Unit 3: Algorithms

Dave Abel

February 15th, 2016

Telescope Science

"Computer Science is no more about computers than astronomy is about telescopes."
- Dijsktra (possibly)

What is Computer Science?

- Abstraction

- Problem Solving!

- Artistic, Creative.
  - E.g. Digital Media, Electronic Music, Games, Animation.
- Science.
  - E.g. Understand and model reality.

- **World Changing!**

Algorithms: Takeaway

- **Definition:** An *algorithm* is a recipe for solving a problem.
- Computer science is (loosely) the study of algorithms.

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.*

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!!!**

Outline

- Algorithms Overview
- Your first algorithm: Search
  - Three flavors of search (Random, Linear, Binary)
- Growth Rates
- Your second algorithm: Sorting
  - Two flavors of sorting (Random, Selection)

Problem Specification

- **A specification defines a problem**

Problem Specification

- **A specification defines a problem**
- **An algorithm solves a problem**

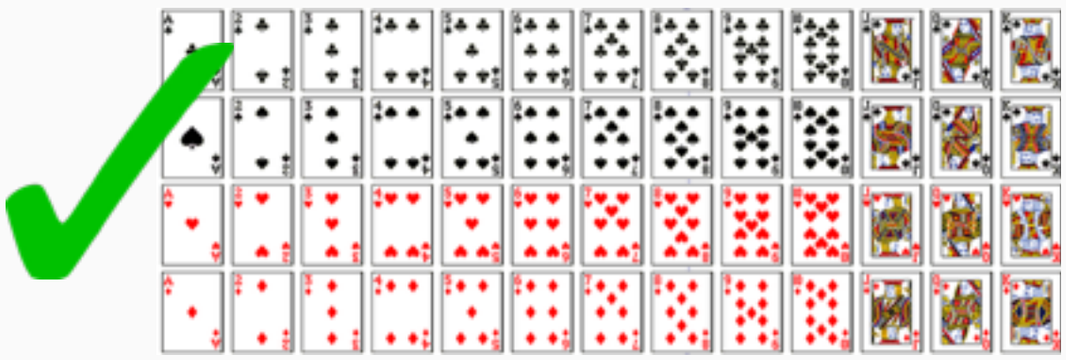
Problem Specification

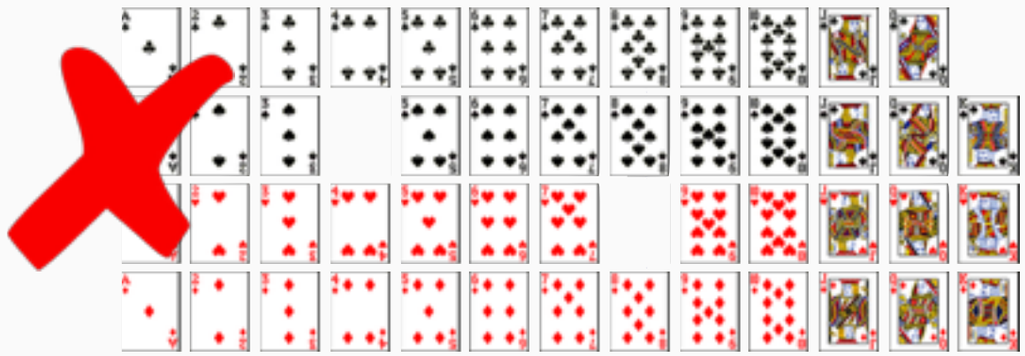
- **A specification defines a problem**
- **An algorithm solves a problem**
- *INPUT: A deck of cards*

Problem Specification

- **A specification defines a problem**
- **An algorithm solves a problem**
- *INPUT: A deck of cards*

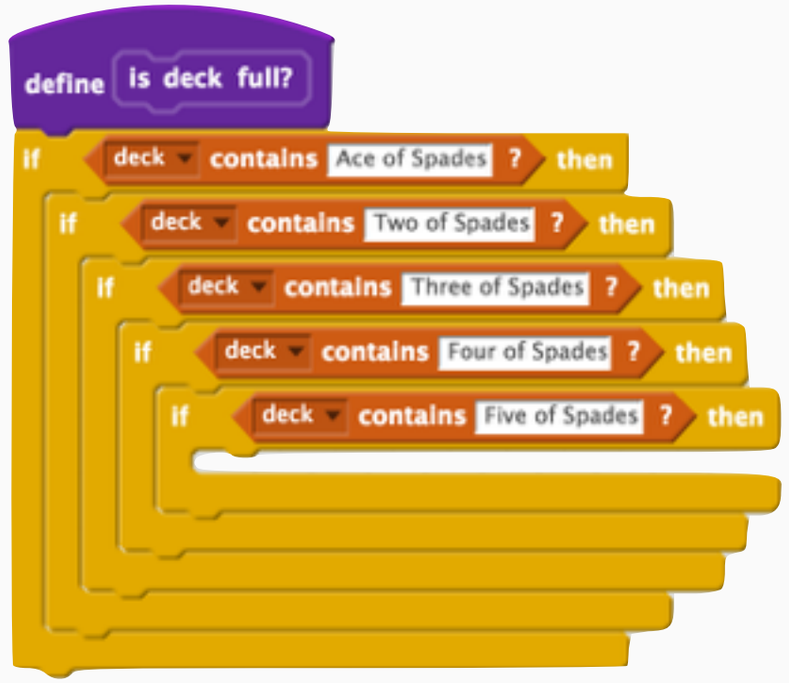
- *OUTPUT: True if the input desk is a complete deck, False otherwise.*





Problem Specification

- *INPUT: A deck of cards*



...

- *OUTPUT: True if the input desk is a complete deck, False otherwise.*

Problem Specification

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

Problem Specification Examples

- *INPUT: Two numbers, X and Y.*
- *OUTPUT: A single number, Z, such that Z = X + Y.*

Problem Specification Examples

- *INPUT: Some Doctor's knowledge about cancer.*
- *OUTPUT: Cure to cancer*

Problem Specification Examples

- *INPUT: The Internet*

- *OUTPUT: The winner of the 2016 election*

Problem Specification Examples

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

Problem Specification Examples

- *INPUT: Data from the stock market.*
- *OUTPUT: Correct predictions about the market.*

Problem Specification Examples

- *INPUT: A bunch of songs from the last 1000 years.*
- *OUTPUT: A new song, guaranteed to be loved.*

Problem Specification Examples

- *INPUT: a number.*
- *OUTPUT: a 0 or a 1, each with equal chance.*

Problem Specification: Abstraction

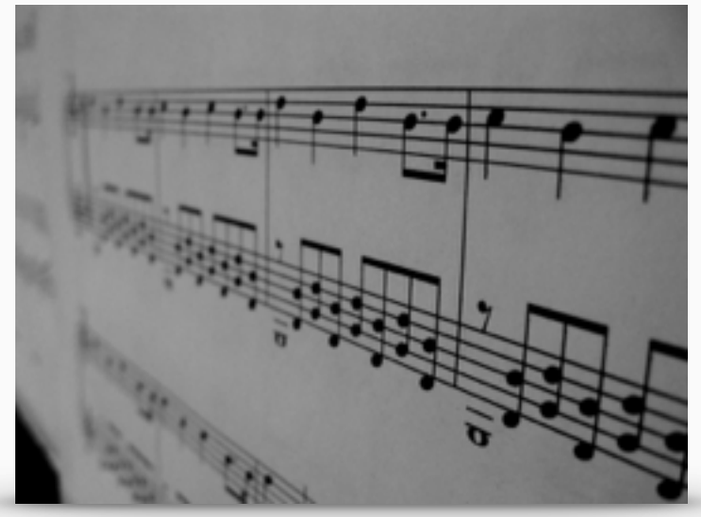
- All of these specifications are extremely nice! But with a computer, as with logic, we need to operate on well defined things.

