YSC2229: Introductory Data Structures and Algorithms

Ilya Sergey

ilya.sergey@yale-nus.edu.sg

Some Terminology

- *Data* represents *information*
- *Computations* represent *data processing*
- An *algorithm* is a sequence of computational steps that transform the *input* data (given) into the *output* data (wanted).

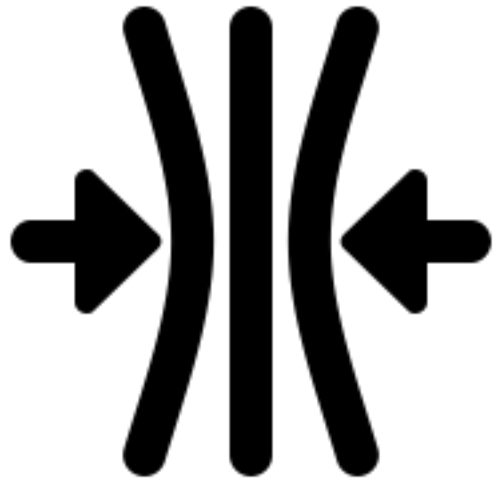
What this course is about?

Solving computational problems

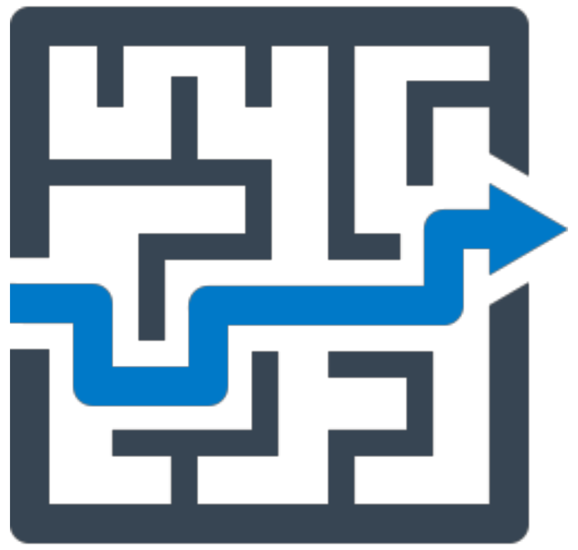
- **Searching:**
  finding a word in a text or an article to buy on Amazon

