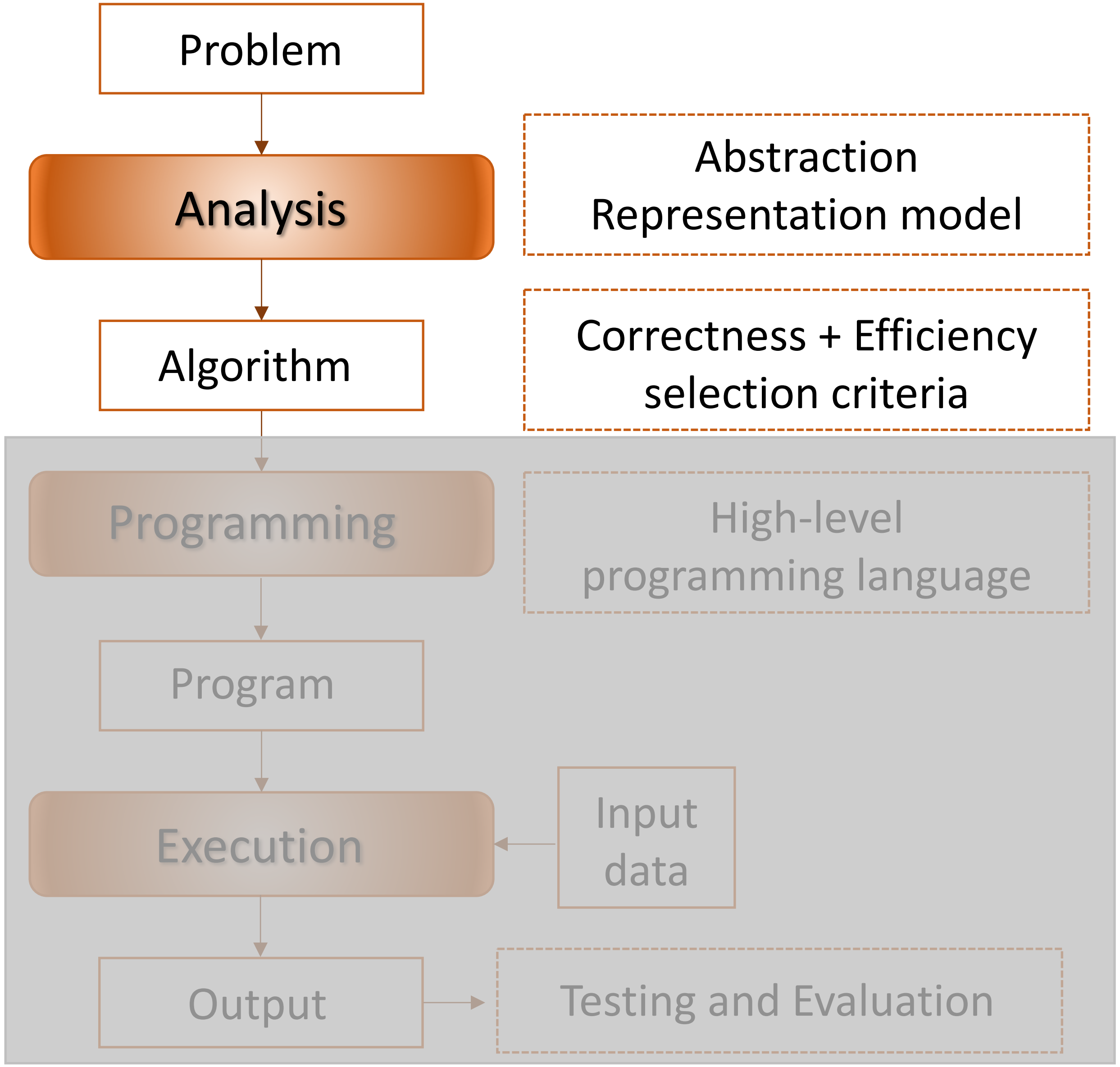
15-110 Principles of Computing – F21

Lecture 2:

Algorithms!

