Recall:

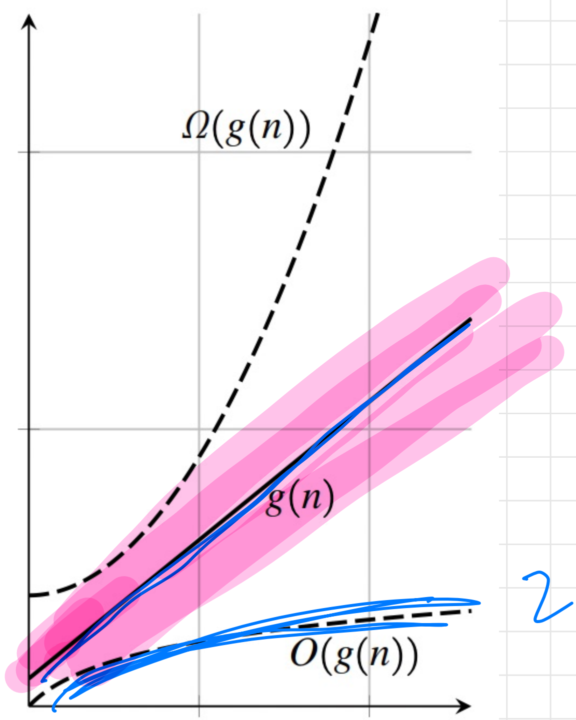
To measure the runtime of an algorithm, we:

① give $f(n)$ counting # of primitive operations on input of size n

② find simplest $g(n)$ s.t. $f(n) = \Theta(g(n))$

we often say big O

That $g(n)$ is runtime of the algorithm.



$2\log_2 n = O(g(n))$

Examples

ex

for $i=1$ to n do ← $n(1+\text{inner})$
  for $j=1$ to n do ← $n(1+\text{inner})$
    for $k=1$ to n ← $n(1+\text{inner})$
      sum = sum + 1 ← 3 operations

$$f(n) = n(1+n(1+n(1+3))) = n + n^2(1+n(4))$$
$$= n + n^2 + n^3(4)$$
$$= 4n^3 + n^2 + n$$

$$= \Theta(n^3)$$
$$= O(n^3)$$
$$= O(n^4)$$

ex

while $n \geq 1$ do $(n-1)(2+\text{inner})$
  $n = n-1$ ← 3 ops

$$f(n) = (n-1)(5) = 5n-5 = \Theta(n)$$

ex

while $n \geq 1$ do ← $\log_2 n(2+\text{inner})$
  $n = n/2$ ← 3 ops

$$f(n) = 5\log_2 n = \Theta(\log n)$$

But what about when runtime depends on specific size n input?

Problem: is x in array A?

ex $x=5$, $A=(4,3,10,7,0)$ F

$x = 5$, $A = (4, 3, 10, 7, 5)$ T

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?

```
for i := 1 to n:          1 op + inner
    if A[i] = x then      4 ops
        return True
return False
```

$:=$ is assignment operator

$\triangleq$

Suppose $x = A[1]$. $f(n) = 5 = \Theta(1)$

Suppose x not in A. $f(n) = 5n + 1 = \Theta(n)$

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

We could be:

easier to define

1. Optimistic — best case
2. pessimistic — worst case — guarantee for all inputs
3. neither — avg case — $O(\cdot)$ for all

some arbitrary function

<u>Def</u> Worst-case runtime of an algorithm is

$$T(n) = \max_{x : |x| = n} (\text{the \# of prim. ops. used on } x)$$

Intuitively: for each n, what is the input that makes the runtime as bad as possible.

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

Input: a sorted array $A[1\ldots n]$; an element x

Output: is x in the (sorted) array A?

$lo := 1$
$hi := n$ — $c_1 > 0$ operations

while $lo \leq hi$:

    $middle := \lfloor \frac{lo+hi}{2} \rfloor$
    if $A[middle] = x$ **then**
        return True
    else if $A[middle] > x$ **then**
        $hi := middle - 1$
    else
        $lo := middle + 1$ — $c_2 > 0$ operations

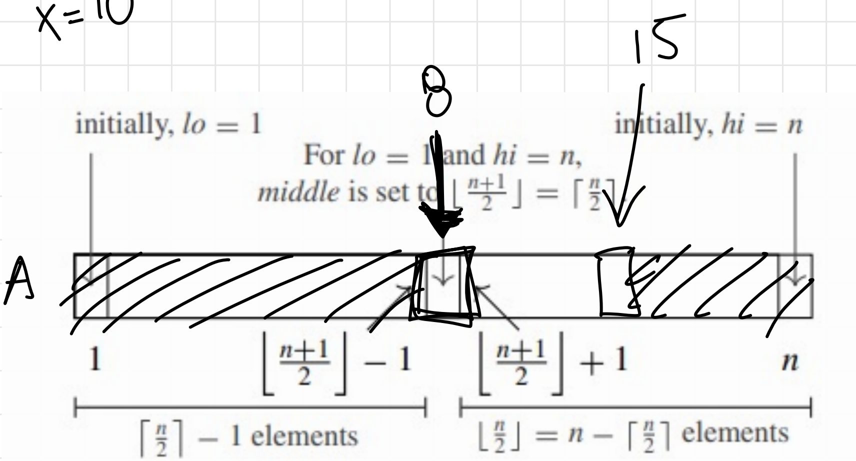
return False — $c_3 > 0$ ops

worst-case runtime (?)

worst-case input
X not in A

best-case input
X is in middle of A

x=10



<u>Claim</u> the worst-case runtime of binary search is $\Theta(\log n)$.

<u>Proof</u> The worst-case input for an array of size n is an array that does not contain x, <u>because</u> the while loop executes the max # of times.

Note that the while loop halves the number of elements under consideration with every iteration, so it executes $\log_2 n$ times.

Before the while loop, we execute $c_1 > 0$ operations. Each iteration of the while loop executes $c_2 > 0$ ops, and the final return takes $c_3 > 0$ ops.

So overall, the worst-case runtime is $f(n) = c_1 + c_2 \log_2 n + c_3 = \Theta(\log n)$.