- **Storing and retrieving data:**
  representing files in you computer

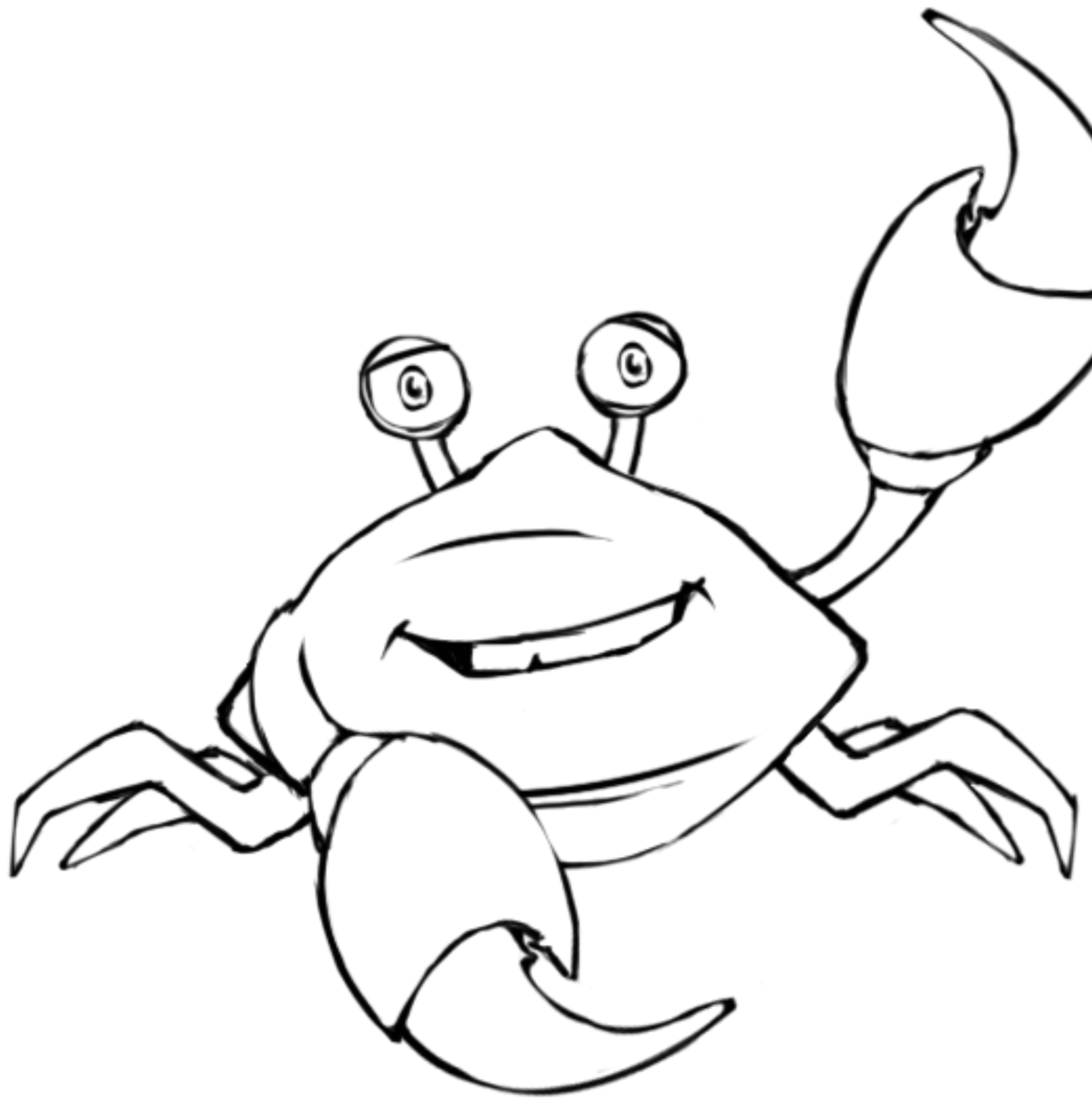
- **Data compression/decompression:**
  transferring files on the internet



- **Path finding:**
  getting from a point A to point B in the most efficient way



