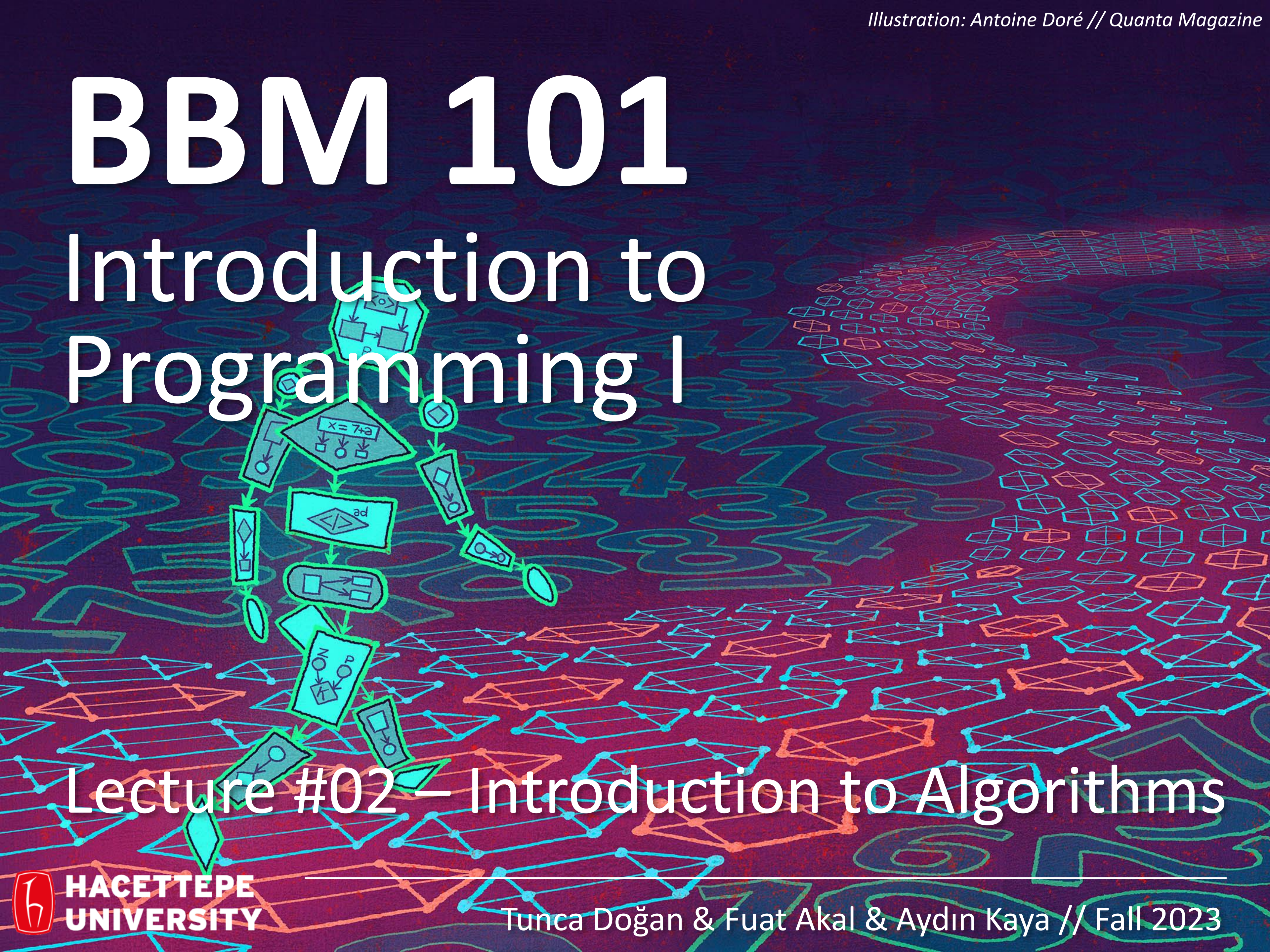


Illustration: Antoine Doré // Quanta Magazine

BBM 101

Introduction to Programming I

Lecture #02 – Introduction to Algorithms

HACETTEPE UNIVERSITY

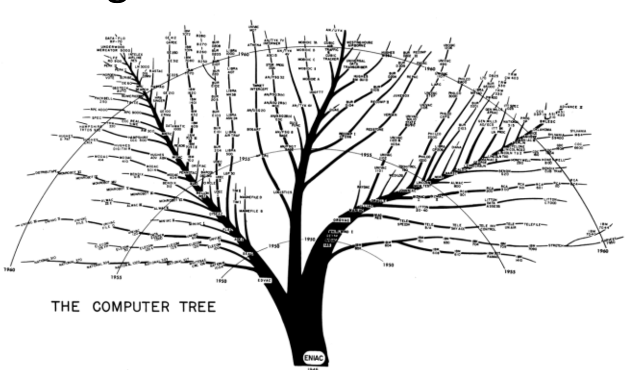
Tunca Doğan & Fuat Akal & Aydın Kaya // Fall 2023