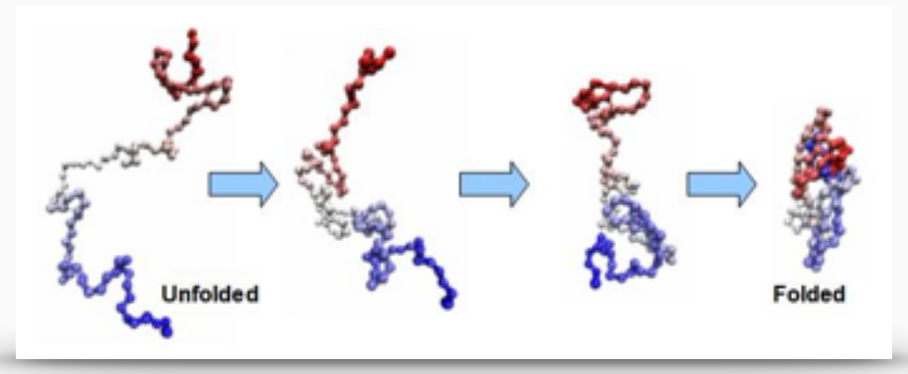
- I.e. a computer would ask, “what’s a stock? what’s mars?”

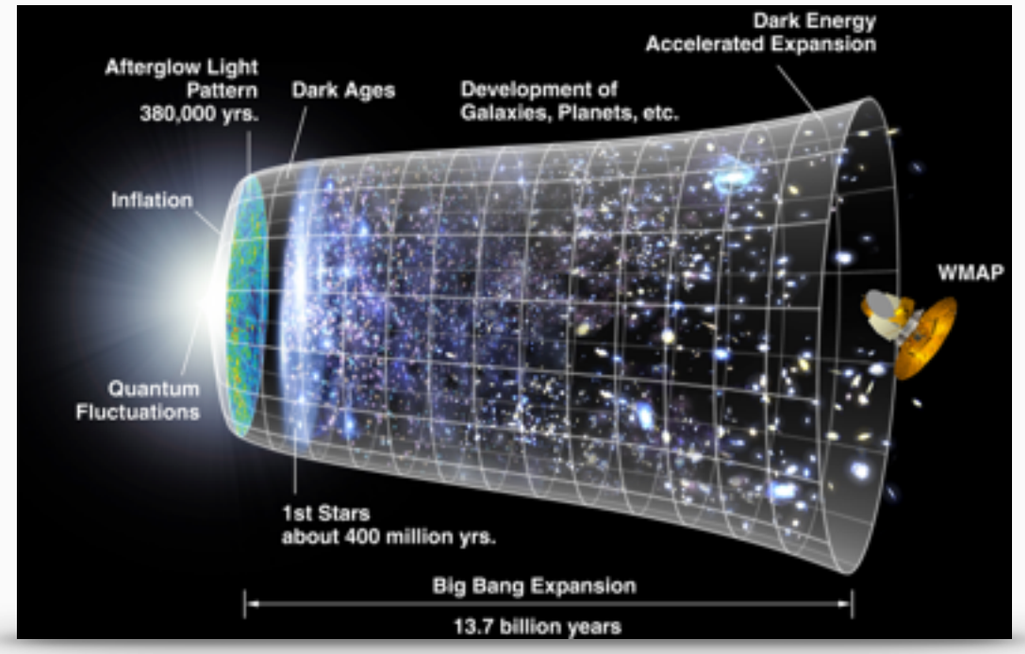
- So, our algorithms are defined with respect to well defined things, like lists, numbers, etc.! We abstract away the details.

- Then we model reality using these abstractions.

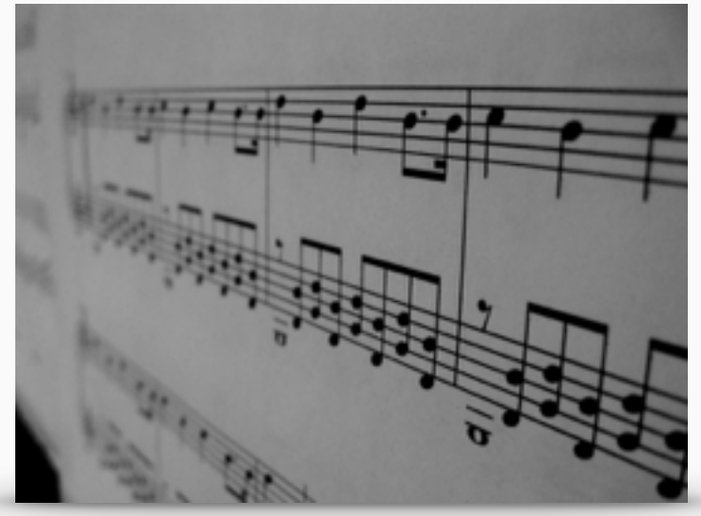
Problem Specification



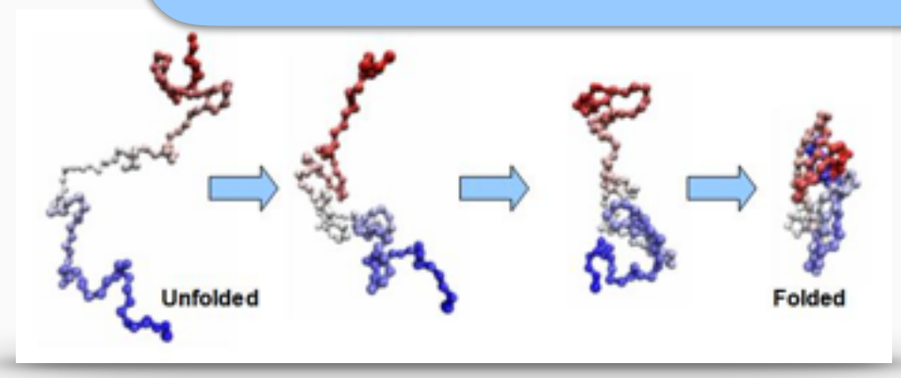


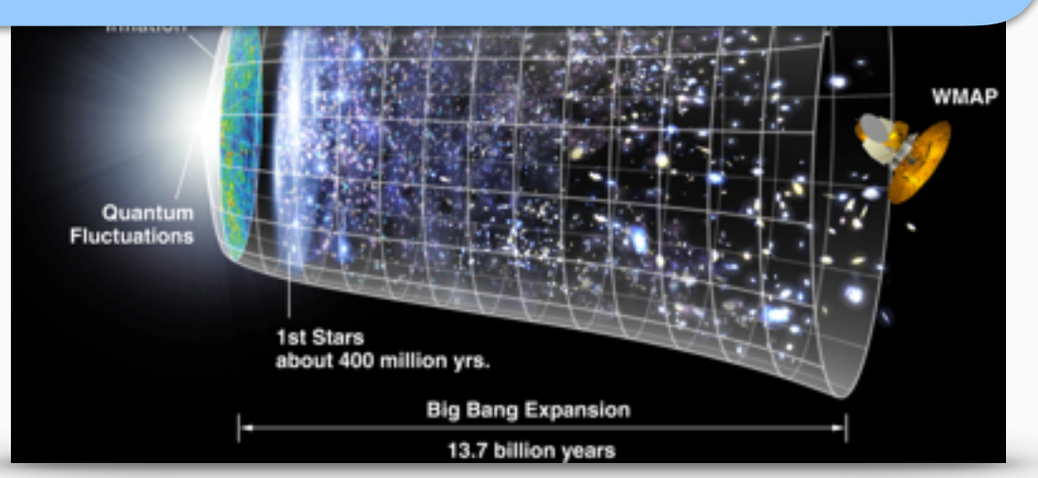


Algorithms:

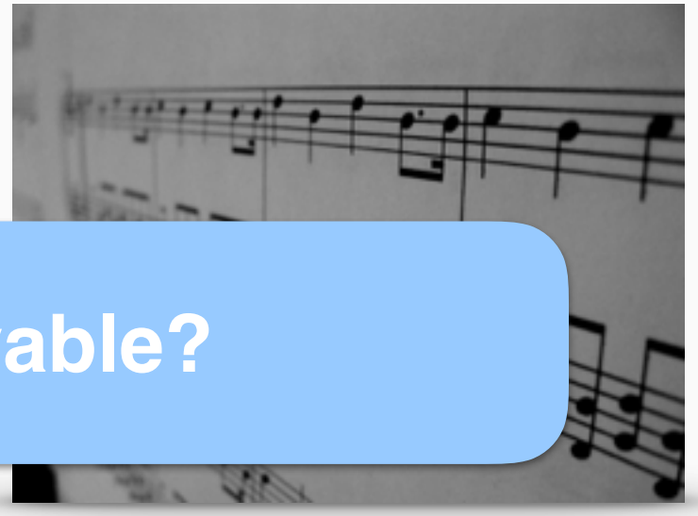


(1) Which of these problems are solvable?