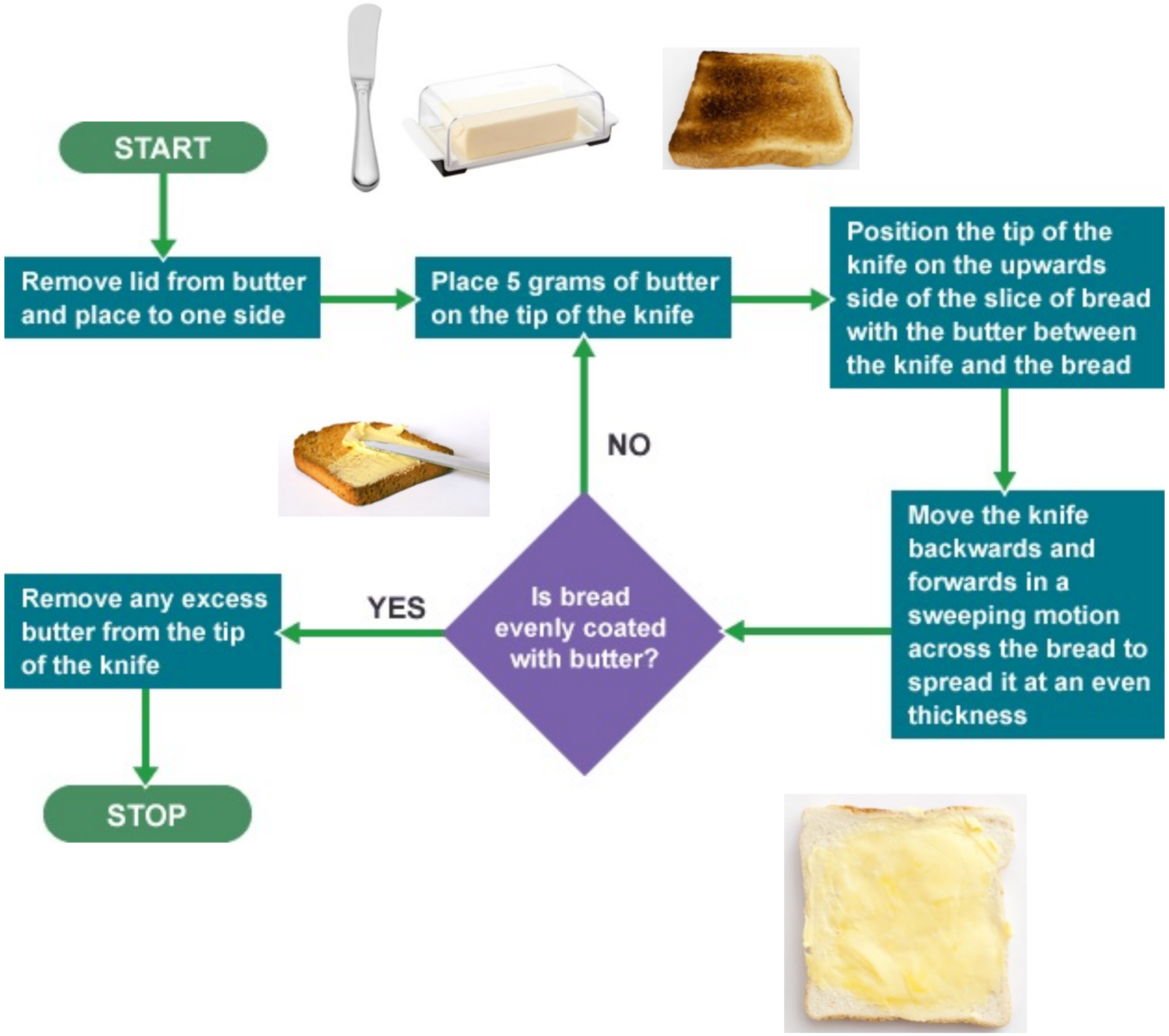
Teacher:
Gianni A. Di Caro

Computational problem solving



- **Algorithm:**
  - ✓ A finite list / sequence of instructions that describe a **computation**
  - ✓ when the instructions are executed on a provided set of inputs, the computation proceeds *step by step* through a set of well-defined states (configurations)
  - ✓ eventually, it ends, with some outputs being produced
- **Program:**
  - ✓ Algorithm encoding using a language that the computer understands
  - ✓ *> 700 programming languages!*
  - ✓ Primitive constructs, syntax, static semantics, semantics

