Unit 3: Algorithms

Dave Abel

February 17th, 2016

Algorithms: Takeaway

- **Definition:** An *algorithm* is a recipe for solving a problem.

- Computer science is (loosely) the study of algorithms.

- I.e., computer science is the study of *automated methods of solving problems.*

- **Programs are ways of carrying out algorithms!!!**

Problem Specification

- *INPUT: Some stuff!*
- *OUTPUT: Information about the stuff!*

Algorithm Properties

1. **Halt:** Does the algorithm eventually *halt?*

2. **Correctness:** Does the algorithm solve the given problem for every possible input of the specified type?

3. **Growth Rate:** How many things does the computer have to do to run the algorithm?

Our First Problem: Search

- *INPUT:*
  - a list of objects, call it “Basket”
  - a specific object, call it “Snozzberry”
- *OUTPUT:*
  - True if “Snozzberry” is in “Basket”.
  - False if “Snozzberry” is *not* in “Basket”

Random Search

1. Pick a random item

2. If that item is the one we want, report True!

3. If not, repeat from step 1

Linear Search

1. Start at the beginning of the list

2. Look at each item in turn. If we find it, stop and report true!

3. If we reach the end of the list, report False!

Binary Search

Assumes a sorted list

1. If the middle item is our item, report True!

2. Otherwise, ask: is our number greater than or less than the middle number?

3. If greater, search the right half.

4. If less, search the left half.

Three Algorithms for Search

1. Random Search

- Slow!
- *Incorrect:* never terminates in some cases.

2. Linear Search

- Fast! *N* operations in the worst case, for a length *N* list.
- *Correct:* always terminates and reports the correct answer.

3. Binary Search

- Assumes a sorted list
- *Correct:* always terminates and reports the correct answer.
- Fast! log(*N*) operations in the worst case, for a length *N* list.

Clicker Question!

Q: For Linear Search, what case forces us to execute the most number of operations?

Clicker Question!

[A] The item we're searching for is not in the list

[B] There is more than one of the item we're searching for in the list

[C] The item we're searching for is in the middle

[D] The item we're searching for is near the end

Q: For Linear Search, what case forces us to execute the most number of operations?

Clicker Answer!

[A] The item we're searching for is not in the list

[B] There is more than one of the item we're searching for in the list

[C] The item we're searching for is in the middle

[D] The item we're searching for is near the end

Q: For Linear Search, what case forces us to execute the most number of operations?

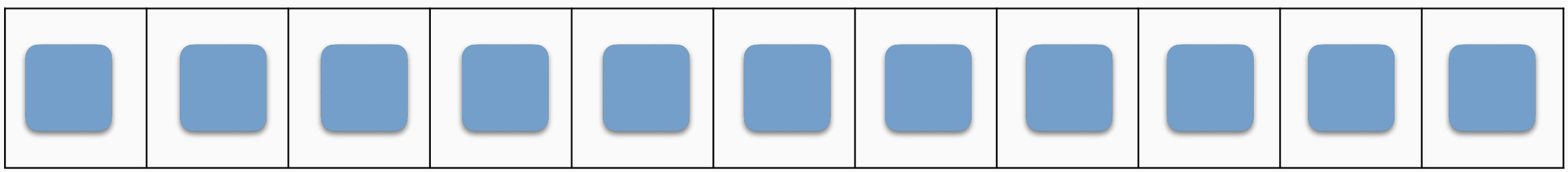
Clicker Answer!

[A] The item we're searching for is not in the list

We'll have to check every item…

Quick Binary Search Review

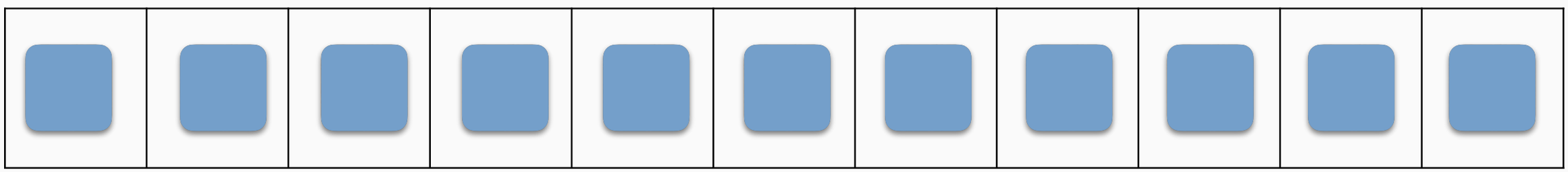
Suppose we're searching for the number "5"



Suppose we are told the list is sorted

Quick Binary Search Review

Suppose we're searching for the number "5"



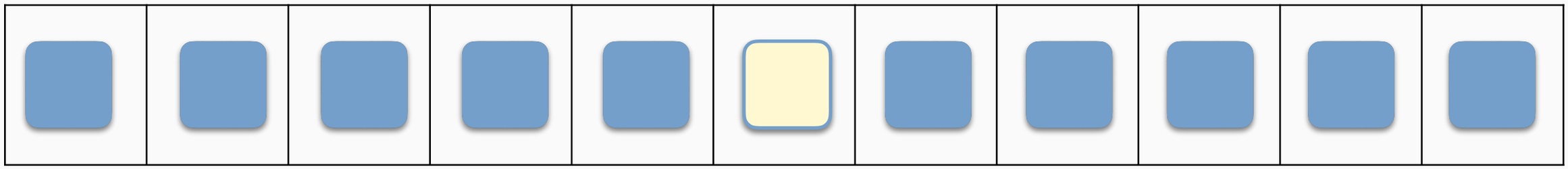
Suppose we are told the list is sorted

Q: What is the best card to turn over?

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted



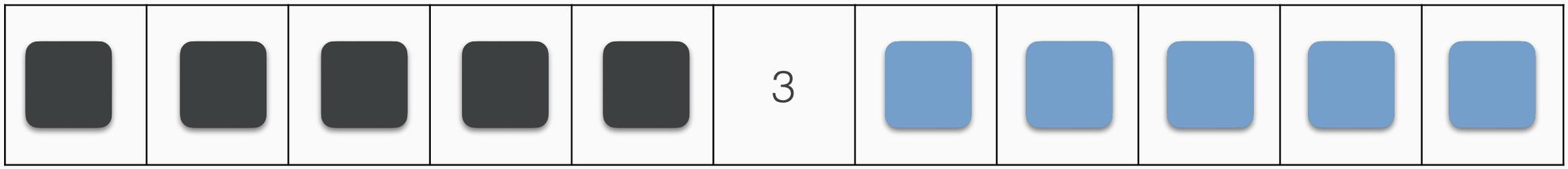
Q: What is the best card to turn over?

A: This one! Whatever the answer is, we get rid of the most options

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted



Q: What is the best card to turn over?

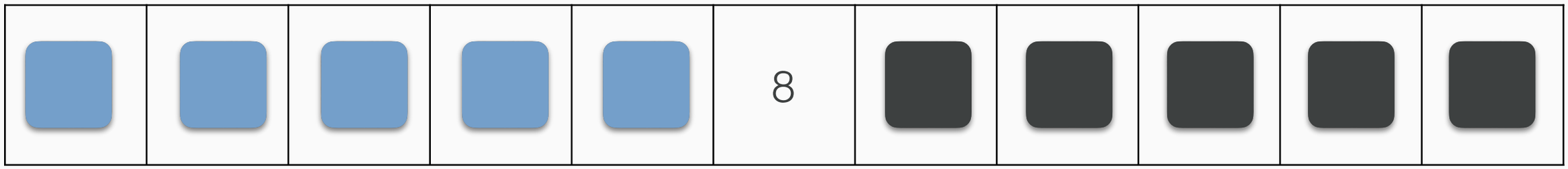
A: This one! Whatever the answer is, we get rid of the most options

3 < 5 So we can get rid of the whole left half…

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted



Q: What is the best card to turn over?

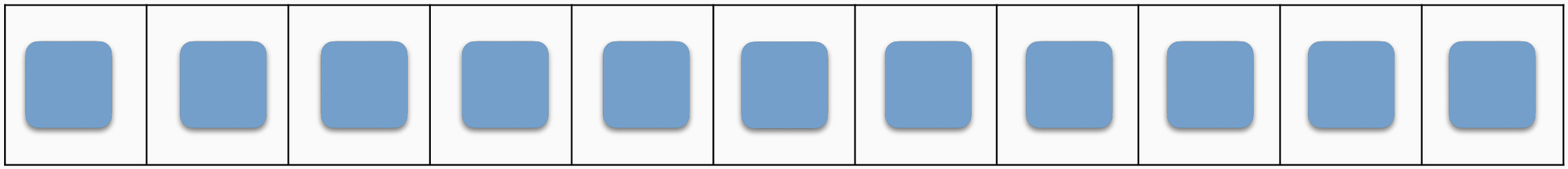
A: This one! Whatever the answer is, we get rid of the most options

5 < 8 So we can get rid of the whole right half…

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted

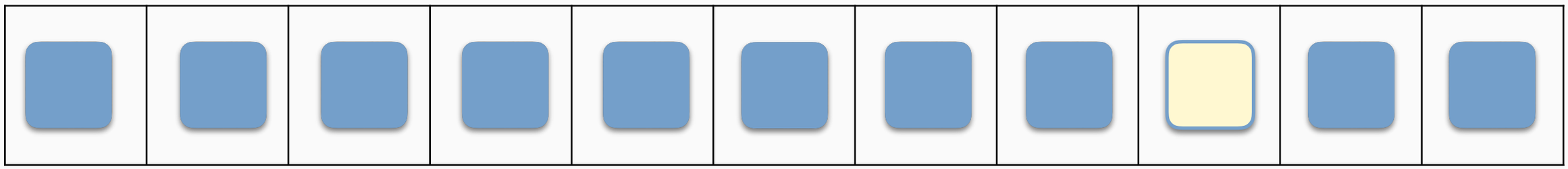


Suppose we try a different one?

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted

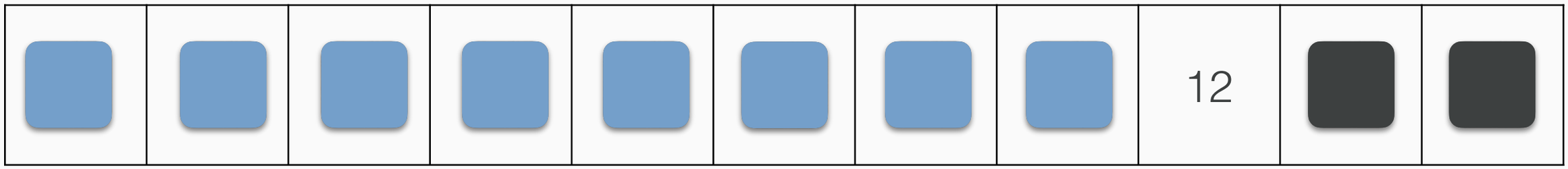


Suppose we try a different one?
Like this one?

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted



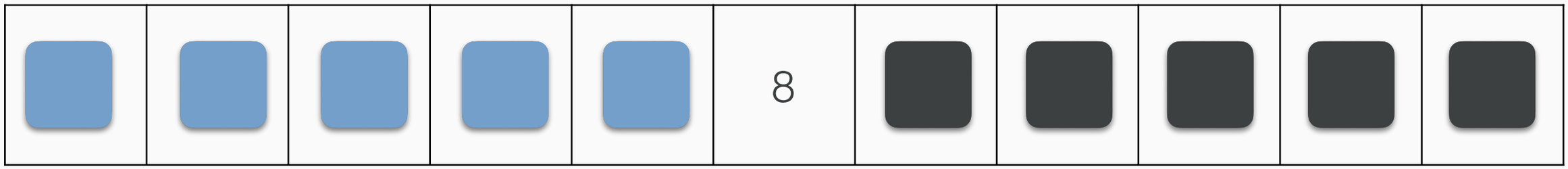
Suppose we try a different one?
Like this one?

Now we can only get rid of *two* options…

Quick Binary Search Review

Suppose we're searching for the number "5"

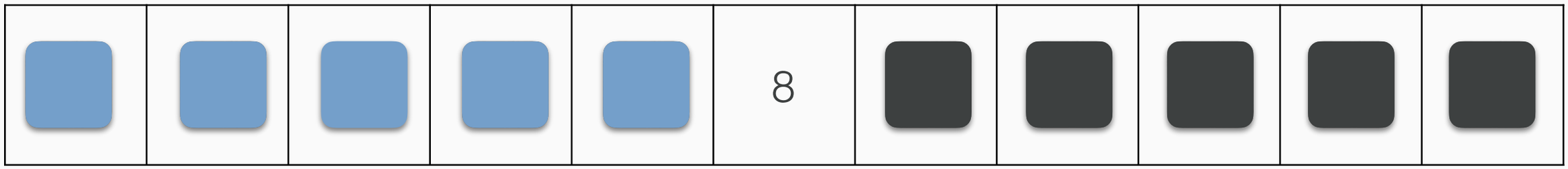
Suppose we are told the list is sorted



Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted

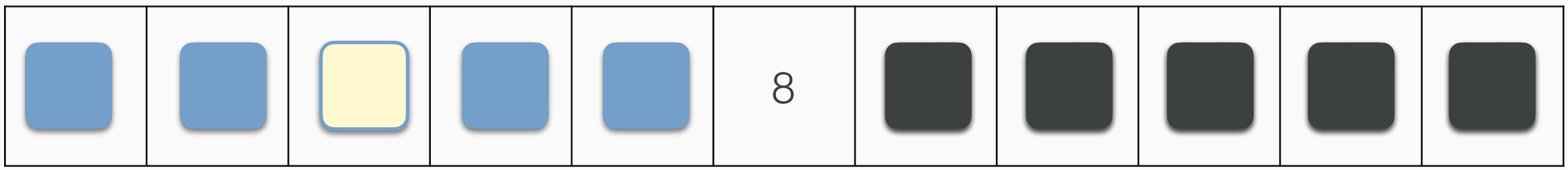


Q: What is the best card to turn over?

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted



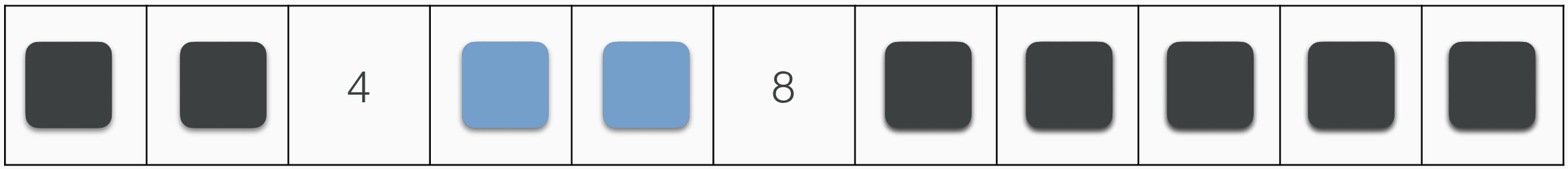
Q: What is the best card to turn over?

