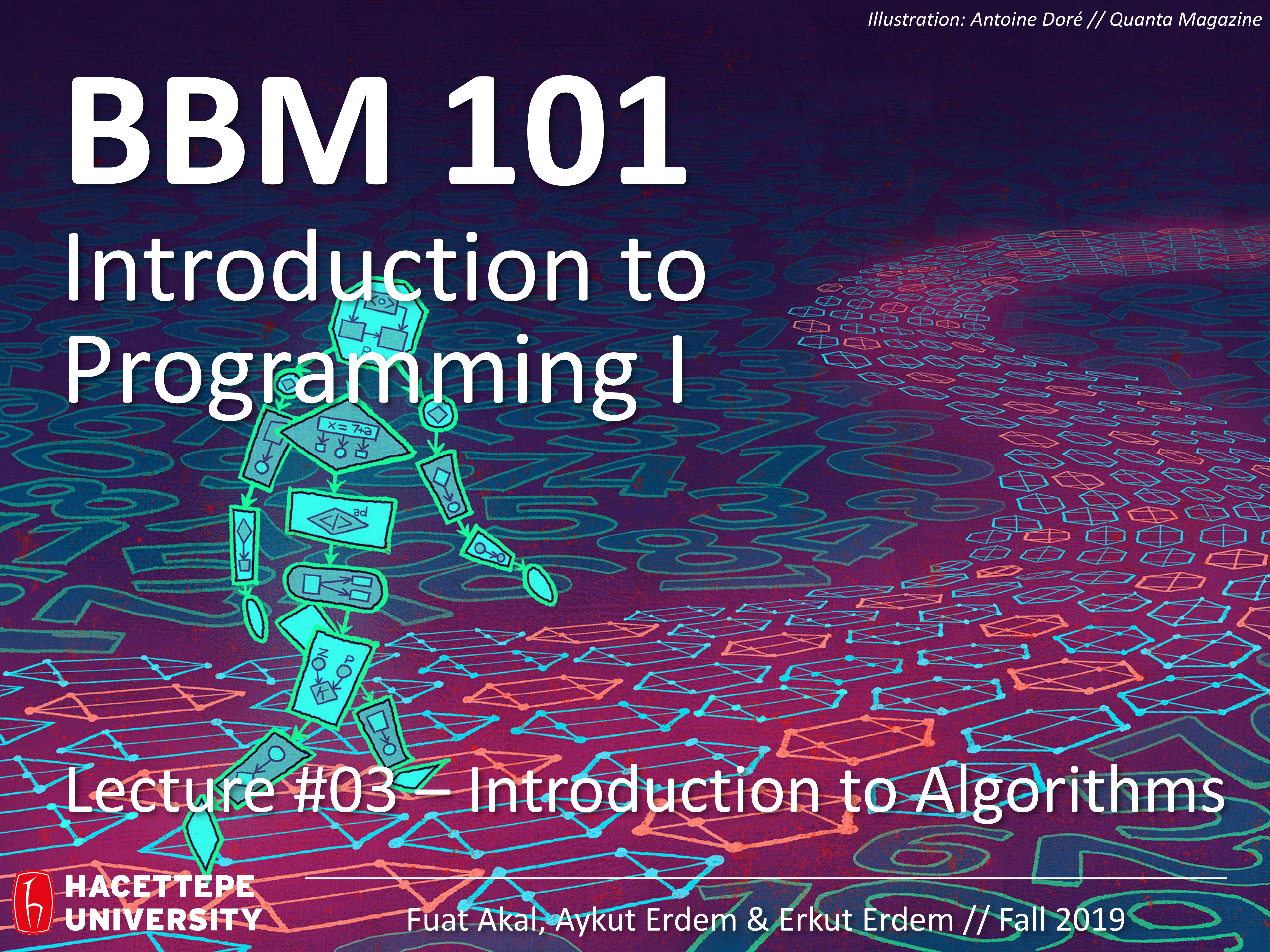
Illustration: Antoine Doré // Quanta Magazine

BBM 101

Introduction to Programming I



Lecture #03 – Introduction to Algorithms

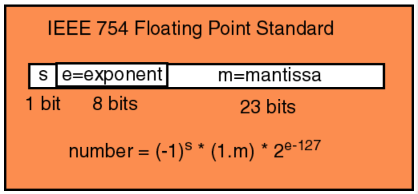
HACETTEPE UNIVERSITY

Fuat Akal, Aykut Erdem & Erkut Erdem // Fall 2019

Last time… **Computers**

Building a Computer

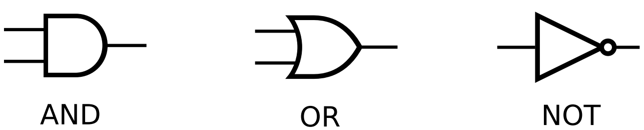
- Numbers
- Letters and Strings
- Structured Information



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Hexadecimal to ASCII conversion table

- Boolean Algebra and Functions
- Logic Using Electrical Circuits
- Computing With Logic
- Memory



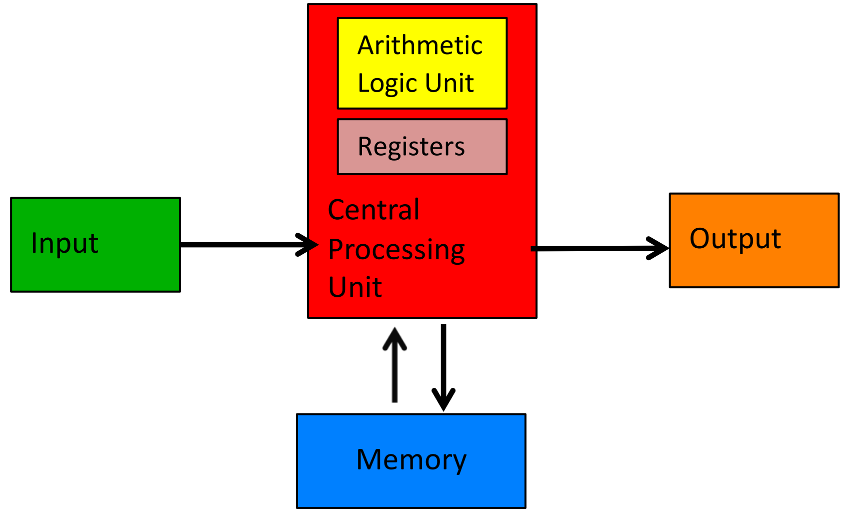
x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1

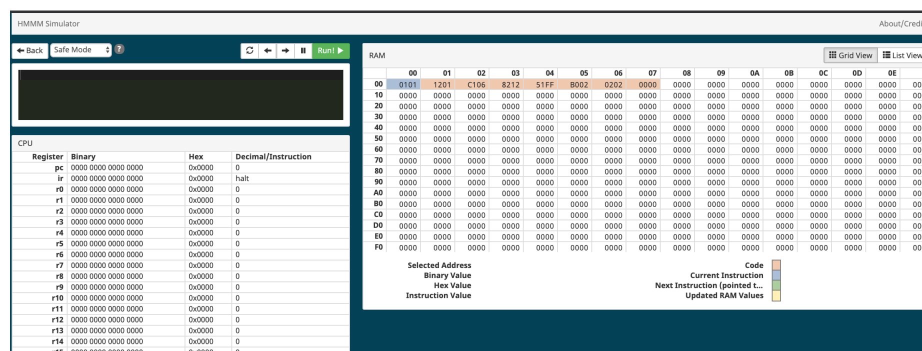
x	NOT x
0	1
1	0

von Neumann Architecture

- Boolean Algebra and Functions



The Harvey Mudd Miniature Machine



Lecture Overview

- Algorithms overview
- Your first algorithm: Search
  - Three flavors of search (Random, Linear, Binary)
- Your second algorithm: Sorting
  - Two flavors of sorting (Random, Selection)
- Program Development Strategies

Disclaimer: Much of the material and slides for this lecture were borrowed from
—Michael Littman's Brown CS8: A First Byte of Computer Science course
—Ruth Anderson's University of Washington CSE 140 course

Lecture Overview

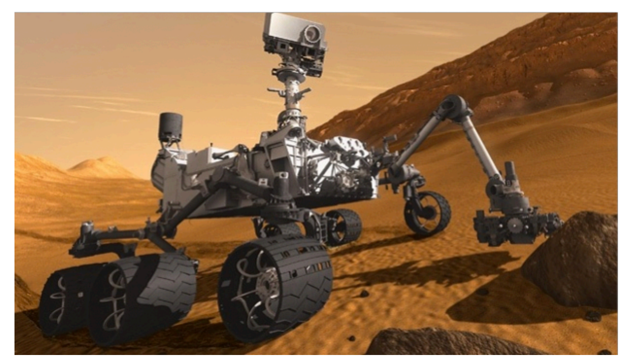
- Algorithms overview
- Your first algorithm: Search
  - Three flavors of search (Random, Linear, Binary)
- Your second algorithm: Sorting
  - Two flavors of sorting (Random, Selection)
- Program Development Strategies

What’s in Computer Science?

- Abstraction

- Problem Solving!
- Artistic, Creative.
  - e.g. Digital Media, Electronic Music, Games, Animation.

