DSC 40B

Lecture 8 : Binary Search, Recurrences

Searching a Database

Next in DSC 40B...

- How do we analyze the time complexity of **recursive** algorithms?
- How do we know that our recursive code is **correct**?

Databases

- Large data sets are often stored in databases.

PID	FullName	Level
A1843	Wan Xuegang	SR
A8293	Deveron Greer	SR
A9821	Vinod Seth	FR
A8172	Aleix Bilbao	JR
A2882	Kayden Sutton	SO
A1829	Raghu Mahanta	FR
A9772	Cui Zemin	SR
⋮	⋮	⋮

Query

What is the name of the student with PID A8172?

Linear Search

- We *could* answer this with a linear search.
- Recall worst-case time complexity: $\Theta(n)$.
- Is there a better way?

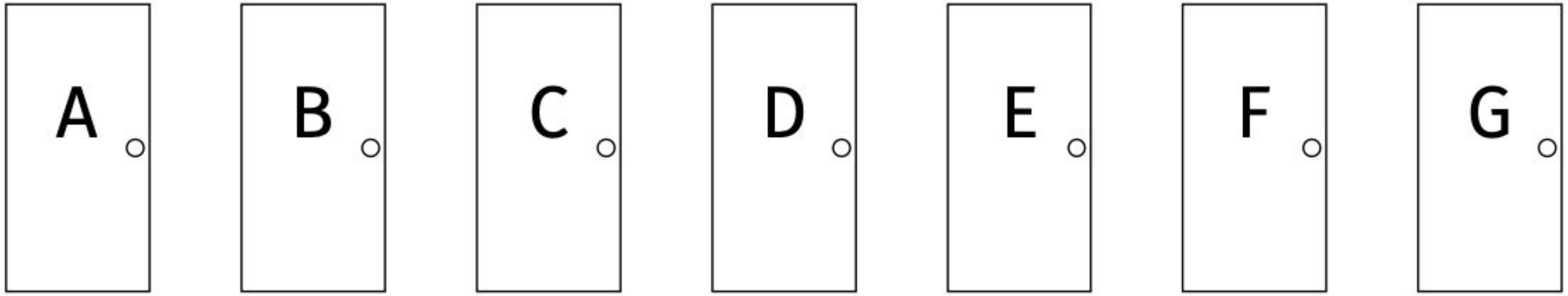
Theoretical Lower Bounds

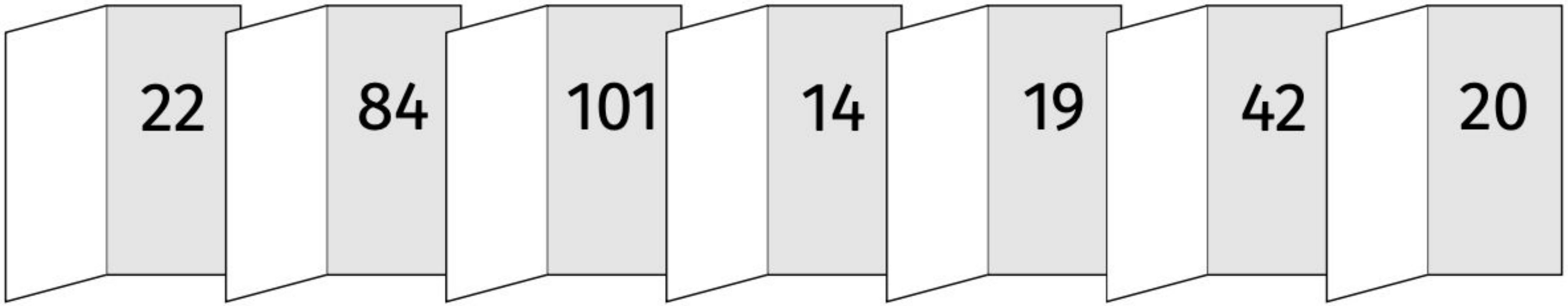
- **Given**: an array `arr` and a target `t`, determine the index of `t` in the array.
- Lower bound: $\Omega(n)$
  - `linear_search` has the best possible worst-case complexity!