A: This one! Again, whatever the answer is, we get rid of the most options

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted



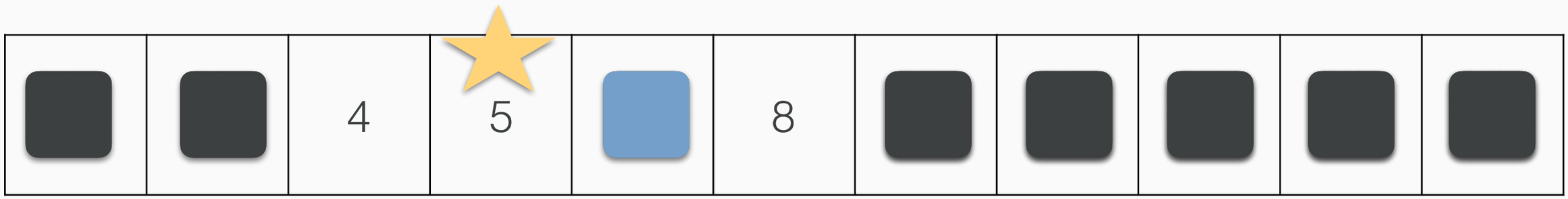
Q: What is the best card to turn over?

A: This one! Again, whatever the answer is, we get rid of the most options

Quick Binary Search Review

Suppose we're searching for the number "5"

Suppose we are told the list is sorted



Q: What is the best card to turn over?

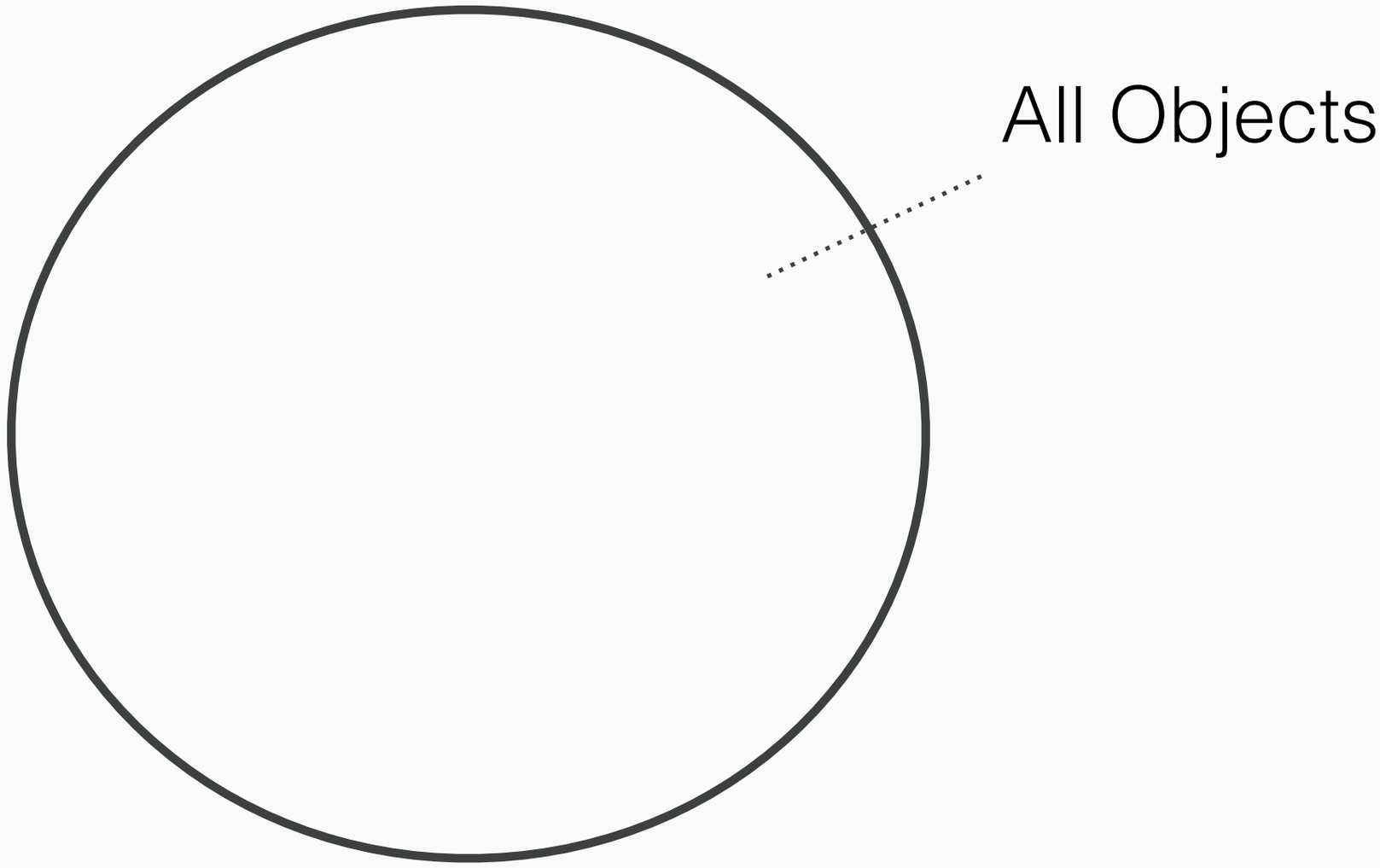
A: This one! Again, whatever the answer is, we get rid of the most options

Remember This?

20 Questions

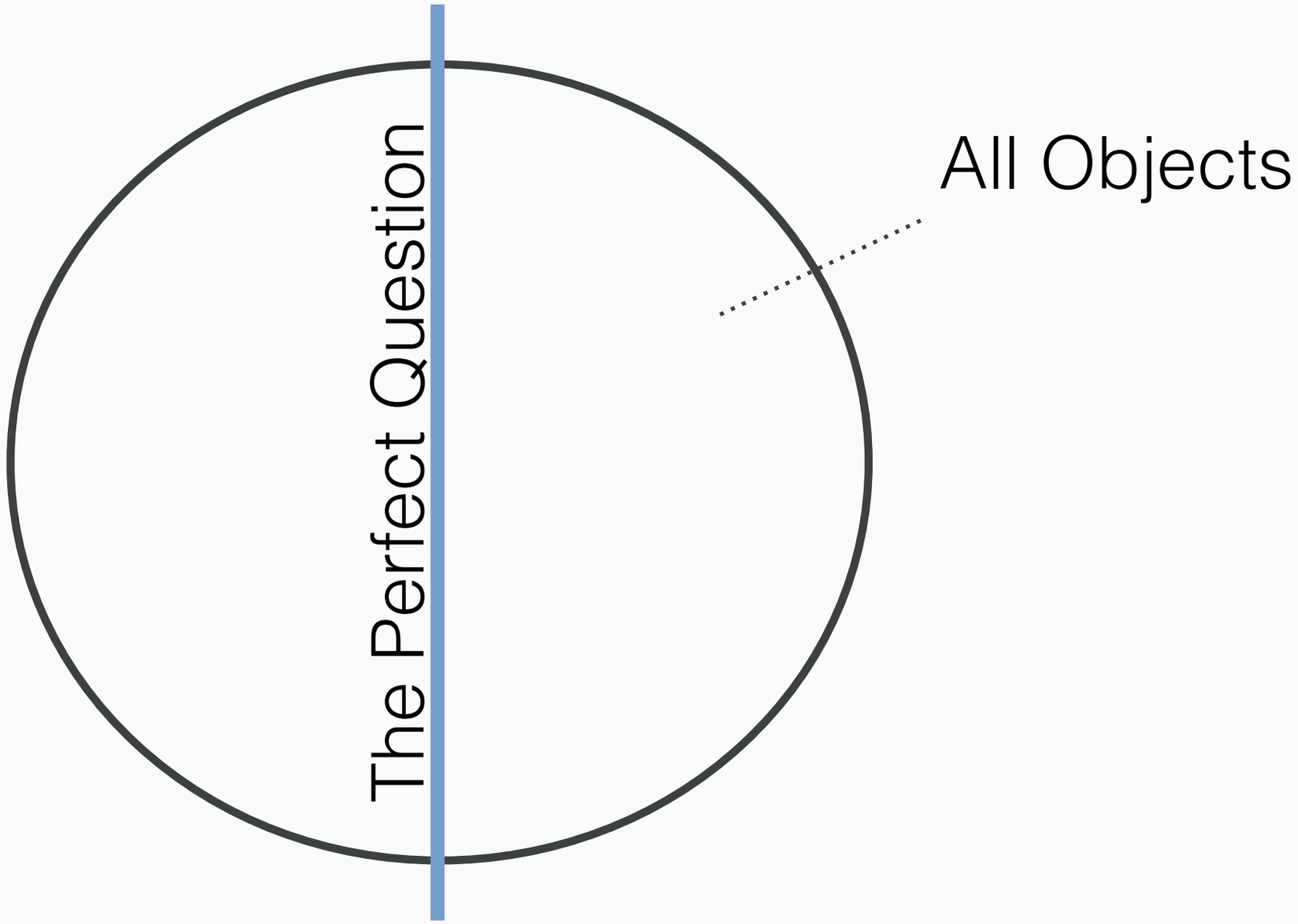
There is a *best strategy* in 20 Questions…

20 Questions



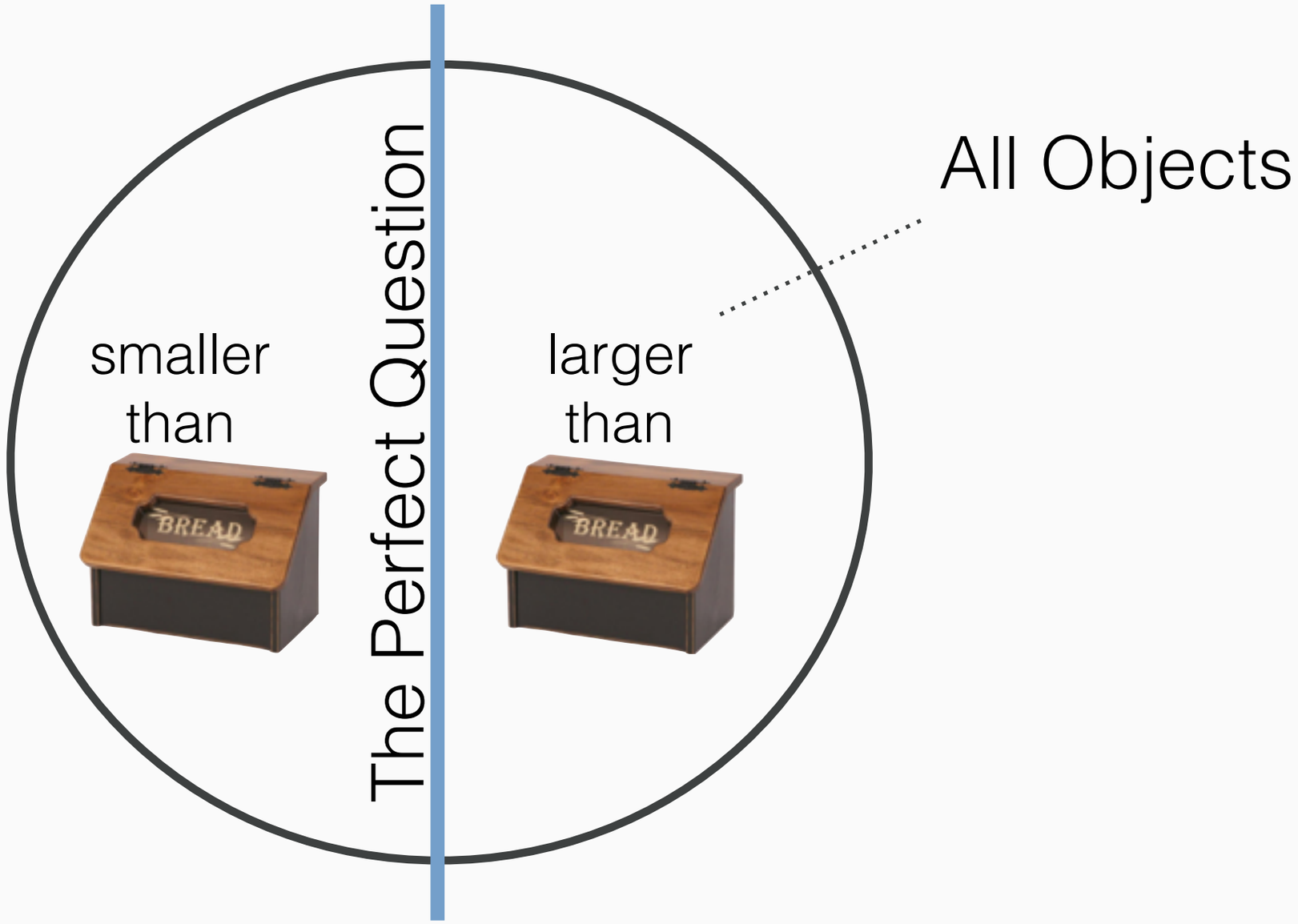
There is a *best strategy* in 20 Questions…