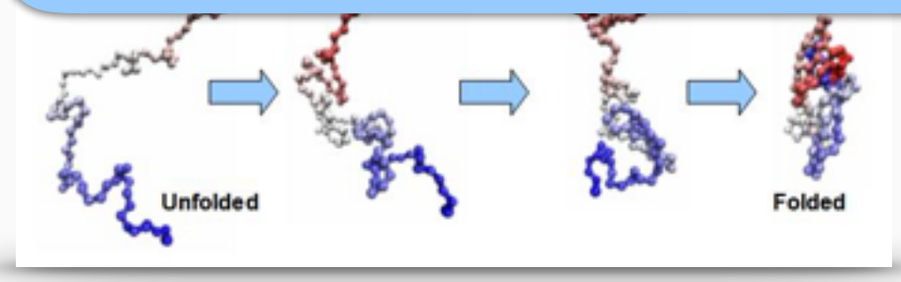


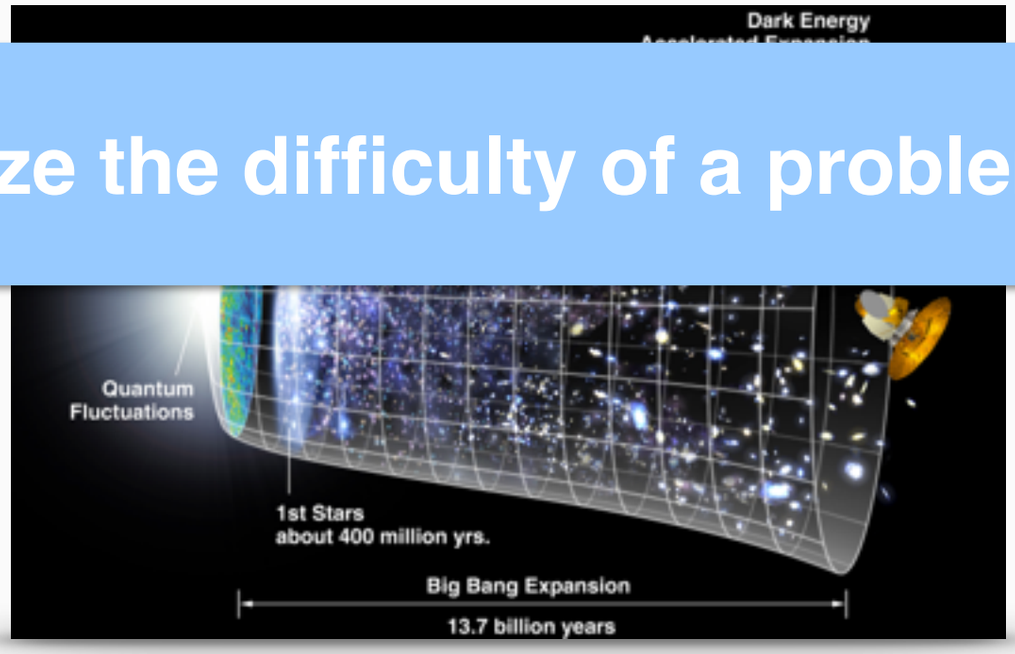
Algorithms:



(1) What problems are solvable?

(2) How can we characterize the difficulty of a problem?





Our First Problem: Search

- *Input:*
  - a collection 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”

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”

Search Algorithm #1

- **Random search**

  1. Pick a random item from “Basket”.

  2. If it’s the item we’re looking for (“Snozzberry”), report True!

  3. Otherwise, go back to step 1.

Clicker Question!

- Q: Does random search solve the search problem?

Random search

1. Pick a random item from “Basket”.

2. If it’s the item we’re looking for (“Snozzberry”), report True!

3. Otherwise, go back to step 1.

Search Problem

- *Input:*
  - a collection 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”

Clicker Question!

- Q: Does random search solve the search problem?

Random search

1. Pick a random item from “Basket”.

2. If it’s the item we’re looking for (“Snozzberry”), return True!

3. Otherwise, go back to step 1.

Search Problem

- *Input:*
  - a collection 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”

[A] Yes! [B] No! [C] I have no idea…

Clicker Answer!

- Q: Does random search solve the search problem?

Random search

1. Pick a random item from “Basket”.

2. If it’s the item we’re looking for (“Snozzberry”), return True!

3. Otherwise, go back to step 1.

Search Problem

- *Input:*
  - a collection 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”

[A] Yes! **[B] No!** [C] I have no idea…

Clicker Answer!

- Q: Does random search solve the search problem?

Random search

1. Pick a random item from “Basket”.

2. If it’s the item we’re looking for (“Snozzberry”), return True!

3. Otherwise, go back to step 1.

Search Problem

- *Input:*
  - a collection 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”

Q: What if the item is not in “Basket”?

[A] Yes! **[B] No!** [C] I have no idea…

Search Algorithm #2

- **Linear search**

1. Put the items from “Basket” in a list

2. Check each item in turn (index 1, then index 2, and so on)

3. If at any point the index we’re looking at in the list contains the item, report True!

4. If we get to the end of the list and haven’t seen it, report False!

Search Algorithm #2

- **Linear search**

  1. Put the items from “Basket” in a list

  2. Check each item in turn (index 1, then index 2, and so on)

  3. If at any point the index we’re looking at in the list contains the item, report True!

  4. If we get to the end of the list and haven’t seen it, report False!

Ask: is “lime” in the list?

Search Algorithm #2

- **Linear search**

1. Put the items from “Basket” in a list

2. Check each item in turn (index 1, then index 2, and so on)

3. If at any point the index we’re looking at in the list contains the item, report True!

4. If we get to the end of the list and haven’t seen it, report False!

Ask: is “lime” in the list?

Search Algorithm #2

- **Linear search**

1. Put the items from “Basket” in a list

2. Check each item in turn (index 1, then index 2, and so on)

3. If at any point the index we’re looking at in the list contains the item, report True!

4. If we get to the end of the list and haven’t seen it, report False!

Ask: is “lime” in the list?

Search Algorithm #2

- **Linear search**

1. Put the items from “Basket” in a list

2. Check each item in turn (index 1, then index 2, and so on)

3. If at any point the index we’re looking at in the list contains the item, report True!

4. If we get to the end of the list and haven’t seen it, report False!

Ask: is “lime” in the list?

Search Algorithm #2

- **Linear search**

  1. Put the items from “Basket” in a list

  2. Check each item in turn (index 1, then index 2, and so on)

  3. If at any point the index we’re looking at in the list contains the item, report True!

  4. If we get to the end of the list and haven’t seen it, report False!

Ask: is “lime” in the list?

Search Algorithm #2

Linear search

1. Put the items from “Basket” in a list
2. Check each item in turn (index 1, then index 2, and so on)
3. If at any point the index we’re looking at in the list contains the item, report True!
4. If we get to the end of the list and haven’t seen it, report False!

Search Problem

- *Input:*
  - a collection 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”

Q: Does Linear Search solve the Search Problem?