- Science.
  - e.g. Understand and model reality.



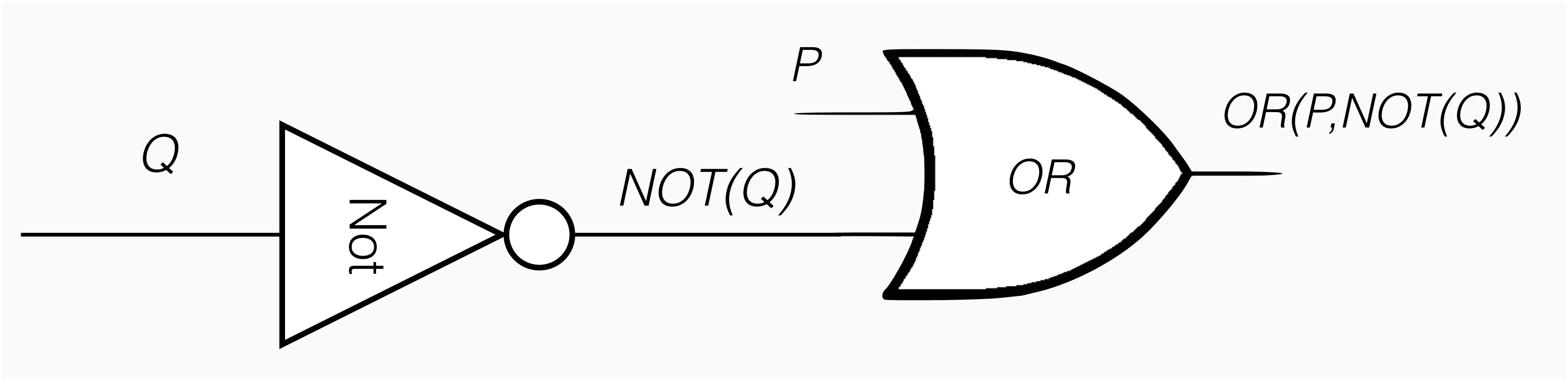
Programming: Take away

1. Physical gates are inflexible.

2. Programming lets us reconfigure what a computer does!

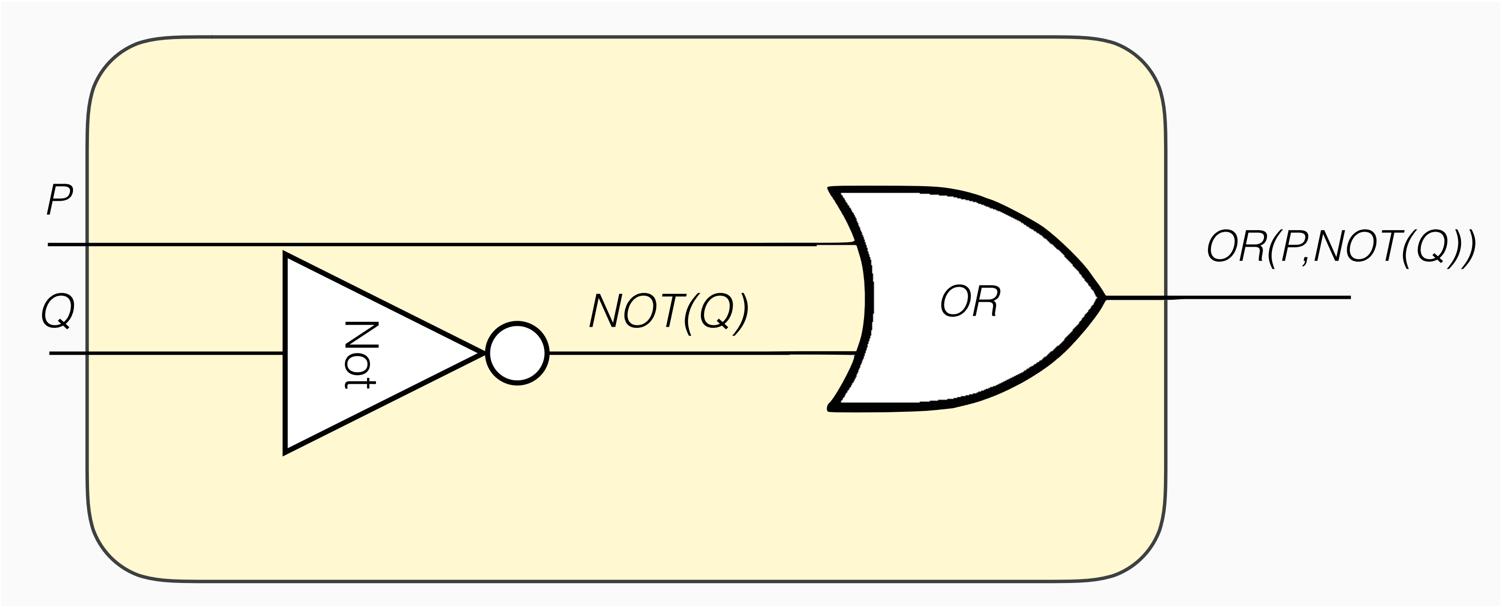
Trouble in Gateland?

OR(P,NOT(Q))



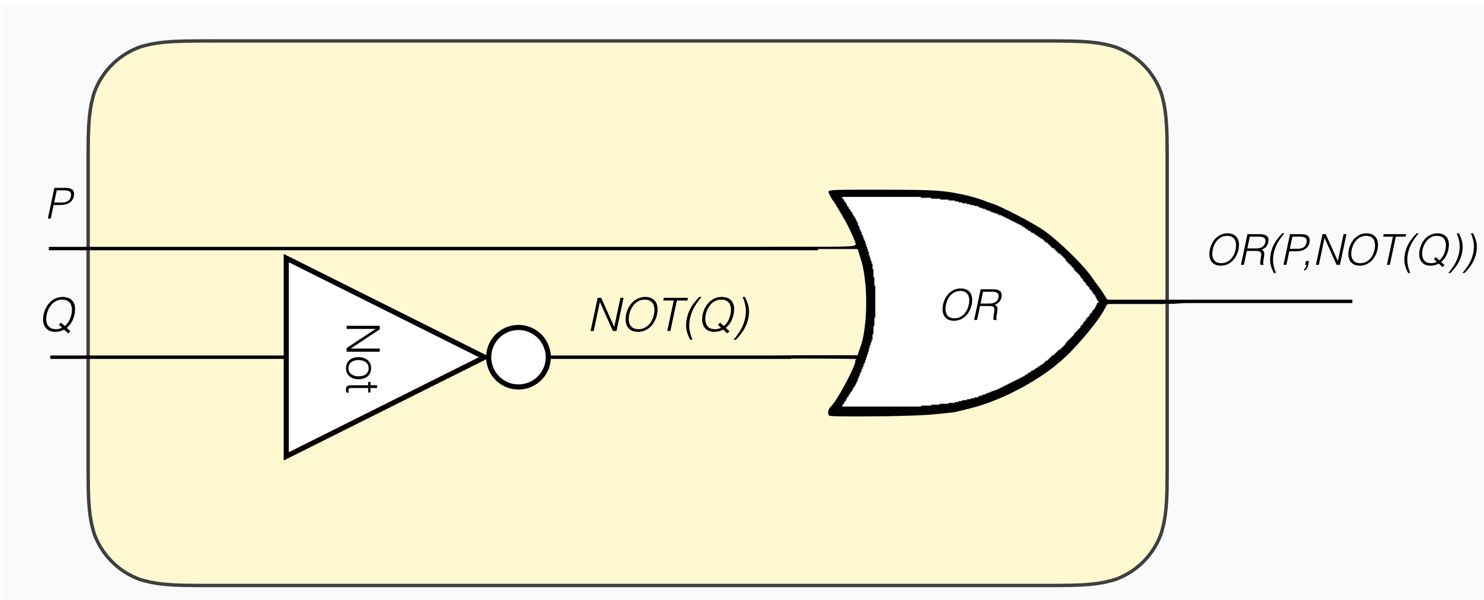
Trouble in Gateland?

Physically represents *OR(P,NOT(Q))*



Trouble in Gateland?