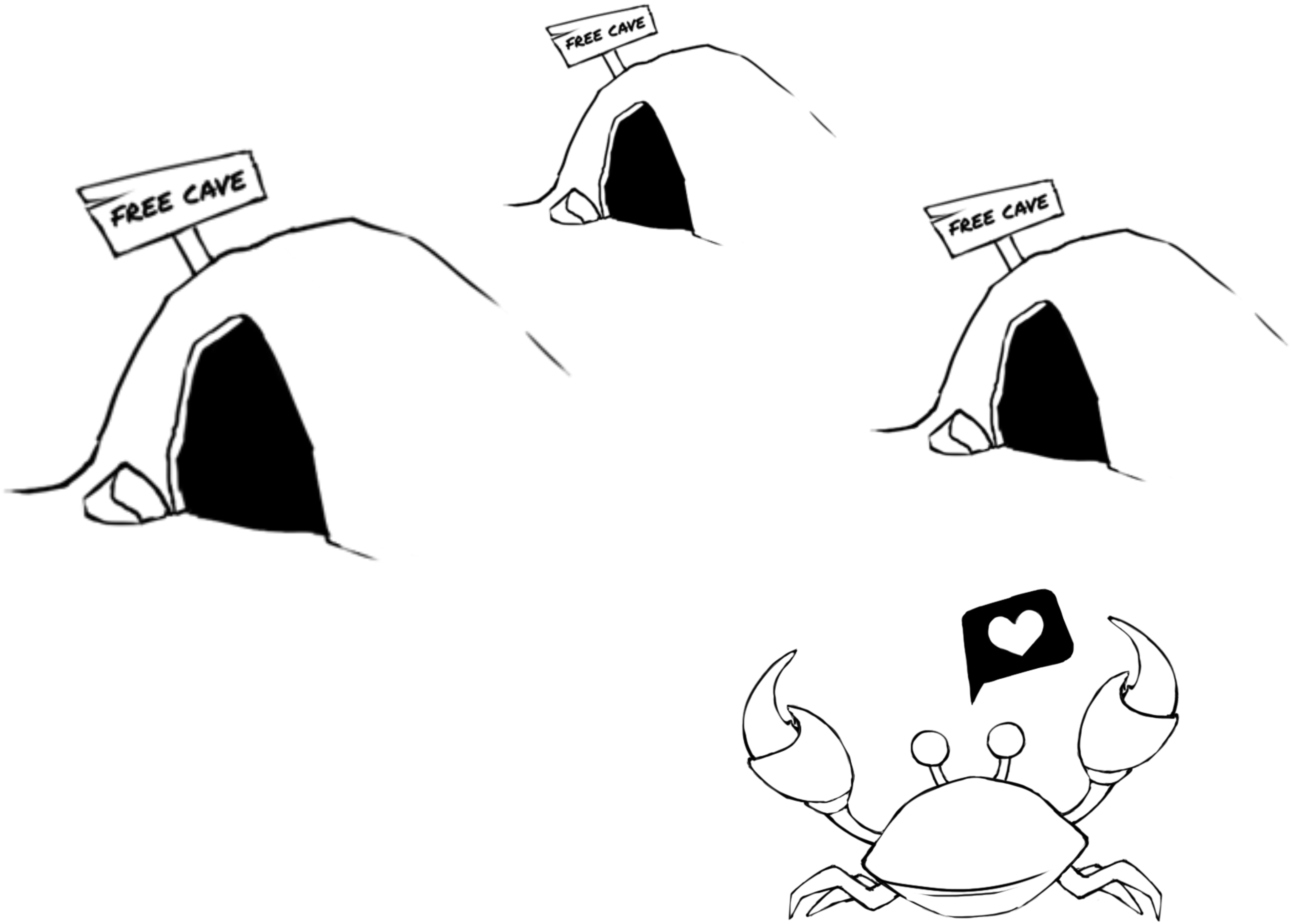
- **Geometric problems**:
  finding the closes fuel station, shape intersection



Torpe

(the prosperous crab)