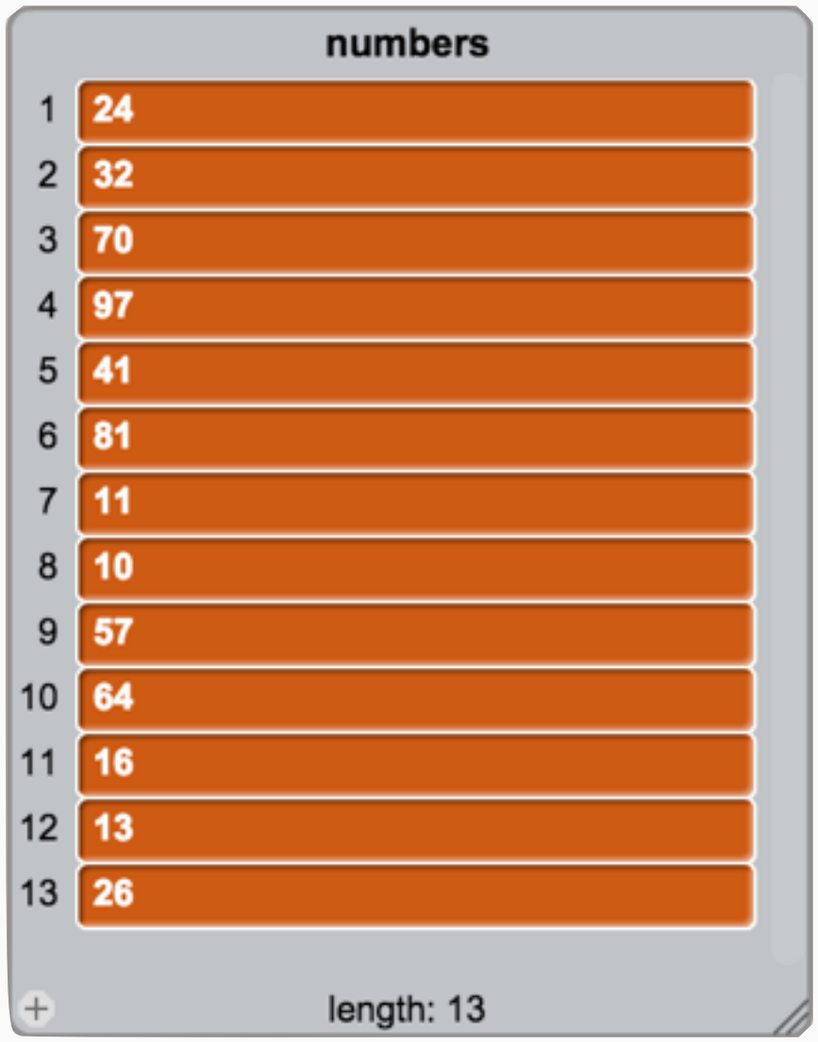
A: Yes! *For any list, for any item, linear search will solve Search!*

Search Algorithm #3

- **Binary Search:** ***assumes a sorted list***

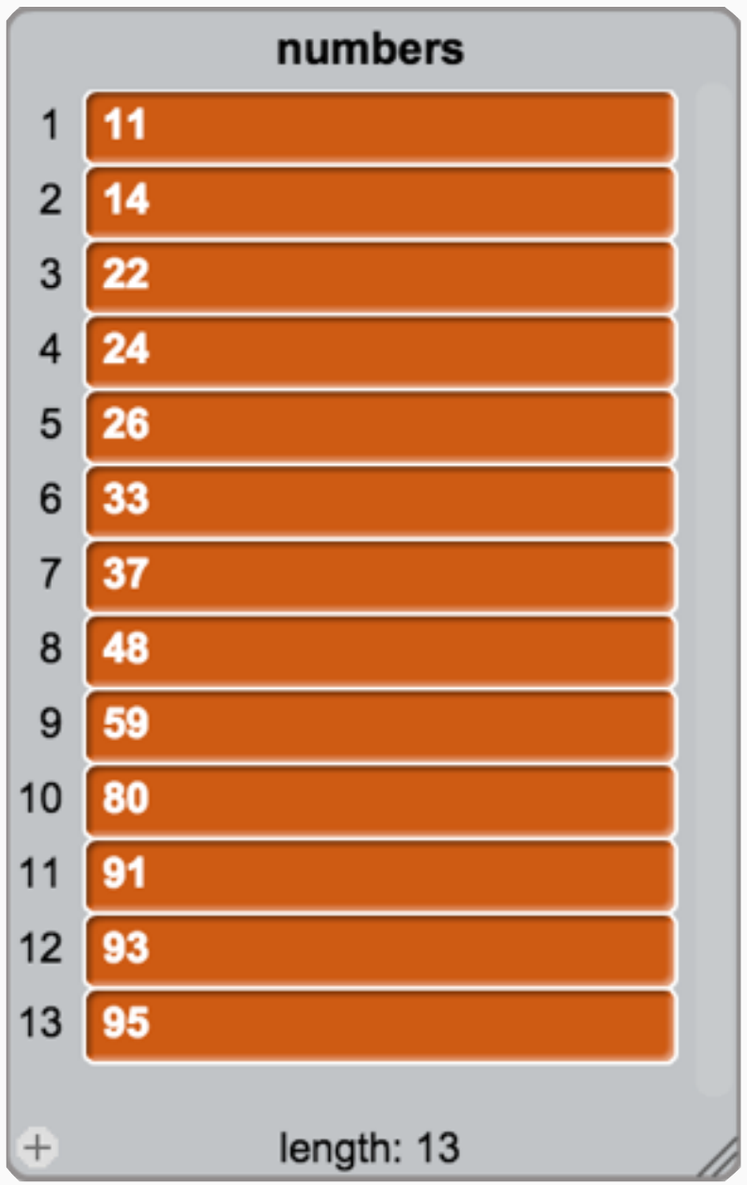
- Idea: if we assume the list is sorted, surely finding our item is easier!

You Try It



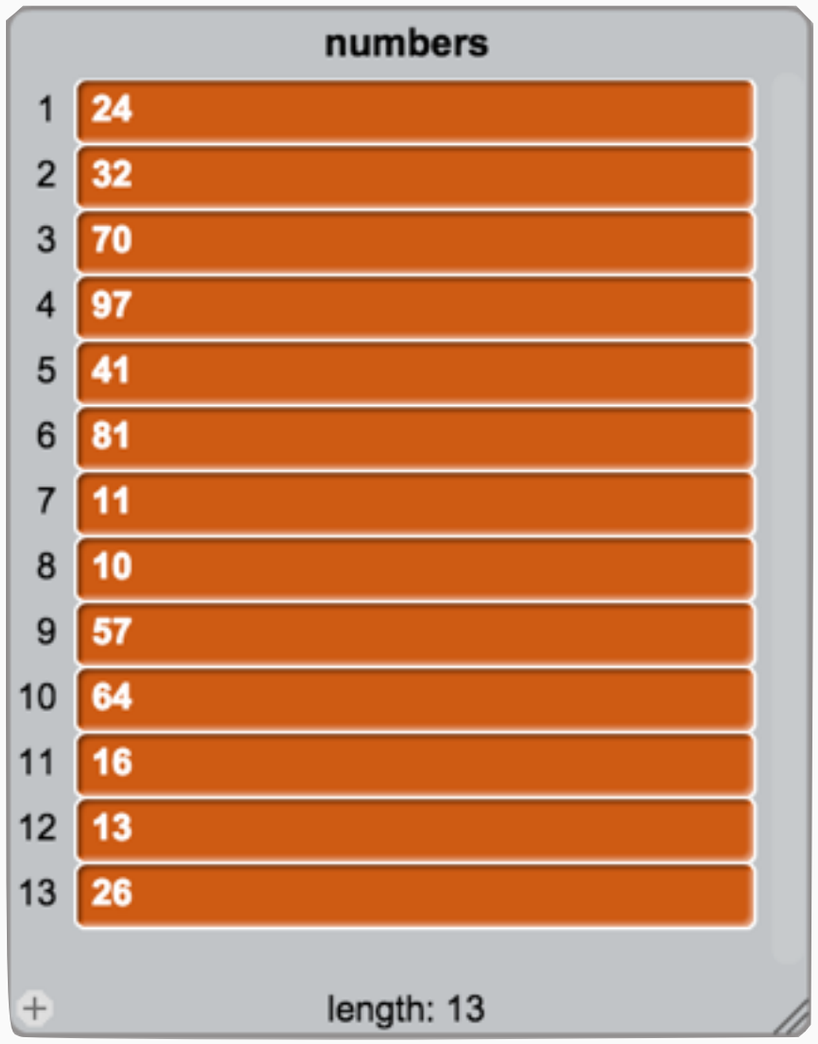
Q: Is 16 in the list?