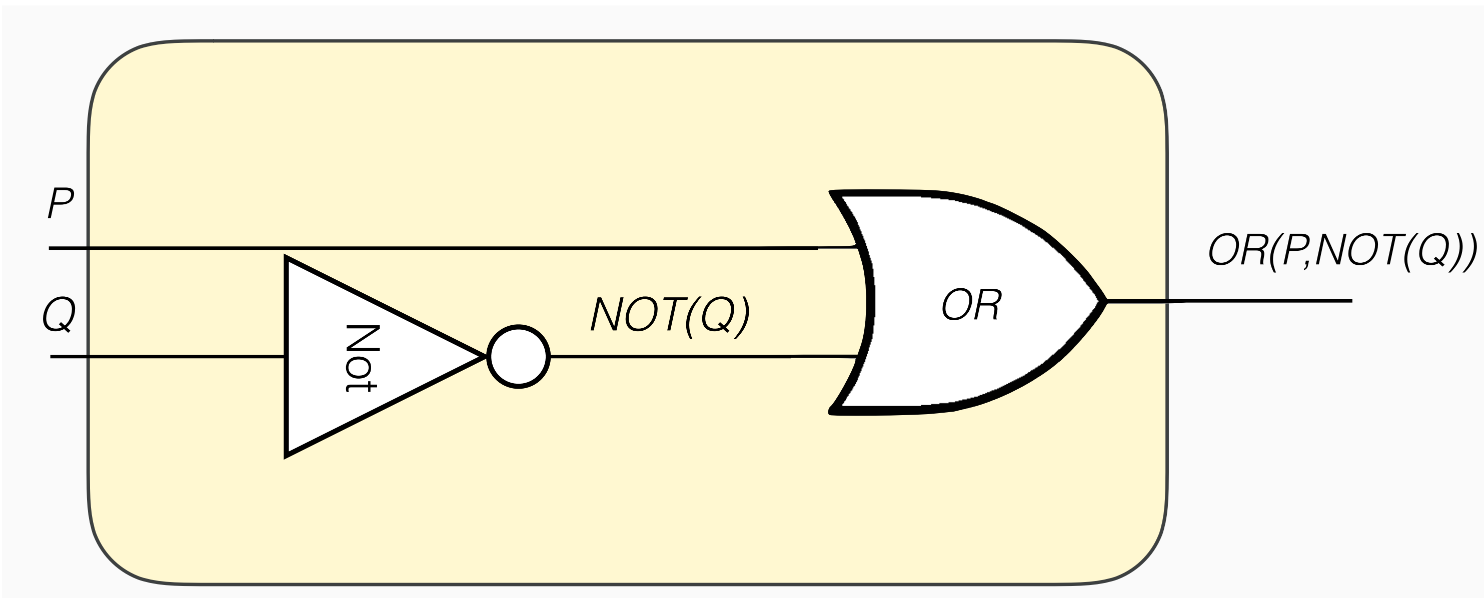
Physically represents *OR(P,NOT(Q))*



Q: What if we want to reconfigure things?

Trouble in Gateland?

Physically represents *OR(P,NOT(Q))*



Q: What if we want to reconfigure things?

A. Programming

Programming

- **Central Idea:** The hardware does not have to change for a computer to change its behavior.

- "Stored program" computers.

Programming

- **Central Idea:** The hardware does not have to change for a computer to change its behavior.

- A fixed set of gates can *change its behavior* to represent any desired function! Build one, **reprogram** into anything.

Programming

- **Central Idea:** The hardware does not have to change for a computer to change its behavior.

- A fixed set of gates can *change its behavior* to represent any desired function! Build one, **reprogram** into anything.

- Drawback: **much slower.**

Programming

- Lots of languages!
- Each language provides a different way to write commands to the computer.
- They all do basically the same thing...
  - "Turing equivalent"

Telescope Science

"Computer Science is no more about computers than astronomy is about telescopes."

- Dijsktra (possibly)

What's in Computer Science?

- Abstraction

- Problem Solving!
- Artistic, Creative.
  - e.g. Digital Media, Electronic Music, Games, Animation.

- Science.
  - e.g. Understand and model reality.

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.

- That is, 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.
- That is, computer science is the study of *automated methods of solving problems.*
- **Programs are ways of carrying out algorithms!!!**

Problem Specification

- **A specification defines a problem**

Problem Specification

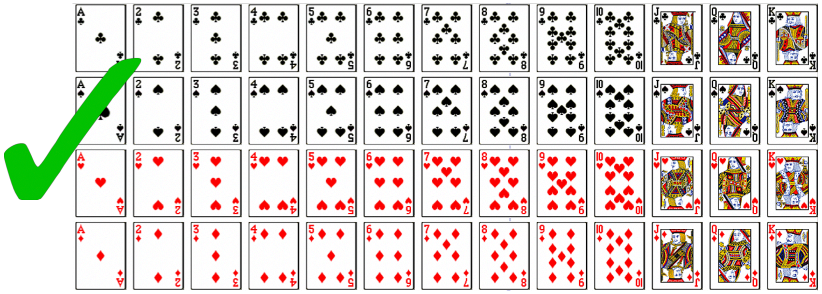
- **A specification defines a problem**
- **An algorithm solves a problem**

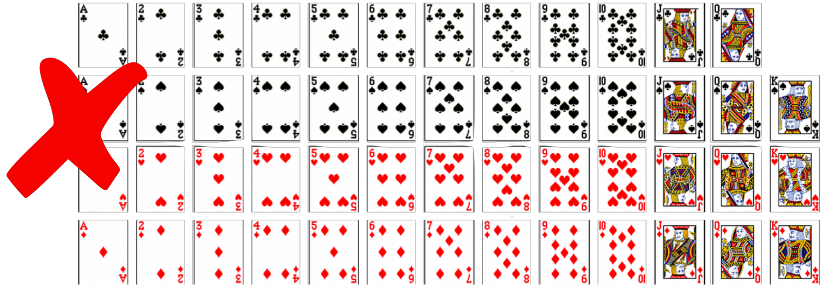
Problem Specification

- **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 2020 election*

Problem Specification Examples

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

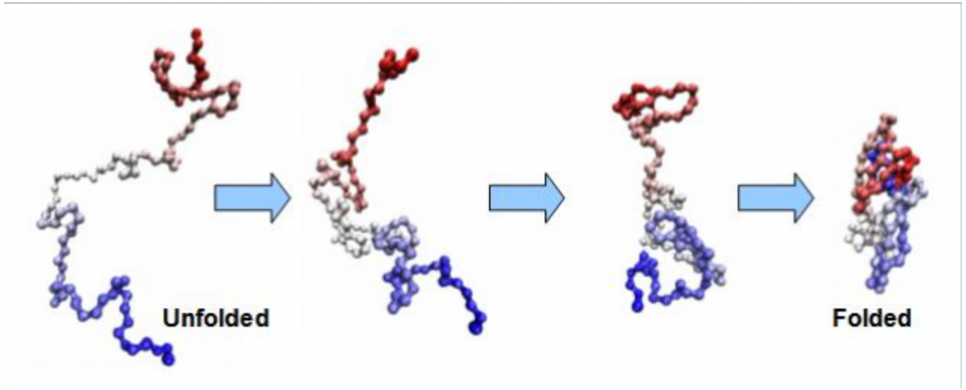
Problem Specification Examples

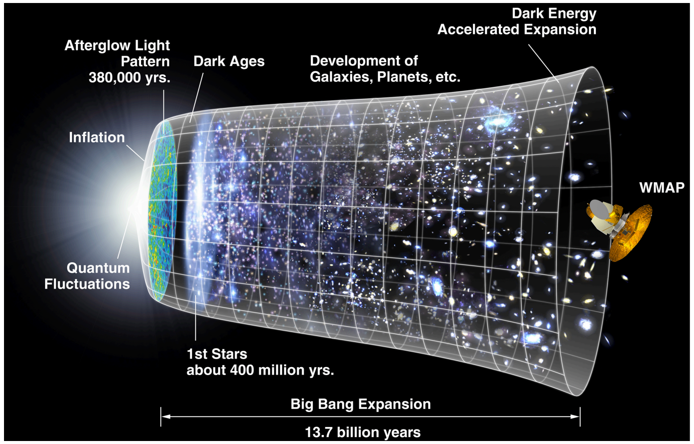
- *INPUT: Data from the stock market.*
- *OUTPUT: Correct predictions about the market.*

Problem Specification Examples

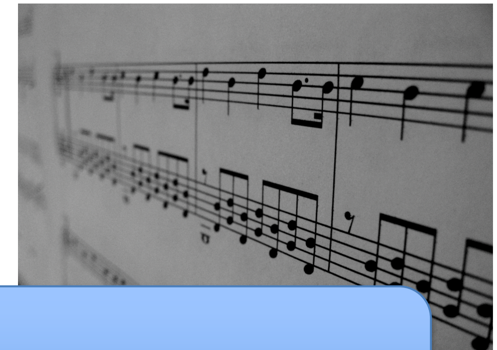
- *INPUT: A bunch of songs from the last 1000 years.*
- *OUTPUT: A new song, guaranteed to be loved.*

