Lecture 15: Searching, Approximation Methods

Comp 102

Forman Christian University

Recap

Search Techniques

Guess and Check

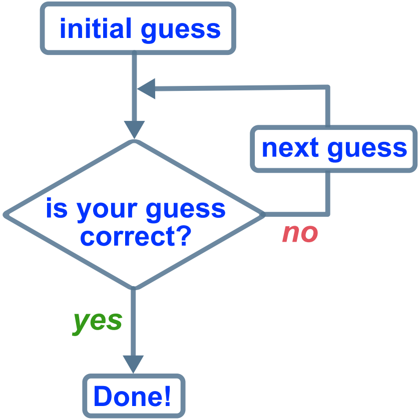
- Applies to a problem when:
  - You know the set of **“all possible solutions”**
  - You are able to check **“if the solution is correct”**

Guess and Check

- Applies to a problem when:
  - You know the set of **"all possible solutions"**
  - You are able to check **"if the solution is correct"**
- Keep guessing until:
  - You find the correct solution
  - You have tried all possible solutions

Guess and Check

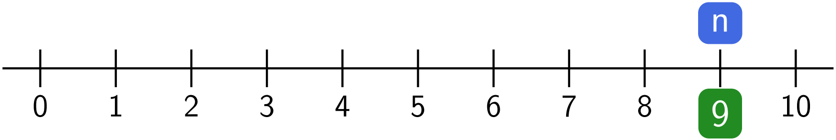
- Applies to a problem when:
  - You know the set of **"all possible solutions"**
  - You are able to check **"if the solution is correct"**
- Keep guessing until:
  - You find the correct solution
  - You have tried all possible solutions



Square Root

Random Search

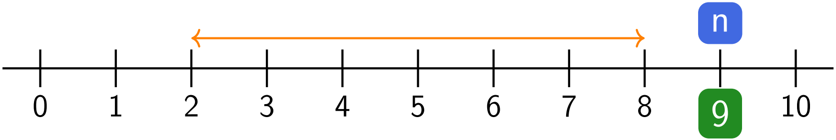
- Square root of $n = 9$



Square Root

Random Search

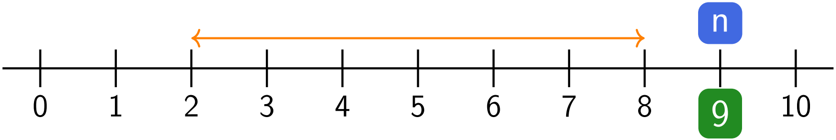
- Square root of $n = 9$
- Possible range in which the square root lies?



Square Root

Random Search

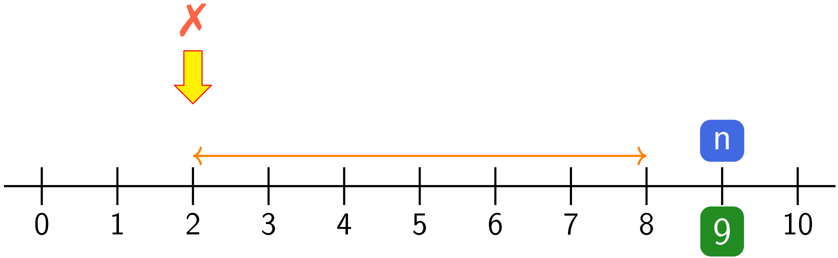
- Square root of $n = 9$
- Possible range in which the square root lies?
- Let's start guessing randomly *(within this range)*



Square Root

Random Search

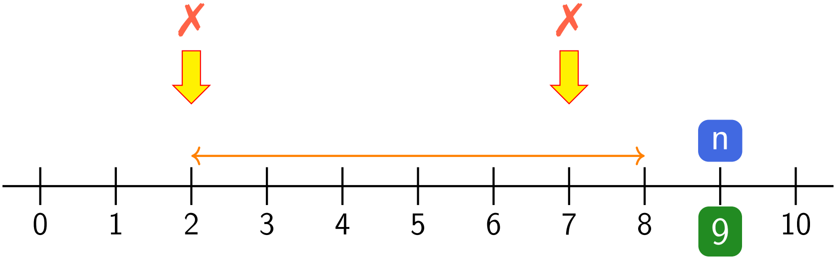
- Square root of $n = 9$
- Possible range in which the square root lies?
- Let's start guessing randomly *(within this range)*



Square Root

Random Search

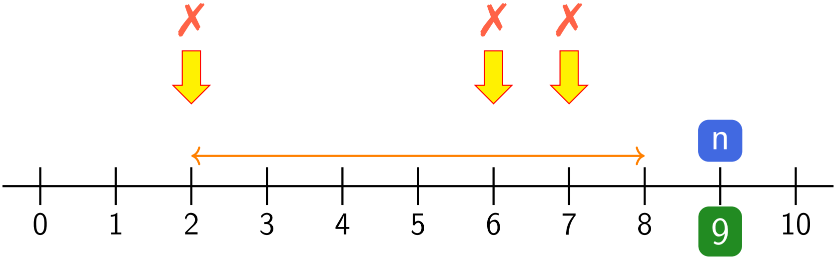
- Square root of $n = 9$
- Possible range in which the square root lies?
- Let's start guessing randomly *(within this range)*



Square Root

Random Search

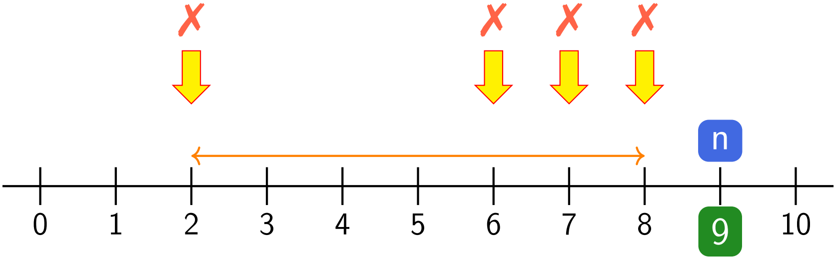
- Square root of $n = 9$
- Possible range in which the square root lies?
- Let's start guessing randomly *(within this range)*



Square Root

Random Search

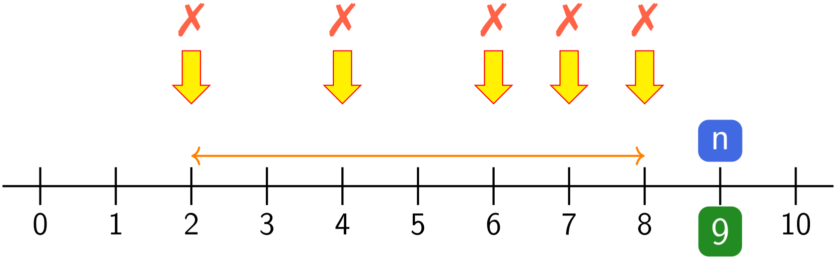
- Square root of $n = 9$
- Possible range in which the square root lies?
- Let's start guessing randomly *(within this range)*



Square Root

Random Search

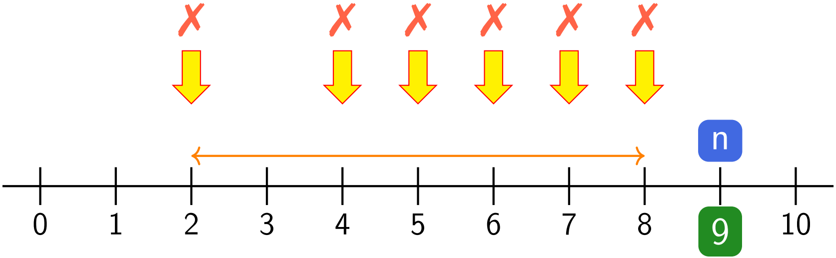
- Square root of $n = 9$
- Possible range in which the square root lies?
- Let's start guessing randomly *(within this range)*



Square Root

Random Search

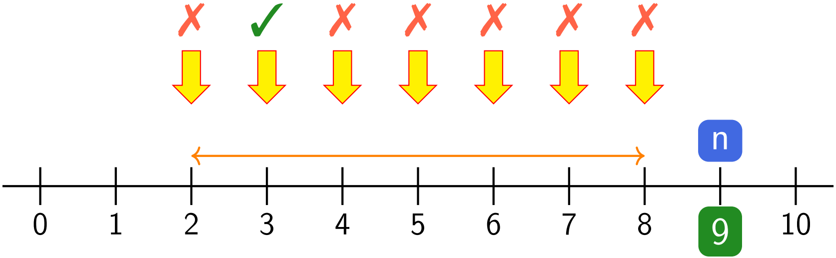
- Square root of $n = 9$
- Possible range in which the square root lies?
- Let's start guessing randomly *(within this range)*



Square Root

Random Search

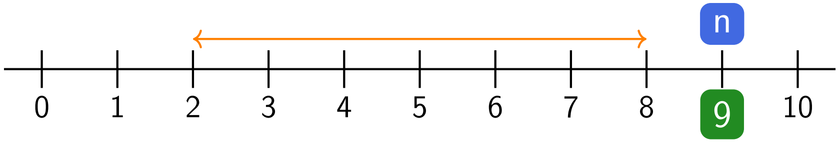
- Square root of $n = 9$
- Possible range in which the square root lies?
- Let's start guessing randomly *(within this range)*



Square Root (9?)

Systematic Search

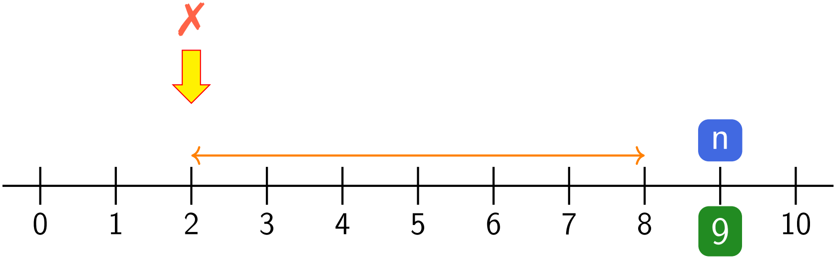
- Let's start guessing **systematically** *(within this range)*
  - Start from the **first possible solution**
  - Move to **next possible solution**



Square Root (9?)

Systematic Search

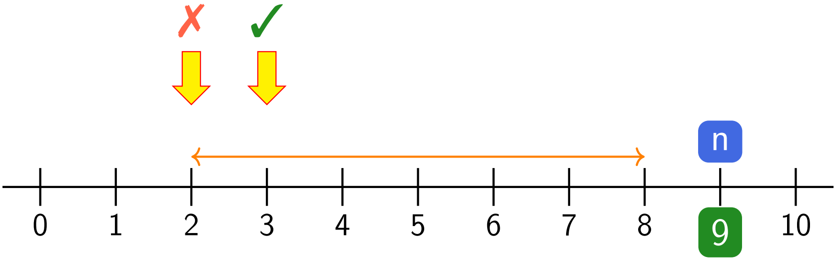
- Let's start guessing **systematically** *(within this range)*
  - Start from the **first possible solution**
  - Move to **next possible solution**



Square Root (9?)

Systematic Search

- Let's start guessing **systematically** *(within this range)*
  - Start from the **first possible solution**
  - Move to **next possible solution**



Square Root with while Loop

```
x = int(input("Enter an integer: "))
guess = 0
while guess**2 < x:
    guess = guess + 1
if guess**2 == x:
    print("Square root of", x, "is", guess)
else:
    print(x, "is not a perfect square")
```

Verify on PythonTutor!

Square Root with while Loop

```
x = int(input("Enter an integer: "))
guess = 0
while guess**2 < x:
    guess = guess + 1
if guess**2 == x:
    print("Square root of", x, "is", guess)
else:
    print(x, "is not a perfect square")
```

*Exit the loop when guess**2 >= x*

Verify on PythonTutor!

Square Root with while Loop

```
x = int(input("Enter an integer: "))
guess = 0
while guess**2 < x:
    guess = guess + 1
if guess**2 == x:
    print("Square root of", x, "is", guess)
else:
    print(x, "is not a perfect square")
```

*Exit the loop when guess**2 >= x*

We found the square root

Verify on PythonTutor!

You Try!

Rewrite the code to find the square root of a number using for loop.

Big Idea

You can't check infinite number of values

You have to stop at some point

You Try!

Ali thought of a secret integer between 1 and 100. Store it in a variable n.

1. Write a program that systematically tries to guess the number e.g. going $0, 1, 2, 3, 4, \cdots, 100$.
2. If the program doesn't find the number, print a message to the user that it didn't find it.

Big Idea

Booleans can be used as signals that something happened

We call them Boolean flags

Searching Example

Remember those word problems from your childhood?

- Ali and Fatima are selling tickets for a charity event.
- Fatima sells 3 fewer tickets than Ali.
- Together, they sell a total of 17 tickets.
- How many tickets did Ali sell?

Searching Example

Remember those word problems from your childhood?

- Ali and Fatima are selling tickets for a charity event.
- Fatima sells 3 fewer tickets than Ali.
- Together, they sell a total of 17 tickets.
- How many tickets did Ali sell?

