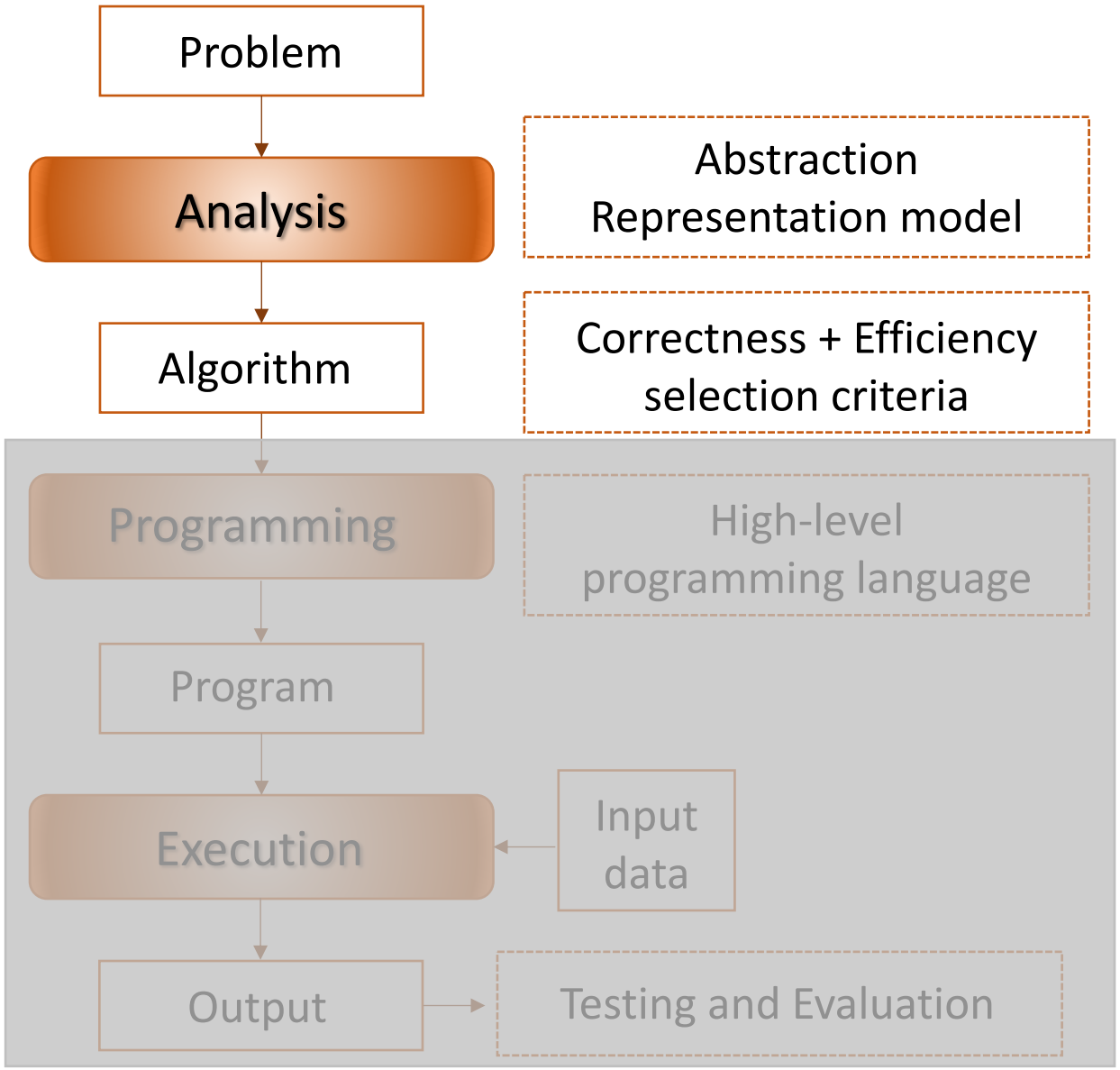
15-110 Principles of Computing – S21

Lecture 2: Algorithms!