Problem Specification

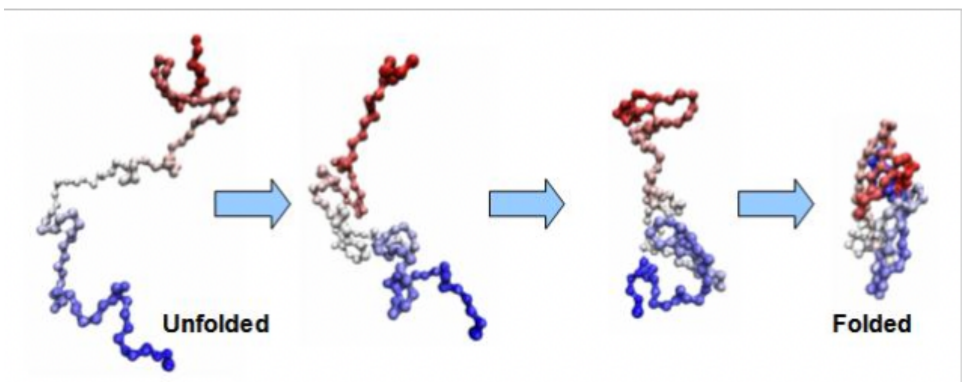


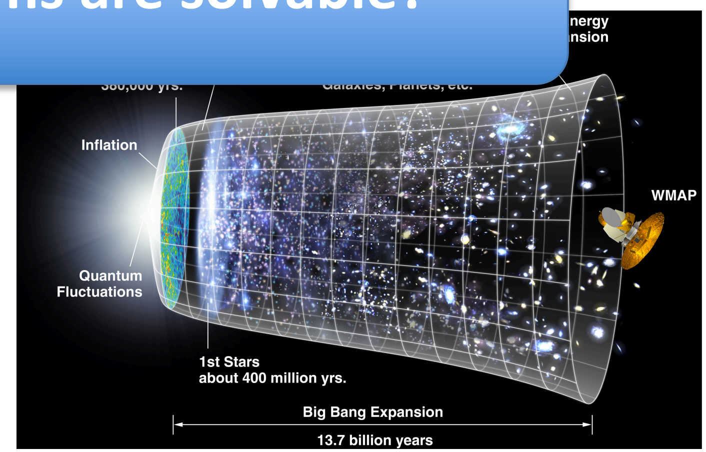


Problem Specification



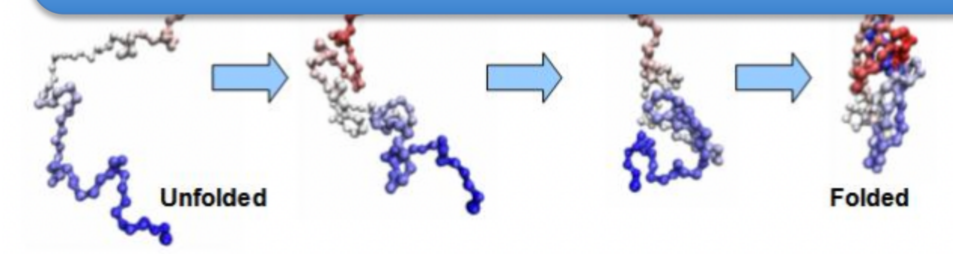
(1) Which of these problems are solvable?

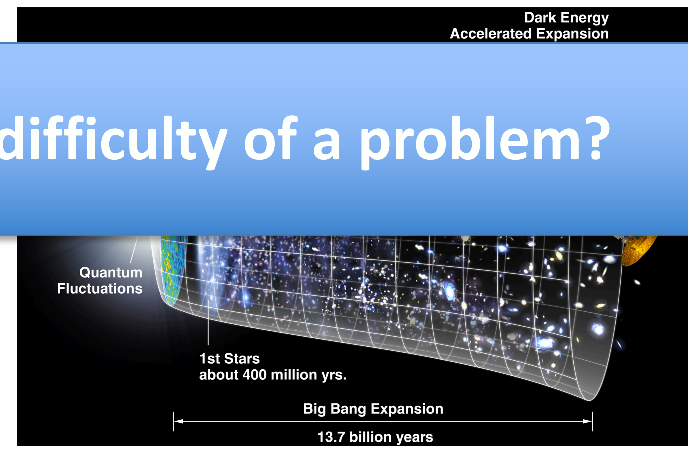




Problem Specification

(1) Which of these problems are solvable?





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

Lecture Overview

- Algorithms overview
- Your first algorithm: Search
  - Three flavors of search (Random, Linear, Binary)
- Your second algorithm: Sorting
  - Two flavors of sorting (Random, Selection)
- Program Development Strategies

Our First Problem: Search

Problem Specification

- *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

Problem Specification

- *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”

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.

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"

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

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”

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

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”

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!

Q: 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!

Q: 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!

Q: 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!

Q: 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!

Q: Is "lime" in the list?

Question!

- Q: Does Linear Search solve the Search Problem?

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”

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

Question!

- Q: Does Linear Search solve the Search Problem?

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"

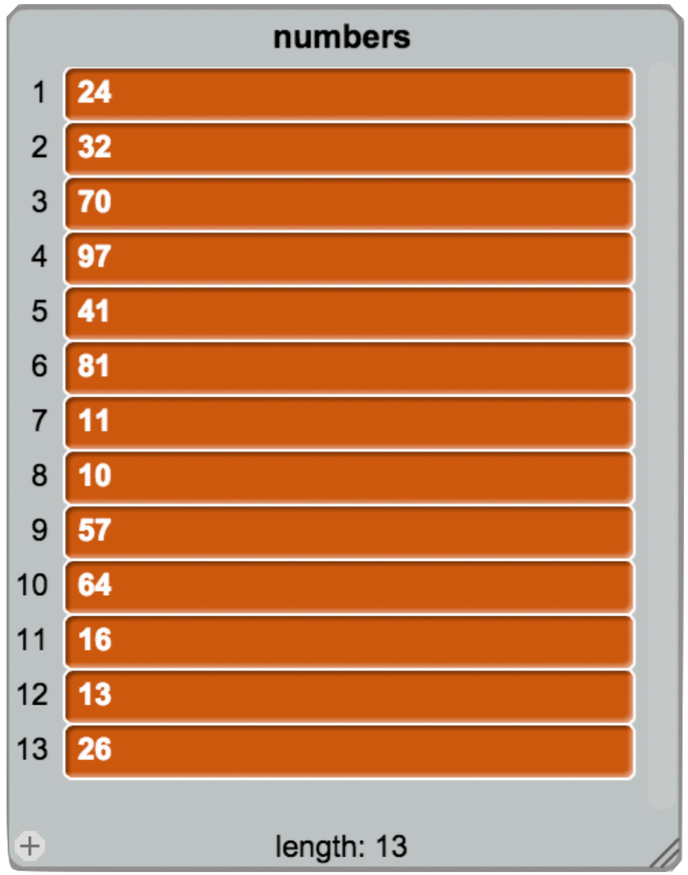
A: Yes! For any list, for any item, linear search will solve Search!

Search Algorithm #3

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

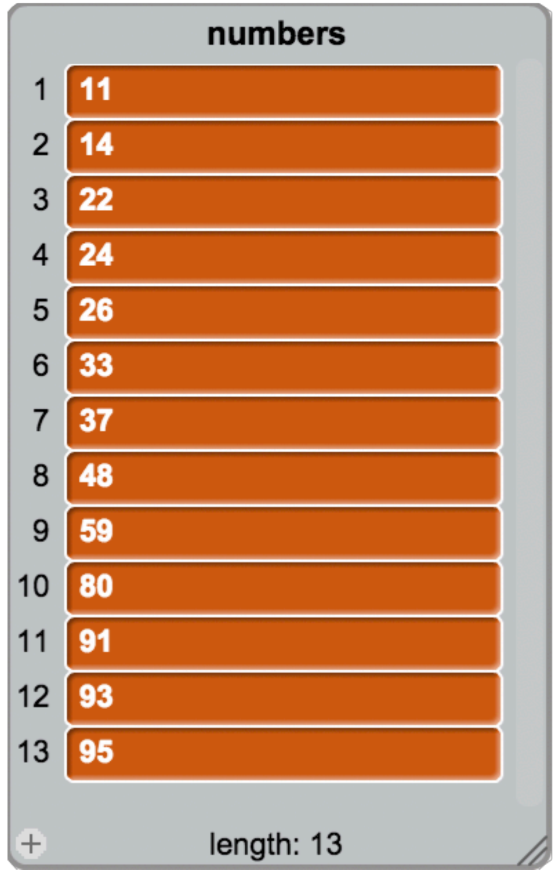
- Idea: if we assume the list is sorted, surely finding our item is easier!

You Try It



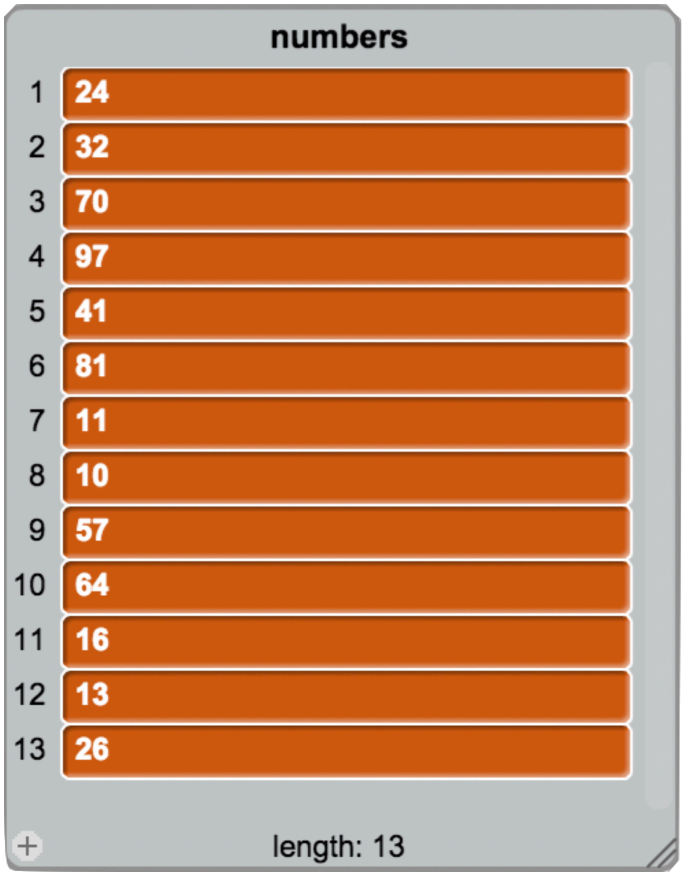
Q: Is 16 in the list?