20 Questions



There is a *best strategy* in 20 Questions…

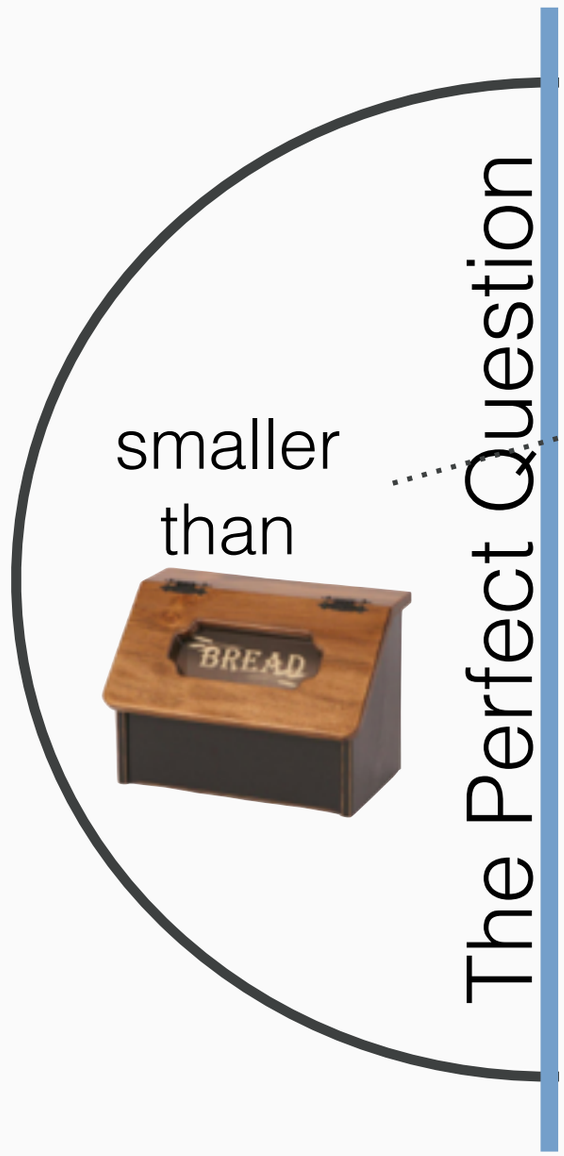
20 Questions



There is a *best strategy* in 20 Questions…

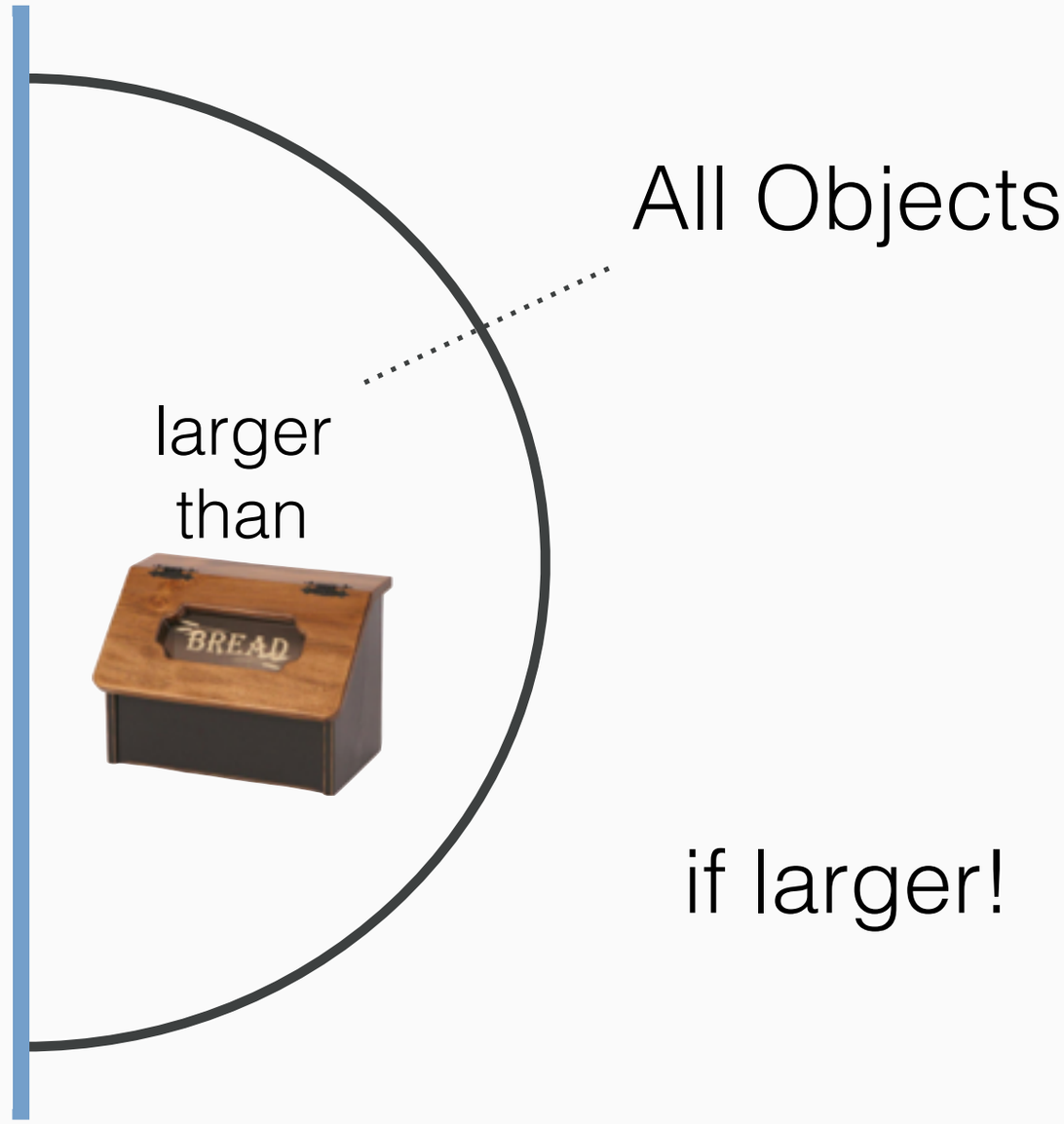
20 Questions

if smaller!



There is a *best strategy* in 20 Questions…

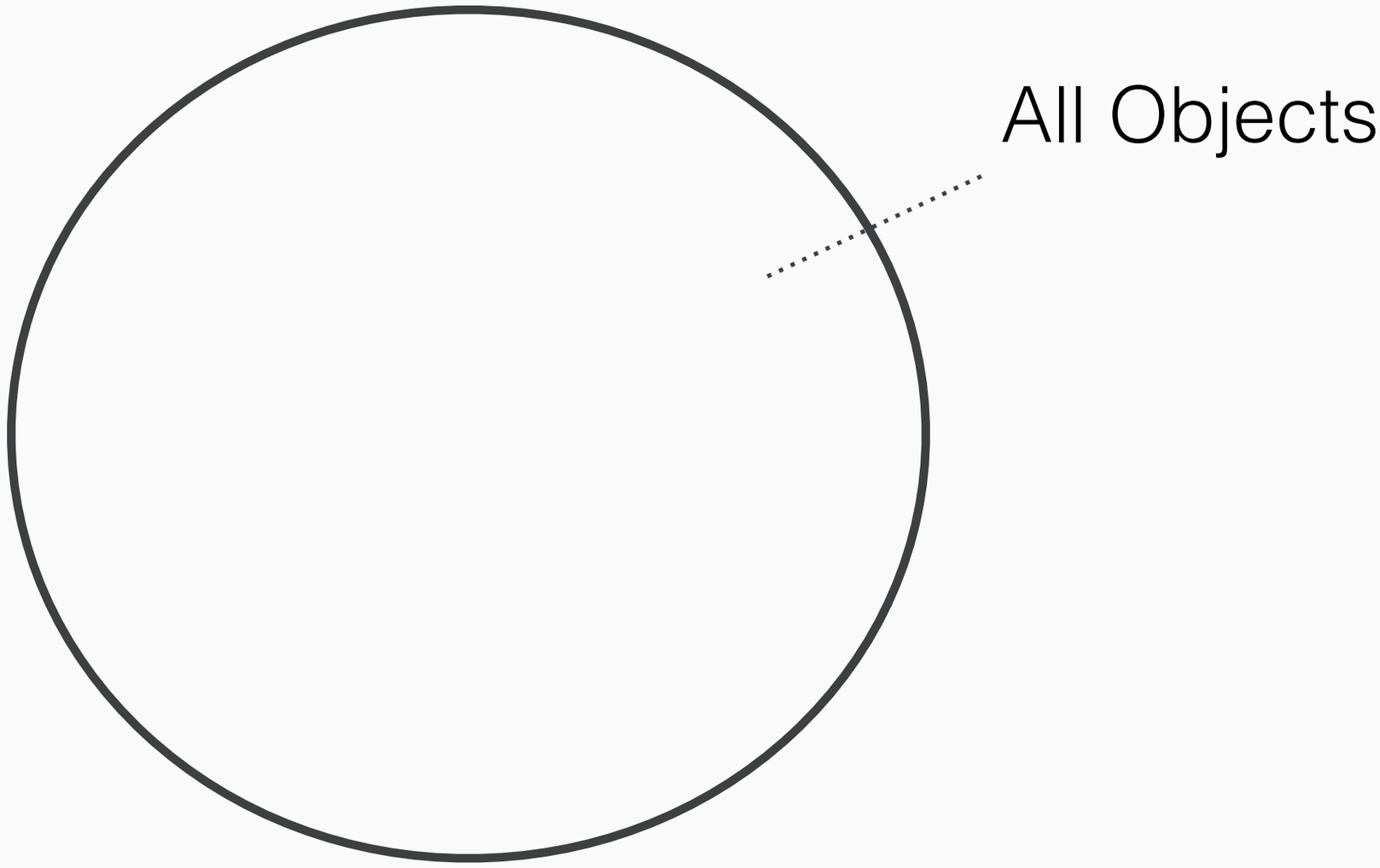
20 Questions



if larger!

There is a *best strategy* in 20 Questions…

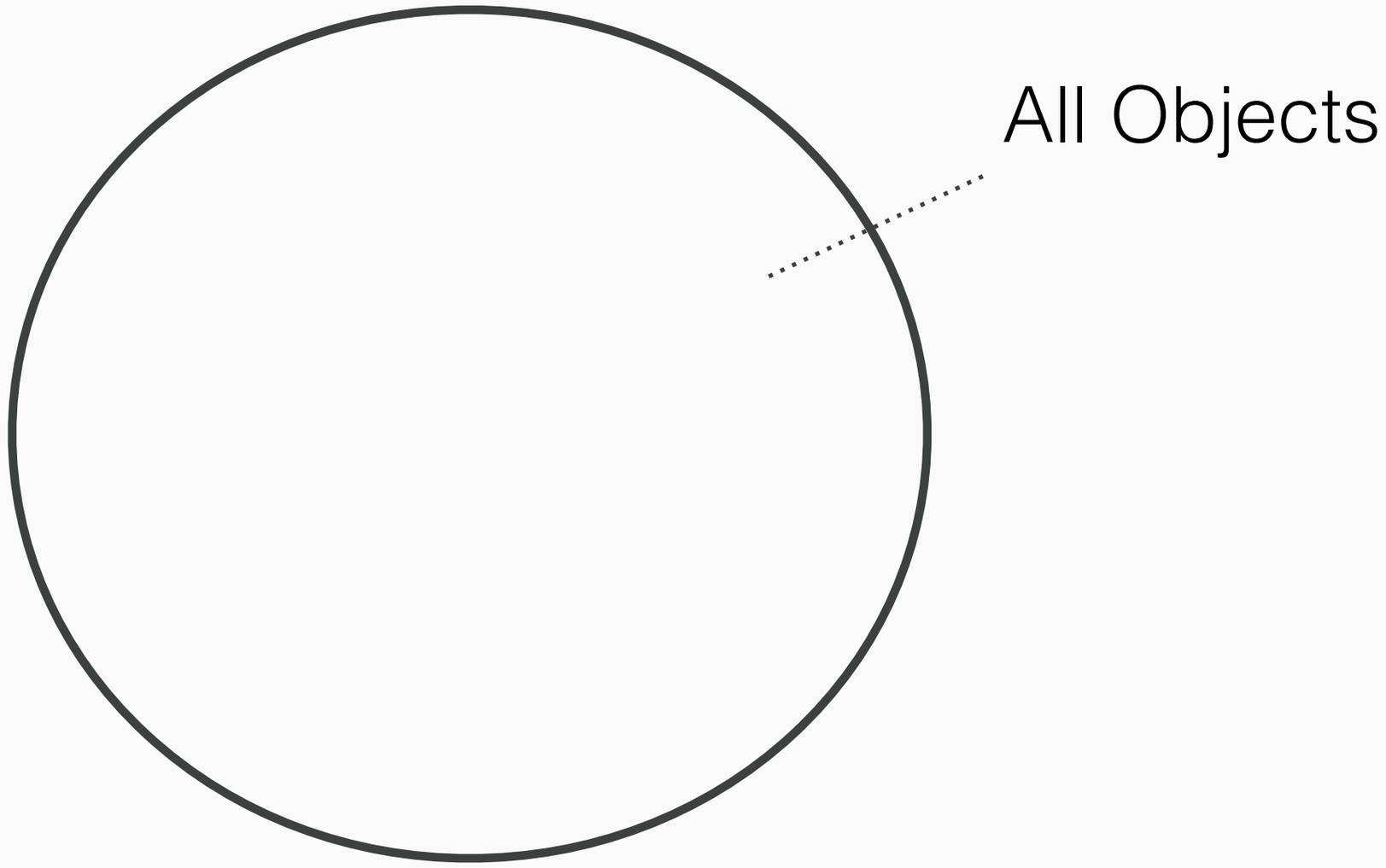
20 Questions



Consider the *Linear Search* equivalent

20 Questions

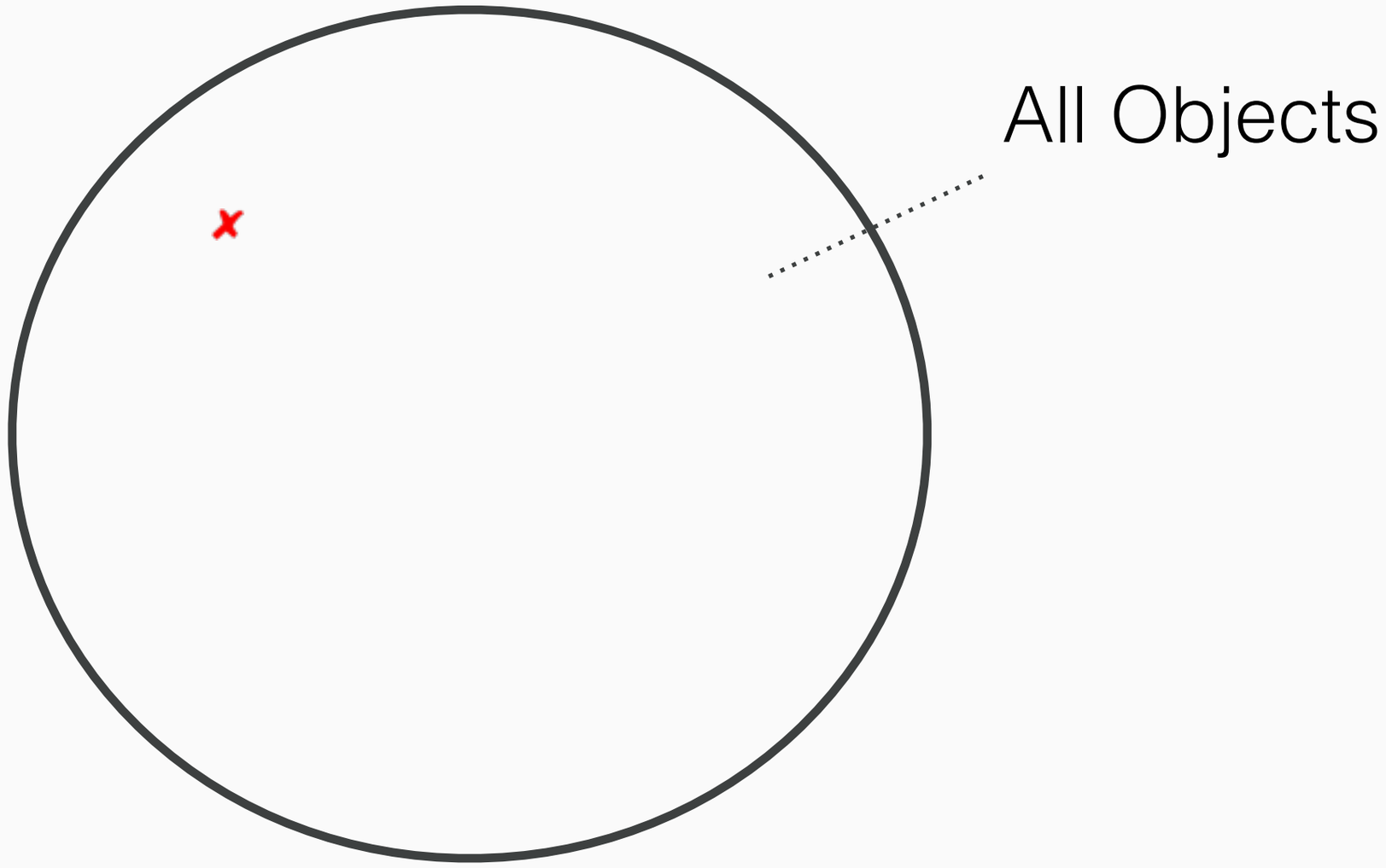
1. Is it a 200ml graduated cylinder?