Teacher:
Gianni A. Di Caro

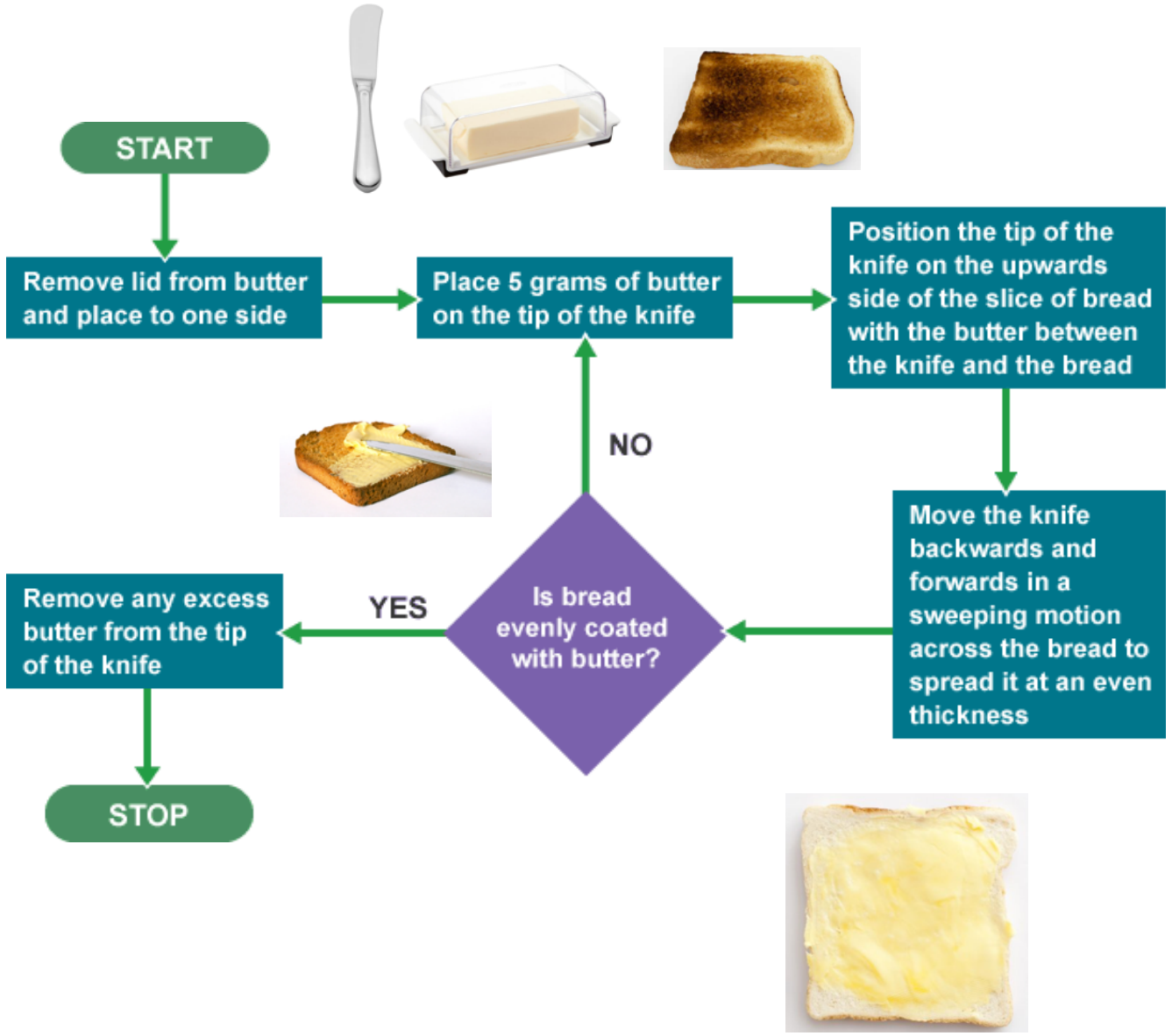
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

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 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 and 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!

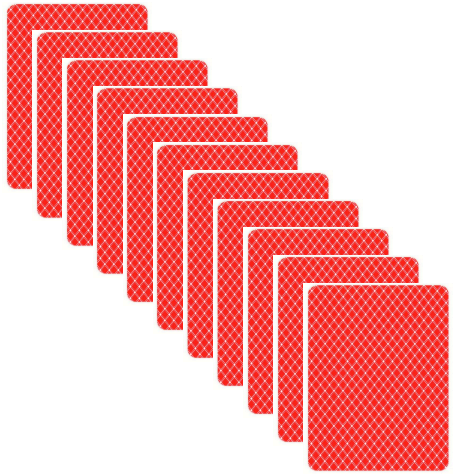
A *selection* problem: Choose a snack with the lowest intake calories

- ❖ 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 algorihtm

- ➢ 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!