You Try It

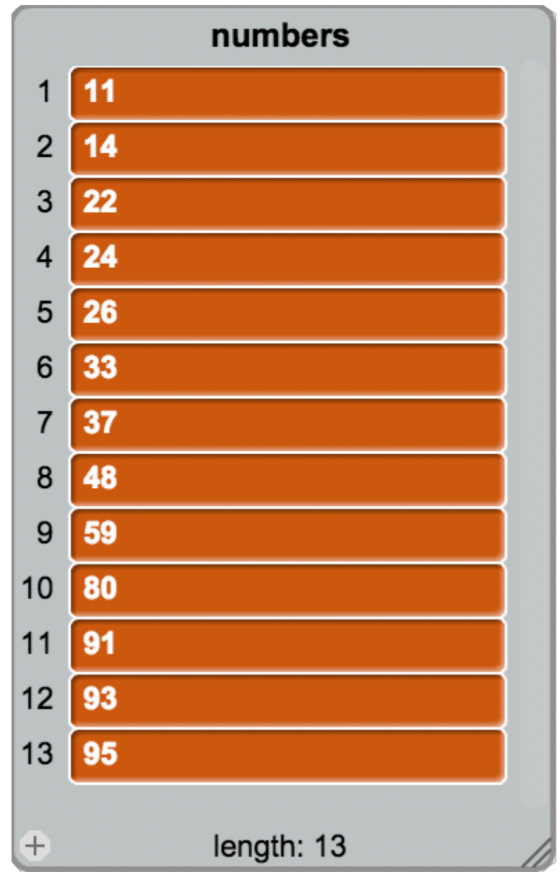


Q: Is 91 in the list?

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.

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 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.

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*

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

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

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

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.

~~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.

$6 < 8$

~~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.

$6 < 8$

~~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.

$6 < 8$

~~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

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?

Question

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

1	3	4	5	7	8	9	11	12	14	16

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

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

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

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

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

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

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

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: 4

Properties of Algorithms

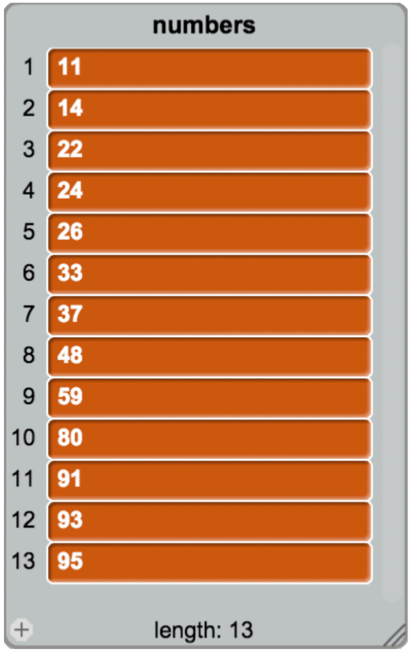
1. Correctness: does the algorithm satisfy the problem specification?

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

Growth Rates

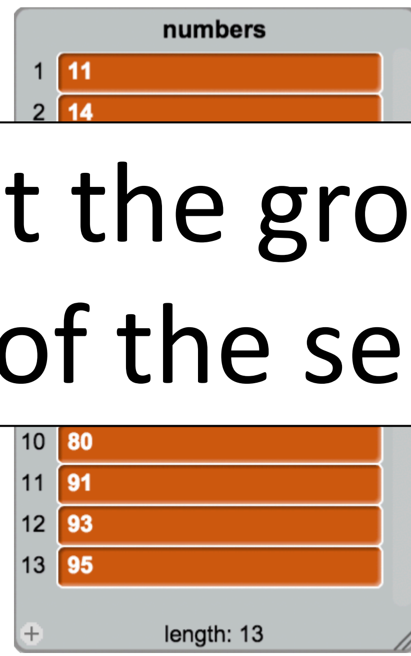
- **Linear Search vs. Binary Search**

- 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?



More about the growth rates at the end of the semester!

Lecture Overview

- Algorithms overview
- Your first algorithm: Search
  - Three flavors of search (Random, Linear, Binary)
- Your second algorithm: Sorting
  - Two flavors of sorting (Random, Selection)
- Program Development Strategies

Our Second Problem: Sorting

Problem Specification

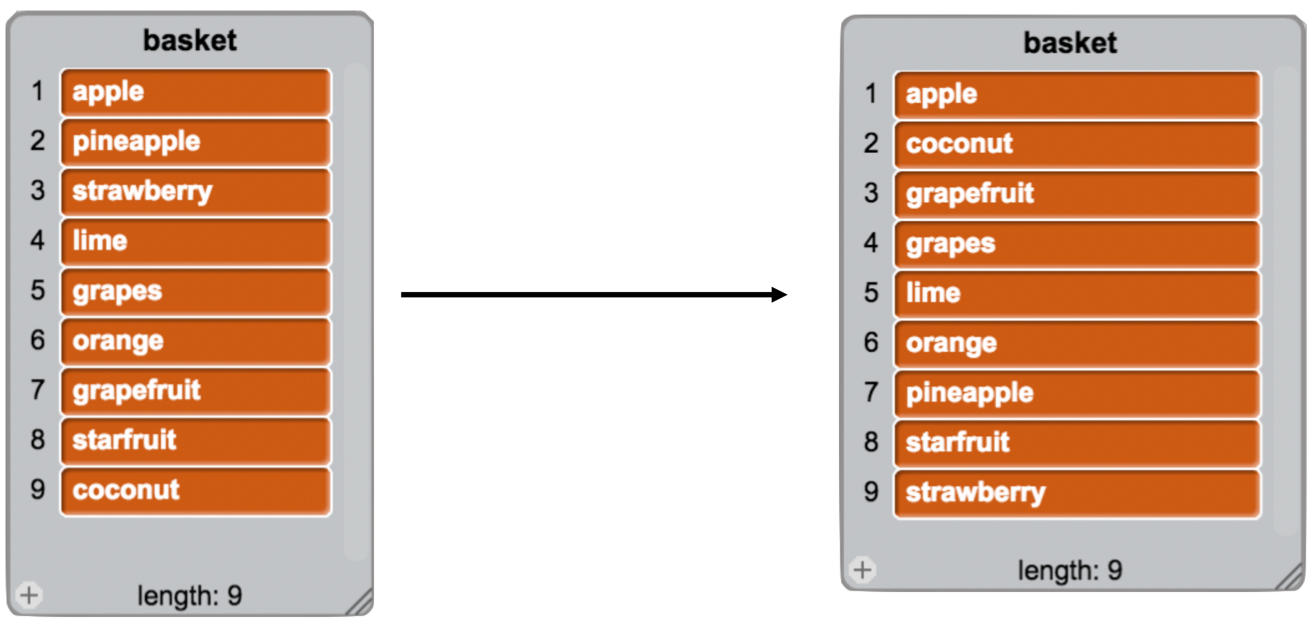
- *Input:*
  - a collection of orderable objects, call it “Basket”

- *Output:*
  - “Basket”, where each item is in order

Our Second Problem: Sorting

Problem Specification

- *Input:*
  - a collection of orderable bjects, 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 Solution #1

Random Sort

https://www.youtube.com/watch?v=C9mdDUutRRg

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....
   (for the 3rd smallest, 4th smallest, …)

Sort Solution #2

Selection Sort

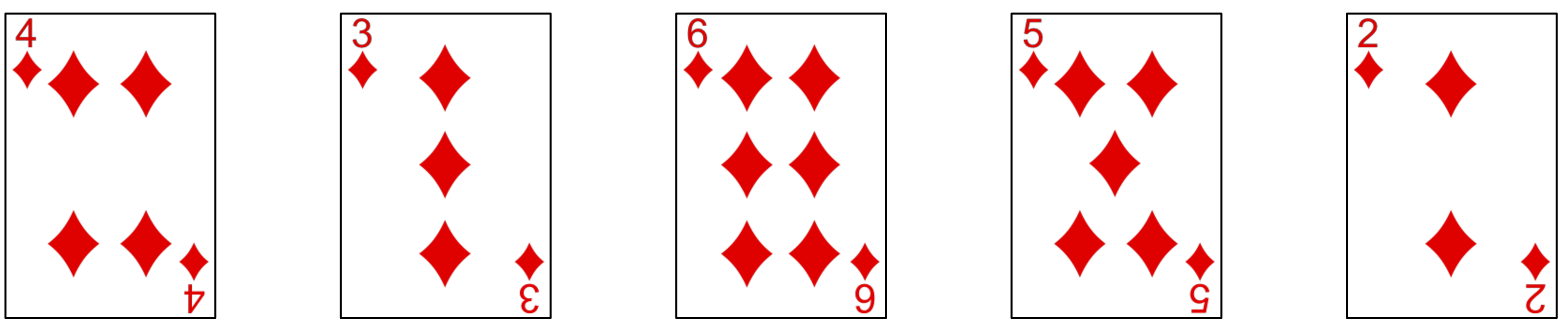
1. “Select” the smallest item in the list.