We could solve this algebraically, but we can also use "**guess-and-check**"

Searching Example

Strategy 1: Random Search ($a + f == 17$)

- Ali: 05, Fatima: 02 $a + f \neq 17$

Searching Example

Strategy 1: Random Search $(a + f == 17)$

- Ali: 05, Fatima: 02 $\quad a + f \neq 17$
- Ali: 10, Fatima: 04 $\quad a + f \neq 17$

Searching Example

Strategy 1: Random Search $(a + f == 17)$

- Ali: 05, Fatima: 02 $a + f \neq 17$
- Ali: 10, Fatima: 04 $a + f \neq 17$
- Ali: 09, Fatima: 09 $a + f \neq 17$

Searching Example

Strategy 1: Random Search $(a + f == 17)$

- Ali: 05, Fatima: 02 $a + f \neq 17$
- Ali: 10, Fatima: 04 $a + f \neq 17$
- Ali: 09, Fatima: 09 $a + f \neq 17$
- Ali: 01, Fatima: 15 $a + f \neq 17$

Searching Example

Strategy 1: Random Search ($a + f == 17$)

- Ali: 05, Fatima: 02 $a + f \neq 17$
- Ali: 10, Fatima: 04 $a + f \neq 17$
- Ali: 09, Fatima: 09 $a + f \neq 17$
- Ali: 01, Fatima: 15 $a + f \neq 17$
- Ali: 14, Fatima: 10 $a + f \neq 17$
- $\cdots$
- $\cdots$

Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

Strategy 2: Systematic Search (*how?*)

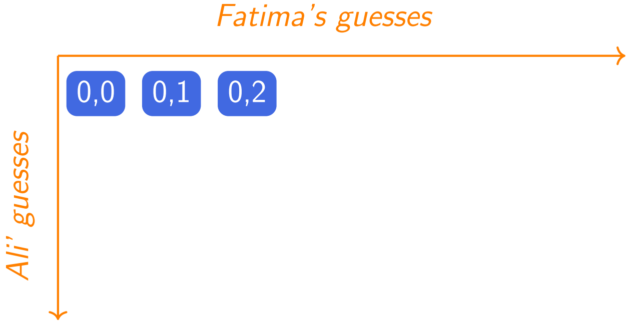
Any ideas before we proceed?

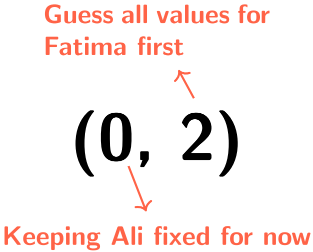
Generate all possible pairs of Ali and Fatima's tickets:

Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

Generate all possible pairs of Ali and Fatima's tickets:

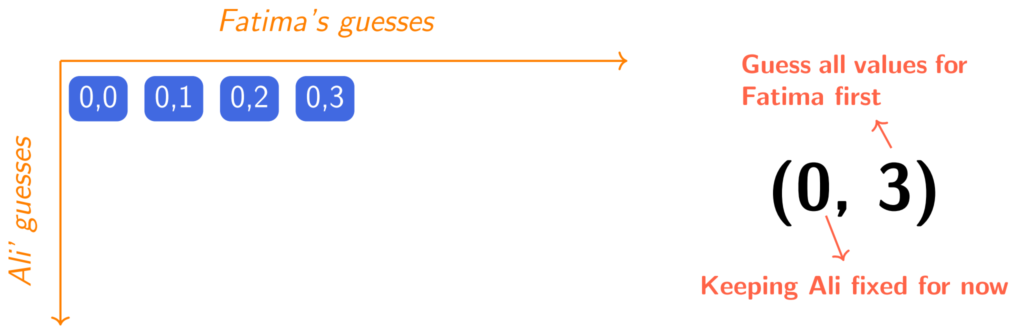




Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

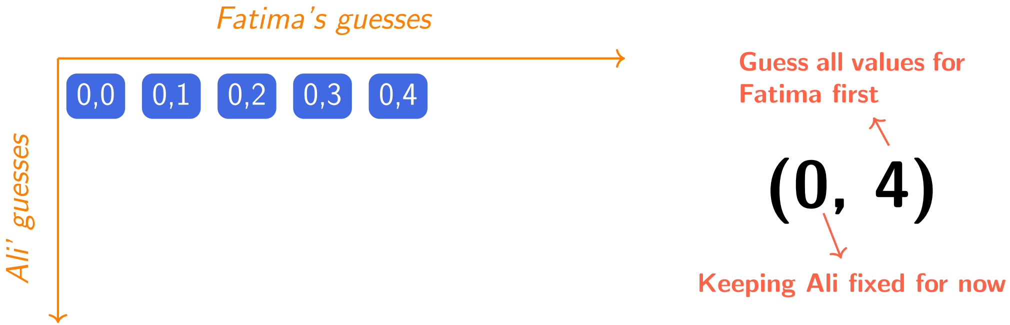
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

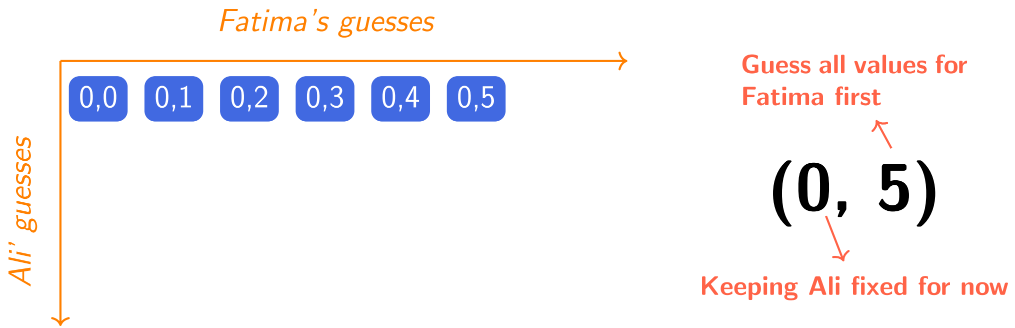
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

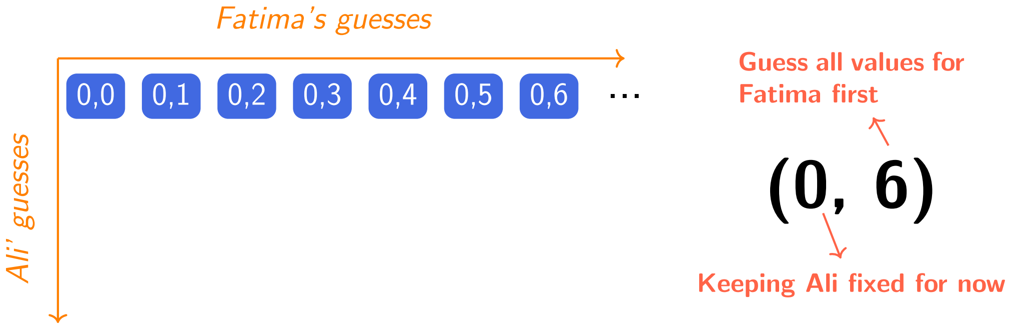
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

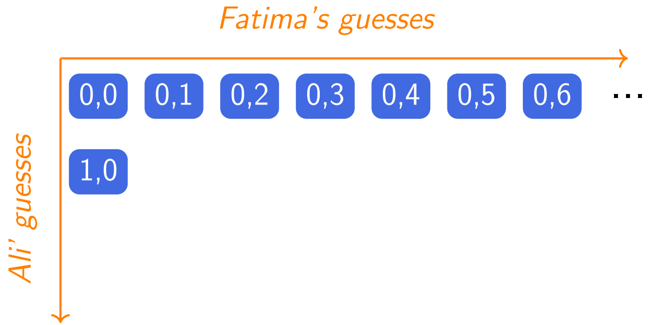
Generate all possible pairs of Ali and Fatima's tickets:

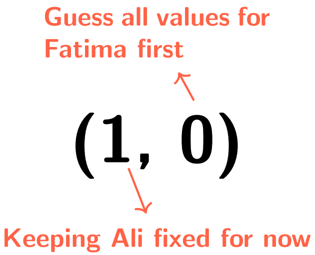


Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

Generate all possible pairs of Ali and Fatima's tickets:

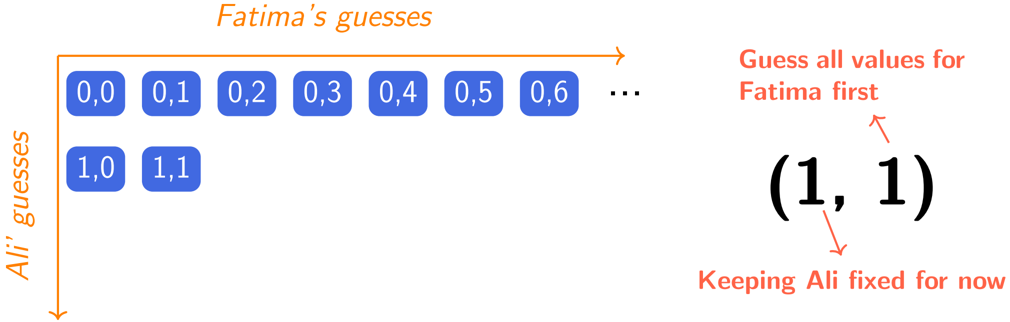




Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

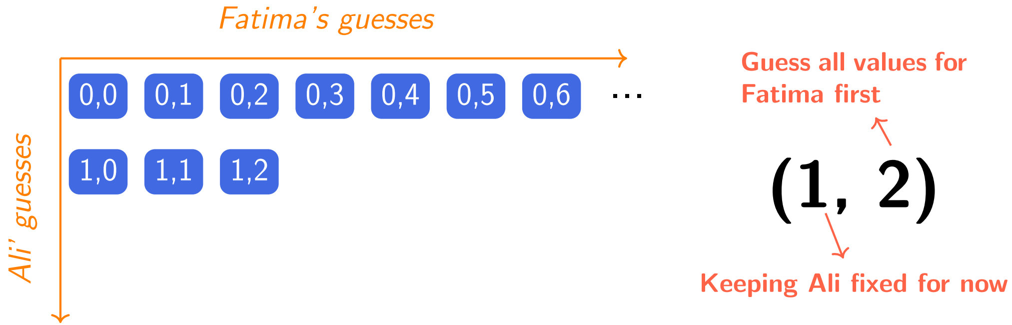
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

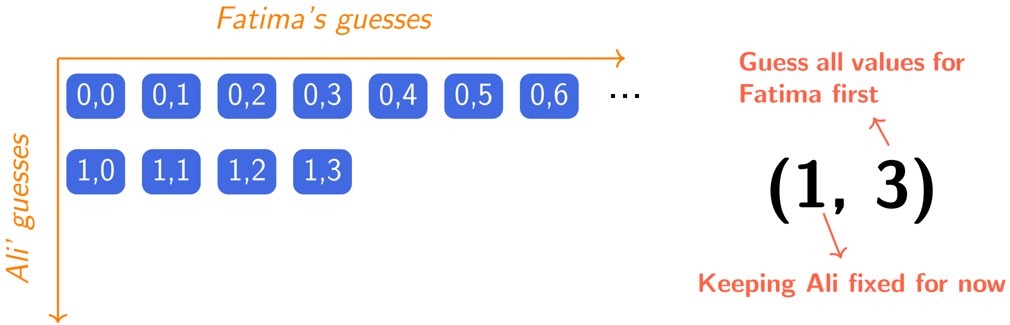
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

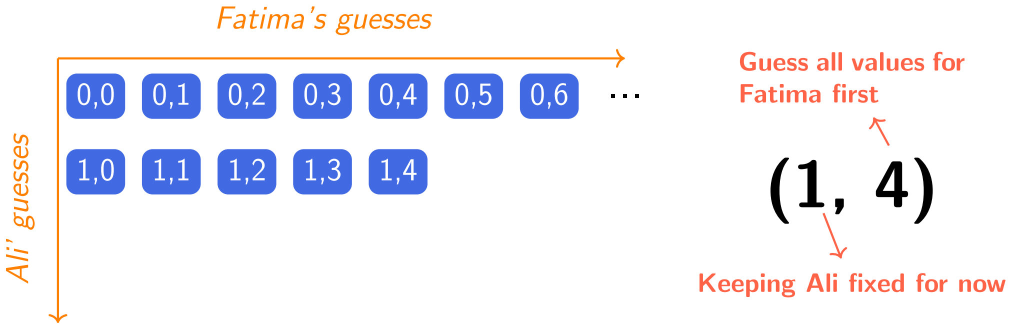
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

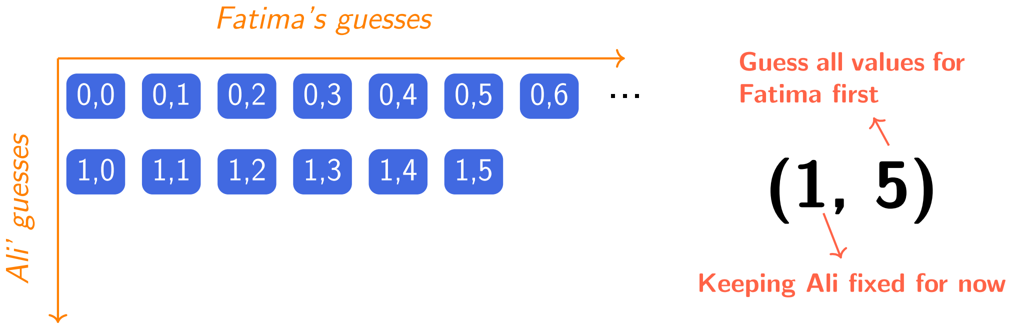
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

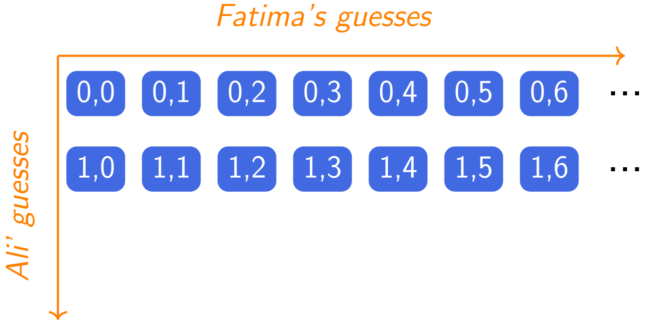
Generate all possible pairs of Ali and Fatima's tickets:

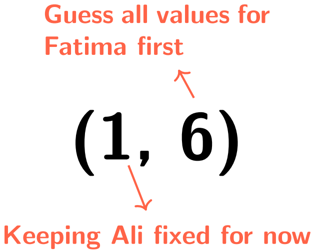


Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

Generate all possible pairs of Ali and Fatima's tickets:

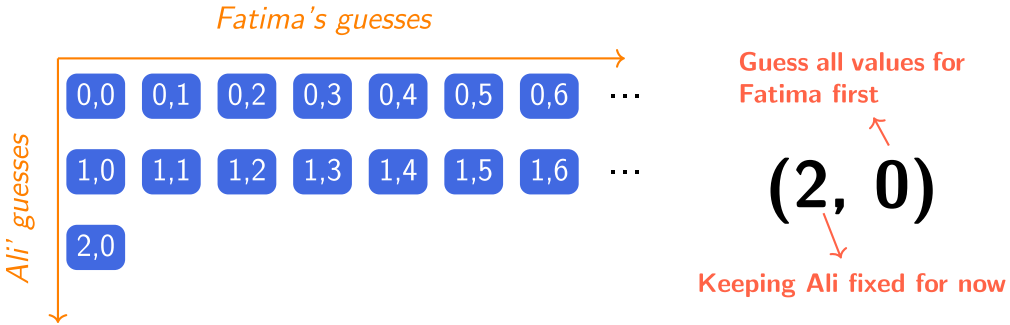




Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

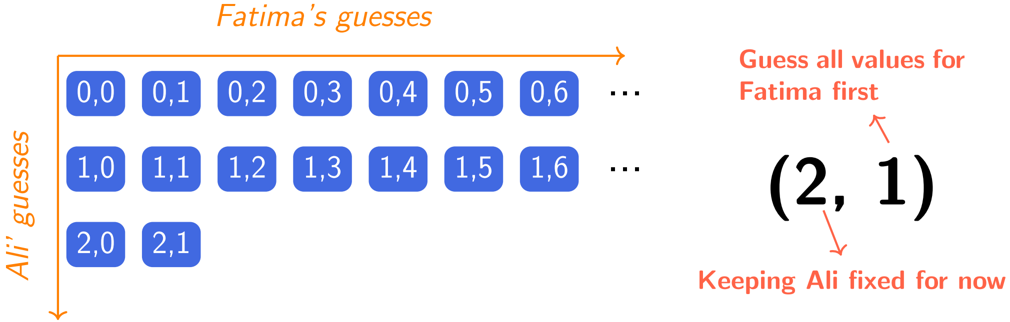
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

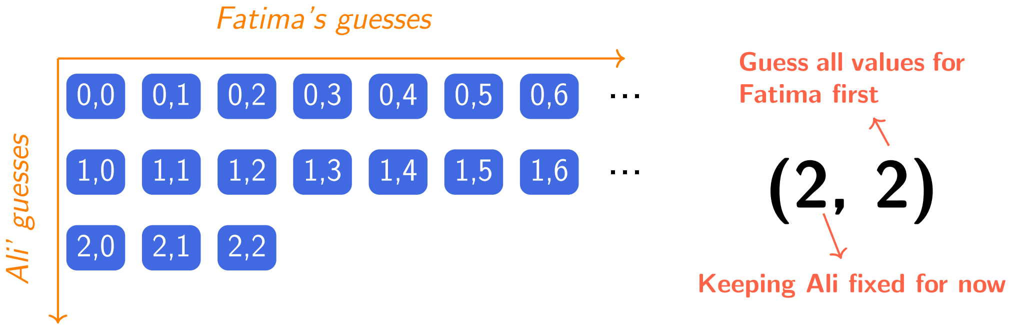
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

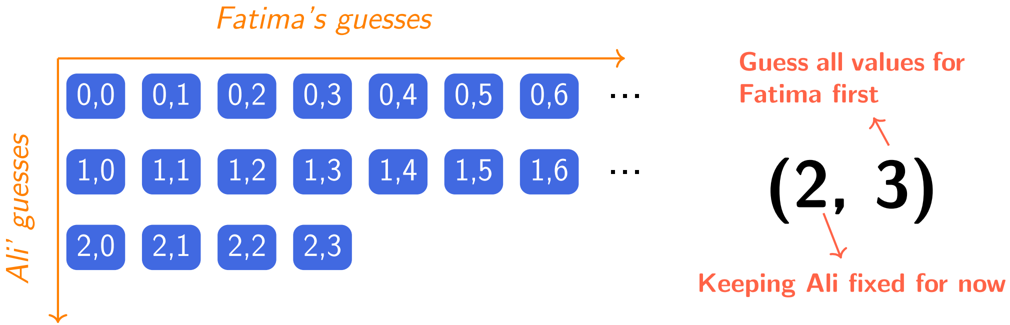
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

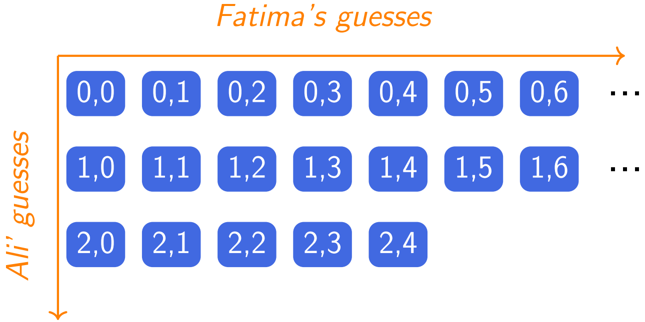
Generate all possible pairs of Ali and Fatima's tickets:

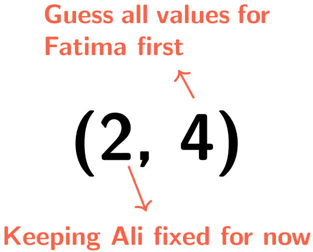


Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

Generate all possible pairs of Ali and Fatima's tickets:

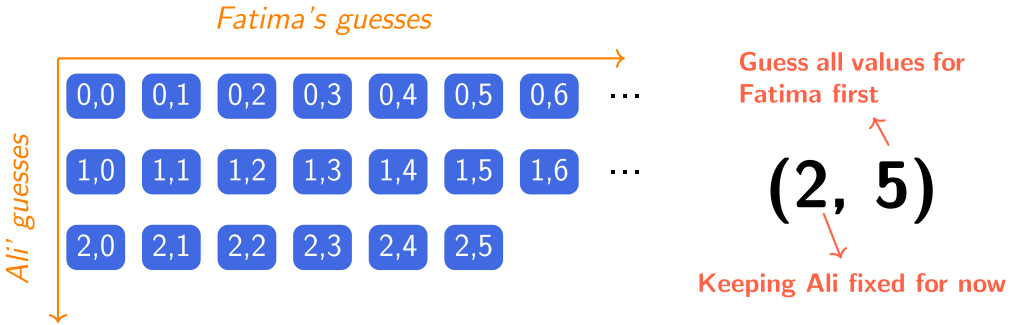




Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

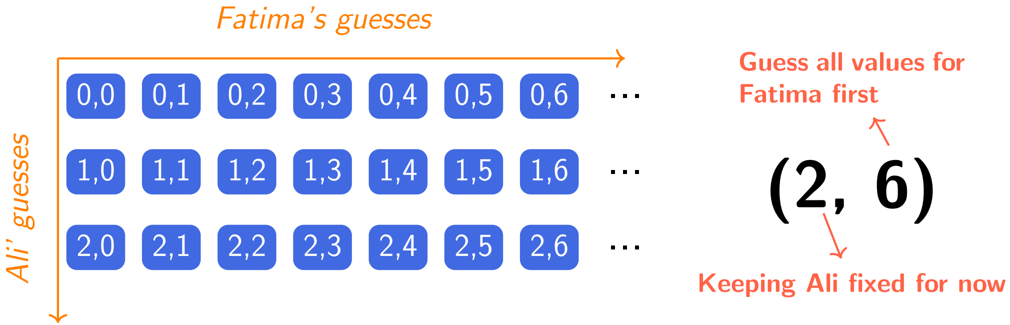
Generate all possible pairs of Ali and Fatima's tickets:



Strategy 2: Systematic Search (*how?*)

Any ideas before we proceed?

Generate all possible pairs of Ali and Fatima's tickets:



Generating Pairs *(Keeping Ali fixed)*

```
ali = 0
for fatima in range(18):
    print(f'{ali},{fatima}')
```

0,0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 ...

Generating Pairs *(Keeping Ali fixed)*

```
ali = 1
for fatima in range(18):
    print(f'{ali},{fatima}')
```

1,0 1,1 1,2 1,3 1,4 1,5 1,6 1,7 ⋯

Generating Pairs *(Keeping Ali fixed)*

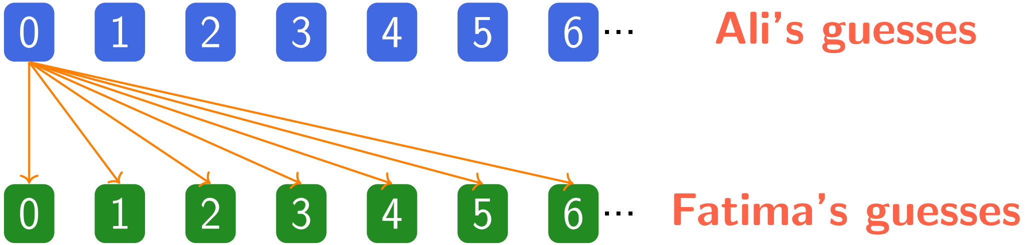
```
ali = 2
for fatima in range(18):
    print(f'{ali},{fatima}')
```

2,0 2,1 2,2 2,3 2,4 2,5 2,6 2,7 ...

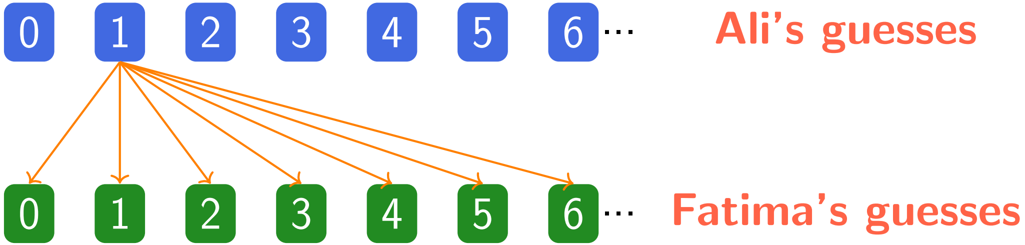
Generating Pairs *(Putting Ali in a loop)*

```
for ali in range(18):
    # for each ali's guess, generate
    #   all possible fatima's guesses
```

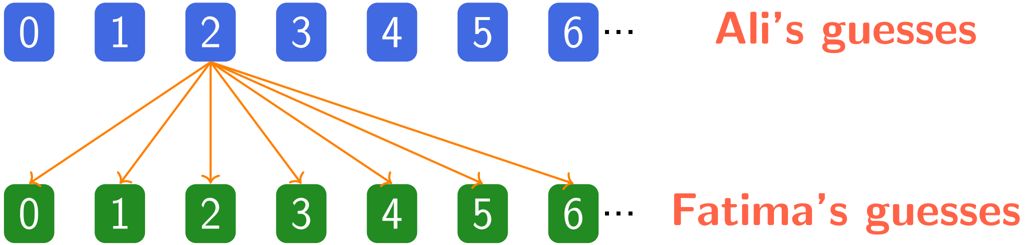
For each of Ali's guess, we try all possible Fatima's guesses:



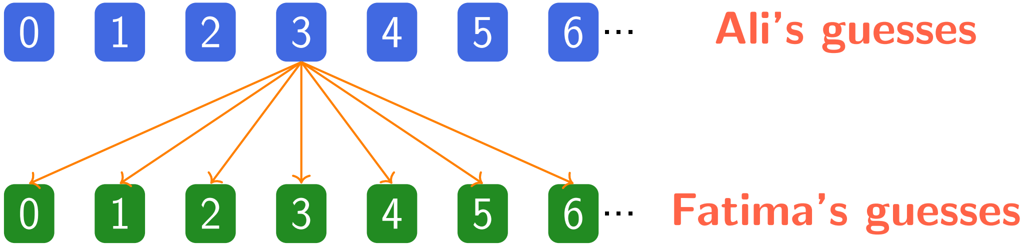
For each of Ali's guess, we try all possible Fatima's guesses:



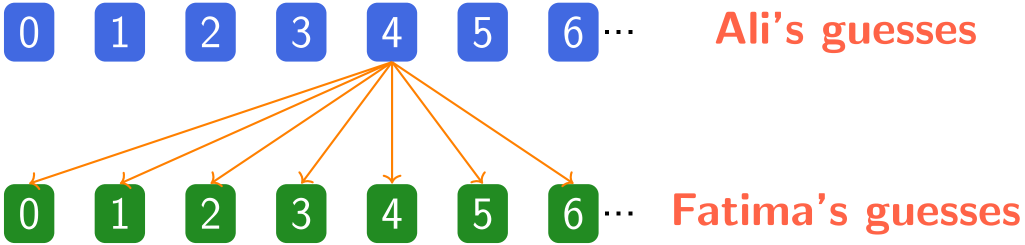
For each of Ali's guess, we try all possible Fatima's guesses:



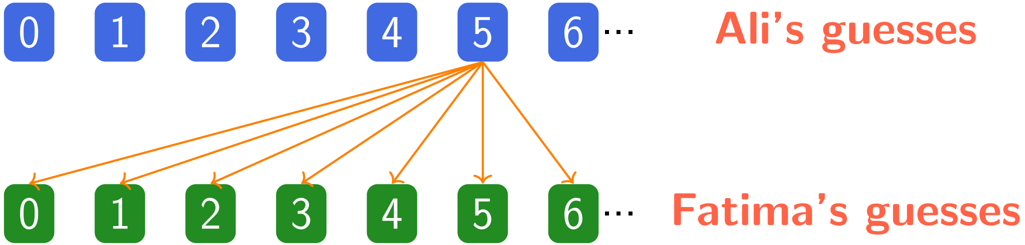
For each of Ali's guess, we try all possible Fatima's guesses:



For each of Ali's guess, we try all possible Fatima's guesses:



For each of Ali's guess, we try all possible Fatima's guesses:



Generating Pairs *(Putting Ali in a loop)*

```
for ali in range(18):
    for fatima in range(18):
        print(f'{ali},{fatima}')
```

0,0	0,1	0,2	0,3	0,4	0,5	⋯	0,17
1,0	1,1	1,2	1,3	1,4	1,5	⋯	1,17
2,0	2,1	2,2	2,3	2,4	2,5	⋯	2,17

Generating Pairs *(Putting Ali in a loop)*

```
for ali in range(18):
    for fatima in range(18):
        print(f'{ali},{fatima}')
```

0,0	0,1	0,2	0,3	0,4	0,5	⋯	0,17
1,0	1,1	1,2	1,3	1,4	1,5	⋯	1,17
2,0	2,1	2,2	2,3	2,4	2,5	⋯	2,17

Nested Loops

When there is a lot of work for a single iteration

Back to our solution

```
for ali in range(18):
    for fatima in range(18):
        if ali + fatima == 17 and ali - fatima == 3:
            print(f"Ali sold {ali} tickets")
```

Back to our solution

```
for ali in range(18):
    for fatima in range(18):
        if ali + fatima == 17 and ali - fatima == 3:
            print(f"Ali sold {ali} tickets")
```

No output when solution is not found. *(fix?)*

Try on PythonTutor!