Theoretical Lower Bounds

- **Given**: an **sorted** array `arr` and a target `t`, determine the index of `t` in the array.
- This is an **easier** problem.
- Theoretical Lower bound: $\Omega(?)$

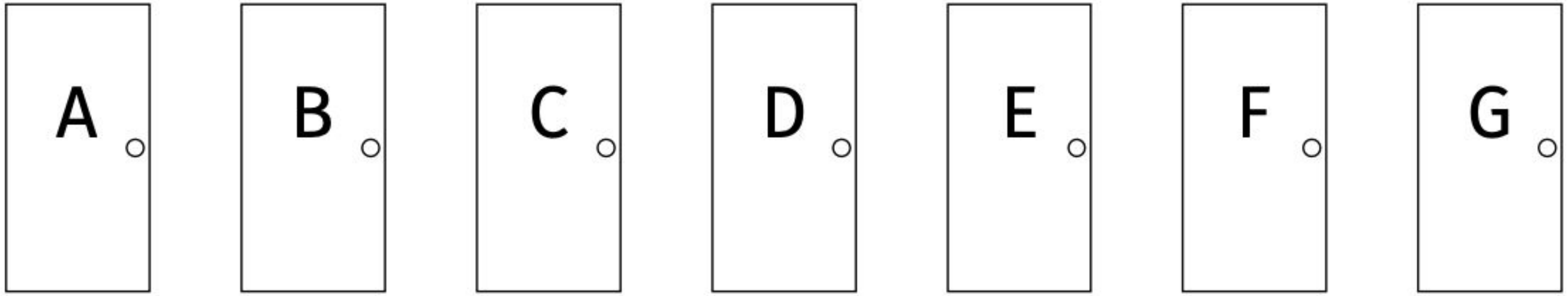
Binary Search


A
B
C
D
E
F
G


22
84
101
14
19
42
20

Game Show

- **Goal**: guess the door with number **42** behind it.
- **Caution**: with every wrong guess, your winnings are reduced.

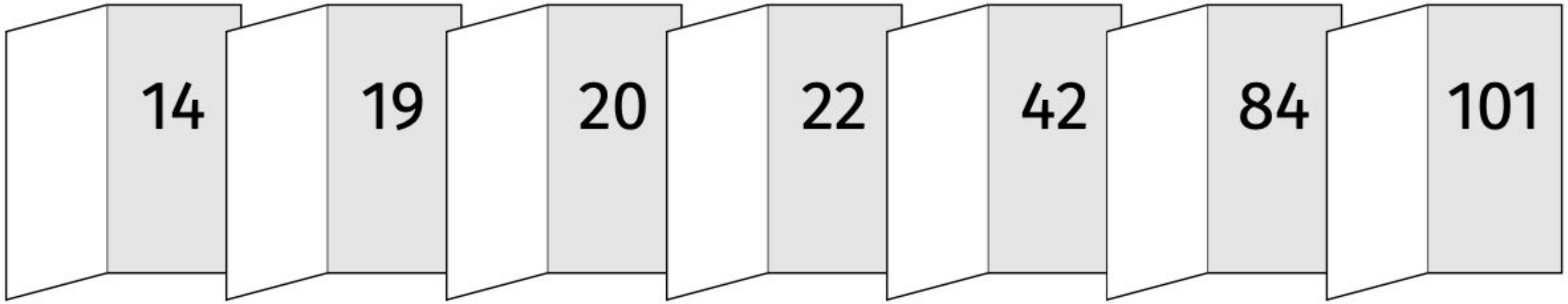

A
B
C
D
E
F
G

Strategy

- Can’t do much better than linear search.
  - “Is it door A?”
  - “OK, is it door B?”
  - “Door C?”
- After an incorrect first guess, the first guess, the right door could be any of the other n – 1 doors!

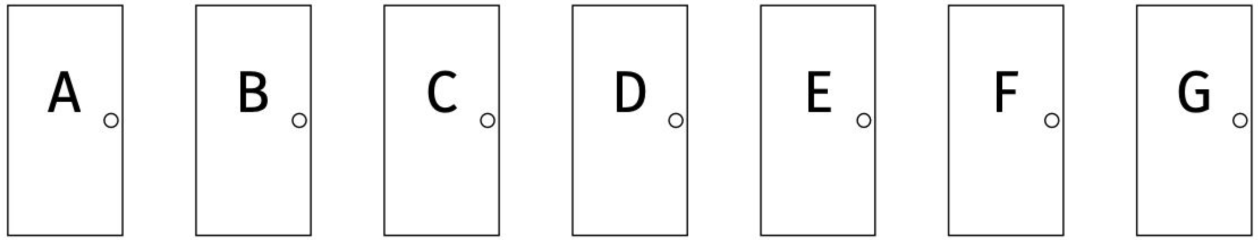
But now...