Consider the *Linear Search* equivalent

20 Questions

1. Is it a 200ml graduated cylinder?

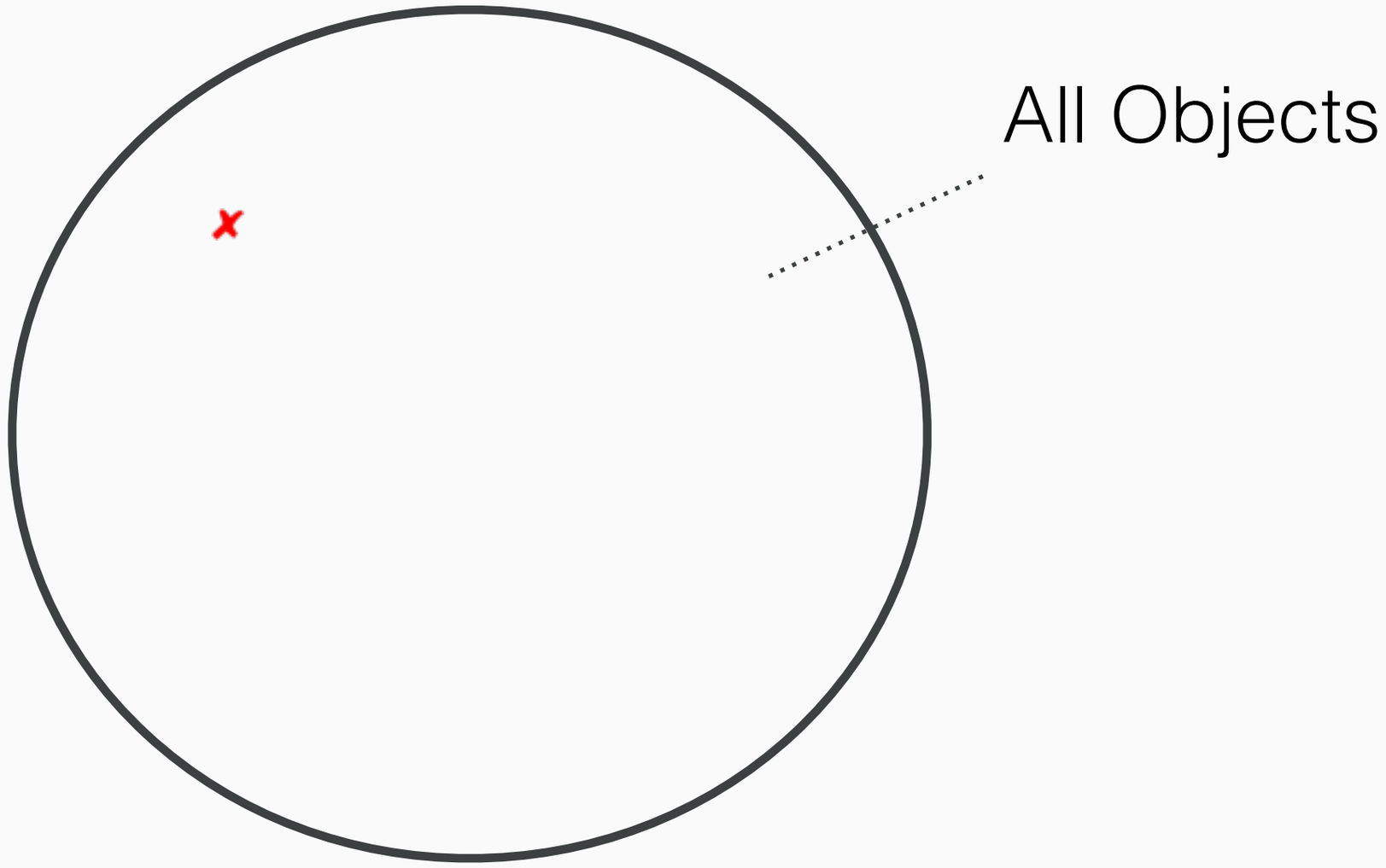


Consider the *Linear Search* equivalent

20 Questions

1. Is it a 200ml graduated cylinder?

2. Is it a granny smith apple?

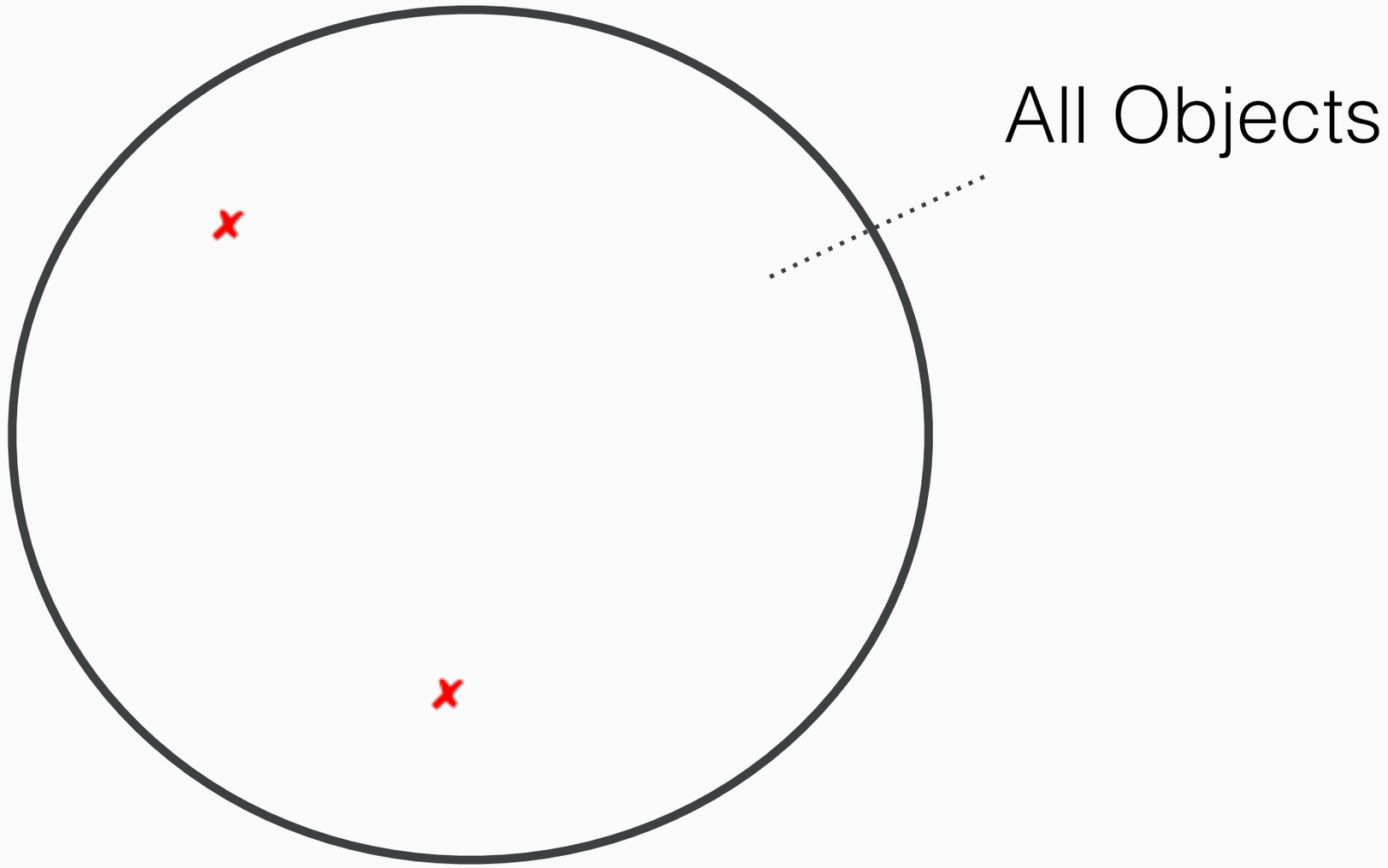


Consider the *Linear Search* equivalent

20 Questions

1. Is it a 200ml graduated cylinder?

2. Is it a granny smith apple?



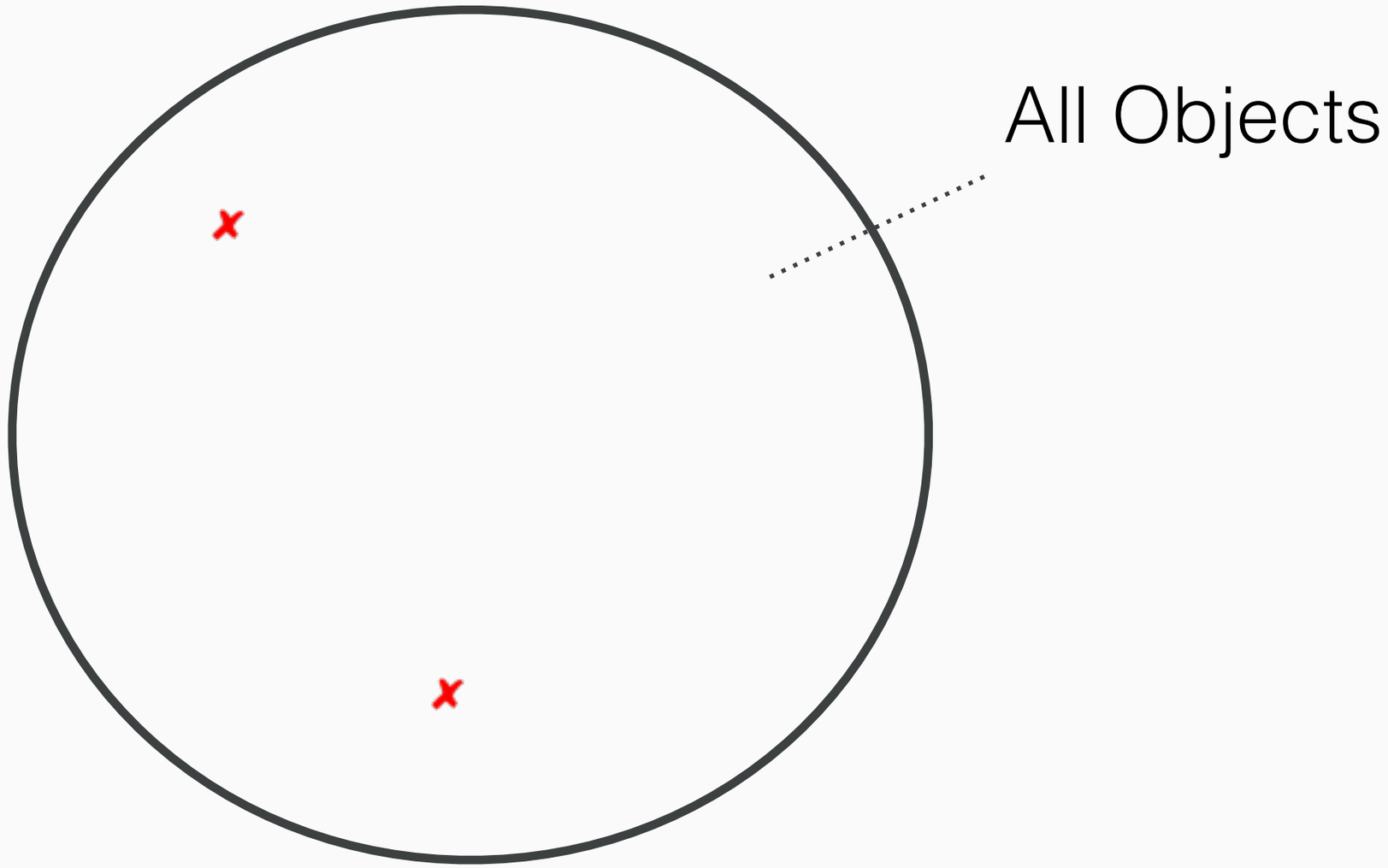
Consider the *Linear Search* equivalent

20 Questions

1. Is it a 200ml graduated cylinder?

2. Is it a granny smith apple?

3. Is it the Cheshire Cat from Alice and Wonderland?



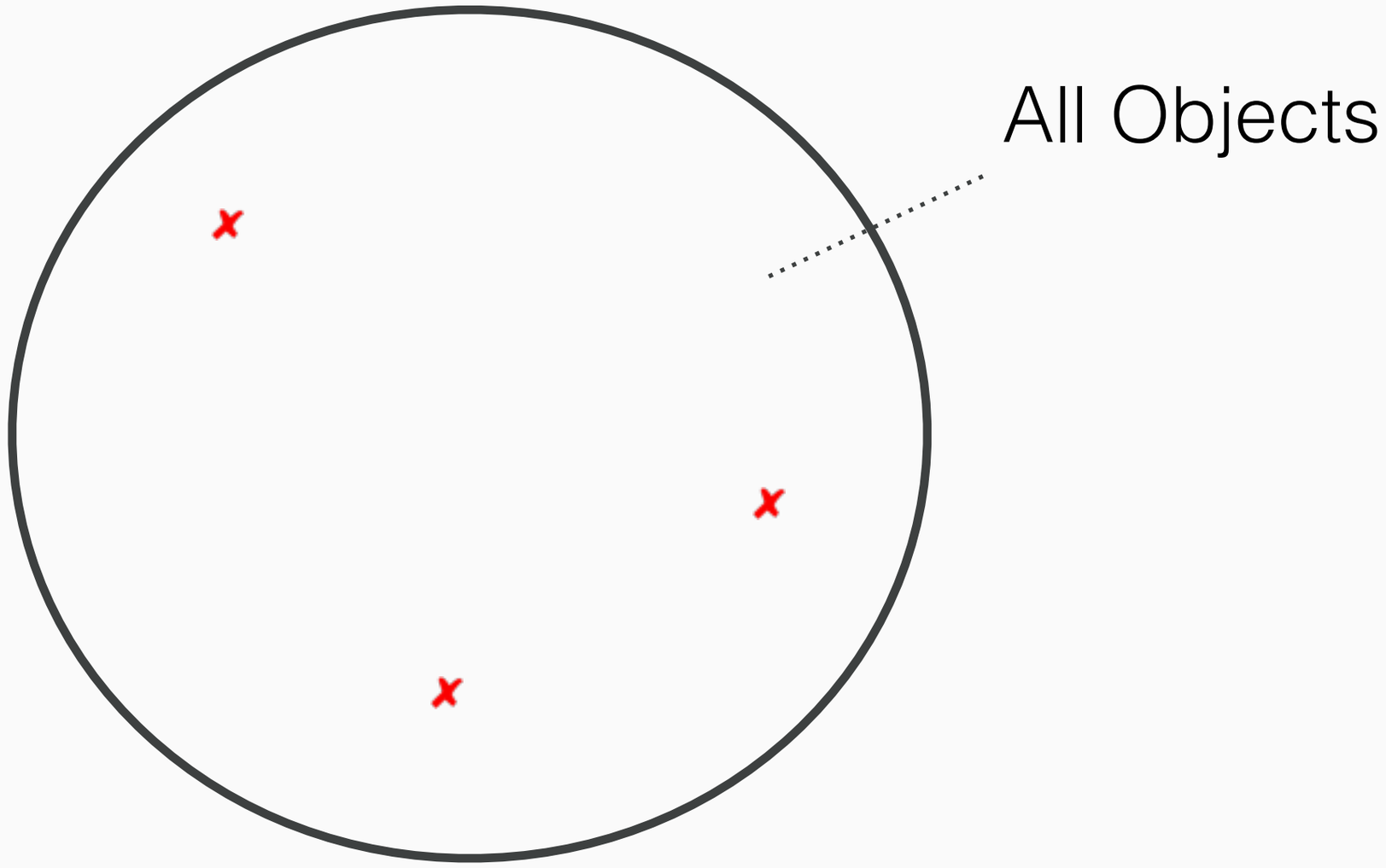