Last time… **What is computation**

Computer science is about logic, problem solving, and creativity

Fixed Program Computers

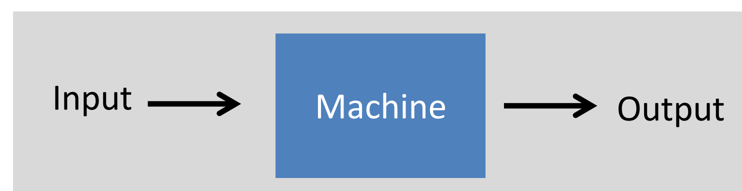
- Abacus
- Antikythera Mechanism
- Pascaline
- Leibniz Wheel
- Jacquard's Loom
- Babbage Difference Engine
- The Hollerith Electric Tabulating System
- Atanasoff-Berry Computer (ABC)
- Turing Bombe



- **Declarative knowledge**
  - Axioms (definitions)
  - Statements of fact
- **Imperative knowledge**
  - How to do something
  - A sequence of specific instructions (what computation is about)

Stored Program Computers

- Problem solving



- What if input is a machine (description) itself?
- Universal Turing machines
  - An abstract general purpose computer

Lecture Overview

- Your first algorithms
- Search algorithms
  - Three flavors of search (Random, Linear, Binary)
- Sorting algorithms
  - 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

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

Your First Algorithms

- Get two integers from the user and print them from smaller to larger.

Algorithm:

```
Input: the first number
Input: the second number
If first < second
        Print first
        Print second
Else
        Print second
        Print first
```

Python Code:

```python
first = input("The first number: ")
second = input("The second number: ")
if first < second:
        print(first)
        print(second)
else:
        print(second)
        print(first)
```

* Notice that arbitrary notations were used to outline your first algorithms.

Your First Algorithms

- Get three integers from the user and print them from smaller to larger.

Algorithm:

```
Input: the first number
Input: the second number
Input: the third number
If first > second
        greater = first
        lesser = second
Else
        greater = second
        lesser = first

If third > greater
        middle = greater
        greater = third
Else
        If  third > lesser
                middle = third
        Else
                middle = lesser
                lesser = third
Print lesser, middle, greater
```

Python Code:

```python
first = input("The first number: ")
second = input("The second number: ")
third = input("The third number: ")
if first > second:
        greater = first
        lesser = second
else:
        greater = second
        lesser = first

if third > greater:
        middle = greater
        greater = third
else:
        if third > lesser:
                middle = third
        else:
                middle = lesser
                lesser = third
print(lesser, middle, greater)
```

Your First Algorithms

- Find the factorial of a given number.

```
Get the number as n
result = 1
If n is 0 OR n is 1
        print result
        end
Else
        while n > 1
                result = result * n
                n = n - 1
print result
```

Your First Algorithms

- Find the Fibonacci sequence for a given number.

```
Input n
Set first to 0
Set second to 1
Set index to 2
print first

If n > 0
    print second

While index <= n
    current <- first + second
    first <- second
    second <- current
    print current
    index <- index + 1
```

The Fibonacci Sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it.

Lecture Overview

- Algorithm Examples
- Search Algorithms
  - Three flavors of search (Random, Linear, Binary)
- Sorting Algorithms
  - Two flavors of sorting (Random, Selection)
- Program Development Strategies

Search Algorithms

Problem Specification

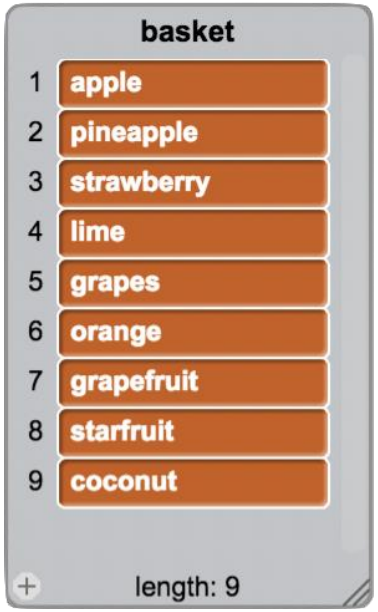
- *Input:*
  - a collection of objects, call it “Basket”
  - a specific object, call it “Snozzberry”

- *Output:*
  - True if “Snozzberry” is in “Basket”
  - False if “Snozzberry” is *not* in “Basket”

Search Algorithms

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”

Search Algorithms

- *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"