Suppose the host tells you that the numbers are **sorted** in increasing order.



Sorted!

Which door do you pick **first**?



A: A

B: G

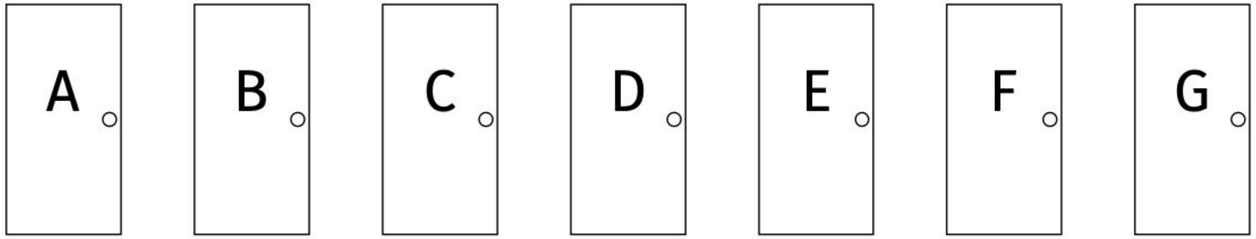
C: D

D: Does not matter

E: Some other logical choice

Sorted!

Which door do you pick **first**?



A: A

B: G

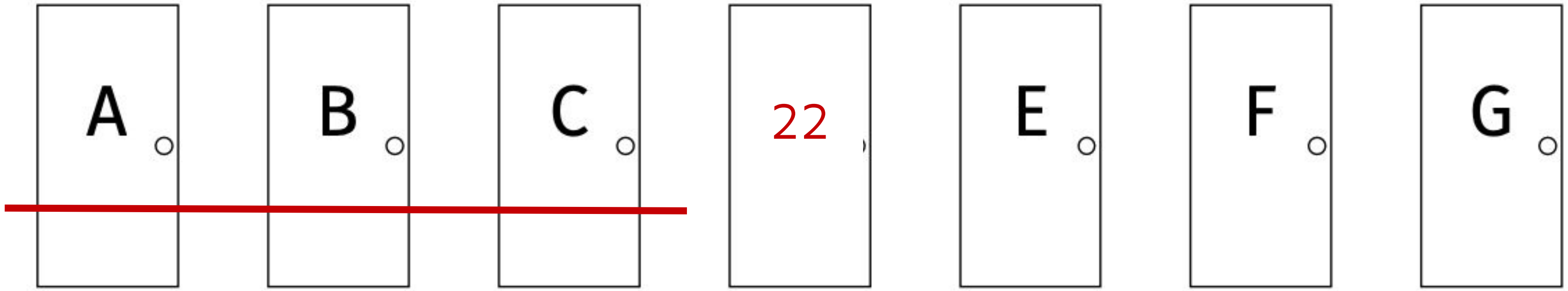
C: D

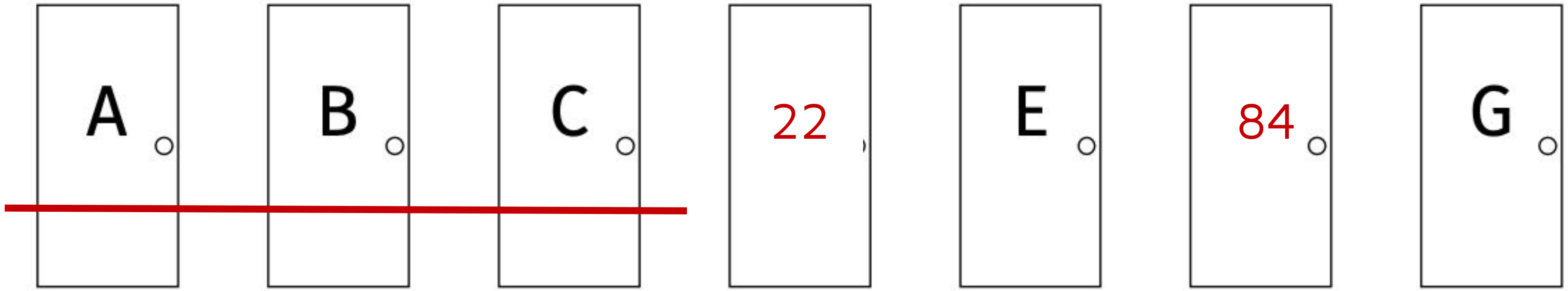
D: Does not matter

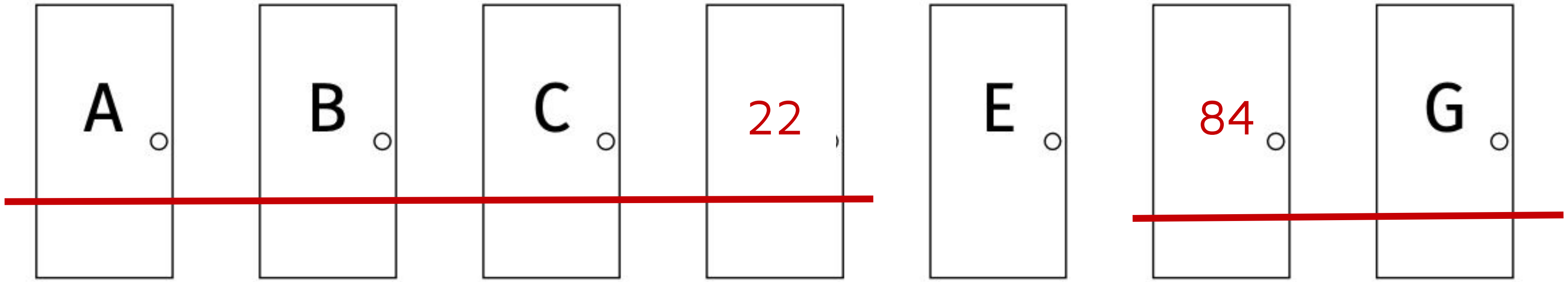
E: Some other logical choice

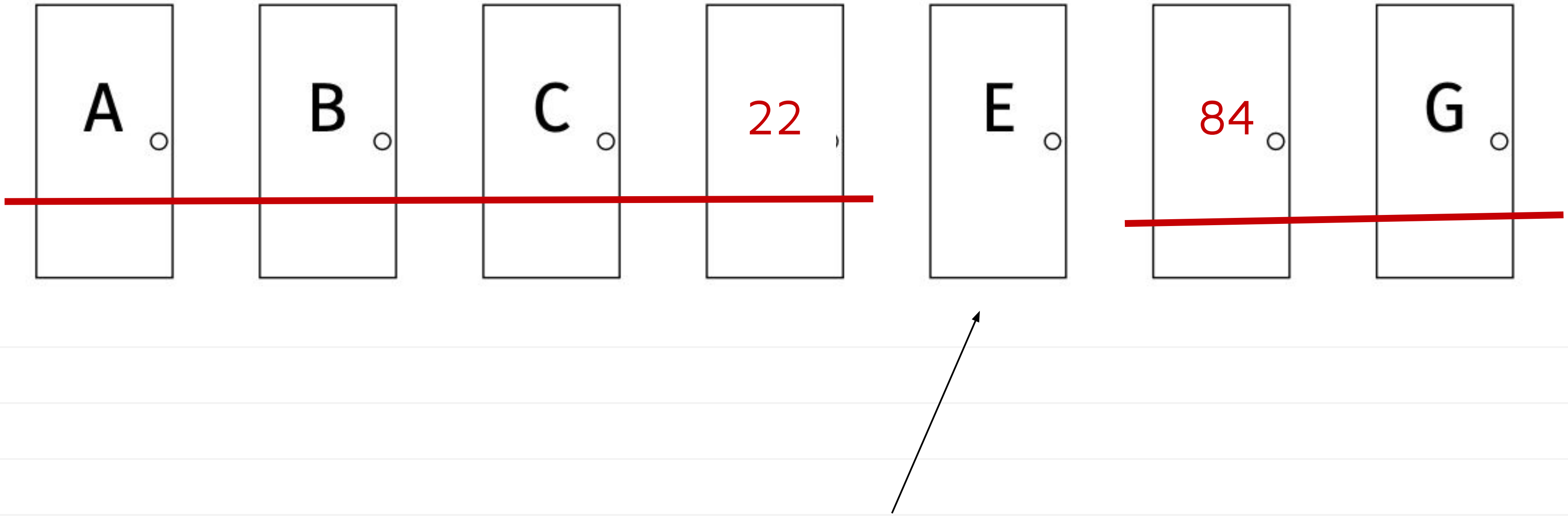
A B C D E F G

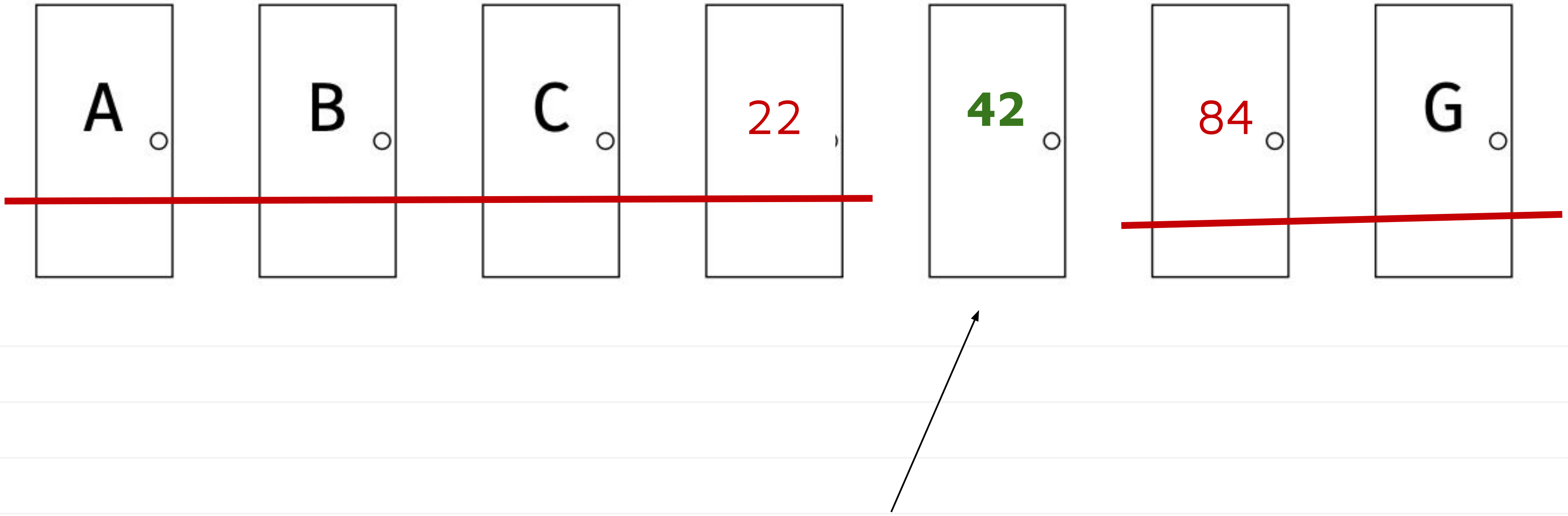
A B C 22 E F G


A
B
C
22
E
F
G


A
B
C
22
E
84
G


A
B
C
22
E
84
G


A
B
C
22
E
84
G


A
B
C
22
42
84
G

Strategy

- First pick the **middle** door.
- Allows you to rule out half of the other doors.