2. Put it at the beginning.

3. “Select” the second smallest item.

4. Put it 2nd from the beginning.

5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

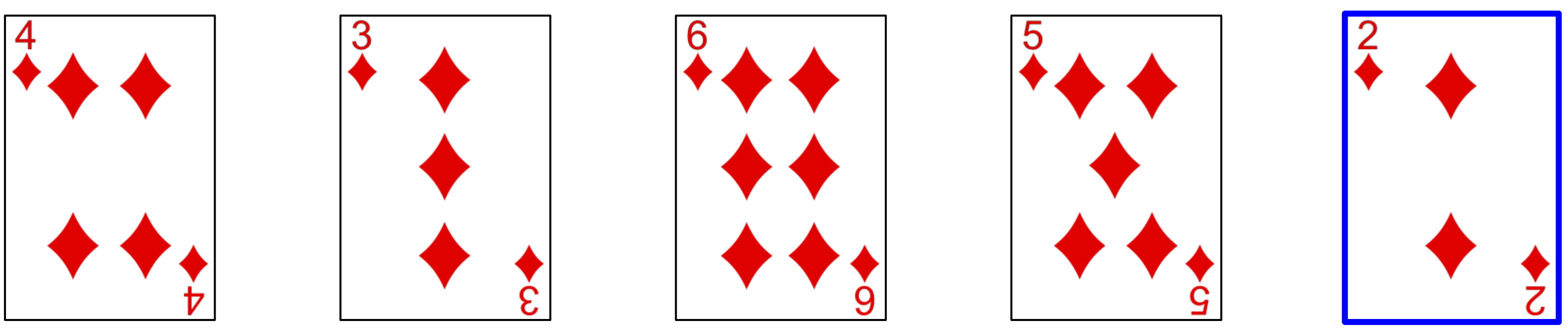
1. “Select” the smallest item in the list.

2. Put it at the beginning.

3. “Select” the second smallest item.

4. Put it 2nd from the beginning.

5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

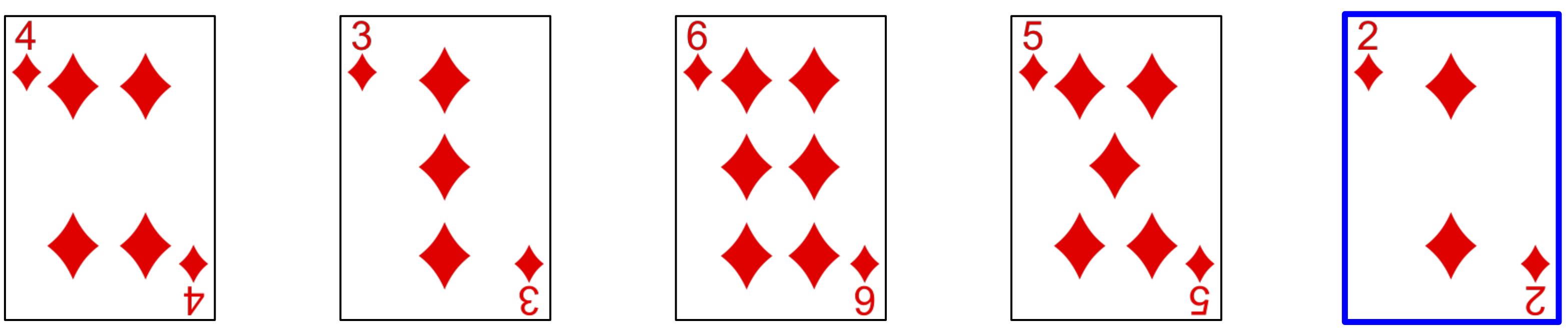
1. “Select” the smallest item in the list.

2. Put it at the beginning.

3. “Select” the second smallest item.

4. Put it 2nd from the beginning.

5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

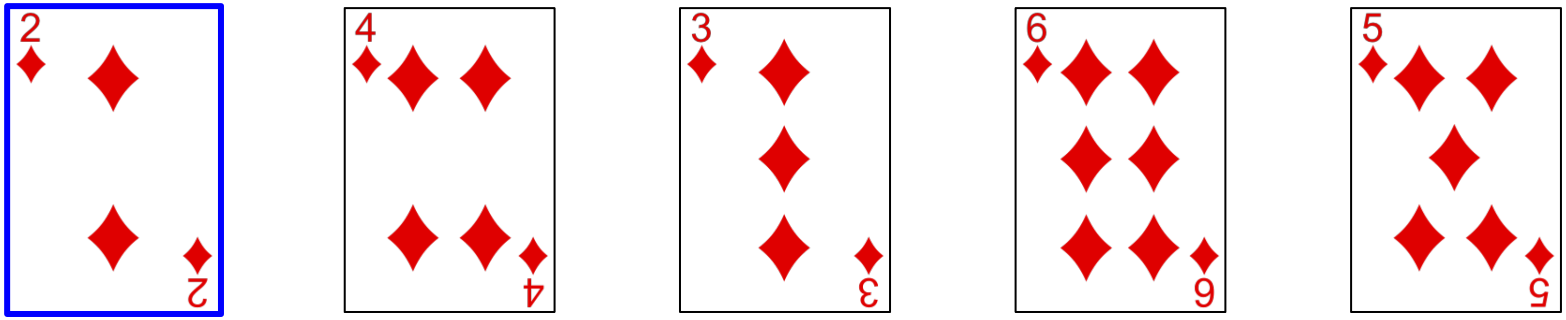
1. “Select” the smallest item in the list.

2. **Put it at the beginning.**

3. “Select” the second smallest item.

4. Put it 2nd from the beginning.

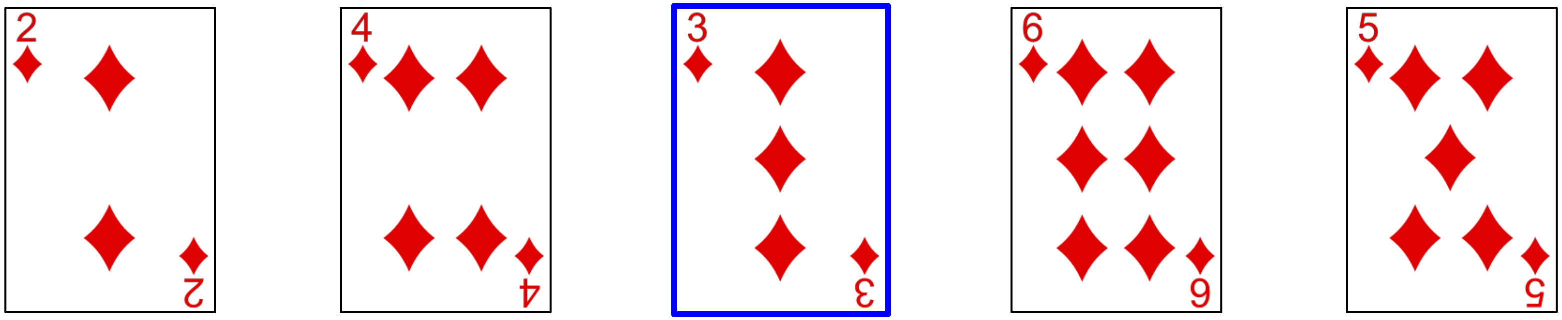
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

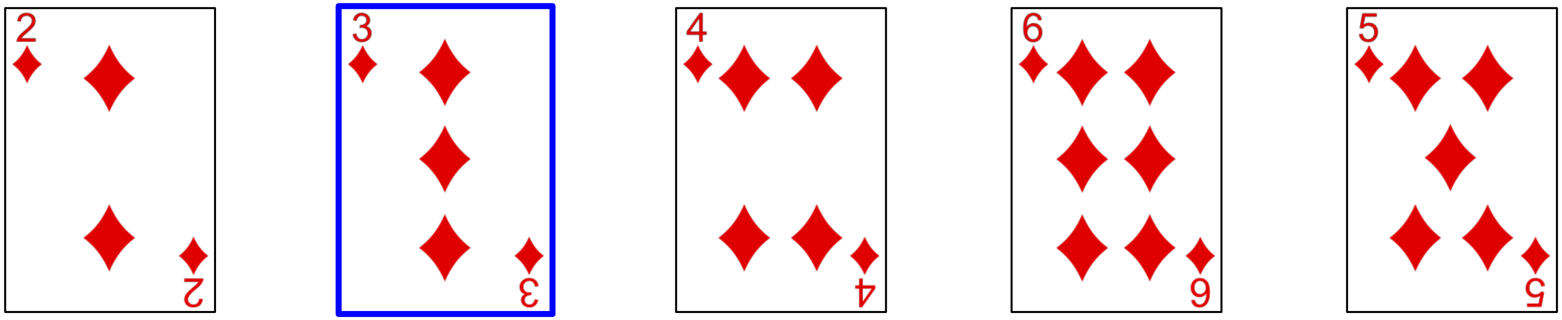
1. “Select” the smallest item in the list.
2. Put it at the beginning.
3. **“Select” the second smallest item.**
4. Put it 2nd from the beginning.
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