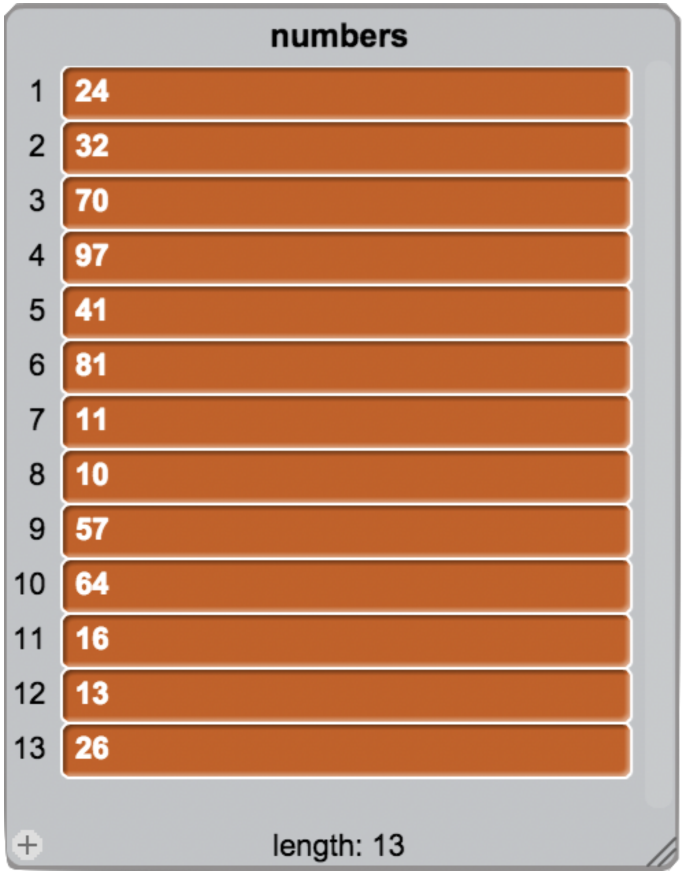
A: Yes! For any list, for any item, linear search will solve the Search problem!

Search Algorithm #3

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

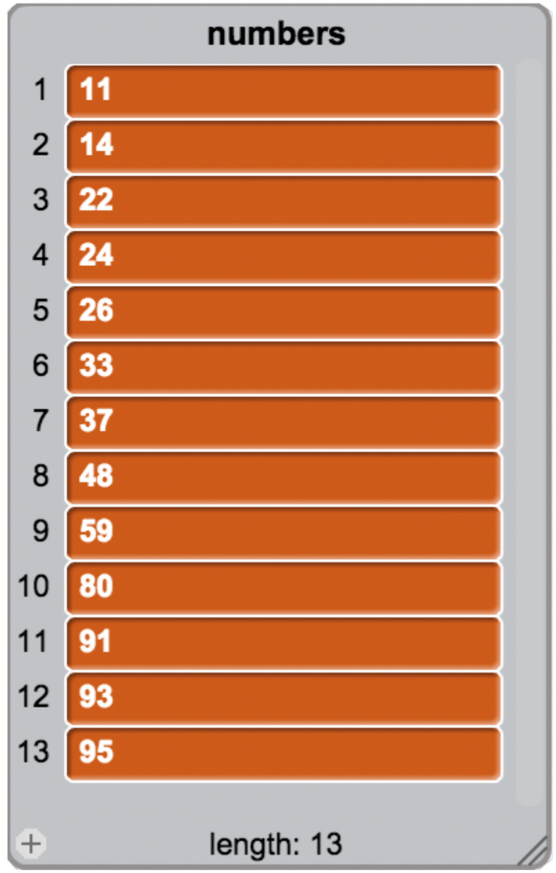
- Idea: if we assume the list is sorted, surely finding our item is easier!

You Try It



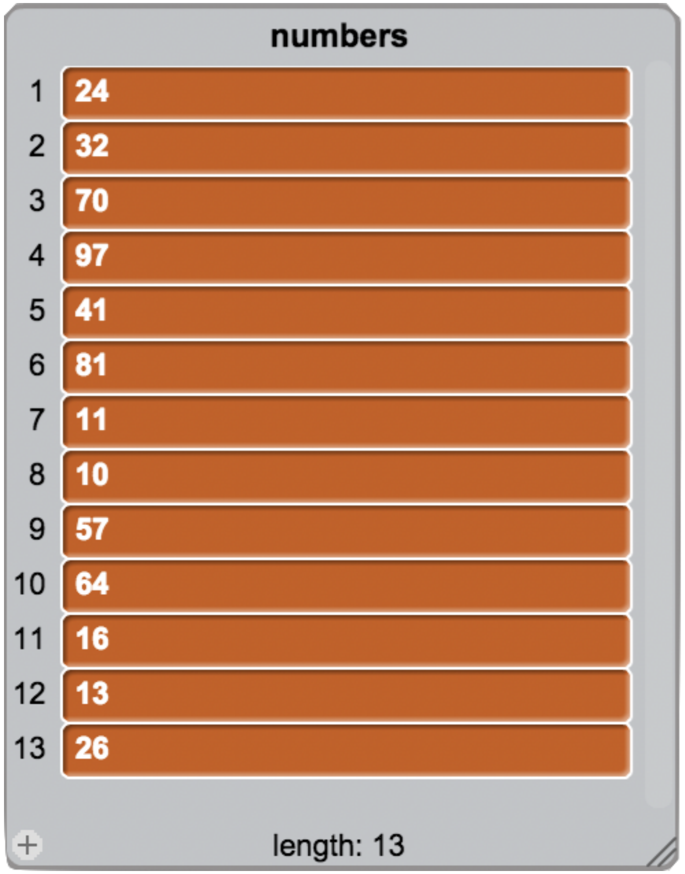
Q: Is 16 in the list?