- Pick door in the middle of what remains.
- Repeat, ***recursively***.

Binary Search in Code: fill in the blanks

```
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = ___________ # index of the middle element
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, ______, ______)
    else:
        return binary_search(arr, t, ______, ______)
```

The middle element

- What is the index of the middle element of `arr[start : stop]`

-20	-13	3	6	10	12	15	34	56	76	89
0	1	2	3	4	5	6	7	8	9	10

The middle element

- What is the index of the middle element of `arr[start : stop]`

-20	-13	3	6	10	12	15	34	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```
start = 0
stop = 11
```

The middle element

- What is the index of the middle element of `arr[start : stop]`

-20	-13	3	6	10	12	15	34	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```
start = 0
stop = 11
```

```
middle = (stop-start)/2
```

The middle element

- What is the index of the middle element of `arr[start : stop]`

-20	-13	3	6	10	12	15	34	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```
start = 0
stop = 11
```

```
middle = (stop-start)/2
```

issue?

The middle element

- What is the index of the middle element of `arr[start : stop]`

-20	-13	3	6	10	12	15	34	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```
start = 0
stop = 11
```

```
middle = (stop-start)/2 + start
```

The middle element

- What is the index of the middle element of `arr[start : stop]`

-20	-13	3	6	10	12	15	34	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```
start = 0
stop = 11
```