Back to our solution

```python
aliTickets = 0
found = False
for aliGuess in range(18):
  for fatimaGuess in range(18):
    if aliGuess+fatimaGuess==17 and aliGuess-fatimaGuess==3:
      aliTickets = aliGuess
      found = True
if found:
  print(f"Ali sold {aliTickets} tickets")
else:
  print("No solution found")
```

Back to our solution

```
aliTickets = 0
found = False
for aliGuess in range(18):
  for fatimaGuess in range(18):
    if aliGuess+fatimaGuess==17 and aliGuess-fatimaGuess==3:
      aliTickets = aliGuess
      found = True
if found:
  print(f"Ali sold {aliTickets} tickets")
else:
  print("No solution found")
```

Shows output even when solution is not found. *(used boolean flags here)*

Searching Example

Here's another word problem you've probably seen before:

- Ali, Zain, and Fatima are selling tickets for a charity event
- Zain sells 2 fewer than Ali
- Fatima sells twice as many as Ali
- 10 total tickets were sold by the three people
- How many did Ali sell?

Searching Example

Here's another word problem you've probably seen before:

- Ali, Zain, and Fatima are selling tickets for a charity event
- Zain sells 2 fewer than Ali
- Fatima sells twice as many as Ali
- 10 total tickets were sold by the three people
- How many did Ali sell?

But instead of two numbers to guess, we have **three**.

Solution:

3 nested loops, represents each person's guess.

```
for ali in range(11):
    for zain in range(11):
        for fatima in range(11):
            total = (ali + zain + fatima == 10)
            two_less = (zain == ali-2)
            twice = (fatima == 2*ali)
            if total and two_less and twice:
                print(f"Ali sold {ali} tickets")
                print(f"Zain sold {zain} tickets")
                print(f"Fatima sold {fatima} tickets")
```

Solution:

3 nested loops, represents each person's guess.

```
for ali in range(11):
    for zain in range(11):
        for fatima in range(11):
            total = (ali + zain + fatima == 10)
            two_less = (zain == ali-2)
            twice = (fatima == 2*ali)
            if total and two_less and twice:
                print(f"Ali sold {ali} tickets")
                print(f"Zain sold {zain} tickets")
                print(f"Fatima sold {fatima} tickets")
```

3 booleans for our word problem equations

Solution:

3 nested loops, represents each person's guess.

```
for ali in range(11):
    for zain in range(11):
        for fatima in range(11):
            total = (ali + zain + fatima == 10)
            two_less = (zain == ali-2)
            twice = (fatima == 2*ali)
            if total and two_less and twice:
                print(f"Ali sold {ali} tickets")
                print(f"Zain sold {zain} tickets")
                print(f"Fatima sold {fatima} tickets")
```

3 booleans for our word problem equations

Solution found when all three are True

Solution:

3 nested loops, represents each person's guess.

```
for ali in range(11):
    for zain in range(11):
        for fatima in range(11):
            total = (ali + zain + fatima == 10)
            two_less = (zain == ali-2)
            twice = (fatima == 2*ali)
            if total and two_less and twice:
                print(f"Ali sold {ali} tickets")
                print(f"Zain sold {zain} tickets")
                print(f"Fatima sold {fatima} tickets")
```

3 booleans for our word problem equations

Solution found when all three are True

Try on PythonTutor!

So Far...

Our search methods have allowed us to find **integer** solutions

Sometimes, there are no integer solutions

Approximation Methods

Consider:

```
x = 0
for i in range(10):
    x += 0.1
print(x == 1)
print(x, '==', 10*0.1)
```

Consider:

```
x = 0
for i in range(10):
    x += 0.1
print(x == 1)
print(x, '==', 10*0.1)
```

False

Consider:

```
x = 0
for i in range(10):
    x += 0.1
print(x == 1)
print(x, '==', 10*0.1)
```

False

0.9999999999999999 == 1.0

You Try!

Type the following in your interpreter window:

- `0.1 + 0.1 + 0.1`
- `0.1 + 0.1 + 0.1 == 0.3`

Big Idea

Some decimal numbers can't be represented exactly in binary

Each time you do a calculation, you introduce a small error

Surprizing Results!

```
x = 0
for i in range(10):
    x += 0.125
print(x == 1.25)
```

True

Surprizing Results!

```python
x = 0
for i in range(10):
    x += 0.125
print(x == 1.25)
```

```python
x = 0
for i in range(10):
    x += 0.1
print(x == 1)
print(x, '==', 10*0.1)
```

True

False

$0.9999999999999999 == 1.0$

Moral of the Story

- **Never** use == to test **floats**
  - Instead test whether they are within small amount of each other

Moral of the Story

- **Never** use $==$ to test **floats**
  - Instead test whether they are within small amount of each other
- What gets **printed** isn't always what is in **memory**
  - e.g: $0.1_{decimal}$ is $0.00011001100110011..._{binary}$, repeating forever

Moral of the Story

- **Never** use == to test **floats**
  - Instead test whether they are within small amount of each other
- What gets **printed** isn't always what is in **memory**
  - e.g: $0.1_{decimal}$ is $0.00011001100110011\ldots_{binary}$, repeating forever
    $0.1_{decimal}$ can never be stored exactly in memory

Moral of the Story

- **Never** use == to test **floats**
  - Instead test whether they are within small amount of each other
- What gets **printed** isn't always what is in **memory**
  - e.g: $0.1_{decimal}$ is $0.00011001100110011\ldots_{binary}$, repeating forever
    $0.1_{decimal}$ can never be stored exactly in memory
    So, $0.1_{decimal}$ is actually stored as $0.10000000149011612_{decimal}$ in memory

Moral of the Story

- **Never** use == to test **floats**
  - Instead test whether they are within small amount of each other
- What gets **printed** isn't always what is in **memory**
  - e.g: $0.1_{decimal}$ is $0.00011001100110011\ldots_{binary}$, repeating forever

    $0.1_{decimal}$ can never be stored exactly in memory

    So, $0.1_{decimal}$ is actually stored as $0.10000000149011612_{decimal}$ in memory
- Need to be **careful** in designing algorithms that use floats

Floats can't be trusted!

```
x = 0
for i in range(10):
    x += 0.125
print(x == 1.25)
```

True

```
x = 0
for i in range(10):
    x += 0.1
print(x == 1)
print(x, '==', 10*0.1)
```

False

0.9999999999999999 == 1.0

Floats can't be trusted!

Infact following decimal numbers can't be represented exactly in binary:

```
0.1,  0.2,  0.3,
0.4,  0.5,  0.6,
0.7,  0.8,  0.9,
...,
```

Floats can't be trusted!

Infact following decimal numbers can't be represented exactly in binary:

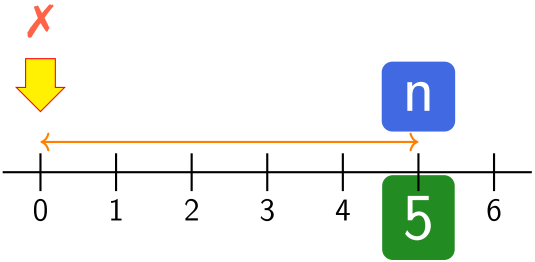
```
0.1,  0.2,  0.3,
0.4,  0.5,  0.6,
0.7,  0.8,  0.9,
...,
```

For example:

```
>>> 0.1 + 0.2 == 0.3
False
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.3 * 3
0.8999999999999999
```

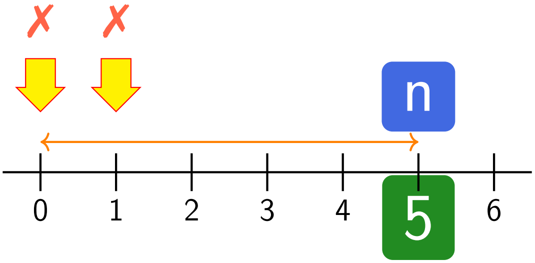
Systematic search: Attempt 1

Let's try to find $\sqrt{5}$.



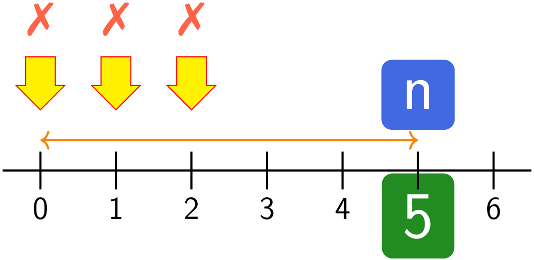
Systematic search: Attempt 1

Let's try to find $\sqrt{5}$.



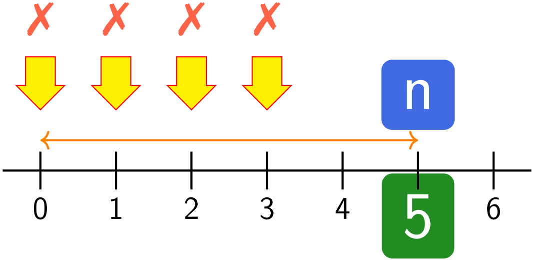
Systematic search: Attempt 1

Let's try to find $\sqrt{5}$.



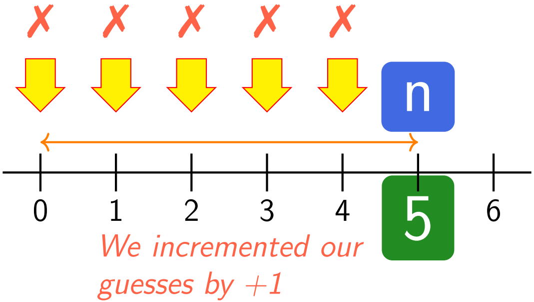
Systematic search: Attempt 1

Let's try to find $\sqrt{5}$.



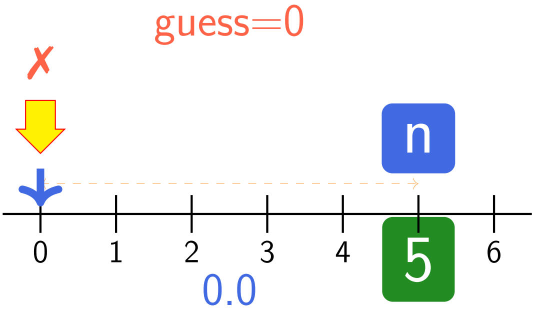
Systematic search: Attempt 1

Let's try to find $\sqrt{5}$.



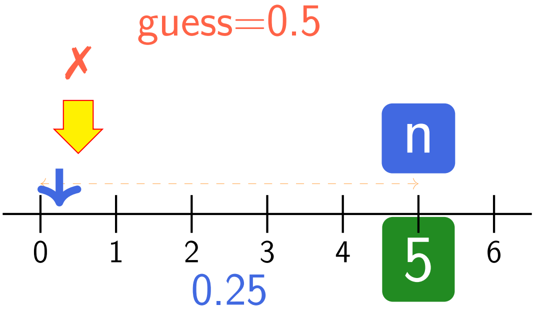
Systematic search: Attempt 2

Decrease increment to **0.5**



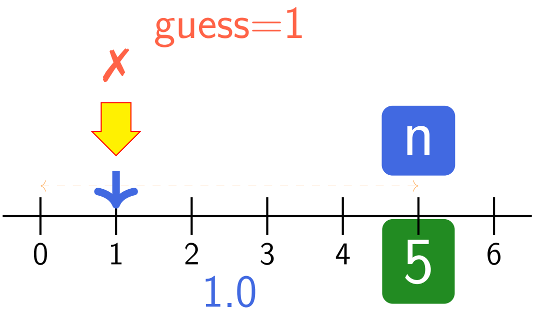
Systematic search: Attempt 2

Decrease increment to **0.5**



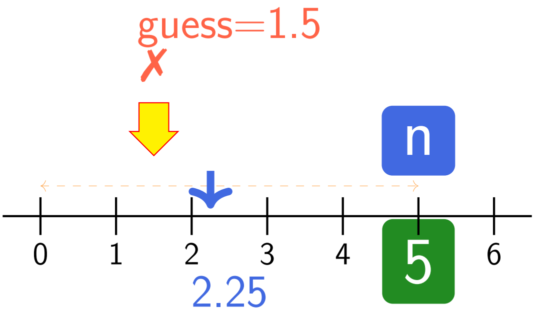
Systematic search: Attempt 2

Decrease increment to 0.5



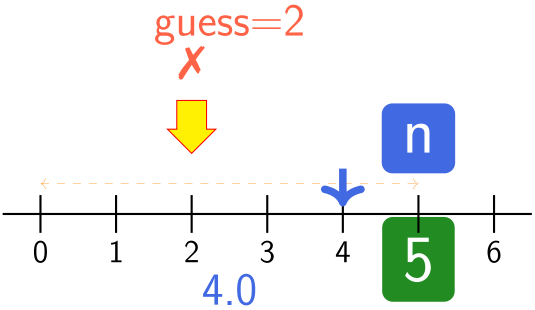
Systematic search: Attempt 2

Decrease increment to 0.5



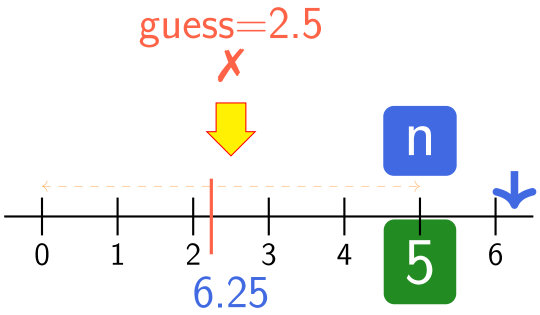
Systematic search: Attempt 2

Decrease increment to **0.5**



Systematic search: Attempt 2

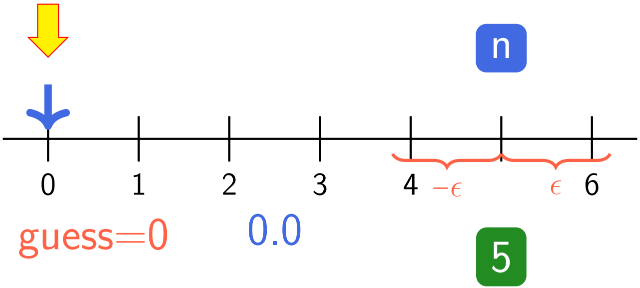
Decrease increment to 0.5



Just missed!