You Try It

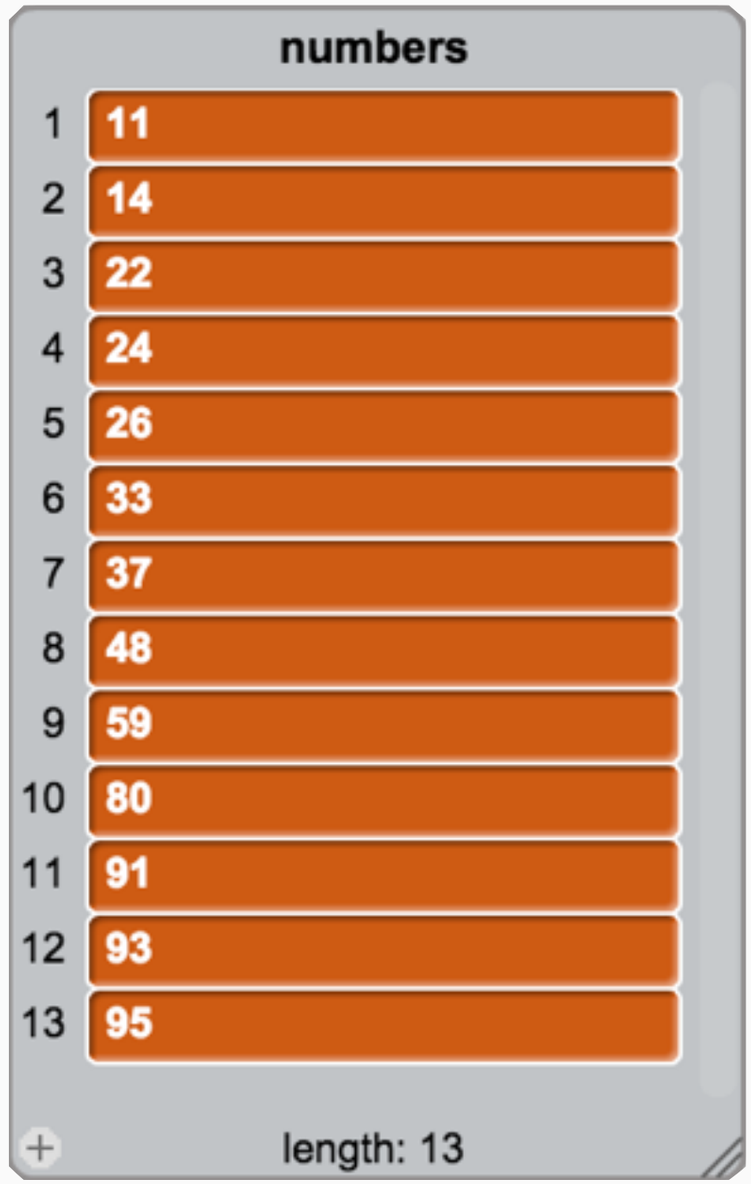


Q: Is 91 in the list?

Which Was Easier?



Q: Is 16 in the list?



Q: Is 91 in the list?

Search Algorithm #3

- **Binary Search:** ***assumes a sorted list***

1. Check the middle of the list

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

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

4. If greater, search the right half.

5. If less, search the left half.

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

1	3	4	5	7	8	9

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

1	3	4	5	7	8	9

Q: is 3 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

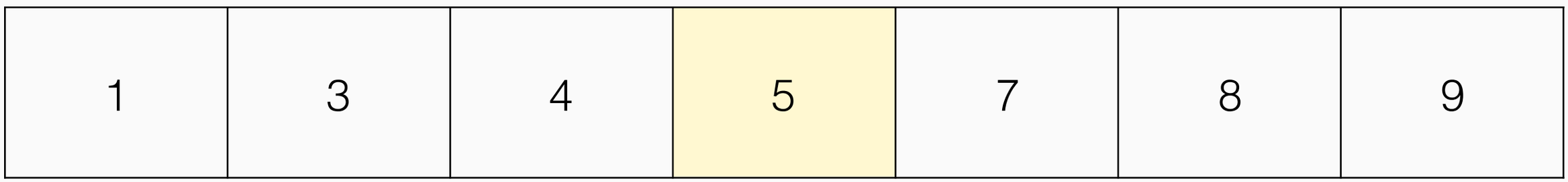
  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.



Q: is 3 in the list?

Binary Search

- **Binary Search: *assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

1	3	4	5	7	8	9

Q: is 3 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half. $3 < 5$

  5. If less, search the left half.

1	3	4	5	7	8	9

Q: is 3 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half. $3 < 5$

  5. If less, search the left half.

1	3	4	5	7	8	9

Q: is 3 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

Because list is sorted, if our number is in the list, it has to be to the left of 5!!!

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

4. If greater, search the right half.

5. If less, search the left half.

$3 < 5$

1	3	4	5	7	8	9

Q: is 3 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

$3 < 5$

1	3	4	~~5~~	~~7~~	~~8~~	~~9~~

Q: is 3 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

1	3	4	~~5~~	~~7~~	~~8~~	~~9~~

Q: is 3 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

1	3	4	5	7	8	9

Q: Is 6 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

1	3	4	5	7	8	9

Q: Is 6 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

$5 < 6$

1	3	4	5	7	8	9

Q: Is 6 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.

$5 < 6$

~~1~~	~~3~~	~~4~~	~~5~~	7	8	9

Q: Is 6 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

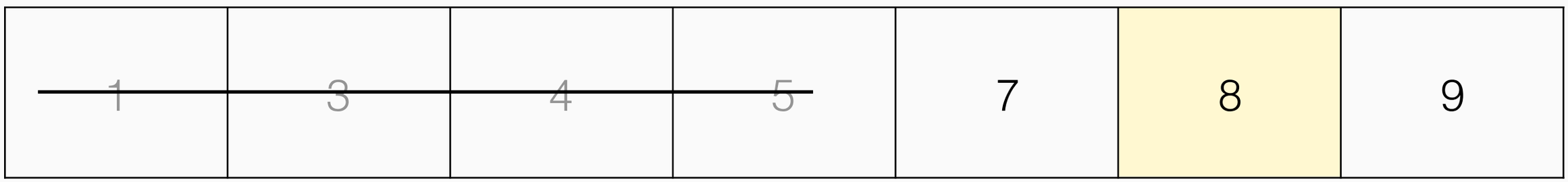
  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.



Q: Is 6 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

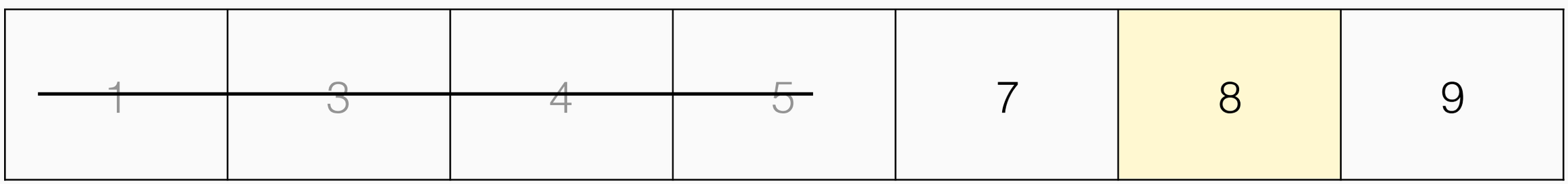
  1. Check the middle of the list

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

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

  4. If greater, search the right half. $6 < 8$

  5. If less, search the left half.



Q: Is 6 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

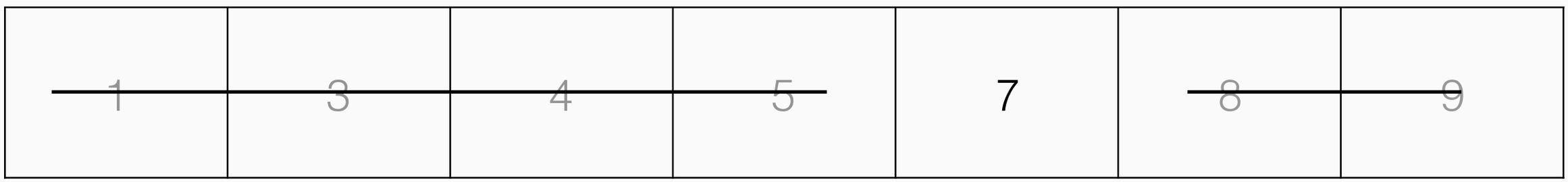
  1. Check the middle of the list

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

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

  4. If greater, search the right half. $6 < 8$

  5. If less, search the left half.



Q: Is 6 in the list?