Consider the *Linear Search* equivalent

20 Questions

1. Is it a 200ml graduated cylinder?

2. Is it a granny smith apple?

3. Is it the Cheshire Cat from Alice and Wonderland?



Consider the *Linear Search* equivalent

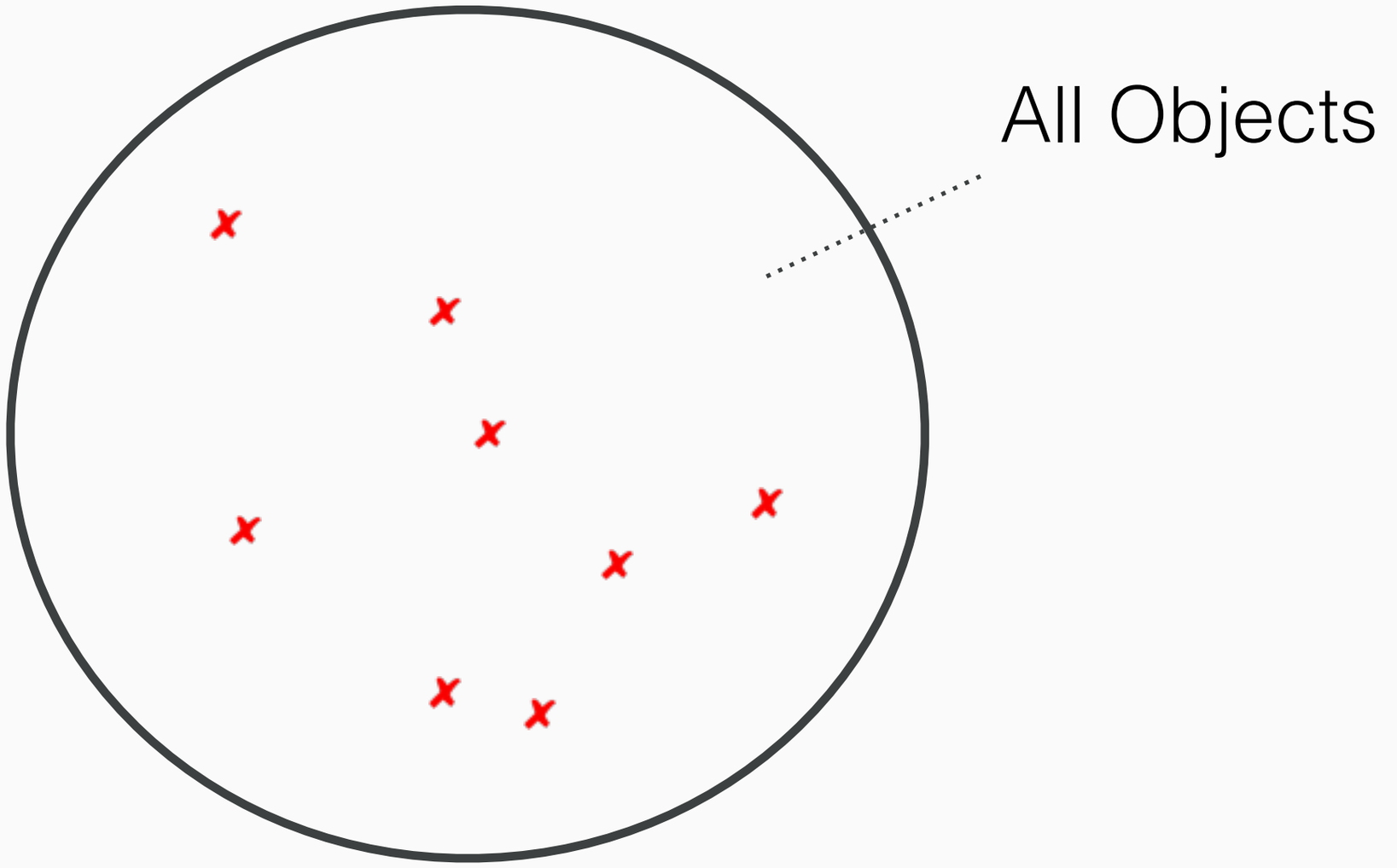
20 Questions

1. Is it a 200ml graduated cylinder?

2. Is it a granny smith apple?

3. Is it the Cheshire Cat from Alice and Wonderland?

4. ….



Consider the *Linear Search* equivalent

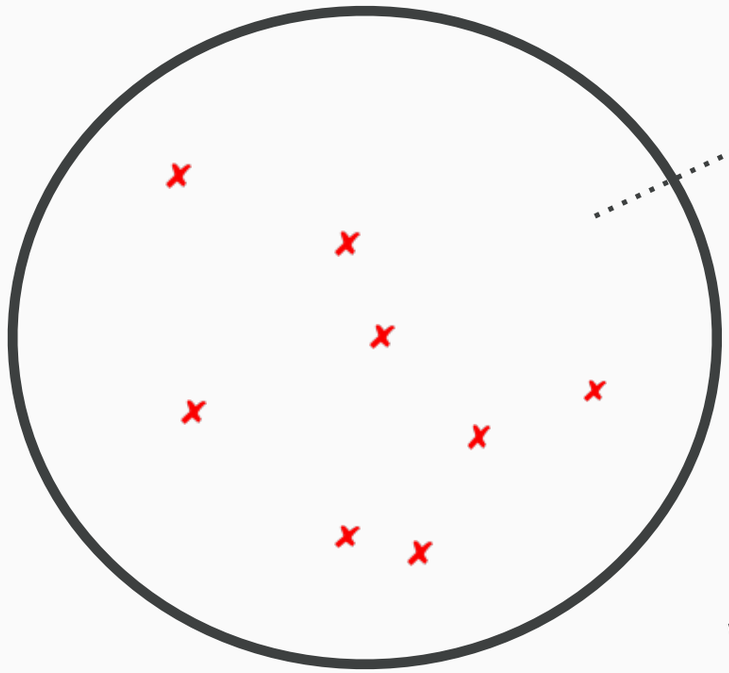
20 Questions

1. Is it a 200ml graduated cylinder?

2. Is it a granny smith apple?

3. Is it the Cheshire Cat from Alice and Wonderland?

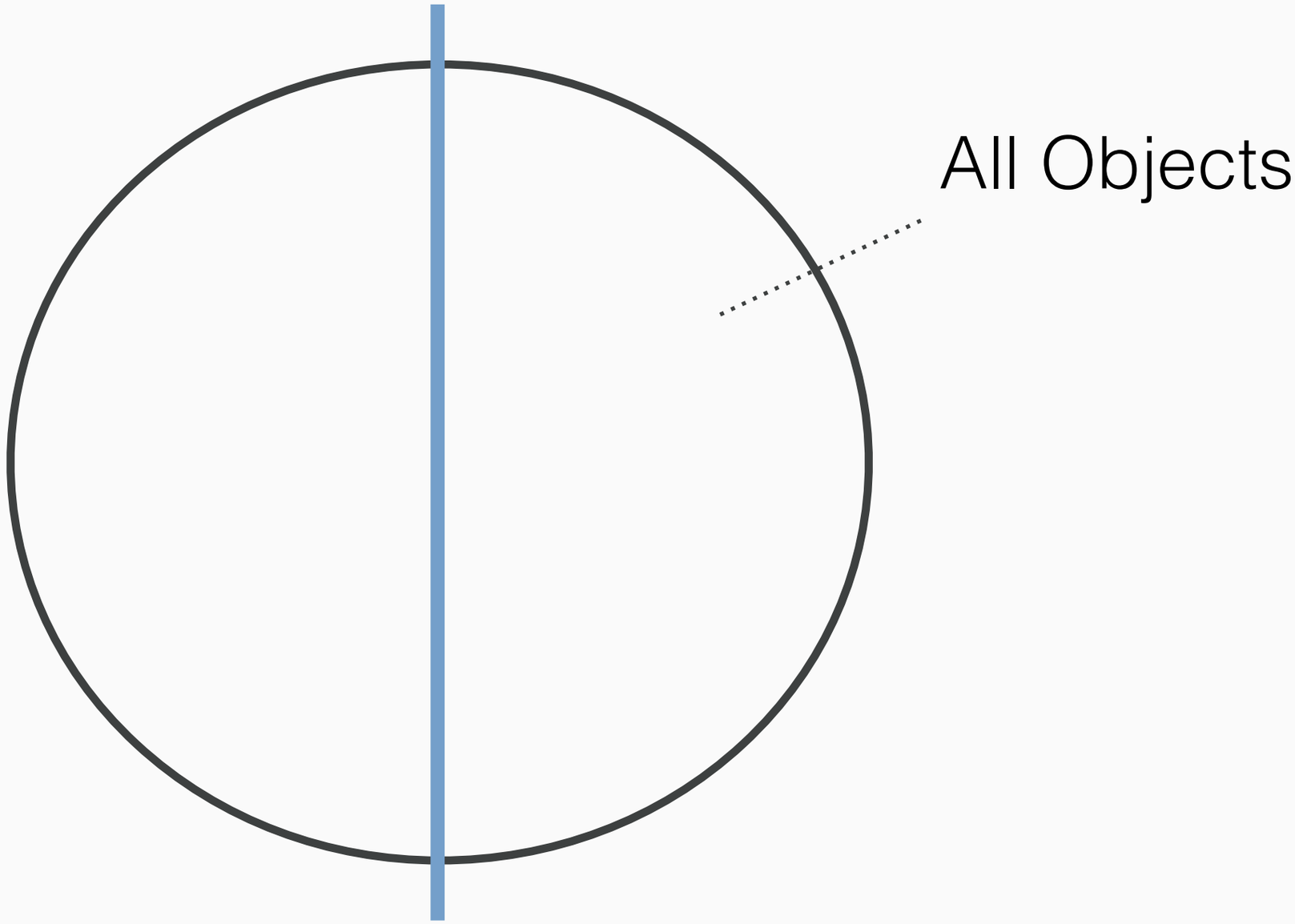
4. ….