You Try It

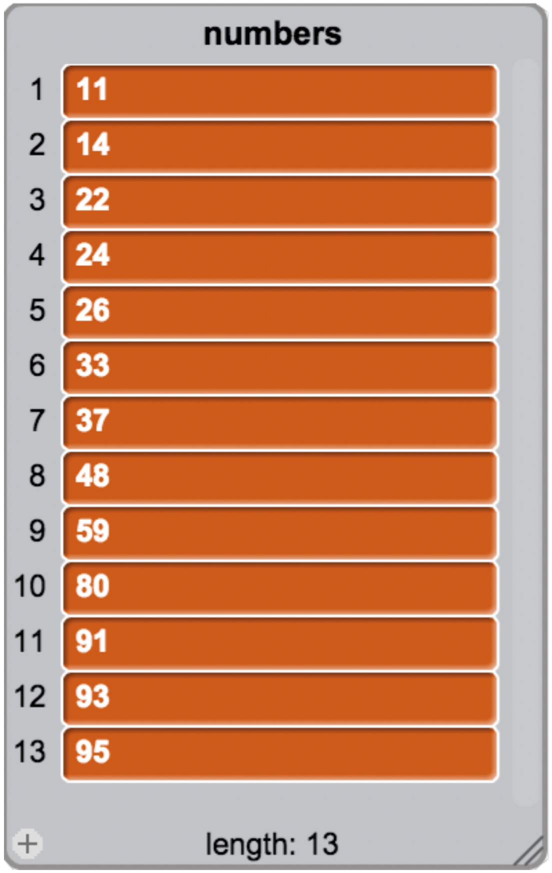


Q: Is 91 in the list?

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

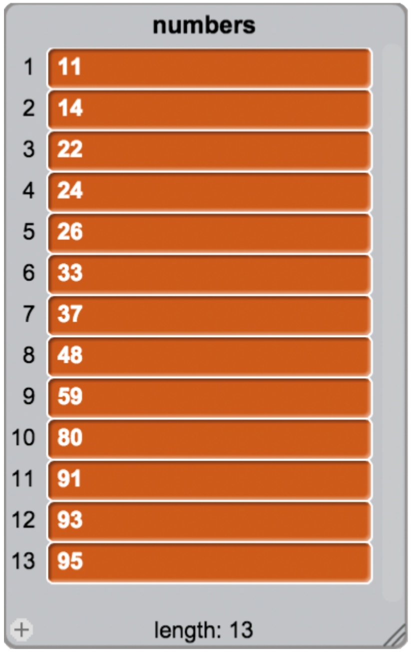
1. Correctness: does the algorithm satisfy the problem specification?

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

Growth Rates

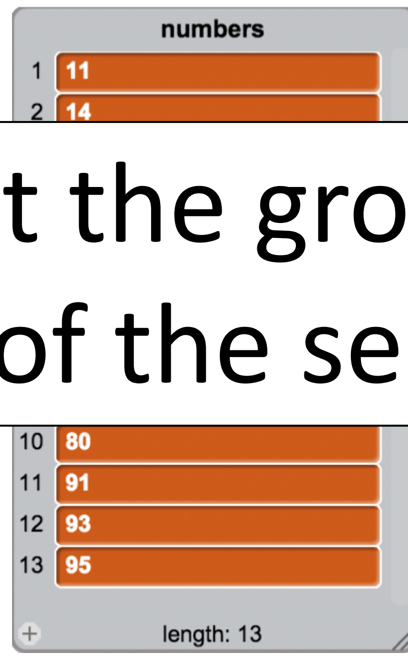
- **Linear Search vs. Binary Search**

- 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

- Algorithm Examples
- Search Algorithms
  - Three flavors of search (Random, Linear, Binary)
- Sorting Algorithms
  - Two flavors of sorting (Random, Selection)
- Program Development Strategies