Algorithms: common traits



- ❖ Identify **essential elements** for solving the problem (*Abstraction* step)
- ✓ **Start up** actions / conditions
- ✓ Actions to execute at **each step**
- ✓ **Inspect** the situation to **make choices**
- ✓ Choices create **decision branches**
- ✓ **Repeat** sub-sets of actions
- ✓ Condition to **stop computation**
- ➢ **Save** intermediate results

A *selection* problem: Choose a snack with the lowest intake calories

- ❖ Identify **essential elements** for solving the problem (*Abstraction* step)
- ✓ **Start up** actions / conditions
- ✓ Actions to execute at **each step**
- ✓ **Inspect** the situation to **make choices**
- ✓ Choices create **decision branches**
- ✓ **Repeat** sub-sets of actions
- ✓ Condition to **stop computation**
- ✓ **Save** intermediate results

❖ You have to choose among **5 snacks**

✓ You want to choose the one with the lowest intake calories

- Snack pack and its nutritional facts are the necessary elements
- Let's assume the snacks are in a heap in front of you

1. At random, pick-up a *first* snack from the heap and check its calories
2. Put the snack aside, in the *selected* location (e.g., to your left)
3. At random, pick-up a snack from the heap
    4. Read its calories
    5. If its calories are lower than the previously selected snack, put the current snack in the *selected* location
        6. Remove the previous snack from *selected* and put it in the *rejected* location (e.g., to your right)
    7. Instead, if the calories are higher than the previously selected snack, put the current snack in the *rejected* location
8. Repeat steps 3 – 7 four times
9. The snack in the *selected* location is the one you'll eat!

A *selection* problem: Choose a snack with the lowest intake calories

- You have to choose among **5 snacks**
- ✓ You want to choose the one with the lowest intake calories

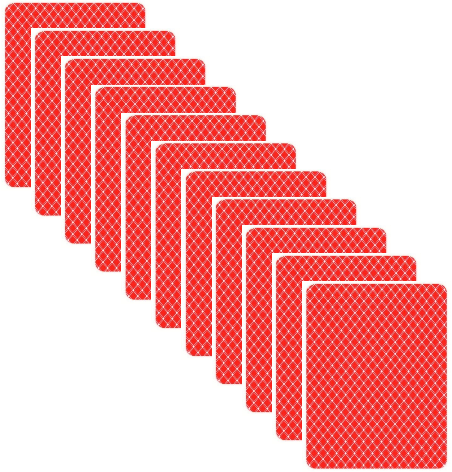
  - ✓ Different variants are possible for the previous algorithm

  - ➢ Don't remove the snack from the initial heap.
    → Need to **save/memorize** the information about the best snack so far to retrieve it at the end

  - ➢ Order (how?) the snacks according to increasing calories and select the first in the ordered list

  - ➢ ...

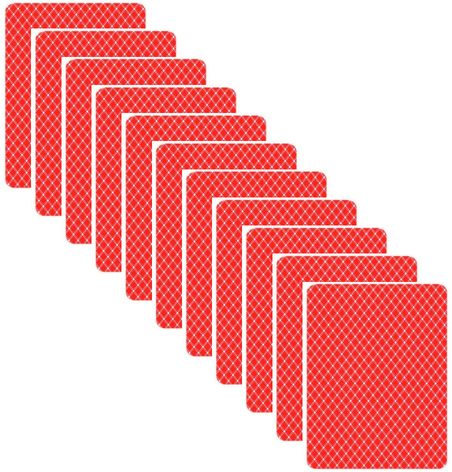
A simple *search* problem: Find the card!



- You are given a set of cards (covered) as show in the figure
- Cards are uniquely numbered from 1 to 100, but of course they aren't necessarily placed in that order!