Binary Search

- **Binary Search:** ***assumes a sorted list***

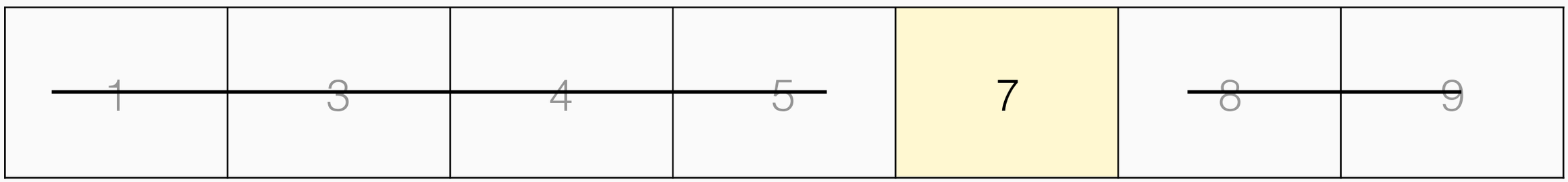
  1. Check the middle of the list

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

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

  4. If greater, search the right half.

  5. If less, search the left half.



Q: Is 6 in the list?

Binary Search

Another way of thinking about it:

Linear Search = check every item in the worst case!

Binary Search = uses sorted property to avoid checking every item

1	3	4	5	7	8	9

Q: Is 6 in the list?

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

1	3	4	5	7	8	9	11	12	14	16

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 [D] 4 [E] 5

1	3	4	5	7	8	9	11	12	14	16

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 **[D] 4** [E] 5

1	3	4	5	7	8	9	11	12	14	16

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 **[D] 4** [E] 5

1	3	4	5	7	8	9	11	12	14	16

Inspections: 1

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 **[D] 4** [E] 5

1	3	4	5	7	~~8~~	~~9~~	~~11~~	~~12~~	~~14~~	~~16~~

Inspections: 1

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 **[D] 4** [E] 5

1	3	4	5	7	~~8~~	~~9~~	~~11~~	~~12~~	~~14~~	~~16~~

Inspections: 2

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 **[D] 4** [E] 5

~~1~~	~~3~~	~~4~~	5	7	~~8~~	~~9~~	~~11~~	~~12~~	~~14~~	~~16~~

Inspections: 2

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 **[D] 4** [E] 5

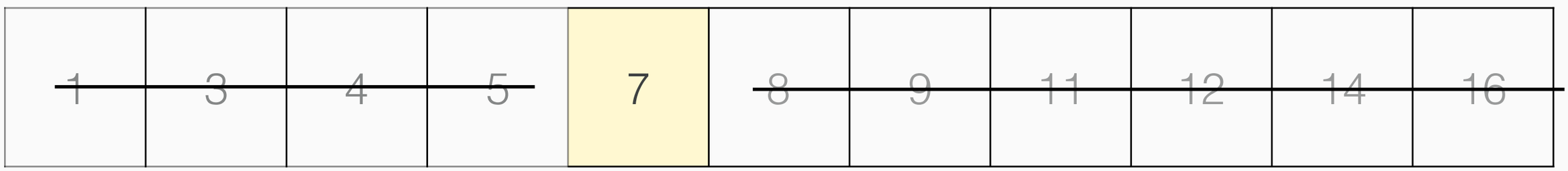
~~1~~	~~3~~	~~4~~	5	7	~~8~~	~~9~~	~~11~~	~~12~~	~~14~~	~~16~~

Inspections: 3

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 **[D] 4** [E] 5

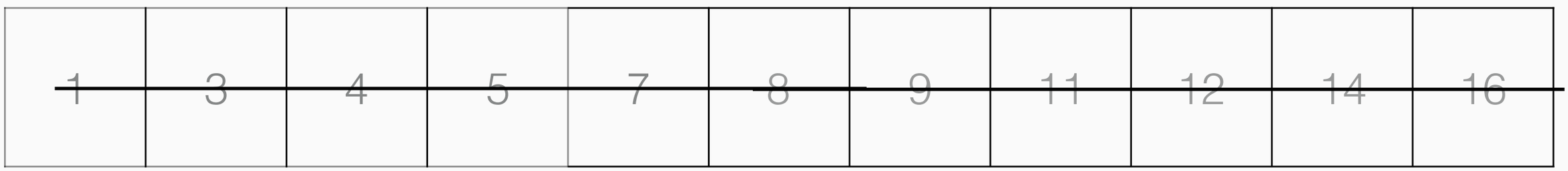


Inspections: 4

Clicker Question!

Q: How many items will Binary Search inspect when searching for 6?

[A] 1 [B] 2 [C] 3 **[D] 4** [E] 5



Inspections: 4

Properties of Algorithms

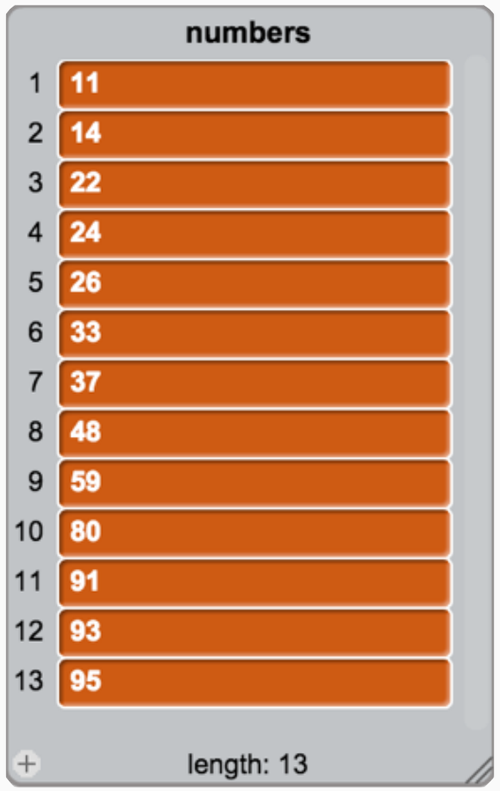
1. Correctness: does the algorithm satisfy the problem specification?

2. Growth Rate: how many “primitive” operations must the computer execute to solve the problem for various sized inputs?

Growth Rates

- **Linear Search vs. Binary Search**

- Well we already said that Binary is faster, but by how much?



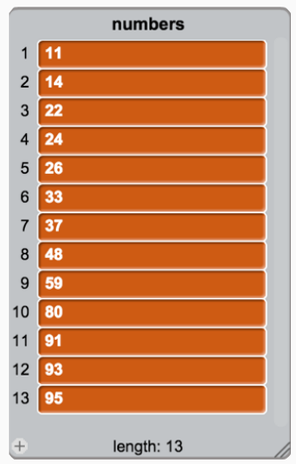
Growth Rates

- **Linear Search vs. Binary Search**

- Well we already said that Binary is faster, but by how much?

- We measure what is called the *growth rate* of an algorithm: how many operations do we need in order to solve the problem?

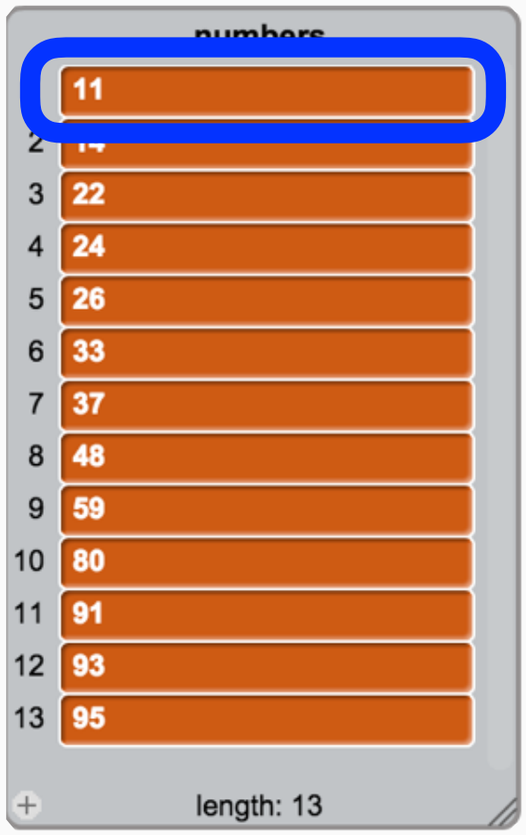
Q: Is 11 in the list?