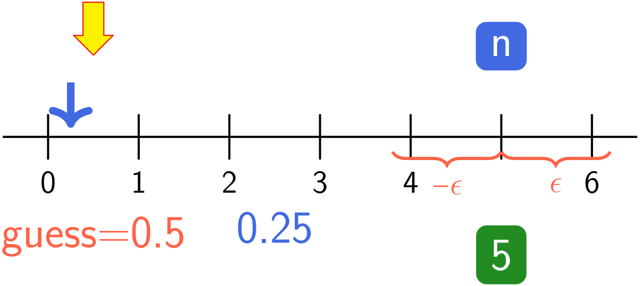
Approximations

- You will **never** get an exact answer using systematic search.
- Idea is to get a "**good enough**" answer.
- Try to hit between $-\epsilon \leq n \leq \epsilon$



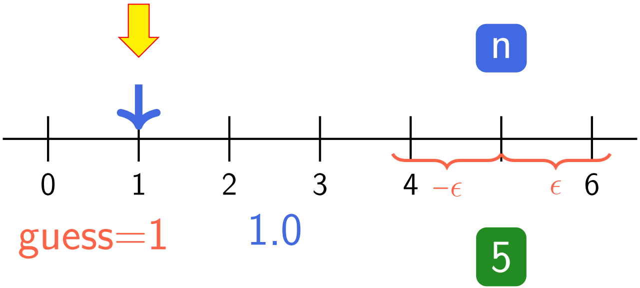
Approximations

- You will **never** get an exact answer using systematic search.
- Idea is to get a "**good enough**" answer.
- Try to hit between $-\epsilon \leq n \leq \epsilon$



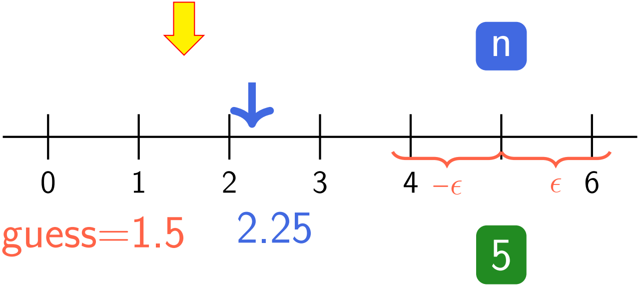
Approximations

- You will **never** get an exact answer using systematic search.
- Idea is to get a "**good enough**" answer.
- Try to hit between $-\epsilon \leq n \leq \epsilon$



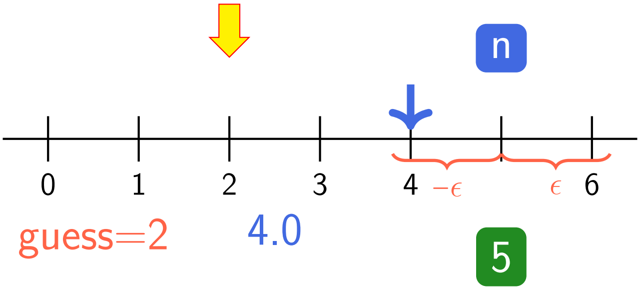
Approximations

- You will **never** get an exact answer using systematic search.
- Idea is to get a "**good enough**" answer.
- Try to hit between $-\epsilon \leq n \leq \epsilon$

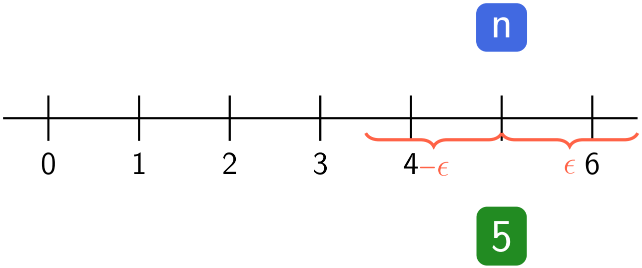


Approximations

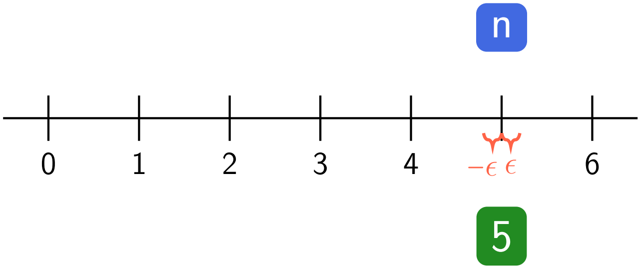
- You will **never** get an exact answer using systematic search.
- Idea is to get a "**good enough**" answer.
- Try to hit between $-\epsilon \leq n \leq \epsilon$



What happens if we **increase** ϵ? **accuracy decreases**

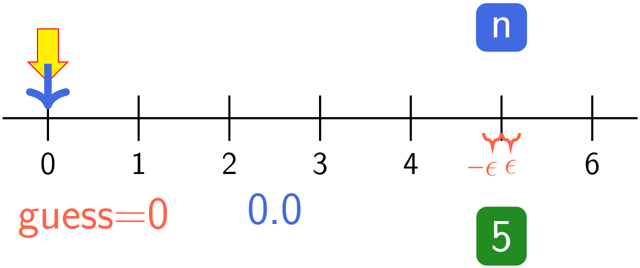


What happens if we decrease ϵ? accuracy increases



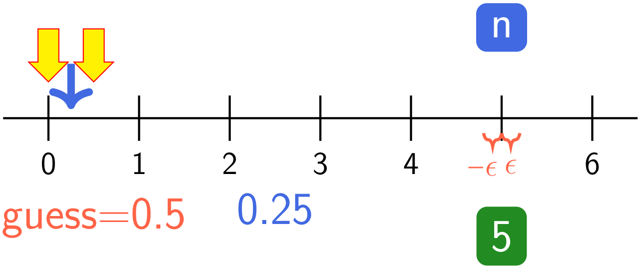
What happens if we **decrease** ϵ? **accuracy increases**

Let's try with increment of **0.5**



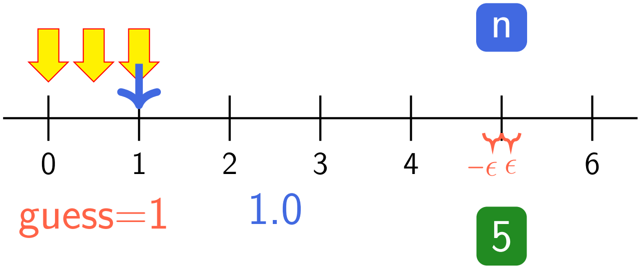
What happens if we **decrease** ϵ? **accuracy increases**

Let's try with increment of **0.5**



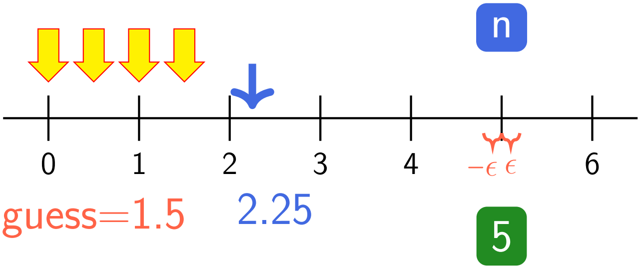
What happens if we **decrease** ϵ? **accuracy increases**

Let's try with increment of **0.5**



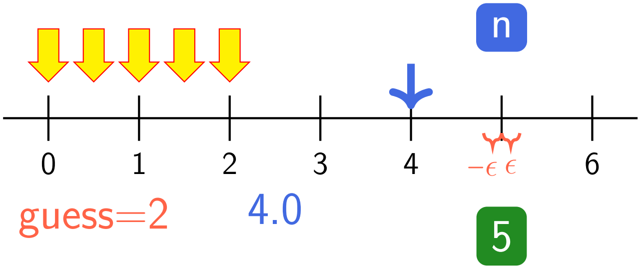
What happens if we **decrease** ϵ? **accuracy increases**

Let's try with increment of **0.5**



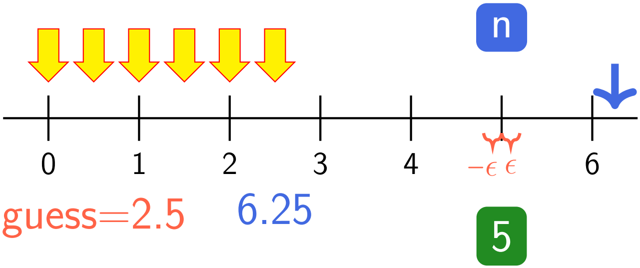
What happens if we **decrease** ϵ? **accuracy increases**

Let's try with increment of **0.5**



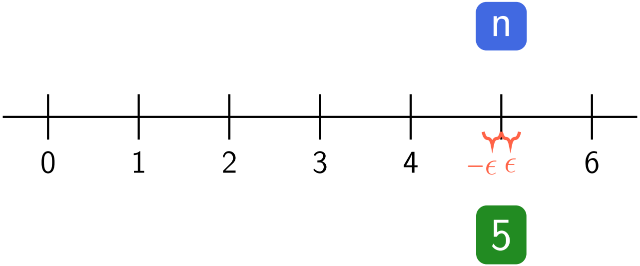
What happens if we **decrease** ϵ? **accuracy increases**

Let's try with increment of **0.5**



Just missed the target!

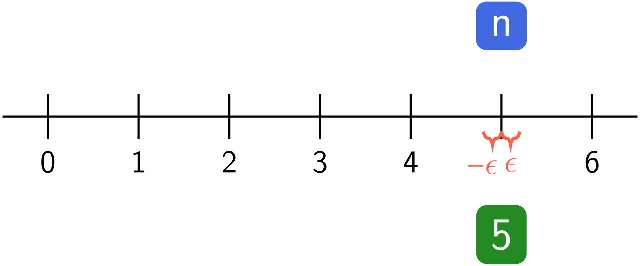
What happens if we decrease ϵ? accuracy increases



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

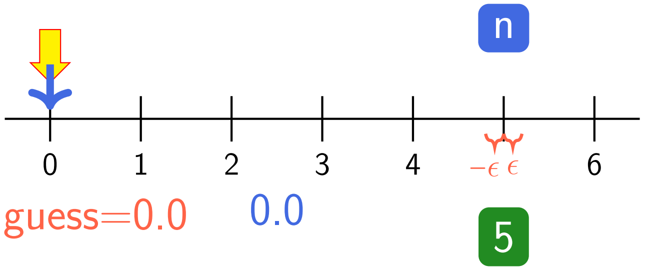
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

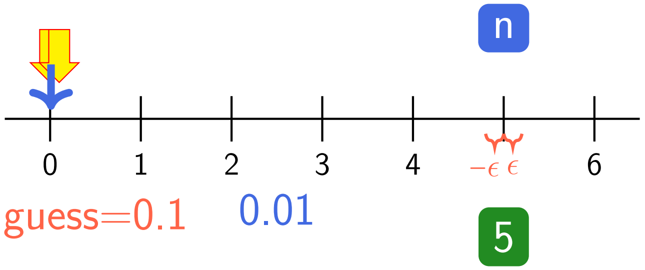
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

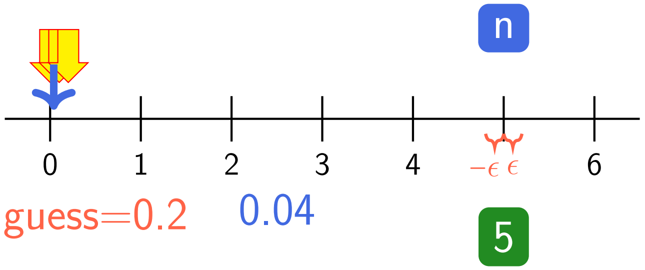
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

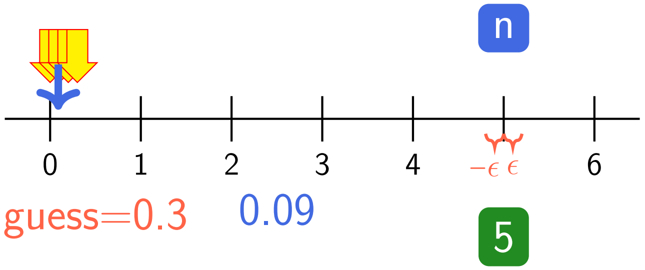
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

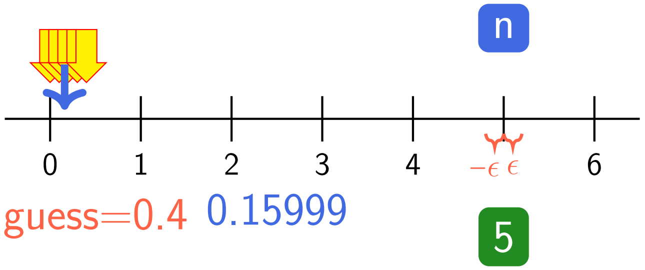
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