Sort Algorithms

Problem Specification

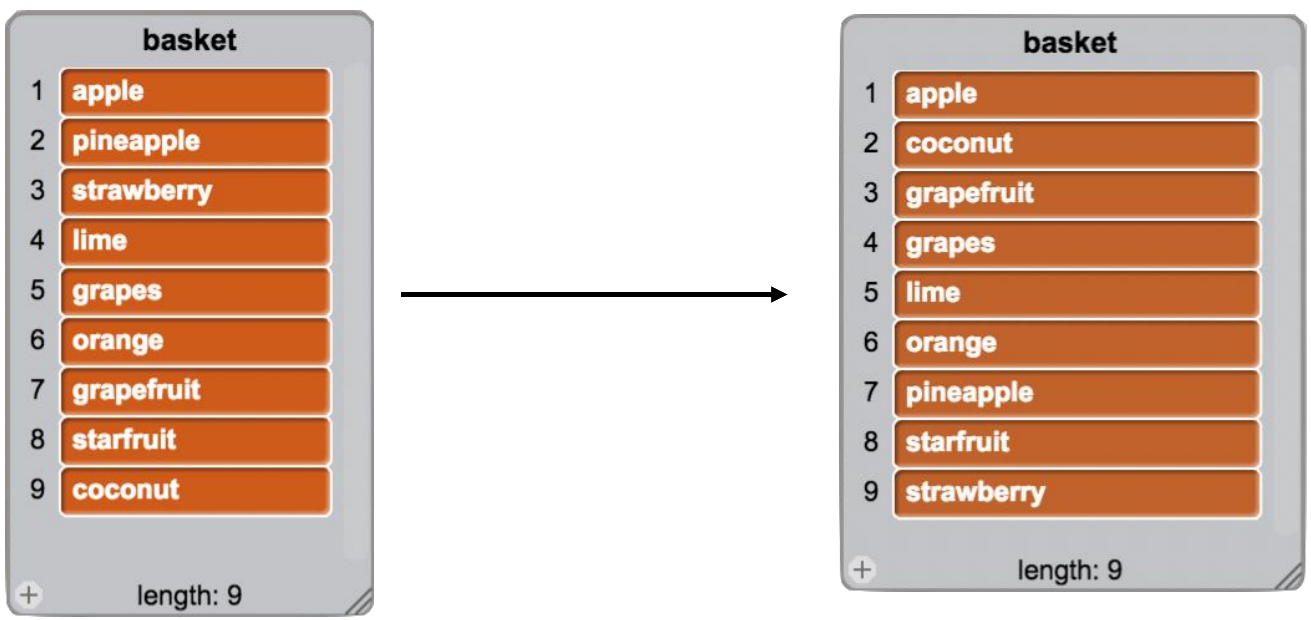
- *Input:*
  - a collection of orderable objects, call it “Basket”

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

Sort Algorithms

Problem Specification

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



Sort Algorithm #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 Algorithm #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 Algorithm #1

Random Sort

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

Sort Suggestions?

Any proposals?

Sort Algorithm #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 at the 2nd position from the beginning.
5. Rinse and repeat....
   (for the 3rd smallest, 4th smallest, …)

Sort Algorithm #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 Algorithm #2

Selection Sort

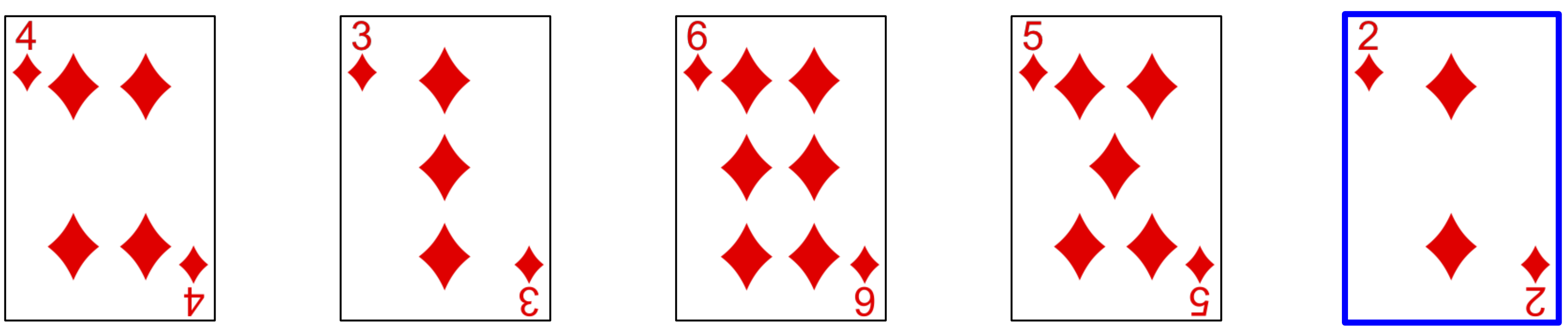
1. “Select” the smallest item in the list.

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 Algorithm #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 Algorithm #2