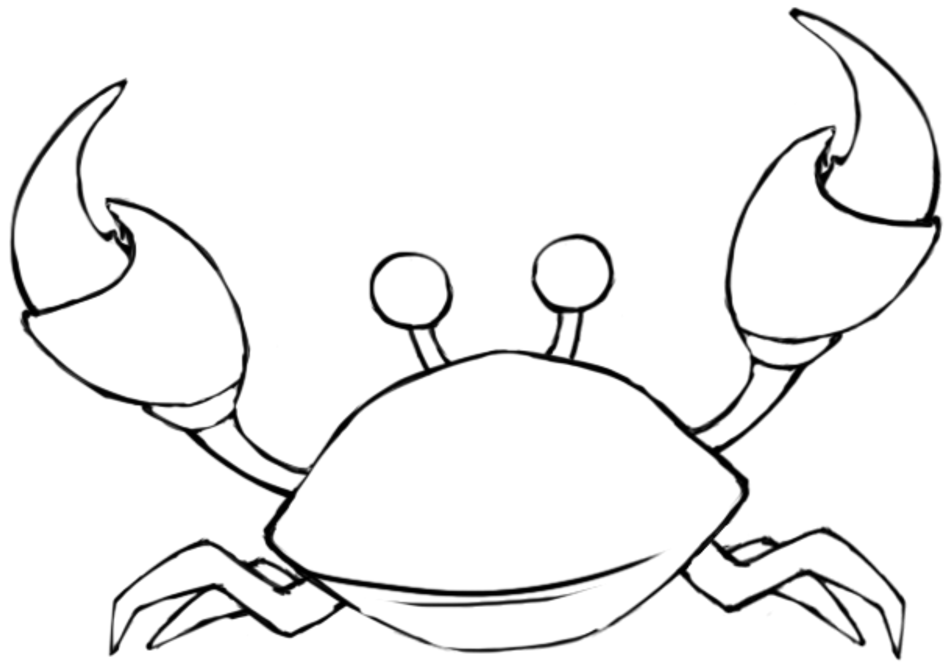

FREE CAVE

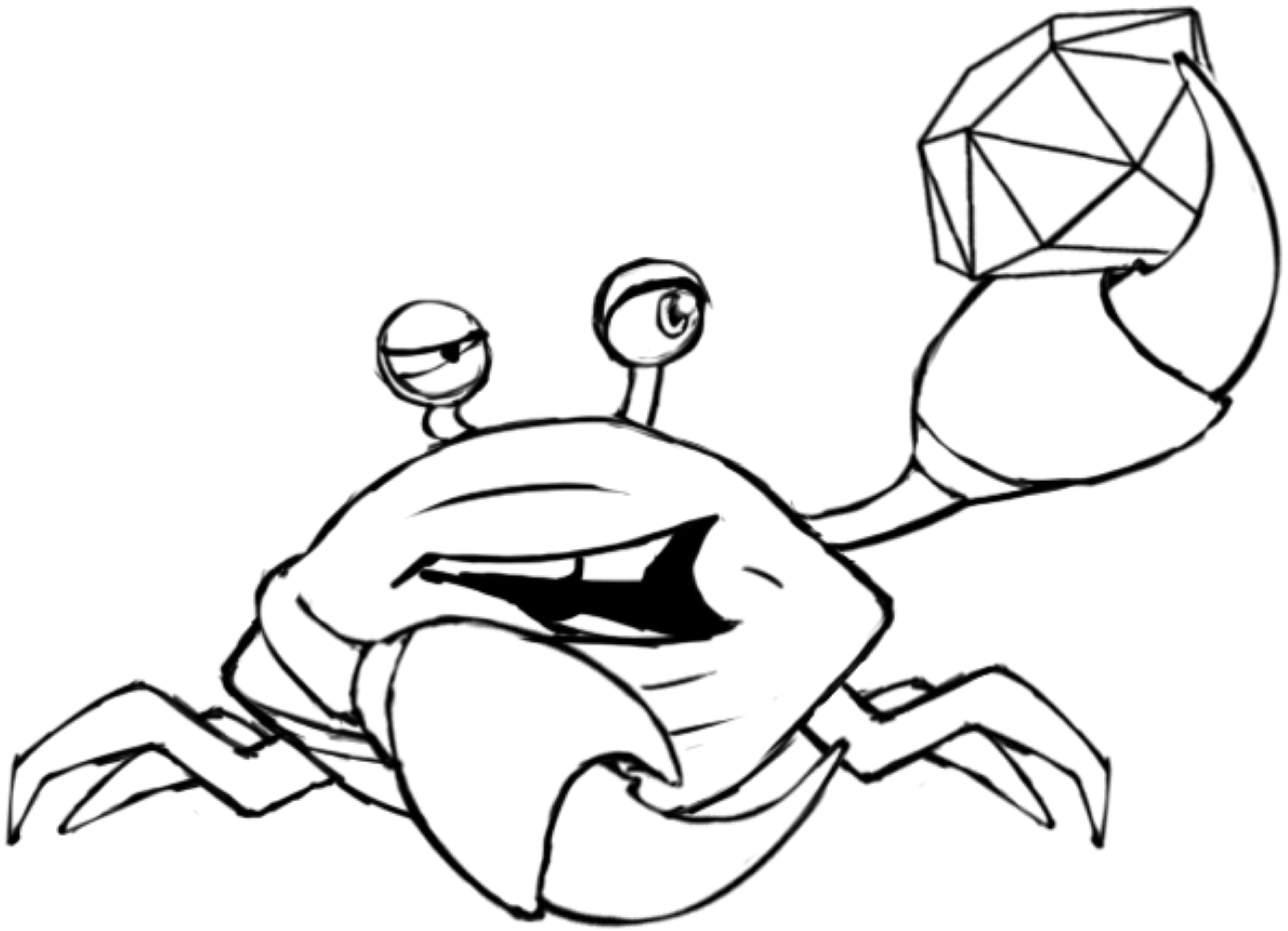