➢ You must find the card with number 100

A simple search for *max/min* problem



- You are given a set of cards (covered) as show in the figure
- Cards are uniquely numbered from 1 to 100, but:
  - You don't know it,
  - You don't know how many cards are there
  - Cards aren't necessarily placed in the 1-100 order!

➢ You must tell the highest and the lowest card values in the set

➢ You can only inspect the cards, but not put them aside (e.g., cards are on a computer screen!)

➢ You can *write* down things / Memory!

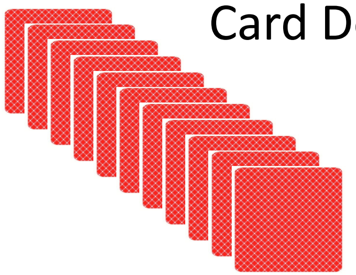
A simple search for *max/min* problem

A detailed step-by-step solution

1. Pick up the first card from the deck pile
2. Record down the number and remove the card from the deck (put it in done pile)
3. Assign the number to min value and to max value

4. Pick up the next card from the deck
5. Look at the number, n, and remove the card from the deck
6. If the number is higher that current max value: max value becomes n
7. If the number is lower that current min value: min value becomes n

8. Repeat 4-7 **until no more cards in deck**
9. Read/Output min value and max value
10. Stop

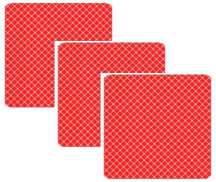


Card Deck

Min value: XX

Max value: YY

Done pile



A simple search for *max/min* problem

Example of proposed solution that has an issue: how do we do sort? → Need another algorithm ☺

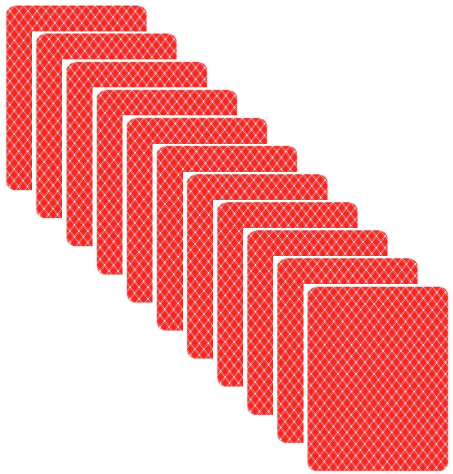
1.) Pick up the first card, pick up the pencil and note down the number that is written on the card on the piece of paper that is on the left side .

2.) repeat the first step by picking up the NEXT card and note down the value.

3.) repeat steps 1-2 until you run out of cards

4.) compare the numbers that you have written down by sorting them in ascending order. Identify the lowest and the highest values.

A *sorting* problem (and you know the numbers)



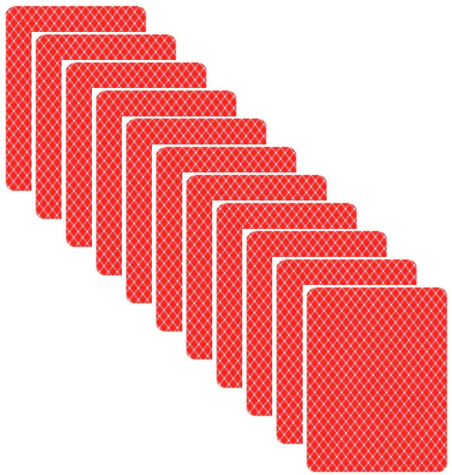
- You are given a set of cards (covered) as show in the figure
- Cards are uniquely numbered from 1 to 100, but cards aren't necessarily placed in the 1-100 order!