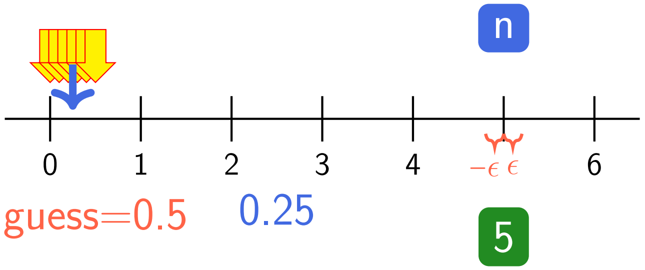
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

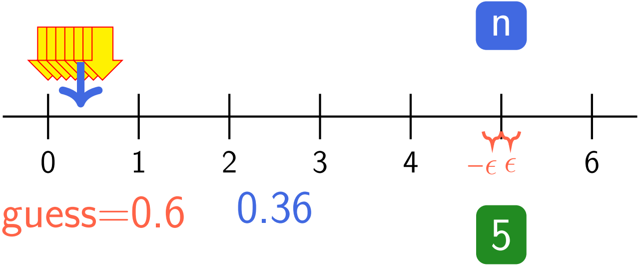
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

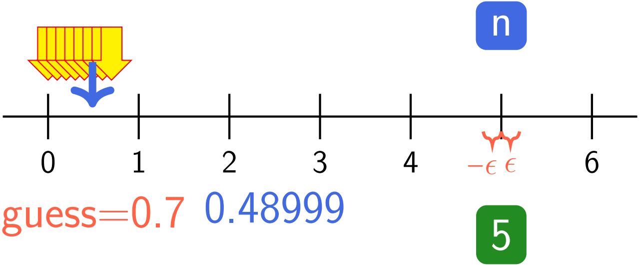
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

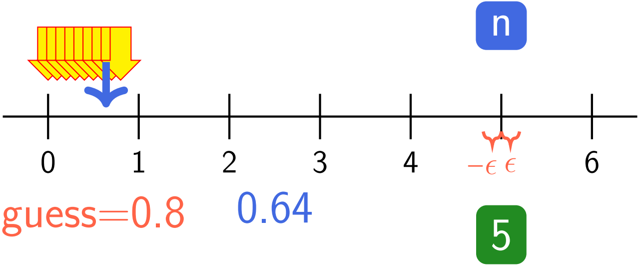
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

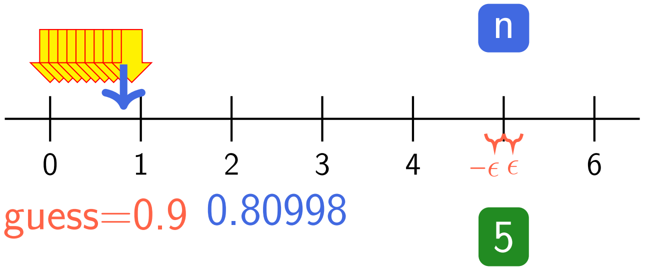
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

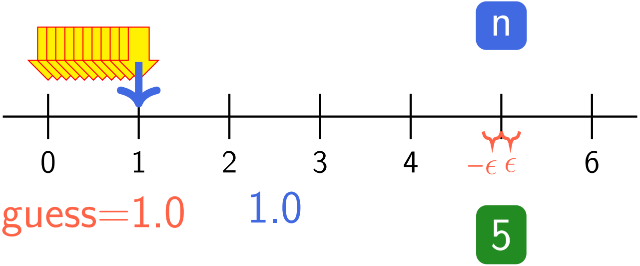
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

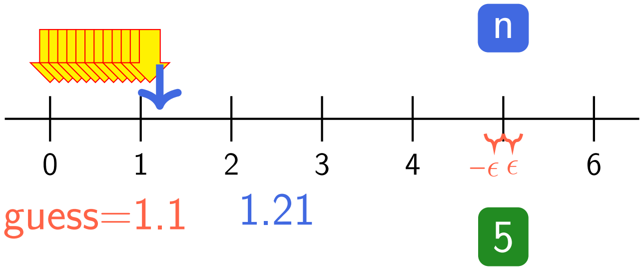
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

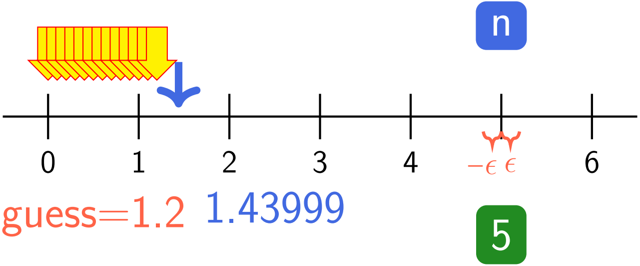
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

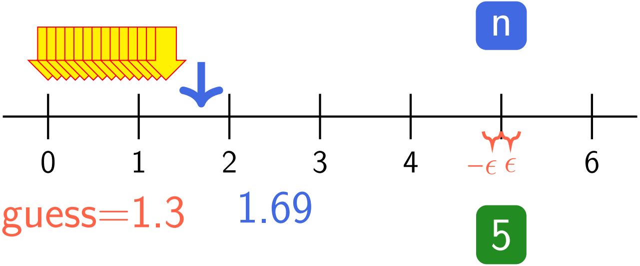
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

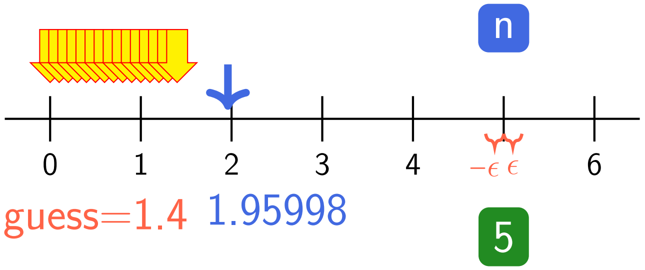
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

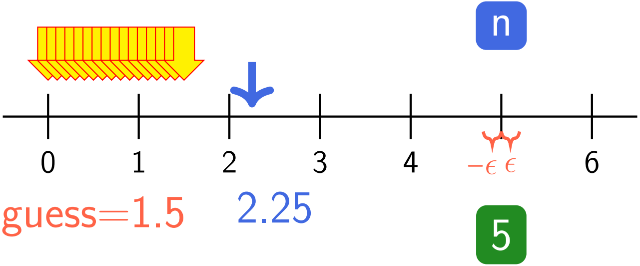
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

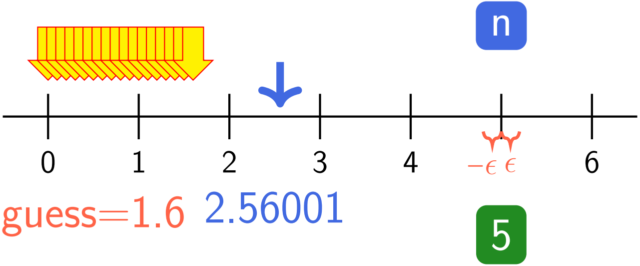
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

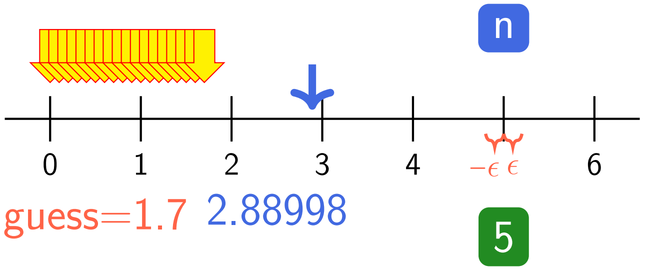
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

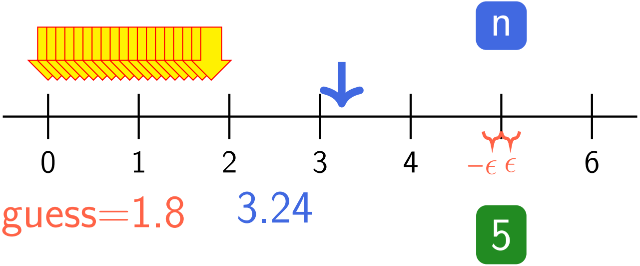
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

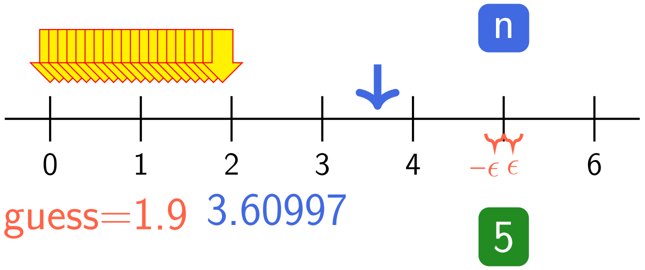
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

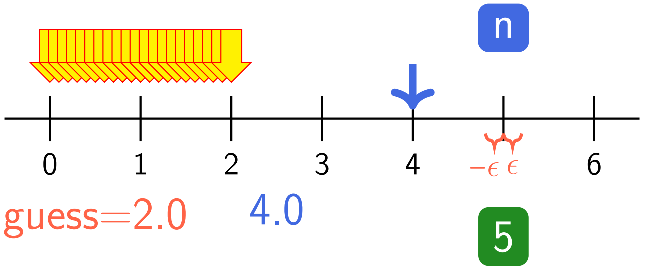
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

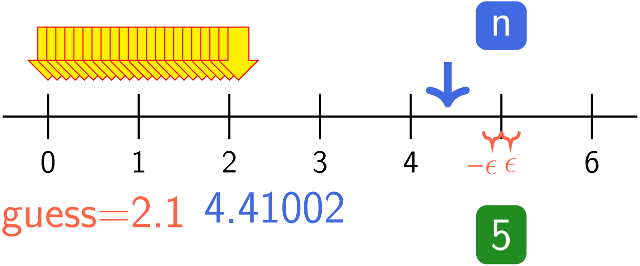
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

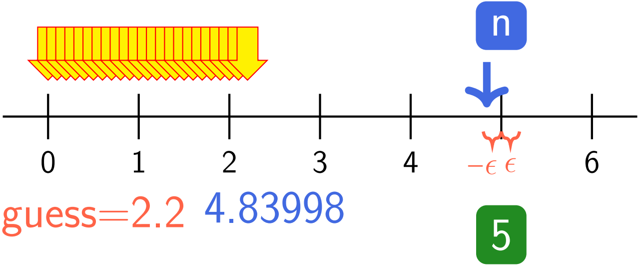
- decreasing the **increment** mean a lot more guesses



What happens if we **decrease** ϵ? **accuracy increases**

With smaller ϵ, we need to **decrease** the **increment**

- decreasing the **increment** mean a lot more guesses



```python
n = int(input())

found = False
increment = 0.001
epsilon = 0.01
guess = 0
while guess**2 < n:
    if (guess**2 >= n-epsilon) and (guess**2 <= n+epsilon):
        found = True
        break
    guess += increment

if found:
    print(f"Square root of {n} is {guess}")
else:
    print(f"Couldn't find square root of {n}")
```

```python
n = int(input())

found = False
increment = 0.001
epsilon = 0.01
guess = 0
while guess**2 < n:
    if (guess**2 >= n-epsilon) and (guess**2 <= n+epsilon):
        found = True
        break
    guess += increment

if found:
    print(f"Square root of {n} is {guess}")
else:
    print(f"Couldn't find square root of {n}")
```

Try on PythonTutor

Big Idea

Approximation is like systematic search for integers except...

1) We increment by some small amount
2) We stop when close enough (exact is not possible)

- Recall the systematic search for integers.
- Suppose Ali thinks of a number between 1 and $10,000$

```
n = int(input('Enter number:'))
guess = 0
while guess < 10000:
    if guess == n:
        break
    guess += 1
print(guess)
```

- Recall the systematic search for integers.
- Suppose Ali thinks of a number between 1 and $10,000$

```
n = int(input('Enter number:'))
guess = 0
while guess < 10000:
    if guess == n:
        break
    guess += 1
print(guess)
```

Ali enters 9999

How many tries till we find the number?

- Count the number of tries till we find answer.

```python
n = int(input('Enter number:'))
guess = 0
tries = 0
while guess < 10000:
    if guess == n:
        break
    guess += 1
    tries += 1
print(f'You guesses {guess} in {tries} tries')
```

- Count the number of tries till we find $\sqrt{5}$.

```
n = int(input())
found = False
increment = 0.0000001
epsilon = 0.000001
guess = 0
tries = 0
while guess**2 < n:
    if (guess**2 >= n-epsilon) and (guess**2 <= n+epsilon):
        found = True
        break
    guess += increment
    tries += 1

if found:
    print(f"Square root is {guess} in {tries} tries")
```

- Count the number of tries till we find $\sqrt{5}$.

```python
n = int(input())
found = False
increment = 0.0000001
epsilon = 0.000001
guess = 0
tries = 0
while guess**2 < n:
    if (guess**2 >= n-epsilon) and (guess**2 <= n+epsilon):
        found = True
        break
    guess += increment
    tries += 1

if found:
    print(f"Square root is {guess} in {tries} tries")
```

It took 4.5 seconds and **22 million tries** to find $\sqrt{5}$ to six decimal places

2.2360677~~999476932~~

- Count the number of tries till we find $\sqrt{5}$.

```
n = int(input())
found = False
increment = 0.0000001
epsilon = 0.000001
guess = 0
tries = 0
while guess**2 < n:
    if (guess**2 >= n-epsilon) and (guess**2 <= n+epsilon):
        found = True
        break
    guess += increment
    tries += 1

if found:
    print(f"Square root is {guess} in {tries} tries")
```

It took 4.5 seconds and **22 million tries** to find $\sqrt{5}$ to six decimal places

Can we Improve?