Linear:

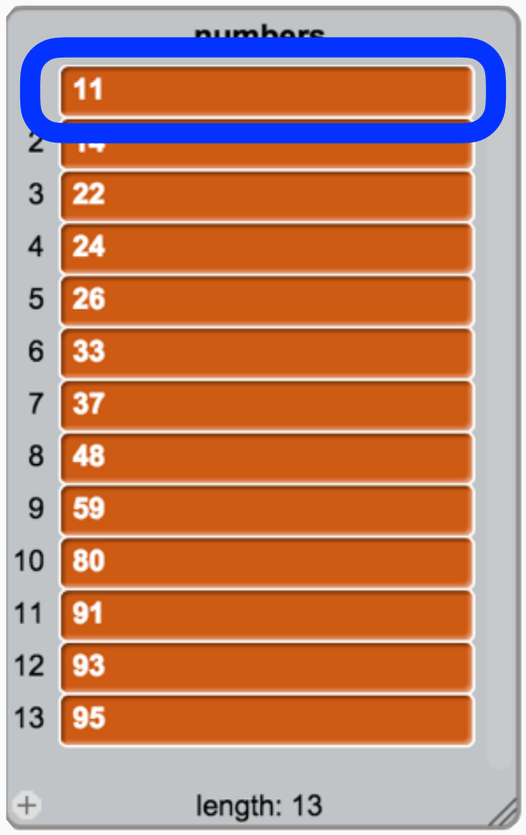
Q: Is 11 in the list?

Operations: 1



Linear:

Q: Is 11 in the list?

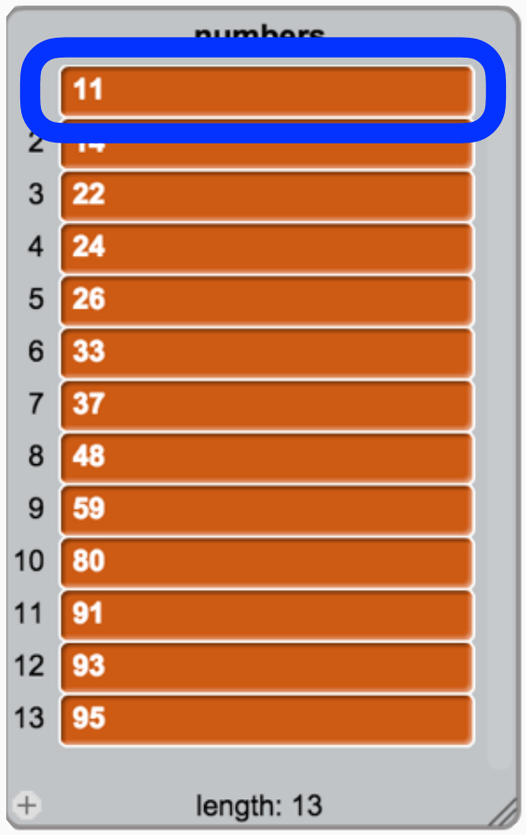


Operations: 1

Linear:

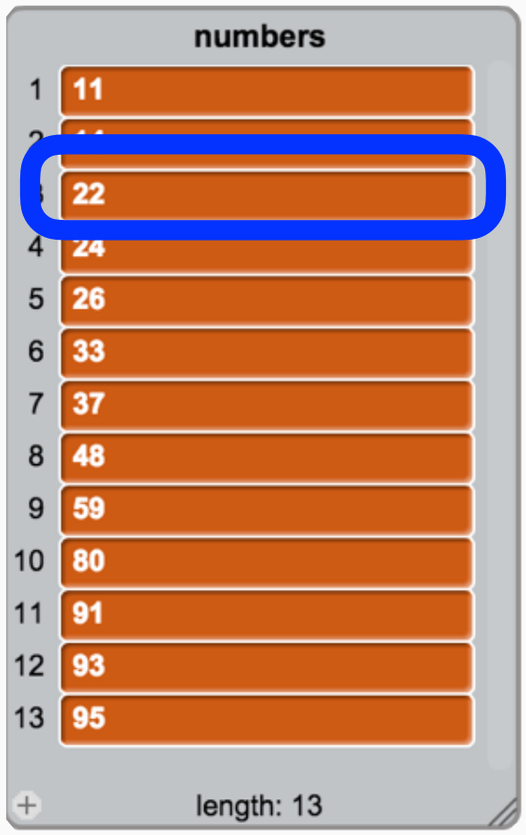
Q: Is 95 in the list?

Operations: 1



Linear:

Q: Is 95 in the list?

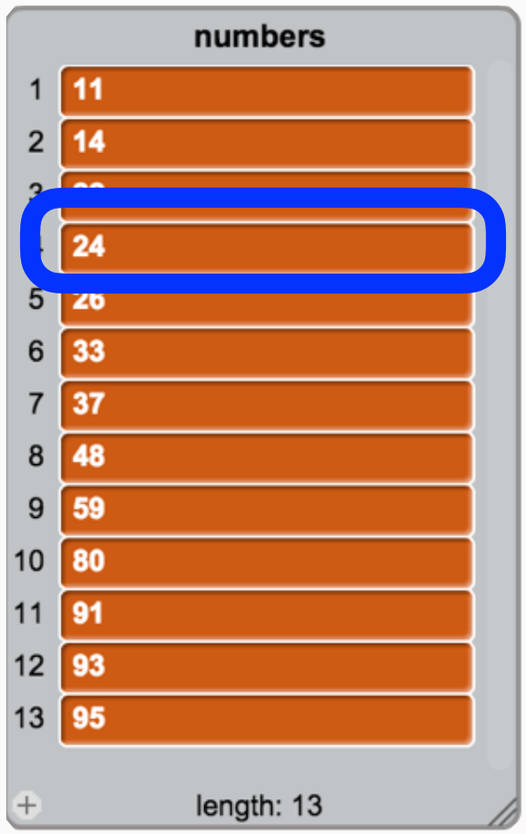


Operations: 2

Linear:

Q: Is 95 in the list?

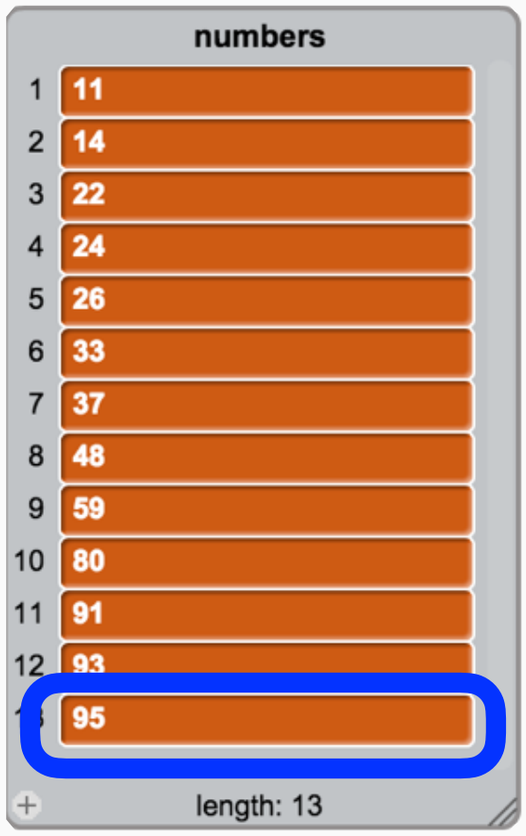
Operations: 3



Linear:

Q: Is 95 in the list?

Operations: 13



More Generally

- Growth Rate is always considered with respect to the worst possible case

- So the growth rate of Linear Search is:

More Generally

- Growth Rate is always considered with respect to the worst possible case

- So the growth rate of Linear Search is:
  - For a size 13 list, we might need to look at all 13 items…
  - For a size 20 list, we might need to look at all 20 items…

More Generally

- Growth Rate is always considered with respect to the worst possible case

- So the growth rate of Linear Search is:
  - For a size 13 list, we might need to look at all 13 items…
  - For a size 20 list, we might need to look at all 20 items…
  - **For a size N list, we might need to look at all N items…**

More Generally

- Growth Rate is always considered with respect to the worst possible case

- Binary Search:
  - Can repeatedly cut the list in half…
  - Worst case we need to cut it in half how many times?
  - Well if we have a length 16 list… (16 -> 8 -> 4 -> 2 -> 1)
  - **For a length N list, generally, this is log(N)**