Worst case: have to guess *every conceivable object*

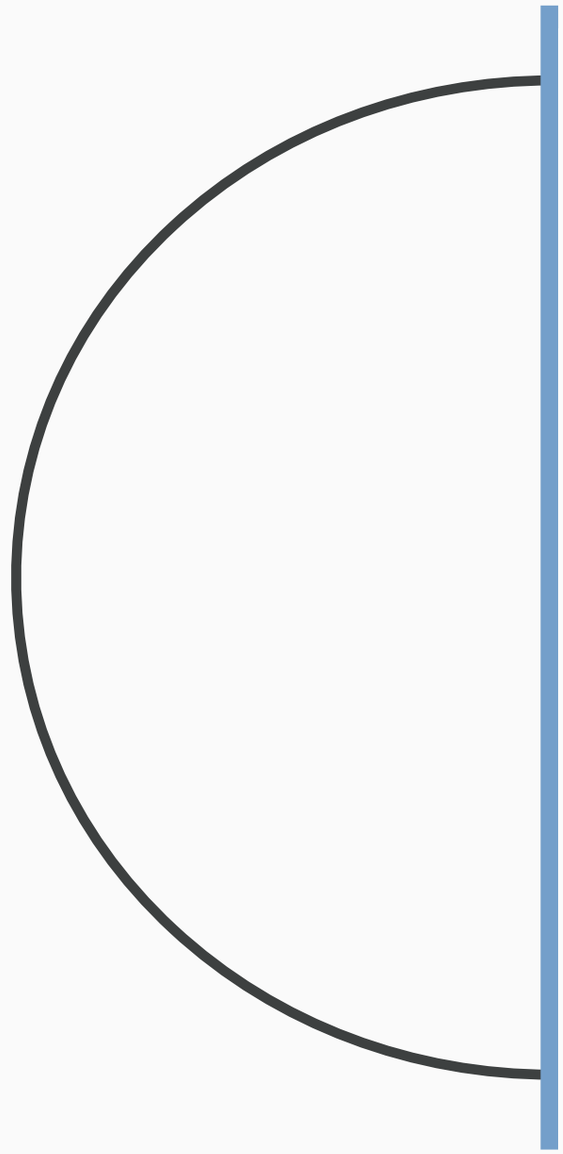
Consider the *Linear Search* equivalent

20 Questions



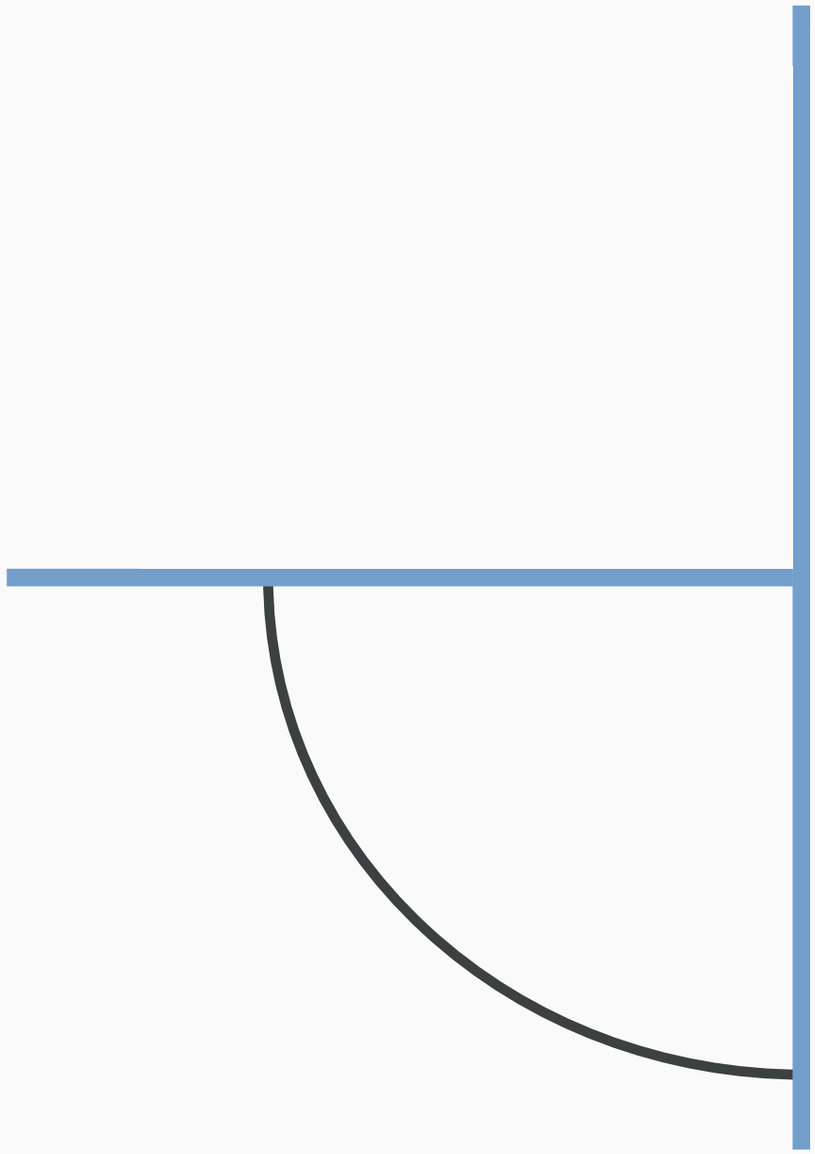
Binary Search

20 Questions



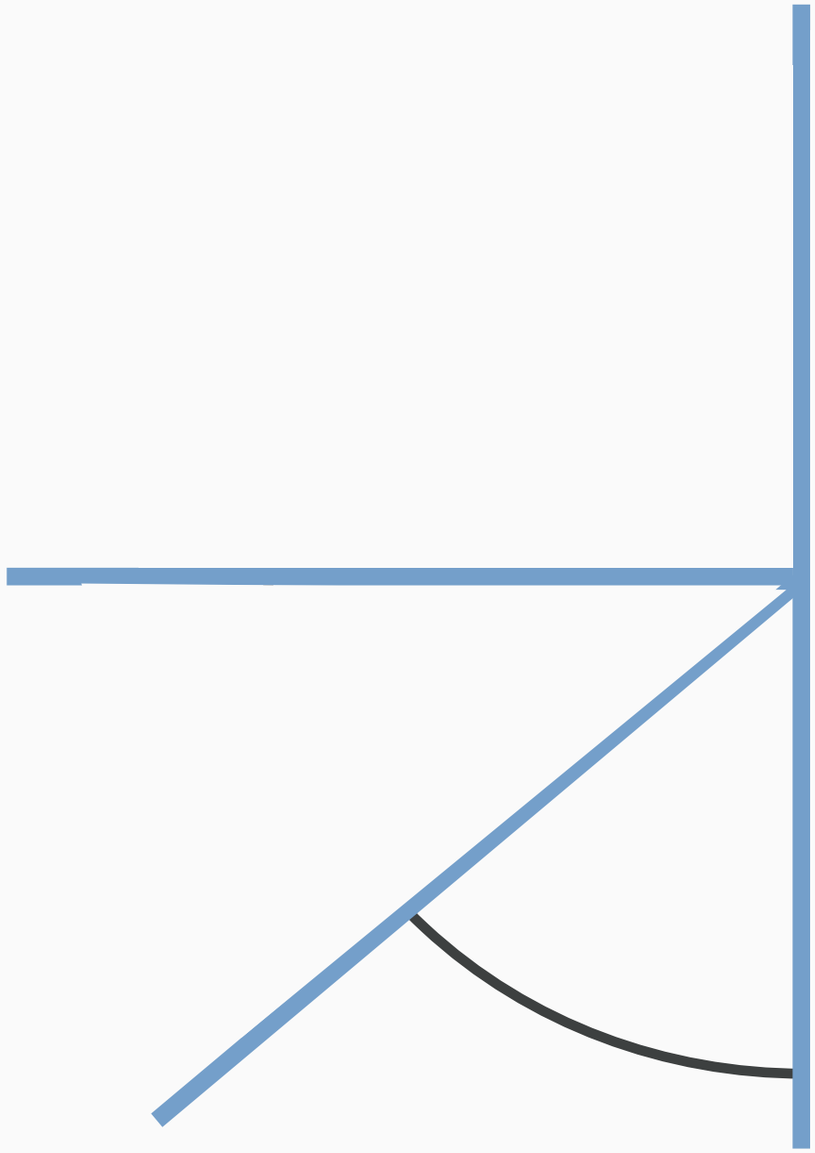
Binary Search

20 Questions



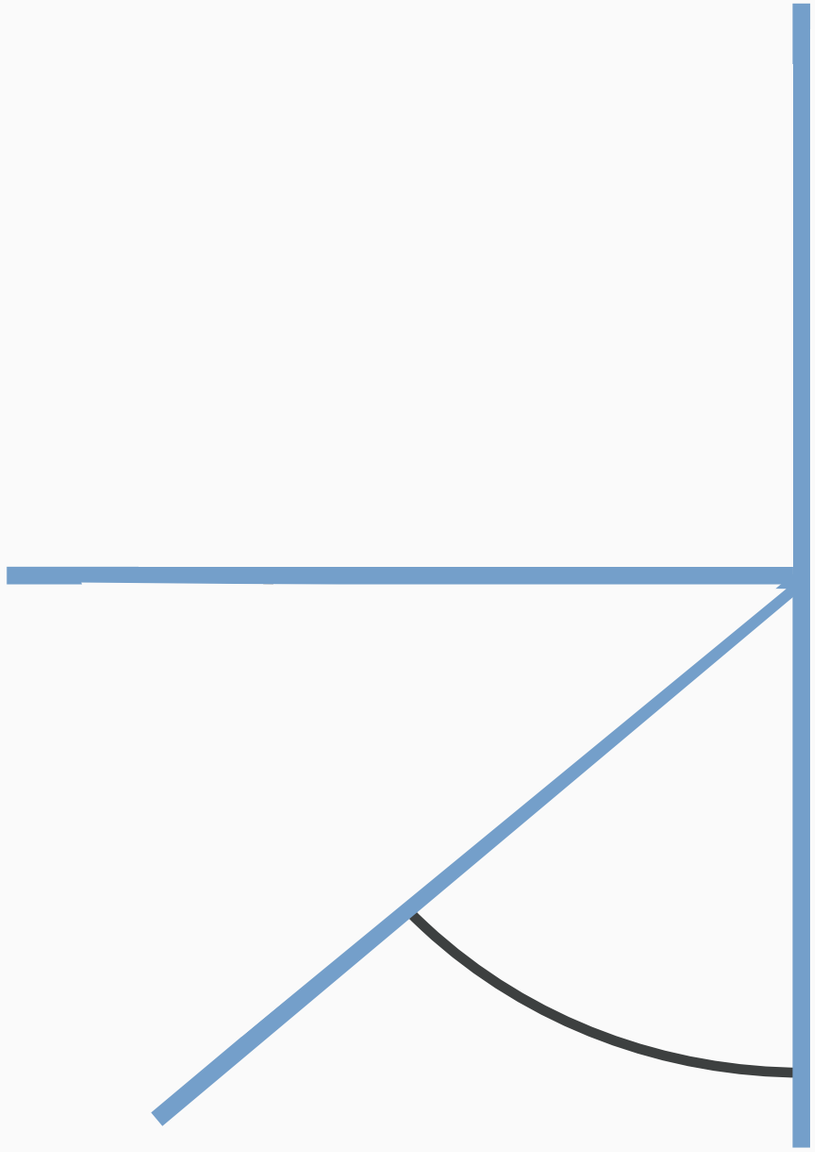
Binary Search

20 Questions



Binary Search

20 Questions



And so on….

(even in the worst case!)

Binary Search

Lets Play!

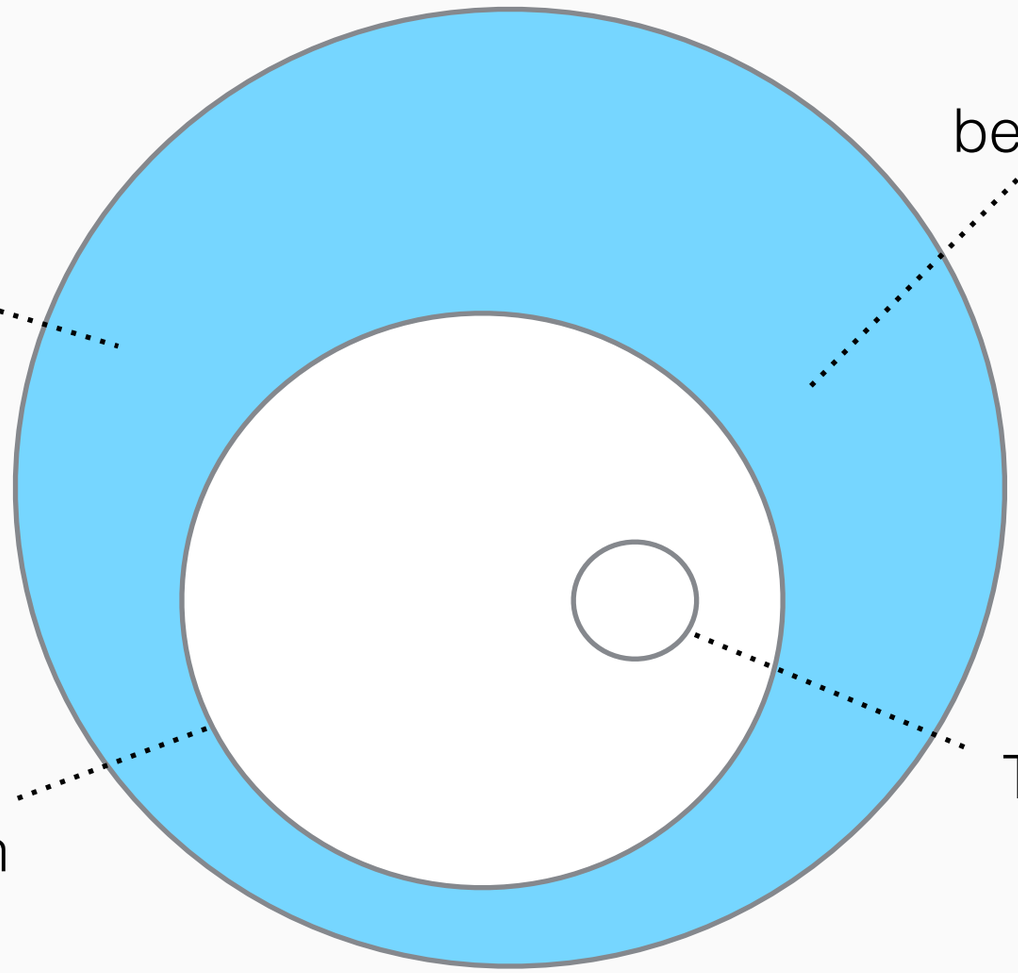
Revisiting Growth Rates

Remember Random Search?

It took *way* longer with a longer list.

Revisiting Growth Rates

Q: What, if anything, is out here?



Things that can be computed, period.