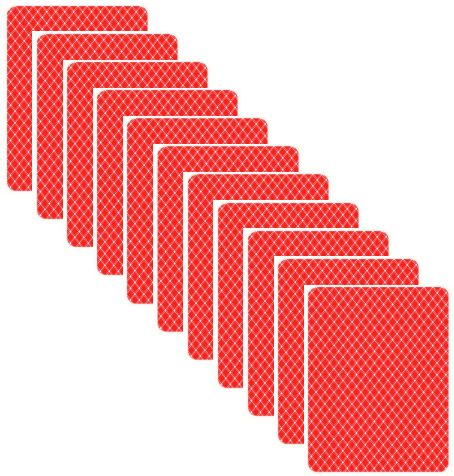
➢ You must find the card with number 100

Two examples of proposed solutions:

1.) Pick up the first card, check if it's 100.
   If It is, hand it to the me.
      Stop.
   If it is not, place it on the left-hand side.
2.) Repeat step one for 99 times.

1) pick up at random a card from deck and check the number written
   2) if the number is 100 put it in the correct pile and stop
   3) if the number is not 100 put it in the rejected pile
4) repeat steps 1-3 99 times

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

Example of proposed solution: 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.

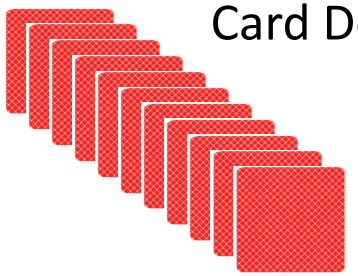
A simple search for *max/min* problem

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

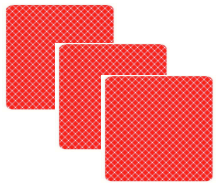
8. Repeat 4-7 **until no more cards in deck**
9. Read/Output min value and max value
10. Stop



Card Deck

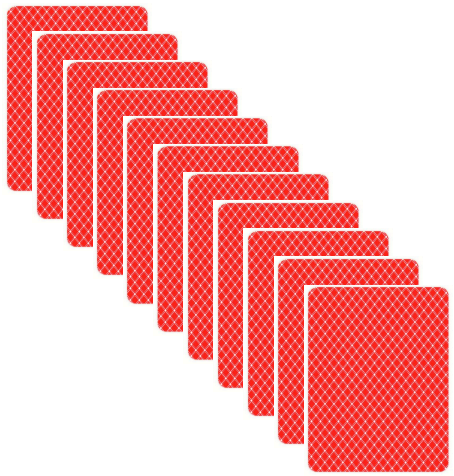
Min value: XX

Max value: YY



Done pile

A *sorting* problem: you know the numbers

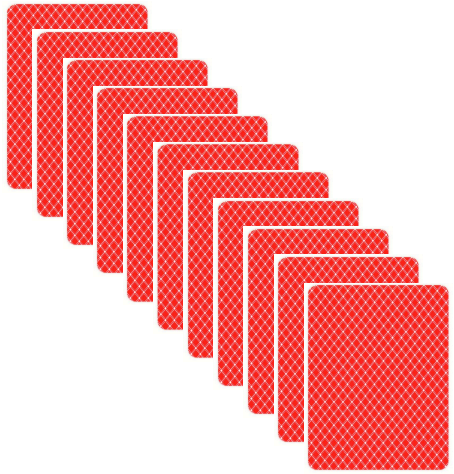


- You are given a set of cards (covered) as show in the figure
- Cards are uniquely numbered from 1 to 100, but cards aren't necessarily placed in the 1-100 order!