2.236067~~7999476932~~

Bisection Search

Bisection Search

Bisection search takes advantage of properties of the problem.

- The search space has an order
- We can tell whether the guess was too low or too high

You Try!

Ask your partner to think of a 3 digit pin code. They only tell you **YES** / **NO** whether your guess is correct or not. Can you use bisection search to quickly and correctly guess the code?

How many tries did it take you to guess the code?

You Try!

Ask your partner to think of a 3 digit pin code. They only tell you **YES** / **NO** whether your guess is correct or not. Can you use bisection search to quickly and correctly guess the code?

How many tries did it take you to guess the code?

No, you CAN NOT use bisection search here. Why?

You Try!

Ask your partner to think of a 3 digit pin code. Now they tell you **LARGER** if your guess was larger than their pin or **SMALLER**. Can you use bisection search to quicklyand correctly guess the code?

How many tries did it take you to guess the code?

You Try!

You are playing an EXTREME guessing game to guess a number **EXACTLY**. A friend has a decimal number between 0 and 10 *(4 decimal digits)* in mind. The feedback on your guess is whether it is correct, too high, or too low. Can you use bisection search to quickly and correctly guess the number?

How many tries did it take you to guess the number?

You Try!

You are playing an EXTREME guessing game to guess a number **EXACTLY**. A friend has a decimal number between 0 and 10 *(4 decimal digits)* in mind. The feedback on your guess is whether it is correct, too high, or too low. Can you use bisection search to quickly and correctly guess the number?

How many tries did it take you to guess the number?

Can you use linear search on this problem?

Sequential Search vs Bisection Search

- Sequential search:
  - Search range is always fixed
  - `next_guess = prev_guess + increment`
  - May take a long time to find answer

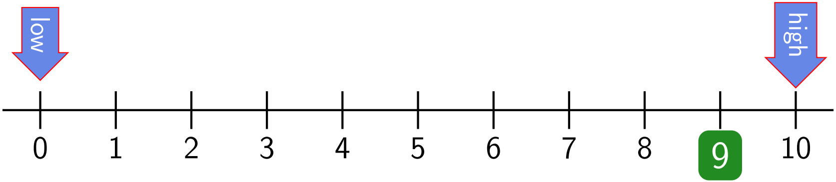
Sequential Search vs Bisection Search

- Sequential search:
  - Search range is always fixed
  - `next_guess = prev_guess + increment`
  - May take a long time to find answer
- Bisection search:
  - Search range shrinks as we get closer to answer
  - `next_guess = (low + high) / 2`
  - Ridiculously Faster than sequential search

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 0
high = 10
guess = ?
```

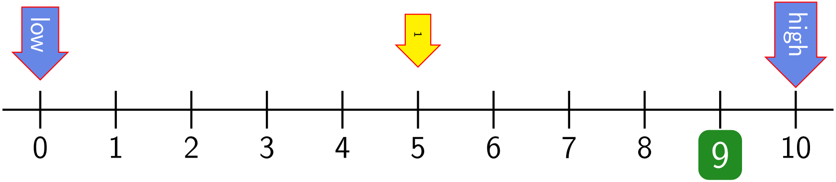


Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

Guess is **exactly** in the **middle** of **low** and **high**.
Hence the name **Bisection Search**.

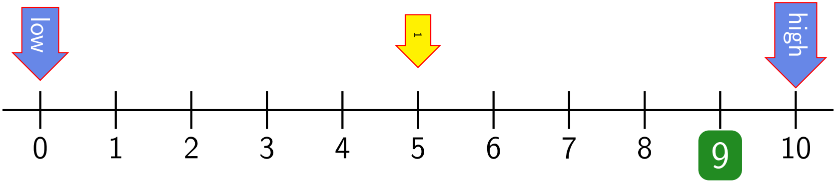
low = 0
high = 10
guess = $\lfloor (0+10)/2 \rfloor$ = 5



Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 0
high = 10
```
guess = $\lfloor (0+10)/2 \rfloor$ = 5

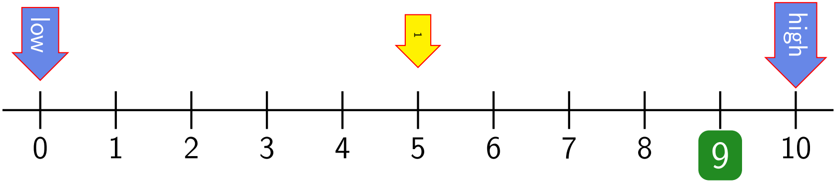


n == guess? **NO**

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 0
high = 10
```
guess = $\lfloor (0+10)/2 \rfloor$ = 5

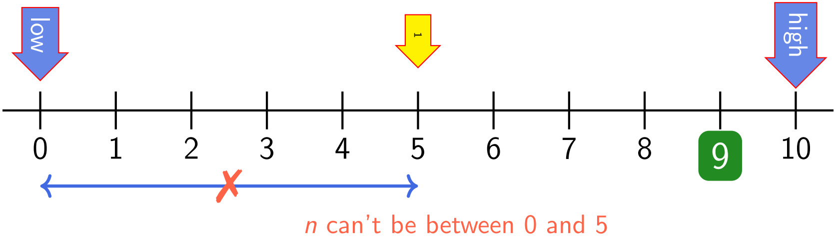


n > guess? **YES**

Bisection Search

Ali guesses a number between 1 and 10 (*say* **9**).

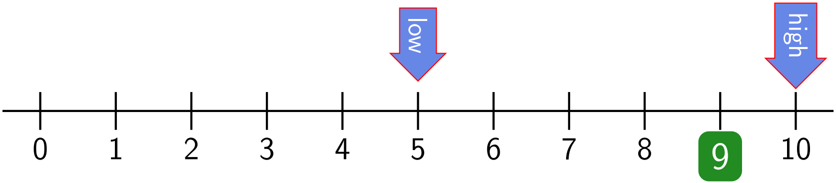
```
low = 0
high = 10
```
guess = $\lfloor (0+10)/2 \rfloor$ = 5



Bisection Search

Ali guesses a number between 1 and 10 (*say* **9**).

```
low = 5
high = 10
```
guess = $\lfloor (0+10)/2 \rfloor$ = 5



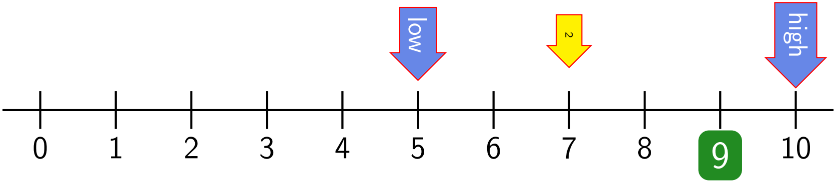
update the lower end to **guess**

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 5
high = 10
```

guess = $\lfloor (5+10)/2 \rfloor$ = 7

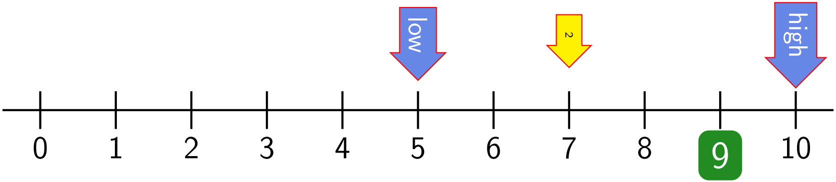


calculate **new guess**

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 5
high = 10
```
guess = $\lfloor (5 + 10)/2 \rfloor$ = 7



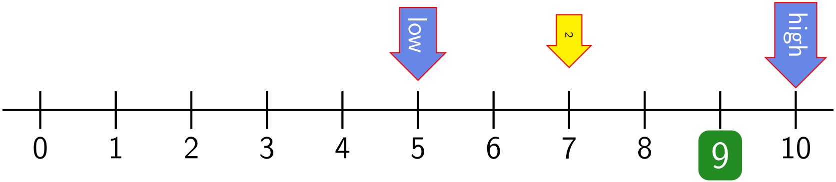
n == guess? **NO**

Bisection Search

Ali guesses a number between 1 and 10 (*say* **9**).

```
low = 5
high = 10
```

guess = $\lfloor (5 + 10)/2 \rfloor$ = 7

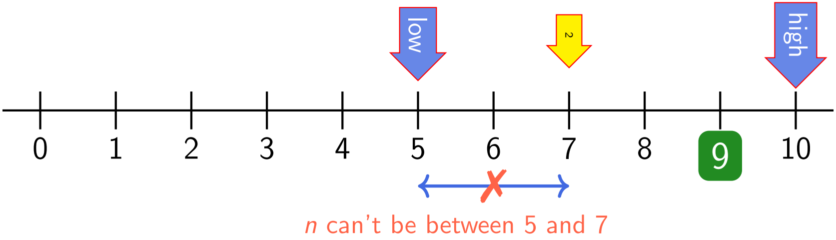


n > guess? **YES**

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

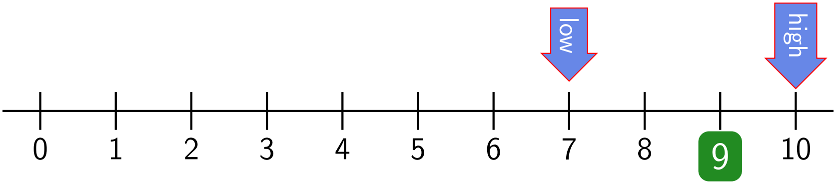
```
low = 5
high = 10
```
guess = $\lfloor (5+10)/2 \rfloor$ = 7



Bisection Search

Ali guesses a number between 1 and 10 (*say* **9**).

```
low = 7
high = 10
```
guess = $\lfloor(5 + 10)/2\rfloor$ = 7



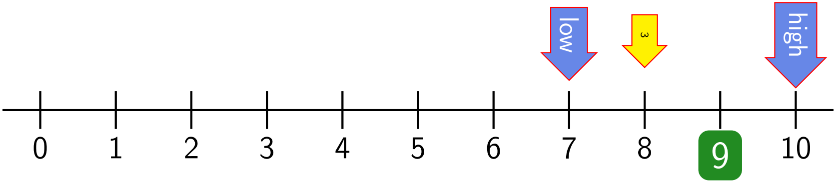
update the lower end to **guess**

Bisection Search

Ali guesses a number between 1 and 10 (*say* **9**).

```
low = 7
high = 10
```

guess = $\lfloor (7+10)/2 \rfloor$ = 8

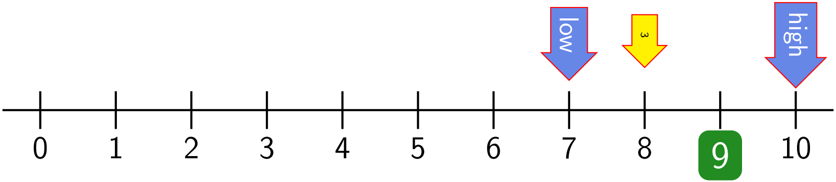


calculate **new guess**

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 7
high = 10
```
guess = $\lfloor (7+10)/2 \rfloor$ = 8

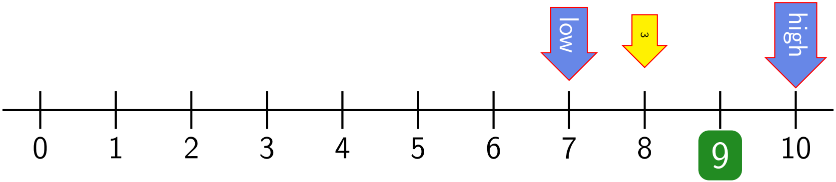


n == guess? **NO**

Bisection Search

Ali guesses a number between 1 and 10 (*say* **9**).

```
low = 7
high = 10
```
guess = $\lfloor (7 + 10)/2 \rfloor$ = 8

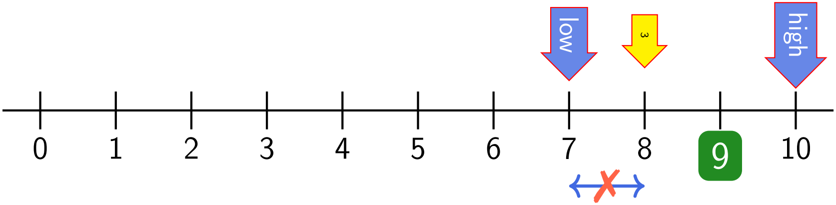


n > guess? **YES**

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 7
high = 10
```
guess = $\lfloor (7 + 10)/2 \rfloor$ = 8

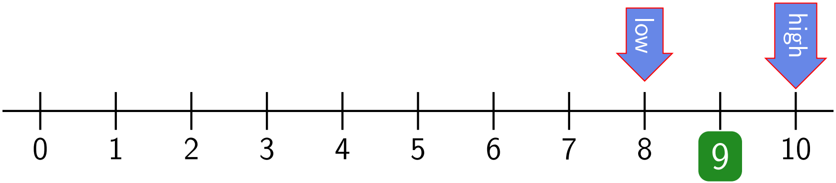


n can't be between 7 and 8

Bisection Search

Ali guesses a number between 1 and 10 (*say* **9**).