➢ You must **sort** the cards in the 1 → 100 order

Find an algorithm that explicitly makes use of the information that card numbers are between 1 and 100

(check the Technical Notes on the course website to find help, if needed)

A more general *sorting* problem: you don't know the numbers



- You are given a set of cards (covered) as show in the figure
- Same problem as before, but now you don't which is the range of values of the cards, you don't know how many cards are there

➢ You must **sort** the cards in increasing order

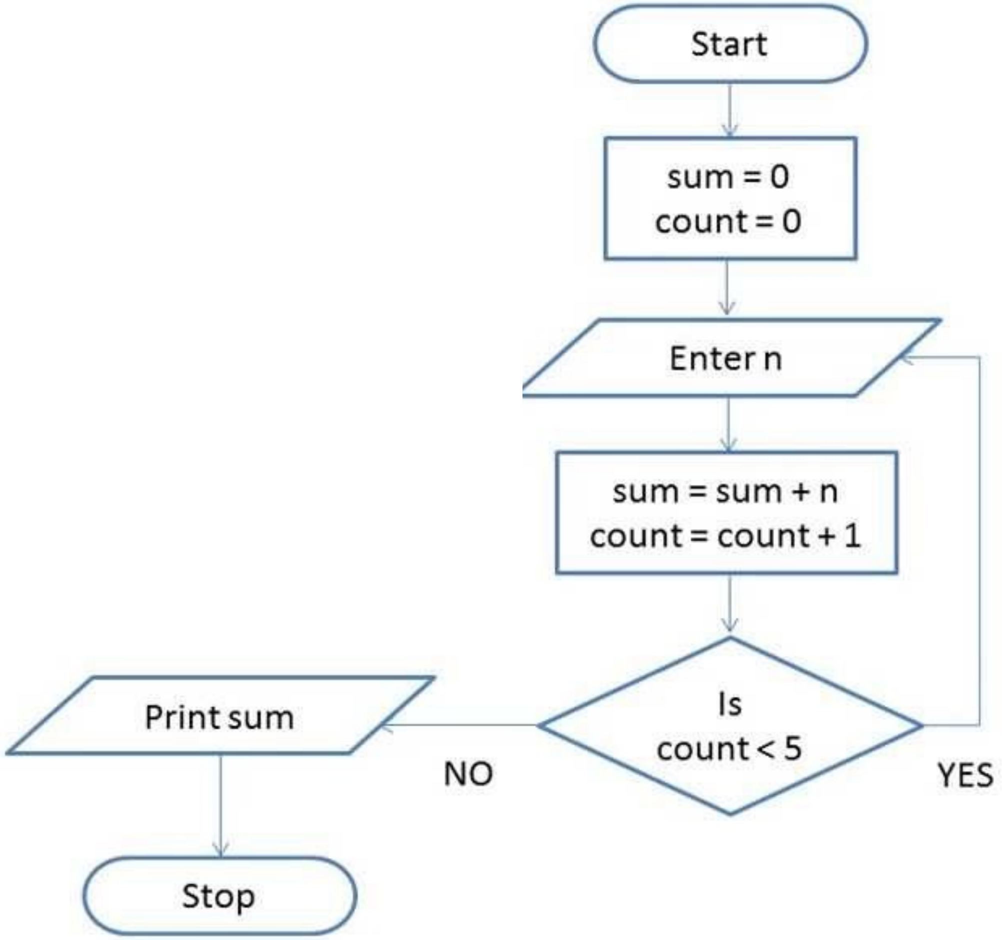
A possible solution:

1. Pick up first card from deck
2. Add the card to sorted pile
3. Pick up first card from deck
4. If card value greater than top card on sorted pile
   - Then add card on top of sorted pile
5. Instead, if card value is lower that bottom card on sorted pile
   - Then add card to the bottom of sorted pile
6. If neither 4 or 5 conditions are satisfied, *insert* card in sorted pile
7. Repeat 3-6 until no cards in card deck

Insert:

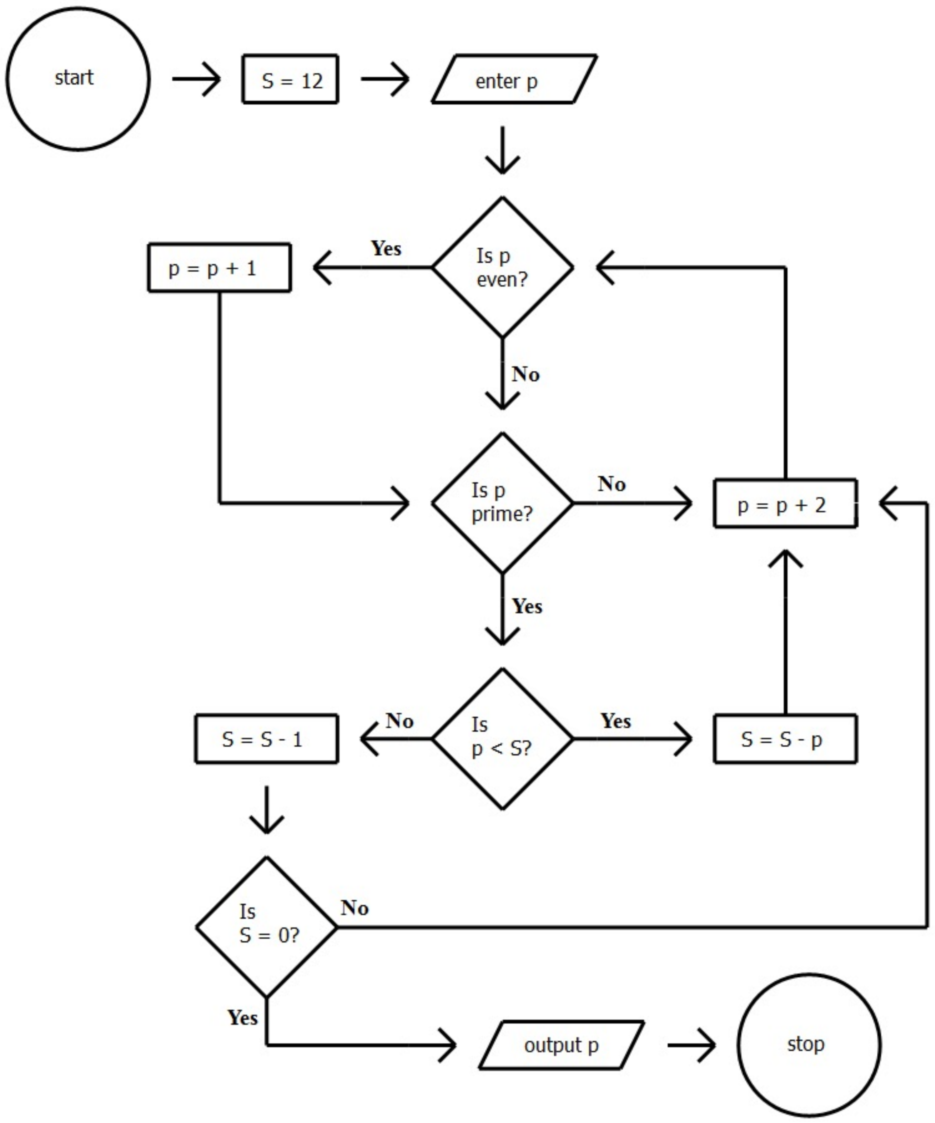
1. If distance from bottom is less than distance from top, start from bottom
2. Otherwise, start from top
3. Inspect the first two cards from start position
4. If card value is lower than first and higher than second, insert card after first
5. Otherwise, set first card as new start position
6. Repeat 3-5 until the new card is inserted

More on algorithms: Decode a flow chart



What is this flow chart doing?

More on algorithms: Decode a flow chart



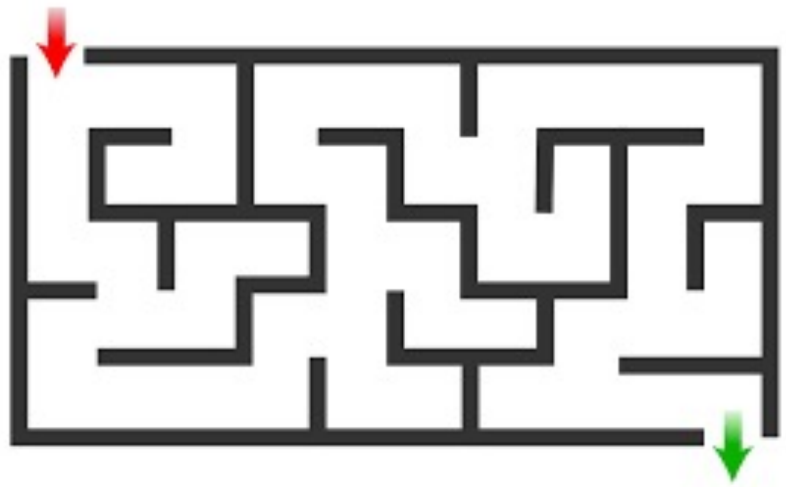
What is this flow chart doing?