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

Tip: two piles, the **deck** and the **sorted** ones

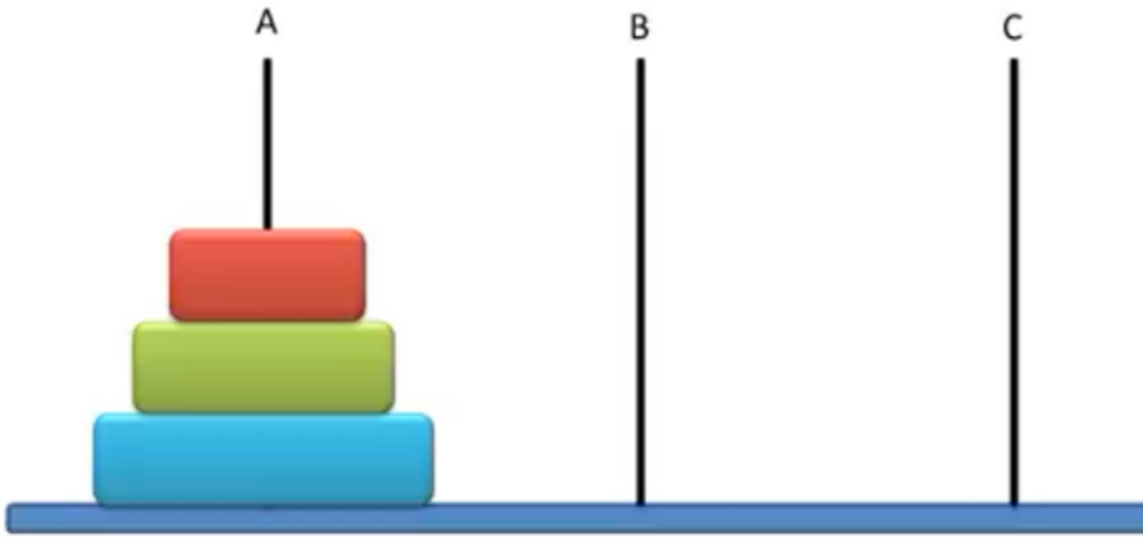
A more general *sorting* problem: you don't know the numbers



- You are given a set of cards (covered) as show in the figure
- Cards are numbered, where each number is a value within some range $n - m$ (e.g., numbers between 1 and 80) but you don't know the range values

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

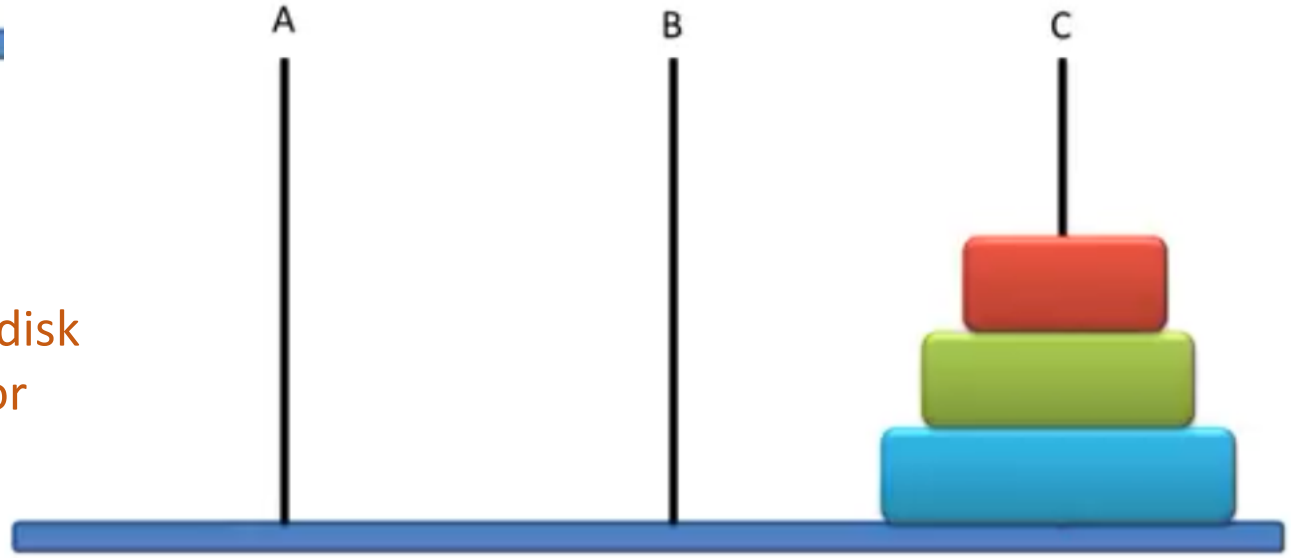
A search problem that needs a *strategy*: Tower of Hanoi



Start configuration

You can move *Disk* (r,b,g) to *Peg* (A,B,C)

Goal configuration



- Rule: A disk can only have on top of it a smaller disk (e.g., the blue disk cannot be on top of the red or green ones)

What if the initial configuration is a random one?

More on algorithms

Moves

A → C

A → B

C → B

A → C

B → A

B → C

A → C