```
low = 8
high = 10
```
guess = $\lfloor (7+10)/2 \rfloor$ = 8



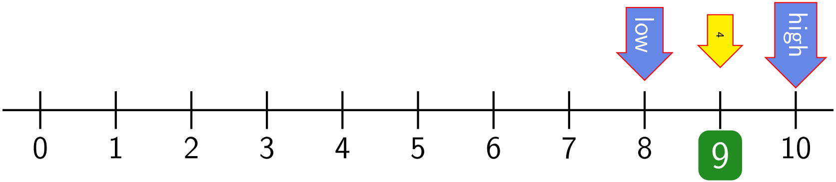
update the lower end to **guess**

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 8
high = 10
```

guess = $\lfloor (8+10)/2 \rfloor$ = 9



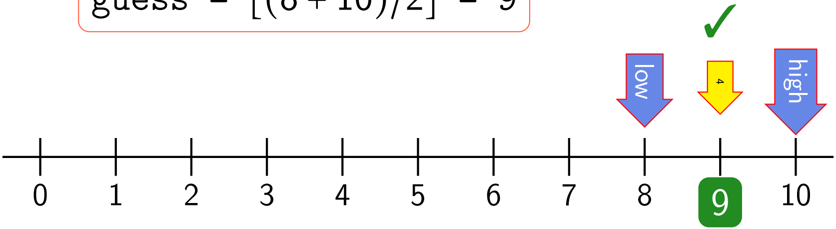
calculate **new guess**

Bisection Search

Ali guesses a number between 1 and 10 *(say* **9***)*.

```
low = 8
high = 10
```

`guess` = $\lfloor (8+10)/2 \rfloor$ = 9

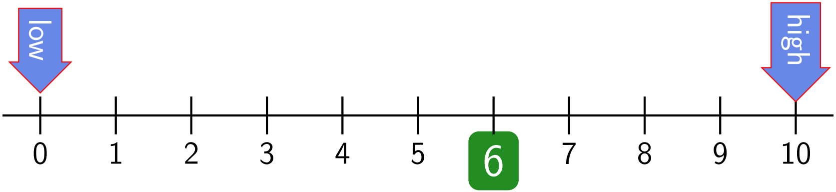


`guess == n?` **YES**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 *(say 6)*.

```
low = 0
high = 10
guess = ?
```

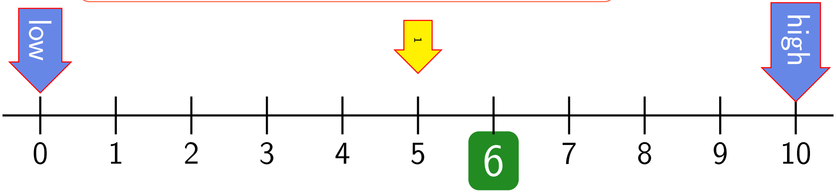


Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 *(say* **6***)*.

```
low = 0
high = 10
```

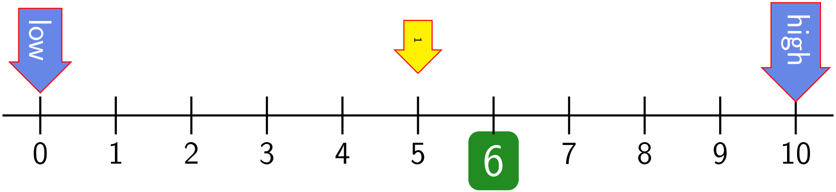
guess = $\lfloor (0+10)/2 \rfloor$ = 5



Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 *(say* **6***)*.

```
low = 0
high = 10
```
guess = $\lfloor (0+10)/2 \rfloor$ = 5



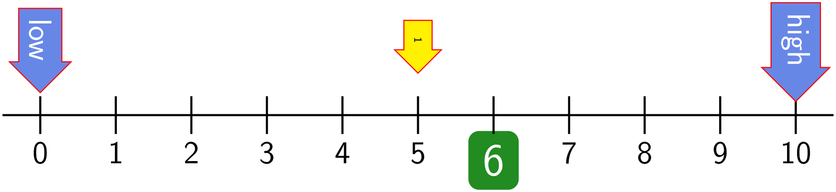
n == guess? **NO**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 (*say* **6**).

```
low = 0
high = 10
```

guess = $\lfloor (0 + 10)/2 \rfloor$ = 5

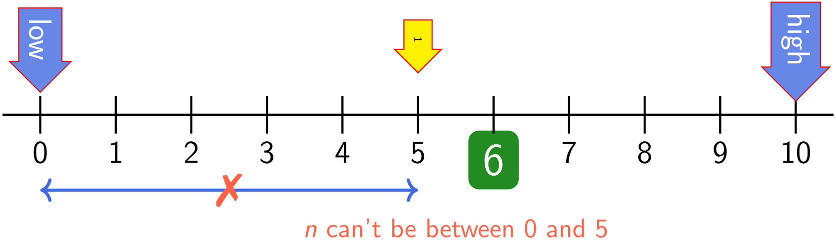


n > guess? **YES**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 *(say* **6***)*.

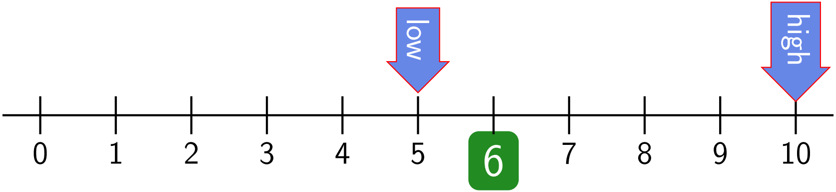
```
low = 0
high = 10
```
guess = $\lfloor (0 + 10)/2 \rfloor$ = 5



Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 (*say* **6**).

```
low = 5
high = 10
```
guess = $\lfloor (0 + 10)/2 \rfloor$ = 5



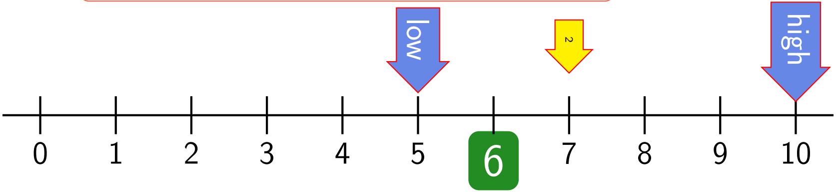
update the lower end to **guess**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 *(say* **6***)*.

```
low = 5
high = 10
```

`guess` = $\lfloor (5+10)/2 \rfloor$ = `7`

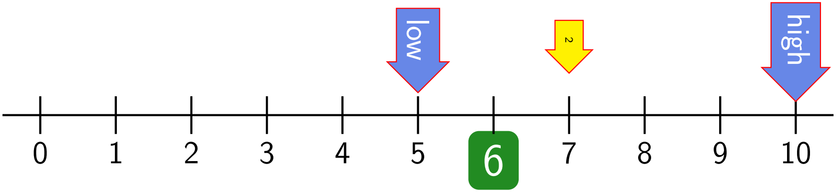


calculate **new guess**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 *(say* **6***)*.

```
low = 5
high = 10
```
guess = $\lfloor (5+10)/2 \rfloor$ = 7

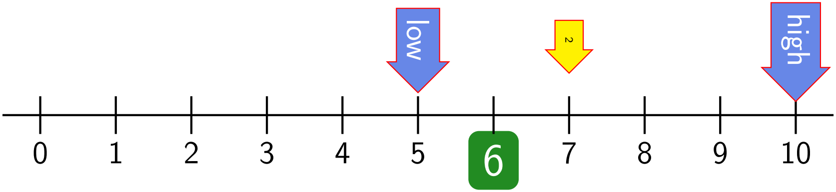


n == guess? **NO**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 (*say* **6**).

```
low = 5
high = 10
```
guess = $\lfloor (5 + 10)/2 \rfloor$ = 7



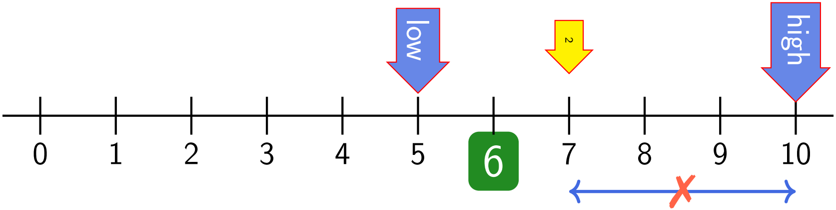
n > guess? **NO**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 *(say* **6***)*.

```
low = 5
high = 10
```

guess = $\lfloor (5+10)/2 \rfloor$ = 7



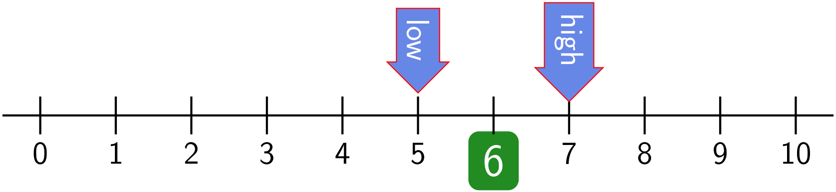
n can't be between 7 and 10

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 *(say **6**)*.

```
low = 5
high = 10
```

guess = $\lfloor (5+10)/2 \rfloor$ = 7



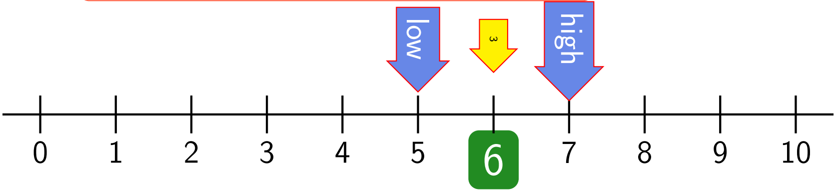
update the upper end to **guess**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 (*say* **6**).

```
low = 5
high = 10
```

guess = $\lfloor (5+7)/2 \rfloor$ = 6



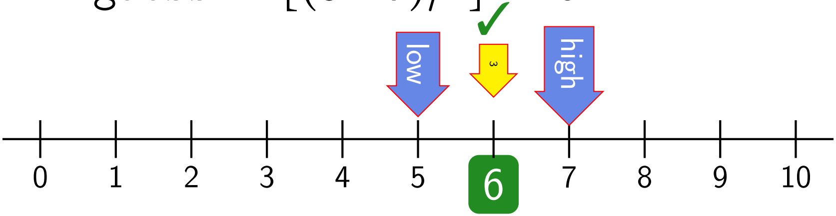
calculate **new guess**

Bisection Search *(example 2)*

Ali guesses a number between 1 and 10 (*say* **6**).

```
low = 5
high = 10
```

guess = $\lfloor (5+7)/2 \rfloor$ = 6



`n == guess?` **YES**

Bisection Search

```python
n = 9999999999 # int(input())
low = 0
high = 10000000000
guess = 
found = False
while            :
    if guess == n:


    elif guess < n:

    else:              # guess > n


if found:
    print(f"Found: {guess}")
else:
    print(f"Not Found")
```

```python
n = 9999999999 # int(input())
low = 0
high = 10000000000
guess = (low + high)//2
found = False
while :
    if guess == n:

    elif guess < n:

    else:            # guess > n

if found:
    print(f"Found: {guess}")
else:
    print(f"Not Found")
```

Bisection Search

```
n = 9999999999 # int(input())
low = 0
high = 10000000000
guess = (low + high)//2
found = False
while low != high:
    if guess == n:


    elif guess < n:

    else:           # guess > n


if found:
    print(f"Found: {guess}")
else:
    print(f"Not Found")
```

Bisection Search

Bisection Search

```
n = 9999999999 # int(input())
low = 0
high = 10000000000
guess = (low + high)//2
found = False
while low != high:
    if guess == n:
        found = True
        break
    elif guess < n:

    else:          # guess > n


if found:
    print(f"Found: {guess}")
else:
    print(f"Not Found")
```

```
n = 9999999999 # int(input())
low = 0
high = 10000000000
guess = (low + high)//2
found = False
while low != high:
    if guess == n:
        found = True
        break
    elif guess < n:
        low = guess
    else:             # guess > n

if found:
    print(f"Found: {guess}")
else:
    print(f"Not Found")
```

Bisection Search

Bisection Search

```
n = 9999999999 # int(input())
low = 0
high = 10000000000
guess = (low + high)//2
found = False
while low != high:
    if guess == n:
        found = True
        break
    elif guess < n:
        low = guess
    else:           # guess > n
        high = guess

if found:
    print(f"Found: {guess}")
else:
    print(f"Not Found")
```

```python
n = 9999999999 # int(input())
low = 0
high = 10000000000
guess = (low + high)//2
found = False
while low != high:
    if guess == n:
        found = True
        break
    elif guess < n:
        low = guess
    else:              # guess > n
        high = guess
    guess = (low + high)//2
if found:
    print(f"Found: {guess}")
else:
    print(f"Not Found")
```

Bisection Search

Guessed in about **33** steps

Slow Square root (**Sequential Search**).

```
n = int(input())
found = False
increment = 0.0000001
epsilon = 0.000001
guess = 0
while guess**2 < n:
    if (guess**2 >= n-epsilon) and (guess**2 <= n+epsilon):
        found = True
        break
    guess += increment

if found:
    print(f"Square root is {guess}")
```

It took 4.5 seconds and **22 million tries** to find $\sqrt{5}$ to **6** decimal places

2.236067~~7999476932~~

Fast Square root (**Bisection Search**).

```
n = int(input("Enter an integer: "))
epsilon = 0.0000000000001
low = 0
high = n

while
    if guess**2 < n:
        low = guess
    else:
        high = guess
    guess = (low + high)/2

print(f"Square root of {n} is {guess}")
```

Fast Square root (**Bisection Search**).

```
n = int(input("Enter an integer: "))
epsilon = 0.0000000000001
low = 0
high = n
guess = (low + high)/2
while
    if guess**2 < n:
        low = guess
    else:
        high = guess
    guess = (low + high)/2

print(f"Square root of {n} is {guess}")
```

Fast Square root (**Bisection Search**).

```
n = int(input("Enter an integer: "))
epsilon = 0.00000000000001
low = 0
high = n
guess = (low + high)/2
while high-low > epsilon:
    if guess**2 < n:
        low = guess
    else:
        high = guess
    guess = (low + high)/2

print(f"Square root of {n} is {guess}")
```

Fast Square root (**Bisection Search**).

```
n = int(input("Enter an integer: "))
epsilon = 0.0000000000001
low = 0
high = n
guess = (low + high)/2
while high-low > epsilon:
    if guess**2 < n:
        low = guess
    else:
        high = guess
    guess = (low + high)/2

print(f"Square root of {n} is {guess}")
```

It took 0 seconds and **49 tries** to find $\sqrt{5}$ to **13** decimal places

2.2360679774997~~87~~

You Try!

Write code to do bisection search to find the cube root of positive cubes within some epsilon. Start with:

```
cube = 27
epsilon = 0.01
low = 0
high = cube
```

Summary

Questions?