Selection Sort

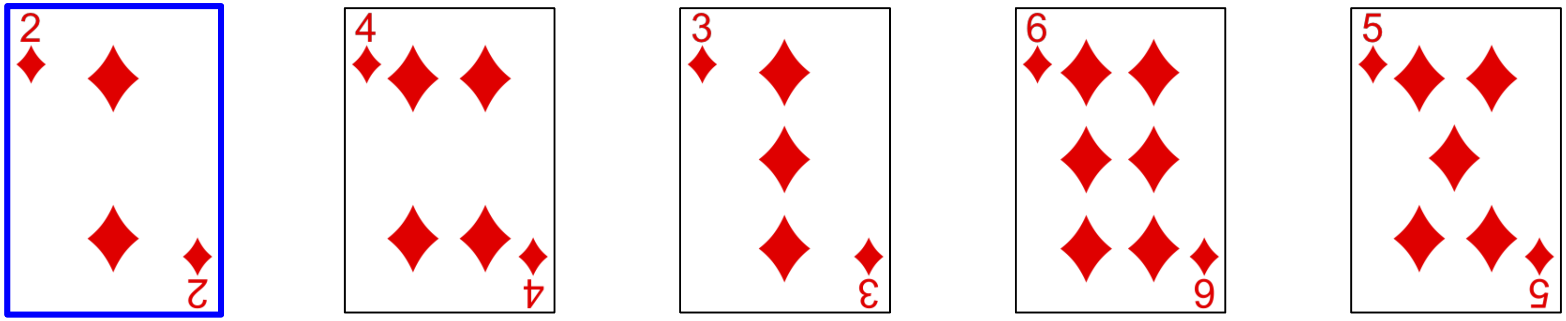
1. “Select” the smallest item in the list.

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 Algorithm #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 Algorithm #2

Selection Sort

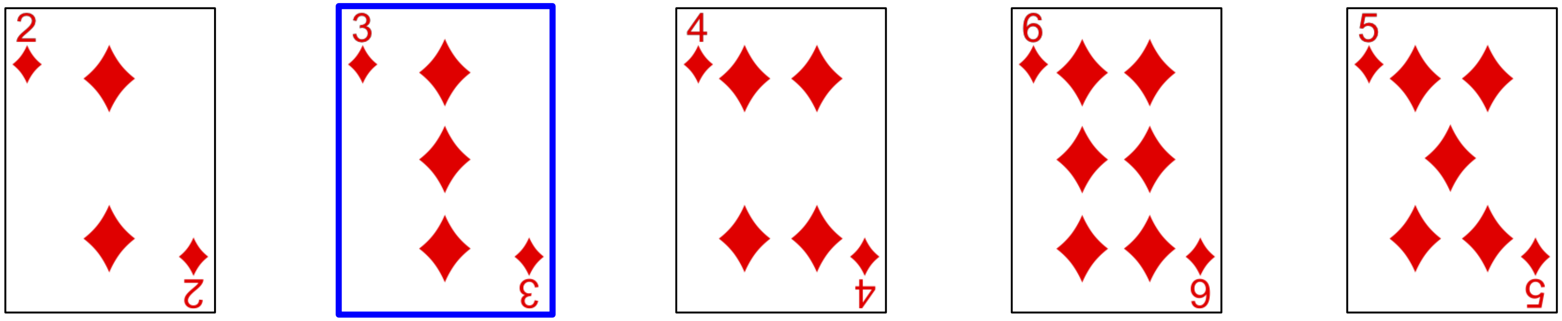
1. “Select” the smallest item in the list.

2. Put it at the beginning.

3. “Select” the second smallest item.

4. Put it 2nd from the beginning.

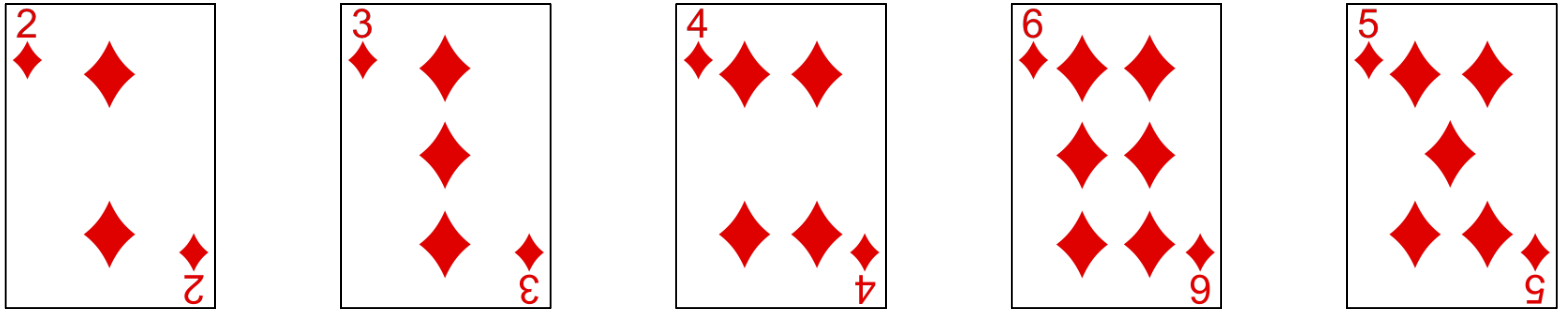
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Algorithm #2

