8 10




$f = 3 + (\log_2 n)20 = O(\log n)$