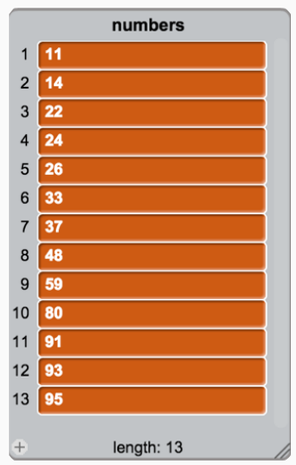
Growth Rates

- **Linear Search vs. Binary Search**

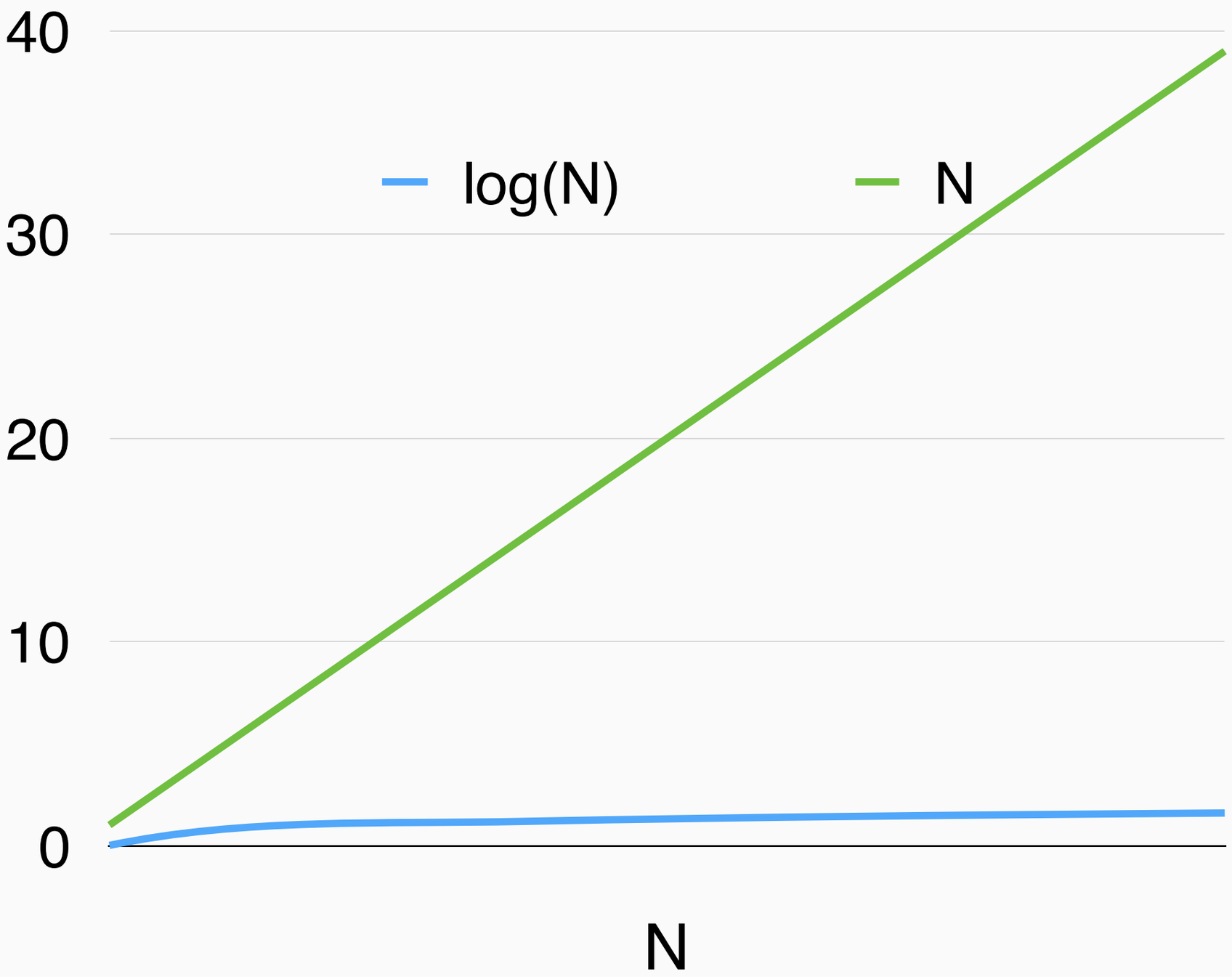
- For an arbitrary list of length N:

  - Linear Search will do N things in the worst case

  - Binary Search will do log(N) things in the worst case



Log vs Linear



Log vs Linear

N = 10000 log(N) = 13.287

Log vs Linear

N = 10000 log(N) = 13.287

(We strongly prefer Binary Search…)

Log vs Linear

(But then our lists need to be sorted!)

N = 10000 log(N) = 13.287

(We strongly prefer Binary Search…)

Our Second Problem: Sorting

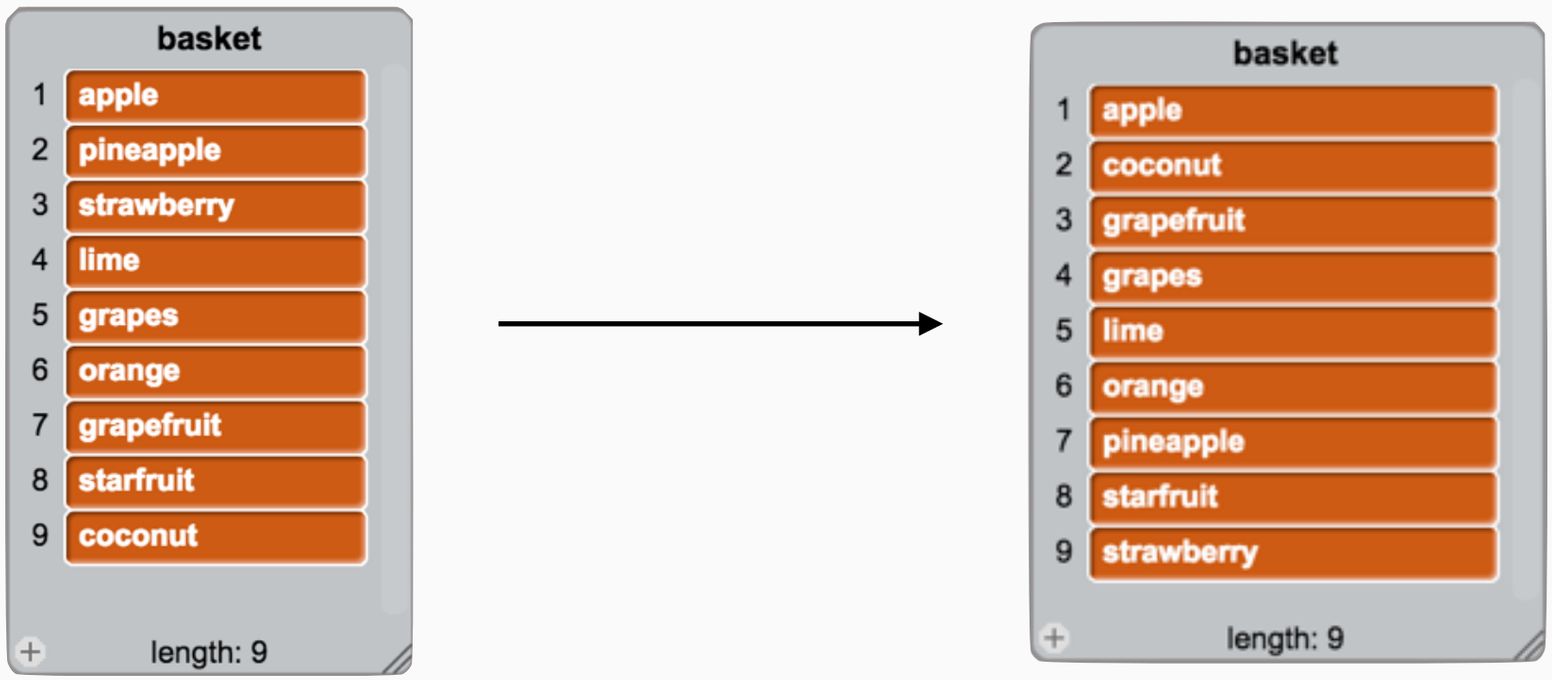
Problem Specification

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

Our Second Problem: Sorting

Problem Specification

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



Sort Solution #1

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.

Sort Solution #1

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.

Let's take a look!

Sort Suggestions?

Any proposals?

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Sort Solution #2

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….

Reflection

- **Definition:** An *algorithm* is a recipe for solving a problem.
- Computer science is (loosely) the study of algorithms.
- Algorithms are *correct* when they solve a specific problem specification
- Search!
- Sort!
- Worst case consideration —> Growth Rate! Some algorithms are faster than others for solving the same problem.