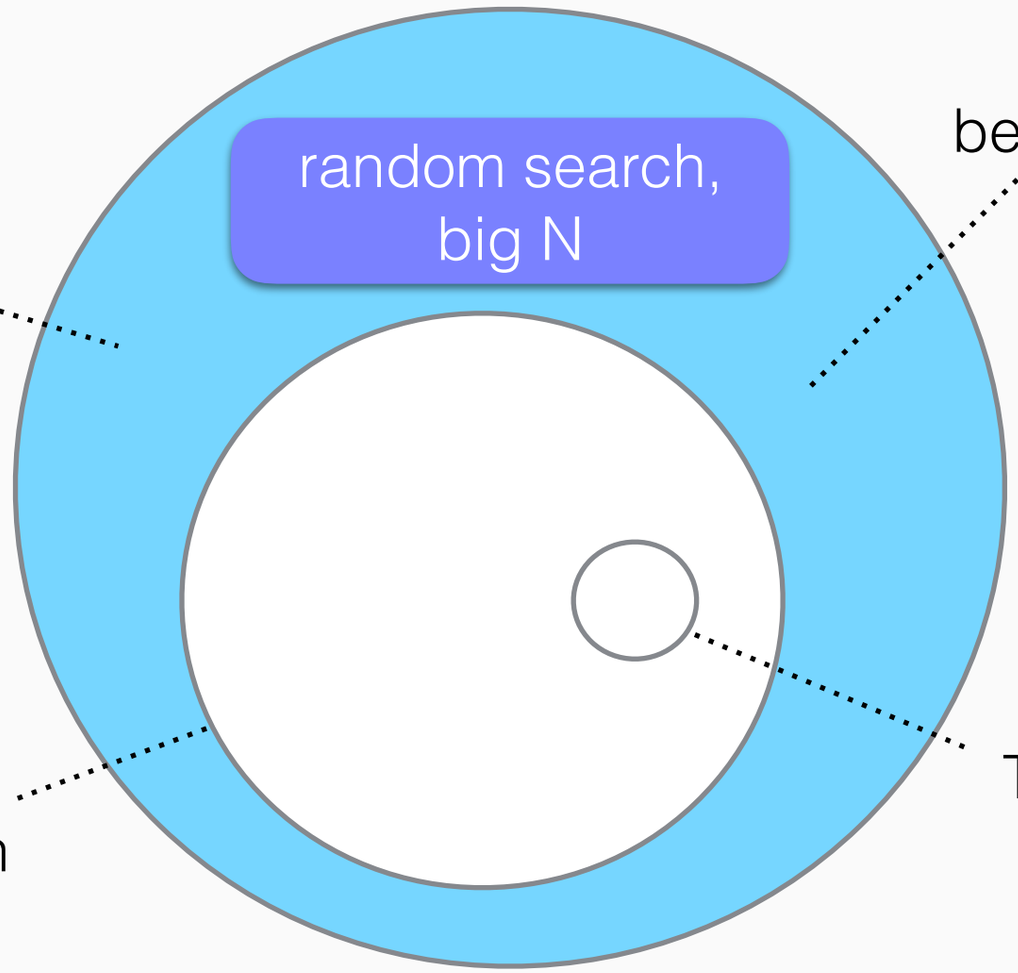
Things a regular computer can compute before the sun goes supernova

Things a domino computer could compute before the sun goes supernova

Remember Random Search? It took *way* longer with a longer list.

Revisiting Growth Rates

Q: What, if anything, is out here?



Things that can be computed, period.

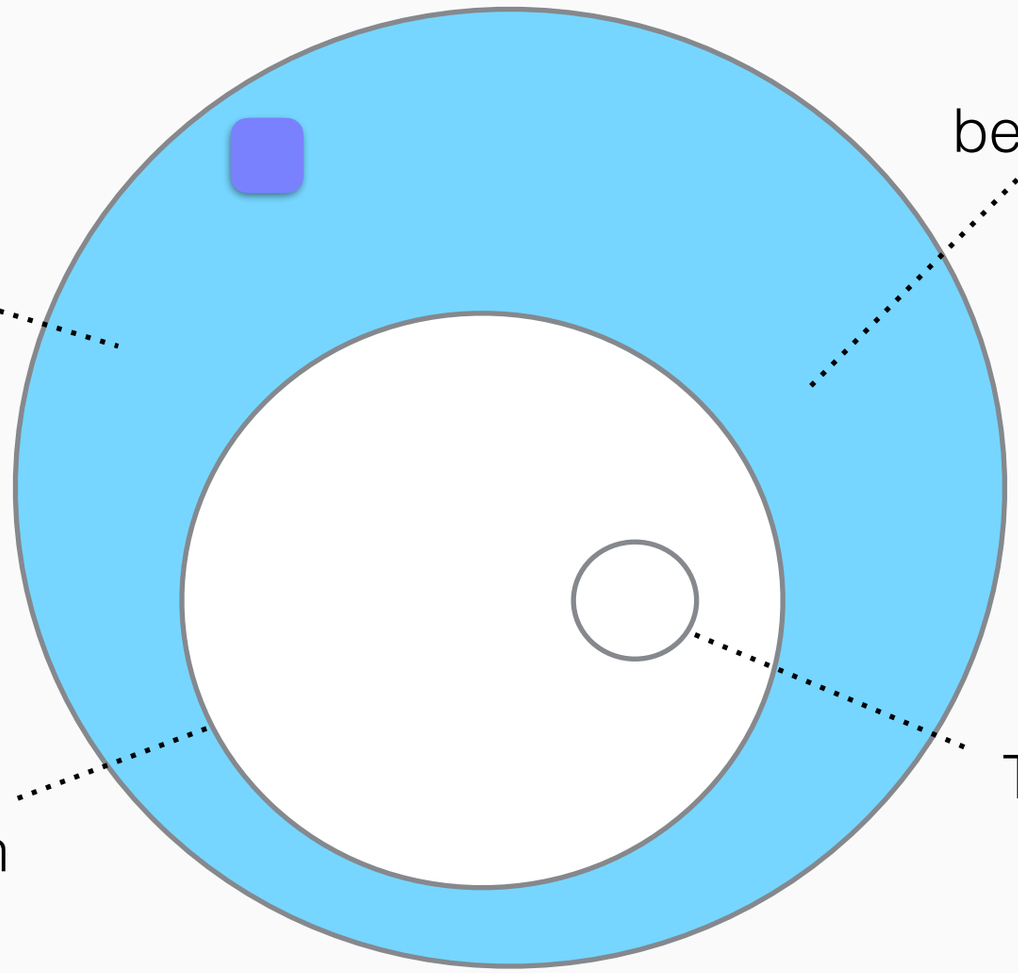
Things a regular computer can compute before the sun goes supernova

Things a domino computer could compute before the sun goes supernova

Remember Random Search? It took *way* longer with a longer list.

Revisiting Growth Rates

Q: What, if anything, is out here?



Things that can be computed, period.

Things a regular computer can compute before the sun goes supernova

Things a domino computer could compute before the sun goes supernova

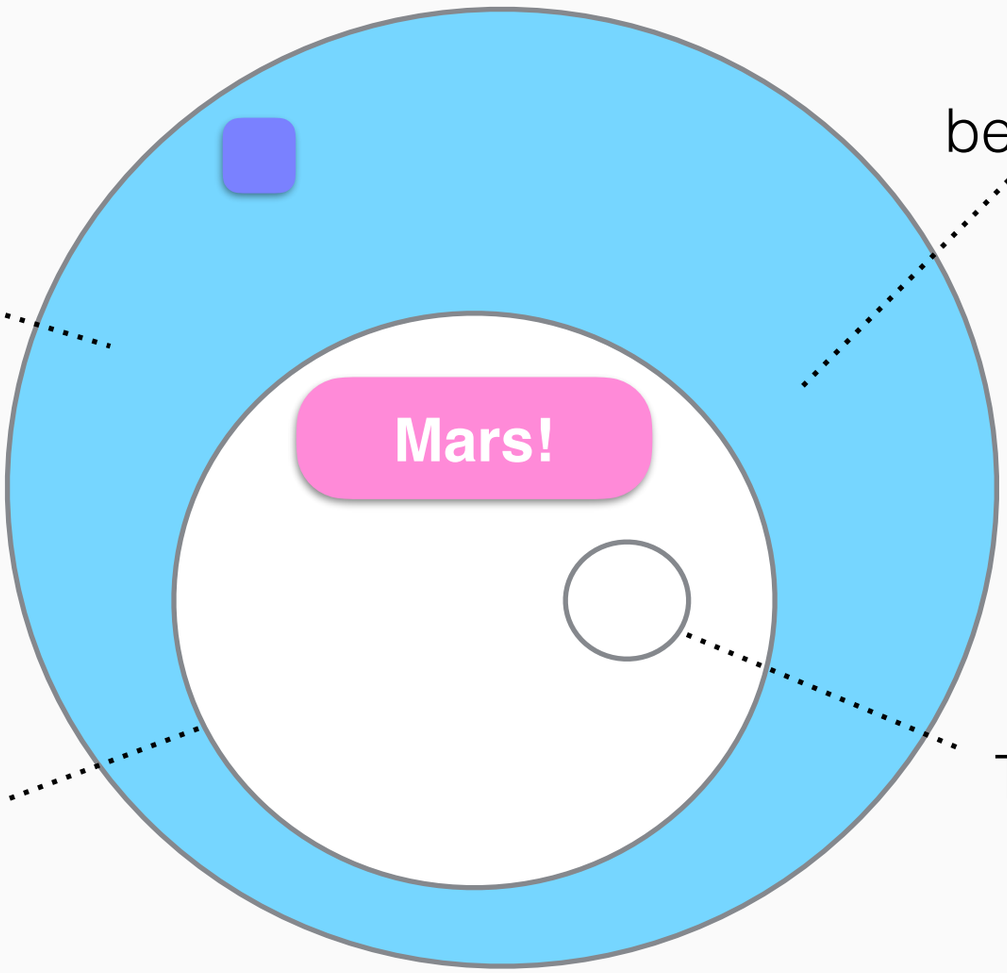
Remember Random Search? It took *way* longer with a longer list.

Problem Specification Example

- *INPUT: Map of solar system, description of physical laws, summary of current technology.*
- *OUTPUT: A method for colonizing Mars.*

Revisiting Growth Rates

Q: What, if anything, is out here?



Things that can be computed, period.

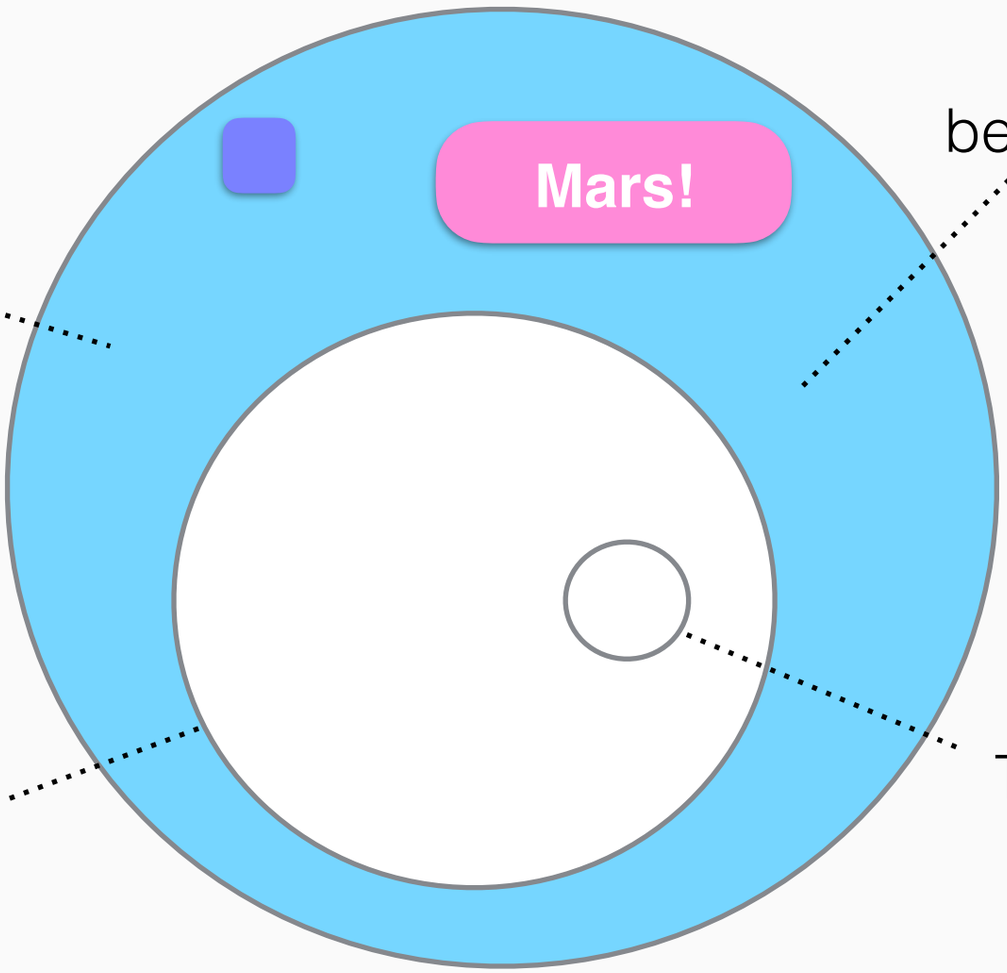
Things a regular computer can compute before the sun goes supernova

Things a domino computer could compute before the sun goes supernova

Remember Random Search? It took *way* longer with a longer list.

Revisiting Growth Rates

Q: What, if anything, is out here?



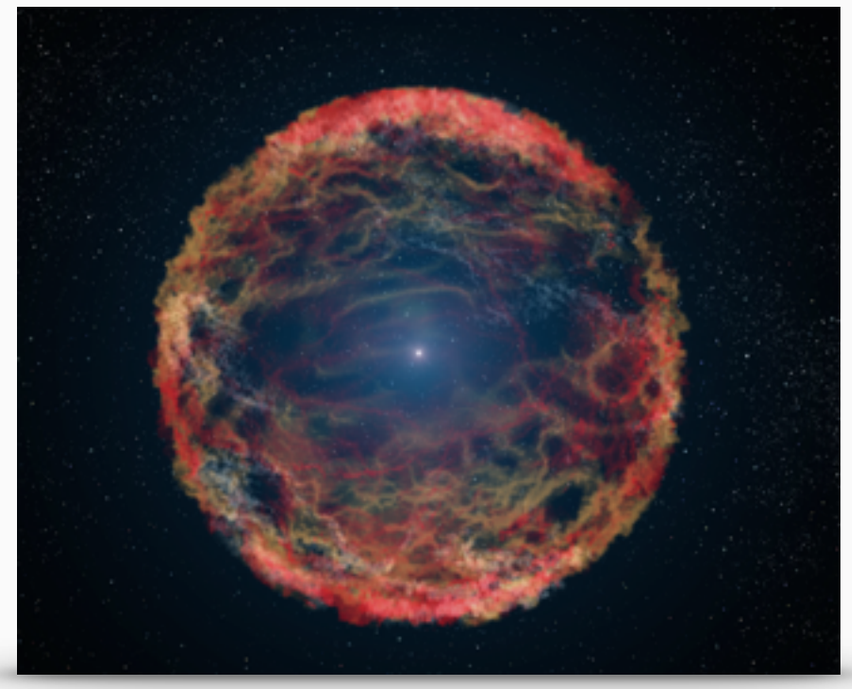
Things that can be computed, period.

Things a regular computer can compute before the sun goes supernova

Things a domino computer could compute before the sun goes supernova

Remember Random Search? It took *way* longer with a longer list.

Growth Rates: The Point



Remember Random Search?
It took *way* longer with a longer list.

The Point: we want to know how many things we have to do as our input grows, because we want to know *what problems are solvable before the sun goes poof! (and which ones will take the drop of a hat)*

Growth Rate: Definition

1. **Definition:** The *growth rate* of an algorithm is the number of primitive operations an algorithm must execute, in the worst case, in order to complete its job.

2. We call it the *growth rate* because it's how the number of operations *grows* as the size of our input grows.

I.e. sort a length 2 list vs. sorting a length 203487 list

Wait… “Primitive” Operation?