Selection Sort

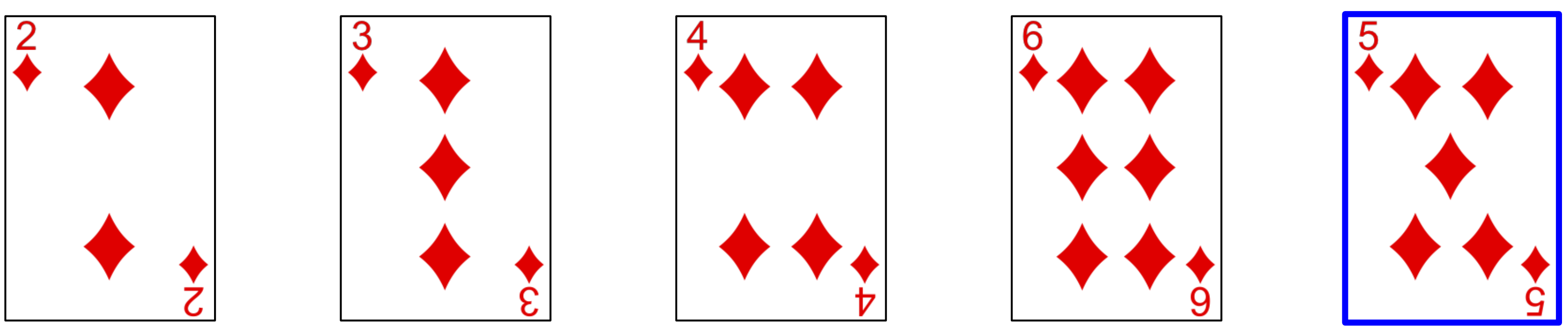
1. “Select” the smallest item in the list.
2. Put it at the beginning.
3. “Select” the second smallest item.
4. Put it 2nd from the beginning.
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Algorithm #2

Selection Sort

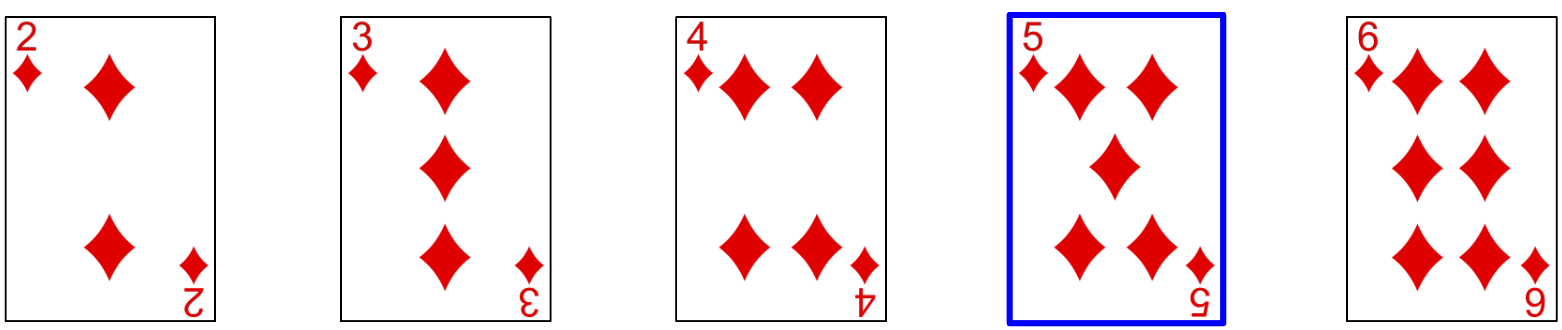
1. “Select” the smallest item in the list.
2. Put it at the beginning.
3. “Select” the second smallest item.
4. Put it 2nd from the beginning.
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Algorithm #2

Selection Sort

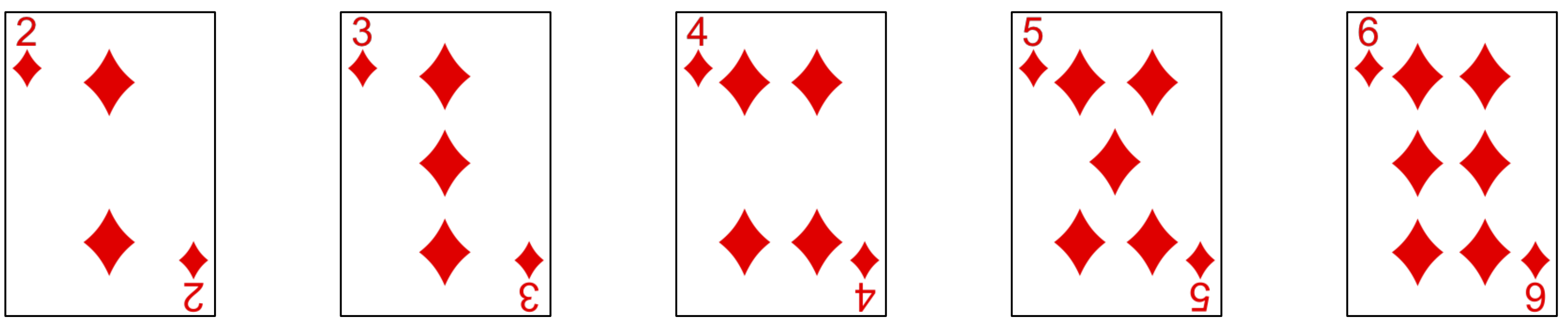
1. “Select” the smallest item in the list.
2. Put it at the beginning.
3. “Select” the second smallest item.
4. Put it 2nd from the beginning.
5. Rinse and repeat.... (for the 3rd smallest, 4th smallest, …)



Sort Algorithm #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 Algorithm #2

Selection Sort

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

Lecture Overview

- Algorithm Examples
- Search Algorithms
  - Three flavors of search (Random, Linear, Binary)
- Sorting Algorithms
  - 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:

      A 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?