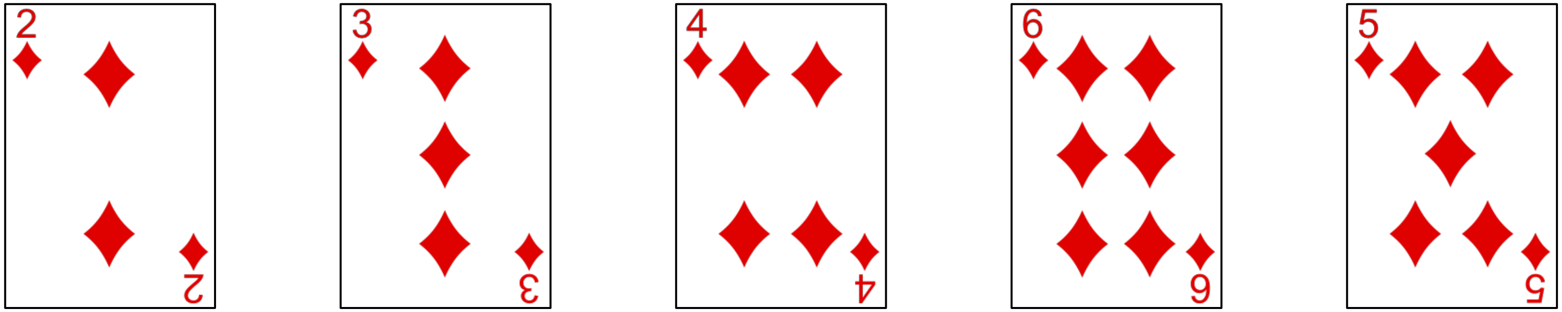
1. "Select" the smallest item in the list.
2. Put it at the beginning.
3. "Select" the second smallest item.
4. **Put it 2nd from the beginning.**
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

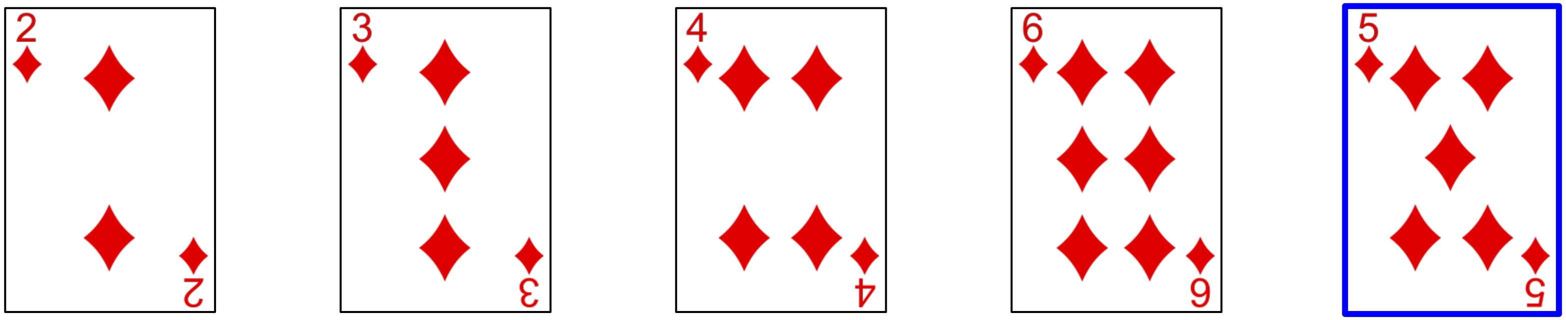
1. “Select” the smallest item in the list.
2. Put it at the beginning.
3. “Select” the second smallest item.
4. Put it 2nd from the beginning.
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

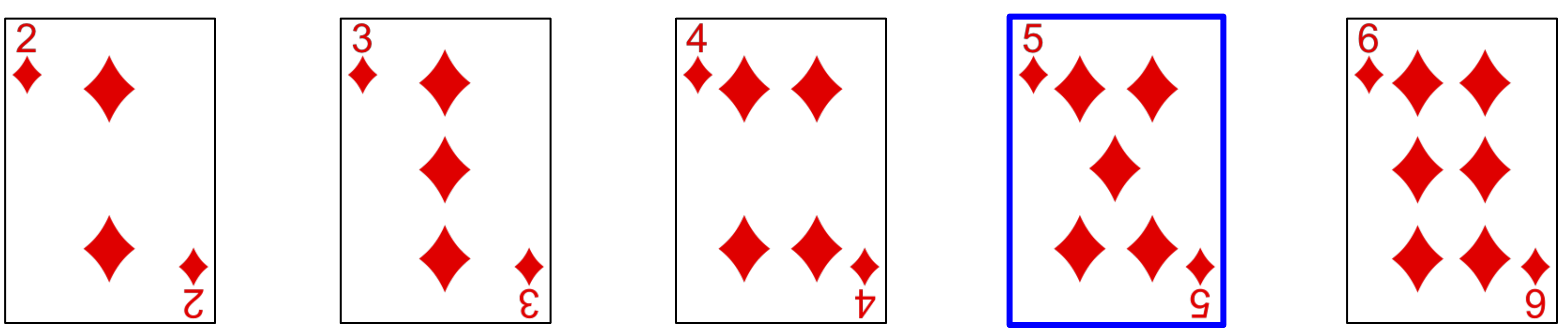
1. “Select” the smallest item in the list.
2. Put it at the beginning.
3. “Select” the second smallest item.
4. Put it 2nd from the beginning.
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

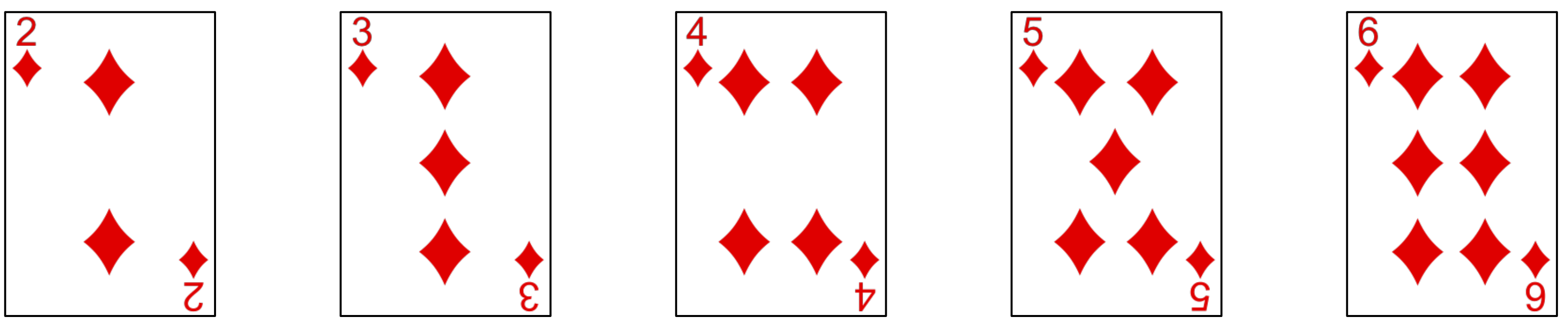
1. “Select” the smallest item in the list.
2. Put it at the beginning.
3. “Select” the second smallest item.
4. Put it 2nd from the beginning.
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



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.... (for the 3rd smallest, 4th smallest, …)



Sort Solution #2

Selection Sort

https://www.youtube.com/watch?v=hqBPYhAQeTI

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 #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....

Our Second Problem: Sorting

Problem Specification

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

Many possible solutions to this problem exist!

Our Second Problem: Sorting

Problem Specification

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

Many possible solutions to this problem exist!

Then, how to develop a computer program?

Lecture Overview

- Algorithms overview
- Your first algorithm: Search
  - Three flavors of search (Random, Linear, Binary)
- Your second algorithm: Sorting
  - Two flavors of sorting (Random, Selection)
- Program Development Strategies

Program development methodology

Algorithm first, then Implementation:

1. Define the problem
2. Decide upon an algorithm
3. Translate it into code

Try to do these steps in order

Program development methodology

Algorithm first, then Implementation:

1. **Define the problem**
   A. Write the problem specification:

      An natural language description of the input and output **for the whole program**. (Do not give details about *how you will compute* the output.)

   B. Create test cases **for the whole program**
      - Input and expected output
2. Decide upon an algorithm
3. Translate it into code

Try to do these steps in order

Program development methodology

Algorithm first, then Implementation:

1. Define the problem
2. **Decide upon an algorithm**
   A. Implement it in an algorithmic manner (e.g. in English)
      - Write the recipe or step-by-step instructions
   B. Test it using paper and pencil
      - Use small but not trivial test cases
      - Play computer, animating the algorithm
      - Be introspective
        - Notice what you really do
        - May be more or less than what you wrote down
        - Make the algorithm more precise
3. Translate it into code

Try to do these steps in order

Program development methodology

Algorithm first, then Implementation:

1. Define the problem
2. Decide upon an algorithm
3. **Translate it into code**
   A. Implement it using a programming language
      - Decompose it into logical units (functions)

Try to do these steps in order

Why functions?