```
middle = (stop + start)/2
```

Definition

The **floor** of a real number x, denoted $\lfloor x \rfloor$, is the **largest** integer that is $\leq x$.

- **Examples**:
  - $\lfloor 3.14 \rfloor = 3$
  - $\lfloor -4.5 \rfloor = -5$
  - $\lfloor 10 \rfloor = 10$

Definition

The **floor** of a real number x, denoted $\lceil x \rceil$, is the **smallest** integer that is $\geq x$.

- **Examples**:
  - $\lceil 3.14 \rceil = 4$
  - $\lceil -4.5 \rceil = -4$
  - $\lceil 10 \rceil = 10$

Binary Search

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

start = 0
stop = 11

-20	-13	3	6	10	12	15	**34**	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

start = 0
stop = 11

middle=5

bs(0, 11)

-20	-13	3	6	10	12	15	**34**	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

start = 0
stop = 11

middle=5
middle=8

bs(0, 11)
bs(6, 11)

~~-20~~	~~-13~~	~~3~~	~~6~~	~~10~~	~~12~~	15	**34**	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

start = 0
stop = 11

middle=5
middle=8

bs(0, 11)
bs(6, 11)
bs(6, 8)

~~-20~~	~~-13~~	~~3~~	~~6~~	~~10~~	~~12~~	15	**34**	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

start = 0
stop = 11

middle=5
middle=8
middle=7

bs(0, 11)
bs(6, 11)
bs(6, 8)

~~-20~~	~~-13~~	~~3~~	~~6~~	~~10~~	~~12~~	15	**34**	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

start = 0
stop = 11

middle=5
middle=8
middle=7

bs(0, 11)
bs(6, 11)
bs(6, 8)
7

~~-20~~	~~-13~~	~~3~~	~~6~~	~~10~~	~~12~~	15	**34**	56	76	89
0	1	2	3	4	5	6	7	8	9	10

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

start = 0
stop = 11

middle=5
middle=8
middle=7

bs(0, 11)
bs(6, 11)
bs(6, 8)
7

~~-20~~	~~-13~~	~~3~~	~~6~~	~~10~~	~~12~~	15	**34**	56	76	89
0	1	2	3	4	5	6	7	8	9	10

Aside: Default Arguments

```
import math
def binary_search(arr, t, start=0, stop=None):
    if stop is None:
        stop = len(arr)
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Aside: Default Arguments

Can't use stop = len(arr)

```
import math
def binary_search(arr, t, start=0, stop=None):
    if stop is None:
        stop = len(arr)
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Other Uses of Binary Search

- Binary search is a useful strategy in “real life”.
- **Example**: finding a bug in your code.
    - You have made 100 git commits (i.e., 100 changes).
    - You know the bug is introduced in one of them.
    - “Rewind” to commit #50.
        - If it bug is there, check in #25;
    - Otherwise, check in #75.
    - The `git bisect` command does this for you.

Thinking Inductively

Recursion

- Recursive algorithms can almost look like **magic**.
- How can we be sure that `binary_search` works?

Tips

1. Make sure algorithm works in the **base case**.

2. Check that all recursive calls are on **smaller problems**.

3. **Assuming** that the recursive calls work, does the whole algorithm work?

Base Case

- **Smallest** input for which you can easily see that the algorithm works.
- Recursion works by making problem **smaller** until base case is reached.
- Usually n = 0 or n = 1 (or even both!)

Base Case: $n = 0$

- Suppose `arr[start:stop]` is empty.
- In this case, the function returns **None**.
    - **Correct!**

Base Case: $n = 1$

- Suppose `arr[start:stop]` has **one** element.
- If that element is the target, the algorithm will find it!
  - **Correct!**
- If it isn't, the algorithm will *recurse* on a problem of size 0 and return **None**.
  - **Correct!**

Recursive Calls

- Recursive calls must be on **smaller problems.**
  - Otherwise, base case never reached. Infinite recursion!

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Is `arr[start:middle]` smaller than `arr[start:stop]`?
Is `arr[middle+1:stop]` smaller than `arr[start:stop]`?

Leap of Faith

- **Assume** the recursive calls work.
- Does the overall algorithm work, then?

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Does this code work? Why or why not?