Not all operations are equal! For instance, on last weeks homework:

$$N \times M \longrightarrow \underbrace{N + N + \ldots + N}_{M \text{ times}}$$

Wait… “Primitive” Operation?

Not all operations are equal! For instance, on last weeks homework:

$$N \times M \longrightarrow \underbrace{N + N + \ldots + N}_{M \text{ times}}$$

Idea: Our computers multiply, add, subtract, and check equality, *really fast*. We have hardware specifically dedicated to doing these super duper fast.

Wait… “Primitive” Operation?

Idea: Our computers multiply, add, subtract, and check equality, *really fast*. We have hardware specifically dedicated to doing these super duper fast.

These are roughly our “primitive” operators

Binary Search Growth Rate

We said the growth rate of Binary Search was *log(N)*….
(where *N* is the length of the list)

Binary Search Growth Rate

We said the growth rate of Binary Search was *log(N)*….
(where *N* is the length of the list)

Okay… Why again?

Binary Search Growth Rate

24,000 x 10 = 240,000

Recall base ten multiplication!

Binary Search Growth Rate

24,000 x 10 = 240,000

240,000 x 10 = 2,400,000

Recall base ten multiplication!

Binary Search Growth Rate

24,000 x 10 = 240,000

240,000 x 10 = 2,400,000

2,400,000 x 10 = 24,000,000

Recall base ten multiplication!

Binary Search Growth Rate

Multiplying by 2 just adds a zero!

24,000 x 10 = 240,000

240,000 x 10 = 2,400,000

2,400,000 x 10 = 24,000,000

Recall base ten multiplication!

Binary Search Growth Rate

(4 x 2 in base ten)

$100_2 \times 10_2$

Recall base **two** multiplication!

Binary Search Growth Rate

(8 x 2 in base ten) (16 in base ten)

$1000_2 \times 10_2 = 10000_2$

Recall base **two** multiplication!

Binary Search Growth Rate

$1000_2 \times 10_2 = 10000_2$

$10000_2 \times 10_2 = 100000_2$

Recall base **two** multiplication!

Binary Search Growth Rate

$1000_2 \times 10_2 = 10000_2$

$10000_2 \times 10_2 = 100000_2$

$100000_2 \times 10_2 = 1000000_2$

Recall base **two** multiplication!

Binary Search Growth Rate

Multiplying by 2 just adds a zero!

$1000_2 \times 10_2 = 10000_2$

$10000_2 \times 10_2 = 100000_2$

$100000_2 \times 10_2 = 1000000_2$

Recall base **two** multiplication!

Binary Search Growth Rate

$1000_2 / 10_2 = 100_2$

$10000_2 / 10_2 = 1000_2$

$100000_2 / 10_2 = 10000_2$

How about division?

Binary Search Growth Rate

Dividing by 2 just removes a zero!

$1000_2 / 10_2 = 100_2$

$10000_2 / 10_2 = 1000_2$

$100000_2 / 10_2 = 10000_2$

How about division?

Binary Search Growth Rate

Dividing by 2 just removes a zero!

1000000000000000000000_2

Q: For a length 22 binary number (assume a 1 followed by 21 0's), how many times do we need to divide to get to just a 1?

Binary Search Growth Rate

Dividing by 2 just removes a zero!

1000000000000000000000_2

A: 21 times

Q: For a length 22 binary number (assume a 1 followed by 21 0's), how many times do we need to divide to get to just a 1?

Binary Search Growth Rate

Dividing by 2 just removes a zero!

More generally: the *logarithm* is just repeated division.

Binary Search Growth Rate

Dividing by 2 just removes a zero!

More generally: the *logarithm* is just repeated division.

So *log(N)* is roughly the number of bits in *N*

Binary Search Growth Rate

Dividing by 2 just removes a zero!

More generally: the *logarithm* is just repeated division.

So *log(N)* is roughly the number of bits in *N*

If *N* is base two, then it's log base two.
If *N* is base ten, then it's log base ten.

Clicker Question!

Q: What's a decent approximation for $log_{10}(45,728,102)$?

Clicker Question!

Q: What's a decent approximation for $log_{10}(45,728,102)$?

[A] 456 [B] 8

[C] 102 [D] 10 [E] 4,572,810

Clicker Answer!

Q: What's a decent approximation for $log_{10}(45,728,102)$?

[A] 456 **[B] 8**

[C] 102 [D] 10 [E] 4,572,810

Since there are 8 digits!

Clicker Answer!

Q: What's a decent approximation for $log_{10}(45,728,102)$?

[A] 456 **[B] 8**

[C] 102 [D] 10 [E] 4,572,810

Actual answer: 7.66

Binary Search Growth Rate

- Each step in Binary Search cuts the list in *half*.
- Q: How many times can we cut something in half?
- If that thing is length *N*, then we can cut it in half $log_2(N)$ times.
- Thus, the growth rate of Binary Search is *log(N)!*

Binary Search Growth Rate

- Do we see why we like Binary Search more?
- Consider *N = 108231027397342983741092812 3*
- Would you rather do *N* operations, or do the number of operations that's the same as the number of digits in *N*?
- *log(N)* is way smaller!

Back to Sorting

Problem Specification

- *Input:*
  - a collection of *orderable* objects, call it “Basket”
- *Output:*
  - “Basket”, where each item is in order.

Random Sort

1. Shuffle the list up randomly (like shuffling a deck).

2. Check to see if the list is in order. If it is, return the list.

3. If it is not, repeat from step 1.

Selection Sort

1. “Select” the smallest item in the list.

2. Put it at the beginning.

3. “Select” the second smallest item.

4. Put it 2nd from the beginning.

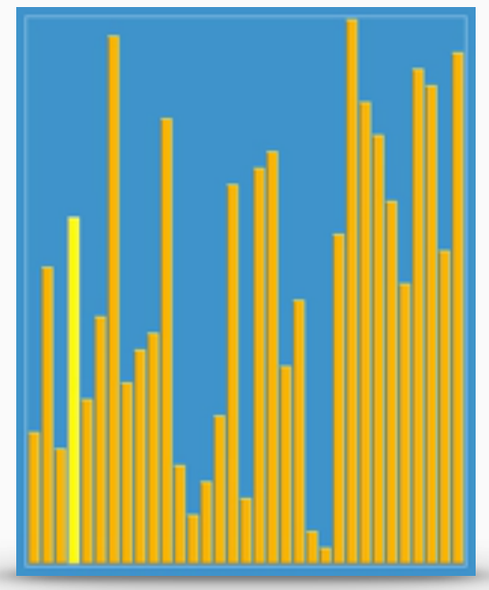
5. Rinse and repeat….

Clicker Question

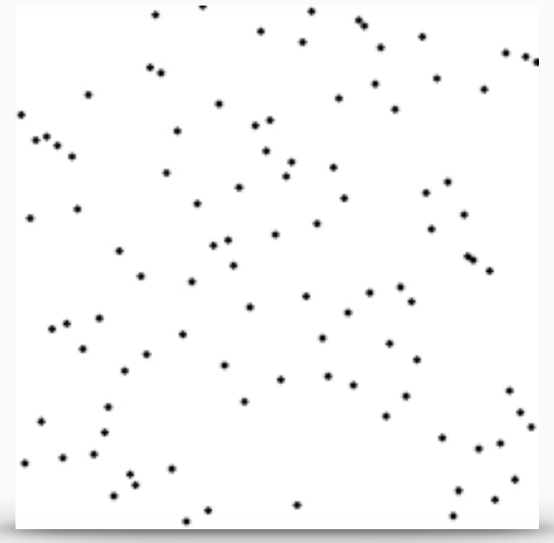
Q: Which of these is a visualization of Selection Sort?

Clicker Question

[A]

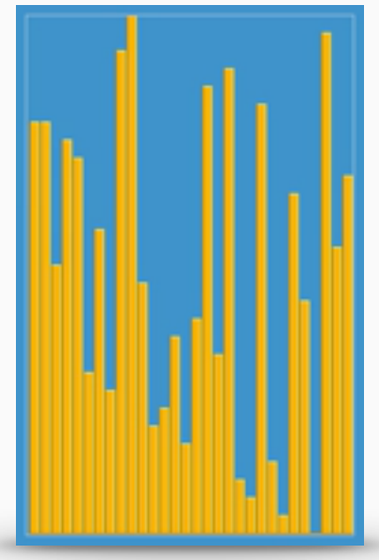


[B]



[C]

[D]

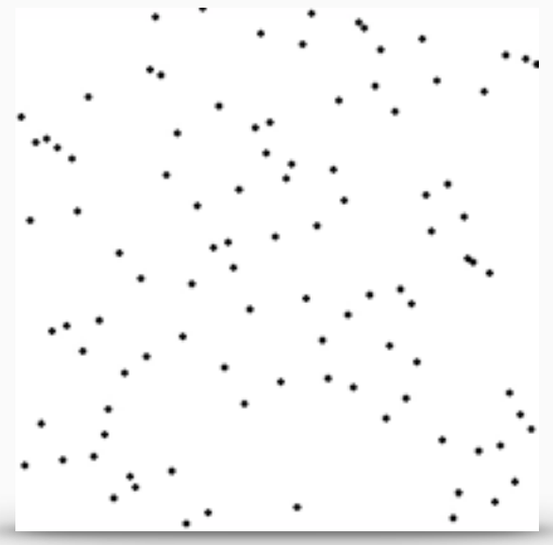


Q: Which of these is a visualization of Selection Sort?

Clicker Question

[A]

[B]



[C]

[D]

Q: Which of these is a visualization of Selection Sort?

Selection Sort

1. “Select” the smallest item in the list.

2. Put it at the beginning.

3. “Select” the second smallest item.

4. Put it 2nd from the beginning.

5. Rinse and repeat….

Selection Sort

1. “Select” the smallest item in the list.

Q: What is the growth rate of this step?

Selection Sort

1. “Select” the smallest item in the list.

Q: What is the growth rate of this step?

A: *N*, since we have an unsorted list, and we’re *basically* searching through it once.

Selection Sort

Q1: Does Selection Sort *halt* for every possible input?

Selection Sort

Q1: Does Selection Sort *halt* for every possible input?

A: yes!

Selection Sort

Q1: Does Selection Sort *halt* for every possible input?

A: yes!

Q2: Is Selection Sort *correct?*

Selection Sort

Q1: Does Selection Sort *halt* for every possible input?

A: yes!

Q2: Is Selection Sort *correct?*

A: yes!

Selection Sort

Q1: Does Selection Sort *halt* for every possible input?

A: yes!

Q2: Is Selection Sort *correct?*

A: yes!

Q3: What is the growth rate of Selection Sort?

Selection Sort

Q1: Does Selection Sort *halt* for every possible input?

A: yes!

Q2: Is Selection Sort *correct?*

A: yes!

Q3: What is the growth rate of Selection Sort?

Clicker question! (take a stab!)

Clicker Question!

[A] N

[B] N^2

[C] 2N

[D] 4N

[E] N^3

Q3: What do you think the growth rate of Selection Sort is?

Clicker Answer!

[A] N

[B] N^2

[C] 2N

[D] 4N

[E] N^3

Q3: What do you think the growth rate of Selection Sort is?

Selection Sort

- Do this for the first smallest, second smallest, all the way up to Nth smallest:
  - “Select” the current smallest item in the list.
  - Put it at the beginning.

Selection Sort

- Do this for the first smallest, second smallest, all the way up to Nth smallest:
  - **“Select” the current smallest item in the list.**
  - Put it at the beginning.

takes N steps

Selection Sort

- Do this for the first smallest, second smallest, all the way up to Nth smallest:
  - **“Select” the current smallest item in the list.**
  - Put it at the beginning.

takes N steps

doing this N times

Selection Sort

- Do this for the first smallest, second smallest, all the way up to Nth smallest:
  - **“Select” the current smallest item in the list.**
  - Put it at the beginning.

takes N steps

doing this N times

So, $N \times N = N^2$

Selection Sort

Let's take a look!

Dancing Selection Sort

A Third Problem:
Can this logical formula be true?

- *Input:*
  - a logical formula
- *Output:*
  - True if there is some way we can make the formula true
  - False if there is no way we can make the formula true

Can this logical formula be true?

I give you: *AND(P,Q)*

Q: Can this be true?

(our tools are how we set the truth value of *P, Q*)

Can this logical formula be true?

I give you: *AND(P,Q)*

Q: Can this be true?

A: Sure! *P = True, Q = True*

(our tools are how we set the truth value of *P, Q*)

Can this logical formula be true?

- Instead of our growth rate capturing the length of the list, now it measures the *number of atomic sentences, e.g. P, Q, R, etc.*

- So, how hard is this problem?

- Let's try a harder one…

Clicker Question!

Q: Can the following logical formula be made True?

NOT(OR(AND(OR(P,Q),NOT(R)),R))

or if you'd prefer the other way of writing things…

not(((P or Q) and not(R)) or R)

[A] Yes, I'm very confident! [B] No, I'm very confident!

[C] I'm guessing randomly!

Clicker Answer!

Q: Can the following logical formula be made True?

NOT(OR(AND(OR(P,Q),NOT(R)),R))

or if you'd prefer the other way of writing things…

not(((P or Q) and not(R)) or R)

A: Yes! *P = True, Q = True, R = False*

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

(P and Q) or not(P)

Algorithm 1:
Can this logical formula be true?

(P and Q) or not(P)

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

P = True, Q = False

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

(P and Q) or not(P)

P = True, Q = False

.....(True and False) or not(True)

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

(P and Q) or not(P)

P = True, Q = False

(True and False) or not(True)

report True!

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

(P and Q) or not(P)

P = True, Q = False

(True and False) or not(True)

report True! ✓

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

The usual questions…

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

The usual questions… Q: Does it halt?

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

The usual questions… Q: Does it halt? A: No!

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

(what if there is no true assignment?)

The usual questions… Q: Does it halt? A: No!

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

The usual questions… Q: It is *correct?* A: No!

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

(what if there is no true assignment?)

The usual questions… Q: It is *correct?* A: No!

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

The usual questions… Q: What's the growth rate?

Algorithm 1:
Can this logical formula be true?

Random Checker

1. Pick a random assignment for each atomic sentence's truth value.

2. Ask, is the sentence True?

3. If it is, report True.

4. If not, repeat from step 1.

The usual questions…

Q: What's the growth rate?

A: Actually hard to say…

Reflection

- Binary search!
  - Twenty Questions
  - Why log(N)?
- Growth Rates
- A new problem: *logical satisfiability*
- Next time: an *unsolvable problem*!

Algorithm 2:
Can this logical formula be true?

Build The Truth Table! (Brute Force)

1. *Build the truth table for the logical formula*
2. *Check to see if it has any row that is True.*
3. *If it does, report True.*
4. *If not, report False.*