What is the output for p=3?

More problems: Navigating out of a maze

Getting out of a maze!

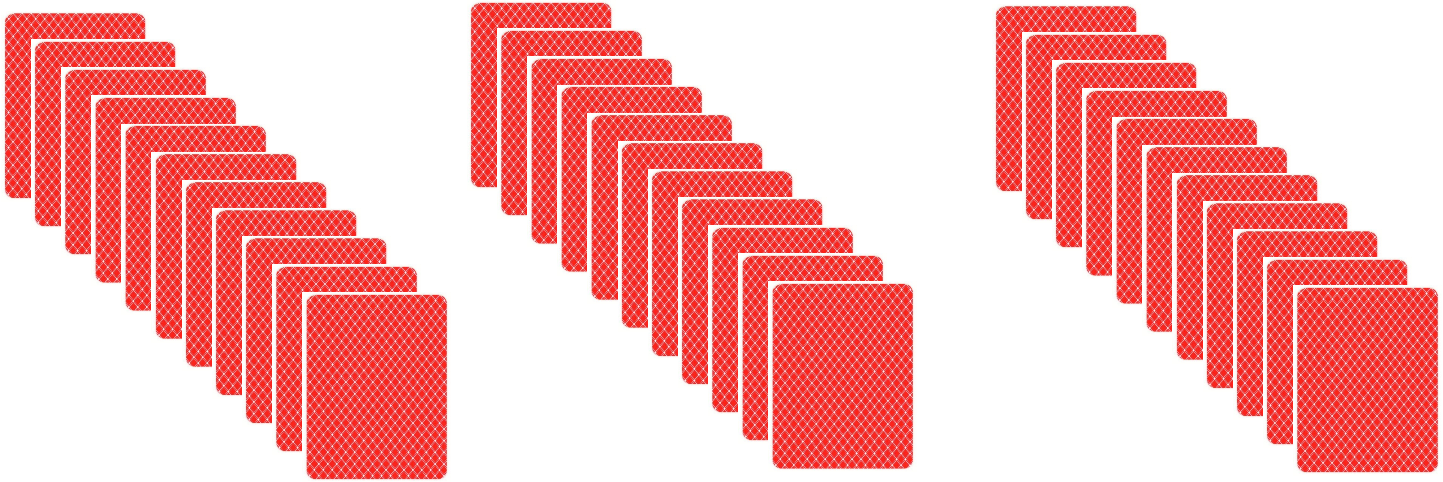
- Describe the step-by-step algorithm (sequence of instructions) to get out of the maze (from the red to the green open accesses)
  - You don’t know distances and don’t know how measure them, but you have a sensor that let you know when you’re hitting a wall
  - You can move straight forward and rotate in place of the desired angle (clockwise or anti-clockwise)



More problems: Merge sorted piles of cards

Split the job among your friends!

- Still need to sort a pile of n cards, and you don't know their ranges.
- Luckily, you have three friends who can help you in the task. You split the cards in three piles, giving each pile to each one of your friends (what if n is not precisely divisible by 3?)
- Each friend know how to sort a pile of cards, such that he/she will return a sorted pile of cards (e.g., sorted in ascending order).
- Now you need to use / merge the sorted piles to create a single sorted pile of cards, which will be your output



More problems: Pair the socks in a heap!

Pair the socks!

- Taking care of organizing clothes at home, sometimes can be a very frustrating job! Image you have an unorganized heap of (many) socks, and you need to pair all the socks and put them aside to bring them to the wardrobe.
- Socks might have different colors, patterns, shape.
- Define two different algorithms that can achieve the task.
- Discuss which one of the two algorithms you think is more efficient (i.e., it would take less time to complete)