```
import math
def summation(numbers):
    n = len(numbers)
    if n == 0:
        return 0
    middle = math.floor(n / 2)
    return (
        summation(numbers[:middle])
        +
        summation(numbers[middle:]
    )
```

A: Yes

B: No (base case)

C: No (rec. call issue)

D: No, issue with putting solution together.

Does this code work? Why or why not?

```
import math
def summation(numbers):
    n = len(numbers)
    if n == 0:
        return 0
    middle = math.floor(n / 2)
    return (
        summation(numbers[:middle])
        +
        summation(numbers[middle:]
    )
```

Induction

- These steps can be turned into a formal proof by **induction**.
- For us, less necessary to prove to other people.
- Instead, prove to **yourself** that your code works.
- We won't be doing formal inductive proofs.

Recurrence Relations

Time Complexity of Binary Search

- What is the time complexity of `binary_search`?
- No loops!

Best case?

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Best case?

Theta(1)

```
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Worst Case

```
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Worst Case

T(n) =

Θ(?)

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Worst Case

T(n) =

$\Theta(1)$

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Worst Case

$T(n) = \Theta(1) +$ *time taken for recursive calls*

$\Theta(1)$

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Worst Case

T(n) = Θ(1) + *time taken for recursive calls*

Θ(1)

```
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

What is the input size for this call?

Worst Case

T(n) = Θ(1) + *time taken for recursive calls*

Θ(1)

```
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

n/2 since we split in the middle

Worst Case

$T(n) = \Theta(1)$ + *time taken for recursive calls*

$\Theta(1)$

```
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Time taken is T(n/2)

Worst Case

T(n) = Θ(1) + *time taken for recursive calls*

```python
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Θ(1)

Same for the second one T(n/2)

Worst Case

$T(n) = \Theta(1) + T(n/2)$

$\Theta(1)$