We will cover functions in week #5.

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:

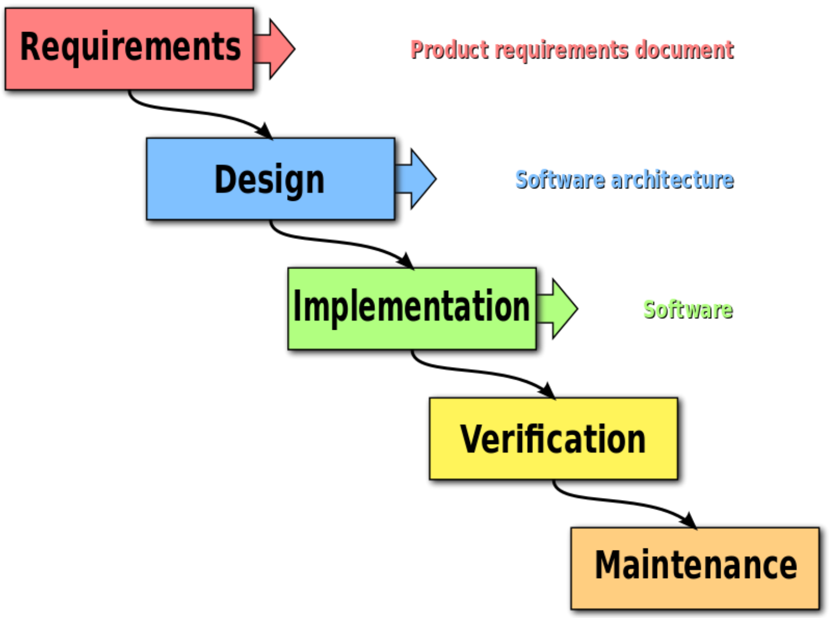
1. Define the problem
2. Decide upon an algorithm
3. Translate it into code

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

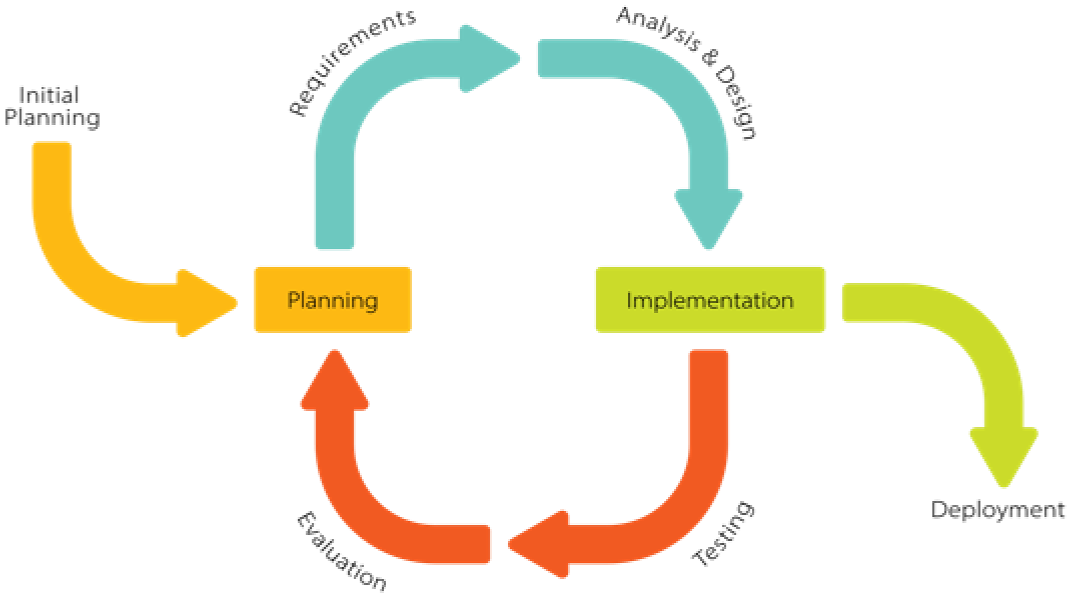
- Before the iterative model, we had the waterfall strategy.
- The **waterfall model** is a breakdown of project activities into linear <u>sequential</u> phases
- Each step handled once.
- The model had a limited capability and received too many criticism.
- Better than nothing!!
- Do not dive in to 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).
- The basic idea behind the iterative development is to develop a system through repeated cycles and in smaller portions at a time (incremental)
- Allows software developers to take advantage of what was learned during development of earlier parts or versions of the system.



** From wikipedia Iterative development model*