There are several reasons:

- Creating a new function gives you an opportunity to name a group of statements, which **makes your program easier to read and debug**.
- Functions **can make a program smaller** by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.
- Dividing a long program into functions allows you to **debug the parts one at a time** and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, **you can reuse it**.

Program development methodology

Algorithm first, then Implementation:

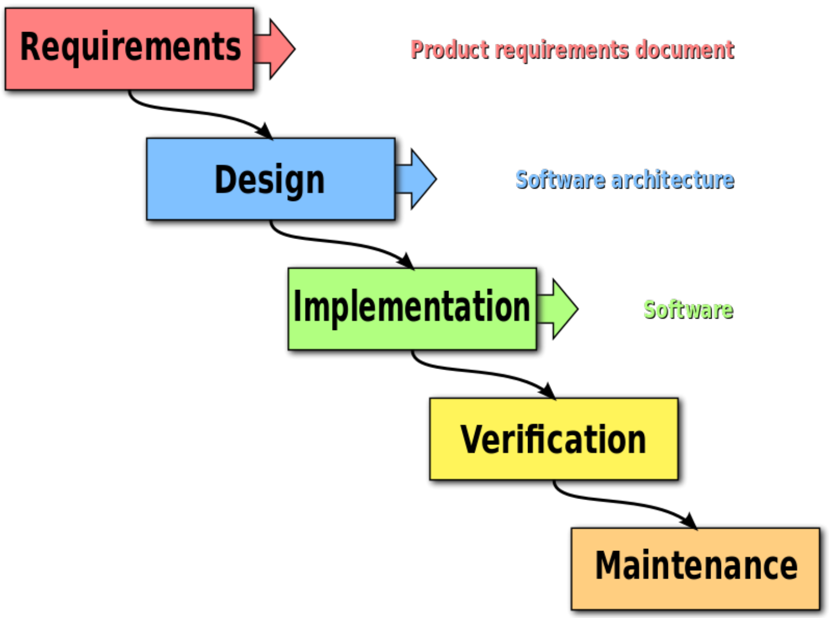
1. Define the problem
2. Decide upon an algorithm
3. Translate it into code

Try to do these steps in order

- It’s OK (even common) to back up to a previous step when you notice a problem
- You are incrementally learning about the problem, the algorithm, and the code
- “Iterative development”

Waterfall Development Strategy

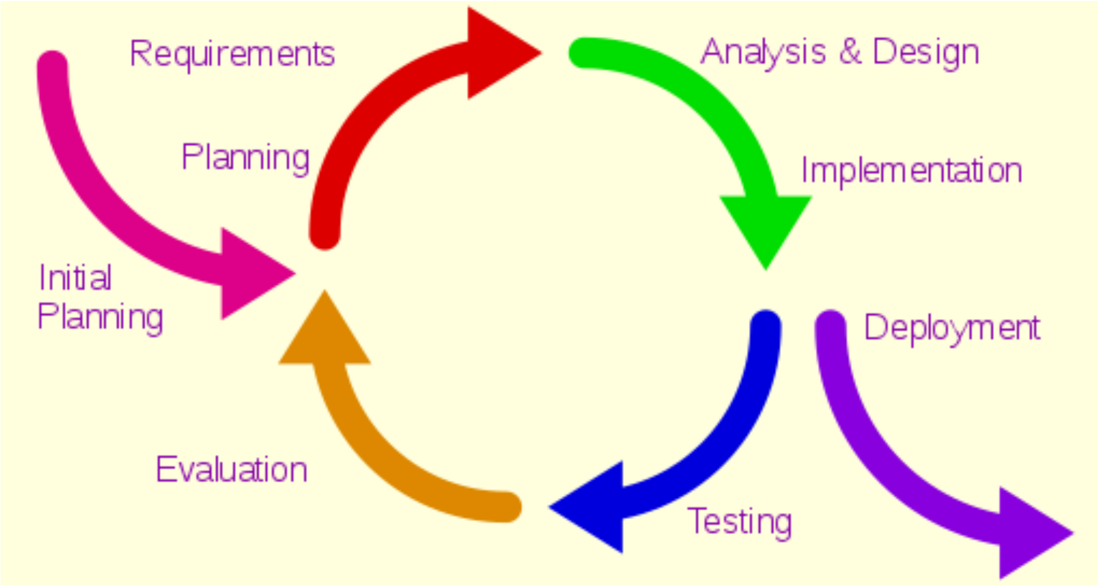
- Before the iterative model, we had the waterfall strategy.
- Each step handled once.
- The model had a limited capability and received too many criticism.
- Better than nothing!!
- Do not dive in to code!!
- Please!!



** From wikipedia waterfall development model*

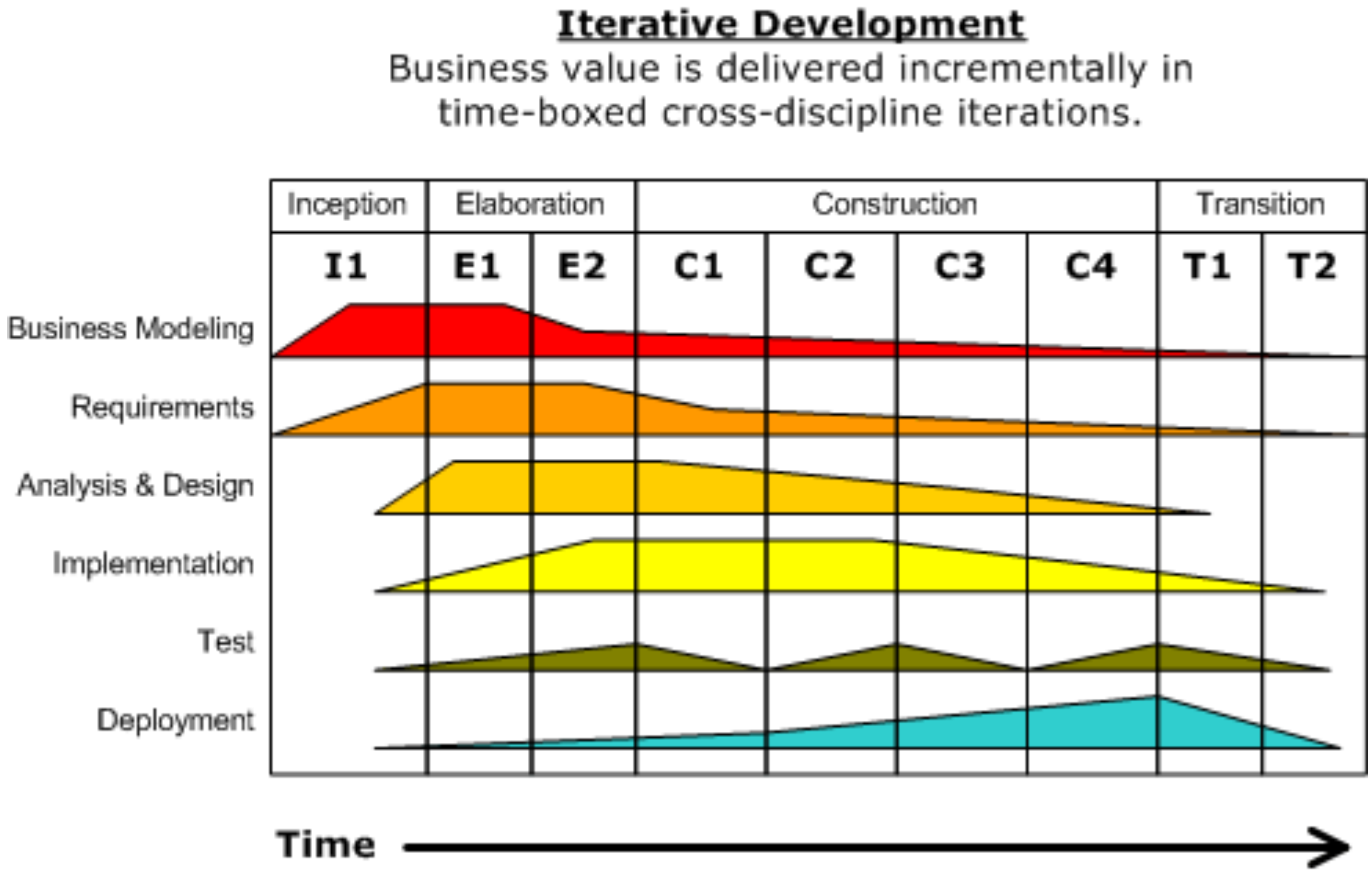
Iterative Development Strategy

- Software development is a living process.
- Pure waterfall model wasn't enough.
- Iterative development strategy suits best to our needs (for now).



** From wikipedia Iterative development model*

Iterative Development Strategy



** From wikipedia Iterative development model*