```
import math
def binary_search(arr, t, start, stop):
    """
    Searches arr[start:stop] for t.
    Assumes arr is sorted.
    """
    if stop - start <= 0:
        return None
    middle = math.floor((start + stop)/2)
    if arr[middle] == t:
        return middle
    elif arr[middle] > t:
        return binary_search(arr, t, start, middle)
    else:
        return binary_search(arr, t, middle+1, stop)
```

Same for the second one T(n/2)

Recurrence Relations

- We found

$$T(n) = \begin{cases} T(n/2) + \Theta(1), & n \geq 2 \\ \Theta(1), & n = 1 \end{cases}$$

- This is a recurrence relation.

Solving Recurrences

- We want simple, non-recursive formula for $T(n)$ so we can see how fast $T(n)$ grows.
  - Is it $\Theta(n)$? $\Theta(n^2)$? Something else?
- Obtaining a simple formula is called solving the recurrence.

Example: Getting Rich

- Suppose on day 1 of job, you are paid $3.
- Each day thereafter, your pay is doubled.
- Let $S(n)$ be your pay on day n:

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

Example: Unrolling

- How much are you paid on day 4?

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

A: $12

B: $8

C: $24

D: $16

E: Something else

Example: Unrolling

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

- How much are you paid on day 4?
- $S(4) = 2 \cdot S(3)$

Example: Unrolling

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

- How much are you paid on day 4?
- S(4) = 2·**S(3)**

  = 2·**2·S(2)**

Example: Unrolling

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

- How much are you paid on day 4?
- $S(4) = 2 \cdot S(3)$

$$= 2 \cdot 2 \cdot \mathbf{S(2)}$$

$$= 2 \cdot 2 \cdot \mathbf{2 \cdot S(1)}$$

Example: Unrolling

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

- How much are you paid on day 4?
- $S(4) = 2 \cdot S(3)$

$= 2 \cdot 2 \cdot S(2)$

$= 2 \cdot 2 \cdot 2 \cdot \mathbf{S(1)}$

$= 2 \cdot 2 \cdot 2 \cdot \mathbf{3}$

$= 24$

Example: Unrolling

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

- How much are you paid on day 4?
- $S(4) = 2 \cdot S(3)$

$= 2 \cdot 2 \cdot S(2)$

$= 2 \cdot 2 \cdot 2 \cdot S(1)$

$= 2 \cdot 2 \cdot 2 \cdot 3$

$= 24$

$S(n) = 2^{?} + 3$

Example: Unrolling

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

- How much are you paid on day 4?
- $S(4) = 2 \cdot S(3)$

$= 2 \cdot 2 \cdot S(2)$

$= 2 \cdot 2 \cdot 2 \cdot S(1)$

$= 2 \cdot 2 \cdot 2 \cdot 3$

$= 24$

$$S(n) = 2^{n-1} \cdot 3$$

Solving Recurrences

We'll use a **four-step** process to solve recurrences:

1. "Unroll" several times to find a **pattern**.
2. Write general **formula** for kth unroll.
3. Solve for # of unrolls needed to reach base case.
4. Plug this number into general formula.

Step 1: Unroll several times

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

$S(n) = 2 \cdot S(n-1)$

Step 1: Unroll several times

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

$$\begin{aligned} S(n) &= 2 \cdot S(n-1) \\ &= 2 \cdot 2 \cdot S(n-2) \end{aligned}$$

Step 1: Unroll several times

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

$S(n) = 2 \cdot S(n-1)$

$= 2 \cdot 2 \cdot S(n-2)$

$= 2 \cdot 2 \cdot 2 \cdot S(n-3)$

Step 1: Unroll several times

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

$$\begin{aligned} S(n) &= 2 \cdot S(n-1) \\ &= 2 \cdot 2 \cdot S(n-2) \\ &= 2 \cdot 2 \cdot 2 \cdot S(n-3) \\ &= 2 \cdot 2 \cdot 2 \cdot 2 \cdot S(n-4) \\ &\ldots\ldots \end{aligned}$$

Step 2: Find general formula

$S(n) = 2 \cdot S(n-1)$ **#k=1**

$= 2 \cdot 2 \cdot S(n-2)$ **#k=2**

$= 2 \cdot 2 \cdot 2 \cdot S(n-3)$ **#k=3**

$= 2 \cdot 2 \cdot 2 \cdot 2 \cdot S(n-4)$

On step k:

Step 2: Find general formula

$S(n) = 2 \cdot S(n-1)$ **#k=1**

$= 2 \cdot 2 \cdot S(n-2)$ **#k=2**

$= 2 \cdot 2 \cdot 2 \cdot S(n-3)$ **#k=3**

$= 2 \cdot 2 \cdot 2 \cdot 2 \cdot S(n-4)$ **#k=4**

On step k:

$S(n) = 2^k \cdot S(n-k)$

Step 3: Find step # of base case

- On step k, $S(n) = 2^k \cdot S(n - k)$.
- When do we see $S(1)$?

Step 3: Find step # of base case

- On step k, $S(n) = 2^k \cdot S(\boldsymbol{n - k})$.
- When do we see $S(1)$?
- When **n - k** =1

Step 3: Find step # of base case

- On step k, $S(n) = 2^k \cdot S(\boldsymbol{n} - \boldsymbol{k})$.
- When do we see $S(1)$?
- When **n - k** =1 => **k = n - 1**

Step 4: Plug into general formula

- From **step 2**: $S(n) = 2^k \cdot S(n-k)$.
- From **step 3**: Base case of $S(1)$ reached when $k = n - 1$.
- So

Step 4: Plug into general formula

- From step 2: $S(n) = 2^k \cdot S(n - k)$.
- From step 3: Base case of $S(1)$ reached when $k = n - 1$.
- So $S(n) = 2^{n-1} \cdot S(n - (n-1))$

Step 4: Plug into general formula

- From step 2: $S(n) = 2^k \cdot S(n-k)$.
- From step 3: Base case of $S(1)$ reached when $k = n - 1$.
- So $S(n) = 2^{n-1} \cdot S(n - (n-1))$

$$= 2^{n-1} \cdot S(1)$$

Step 4: Plug into general formula

- From step 2: $S(n) = 2^k \cdot S(n-k)$.
- From step 3: Base case of $S(1)$ reached when $k = n - 1$.
- So $S(n) = 2^{n-1} \cdot S(n-(n-1))$

$$= 2^{n-1} \cdot S(1)$$

$$= 2^{n-1} \cdot 3$$

$$S(n) = \begin{cases} 2 \cdot S(n-1), & n \geq 2 \\ 3, & n = 1 \end{cases}$$

Solving the Recurrence

- We have solved the recurrence: $S(n) = 3 \cdot 2^{n-1}$
- This is the **exact** solution. The **asymptotic** solution is

$$S(n) = \Theta(2^n).$$

- We'll call this method **"solving by unrolling".**
- **Take the job? Yes! On day 20 you will get ~ 1.5 mill.$**

Thank you!

Do you have any questions?

CampusWire!