FREE CAVE
FREE CAVE
FREE CAVE


FREE CAVE

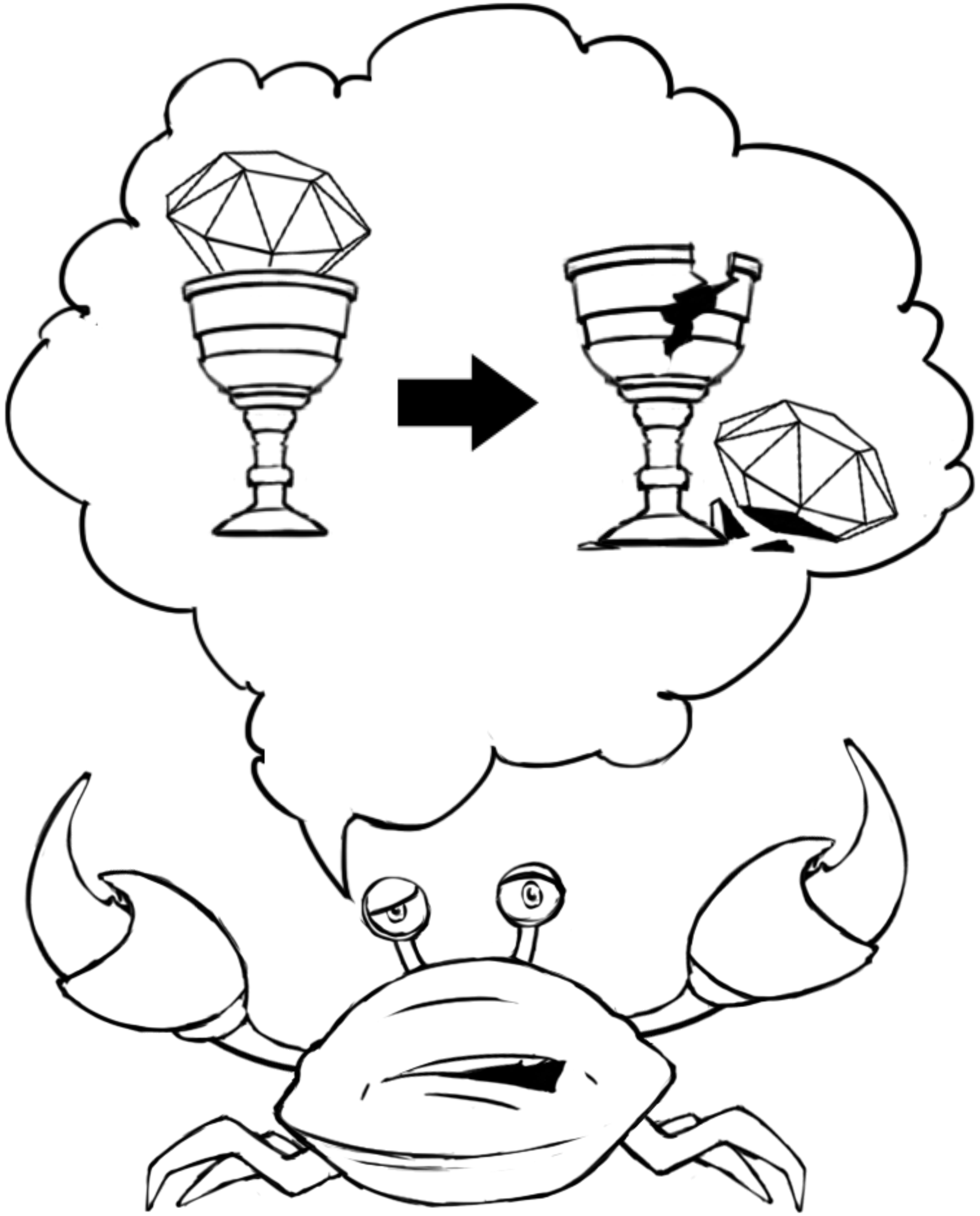

FREE CAVE

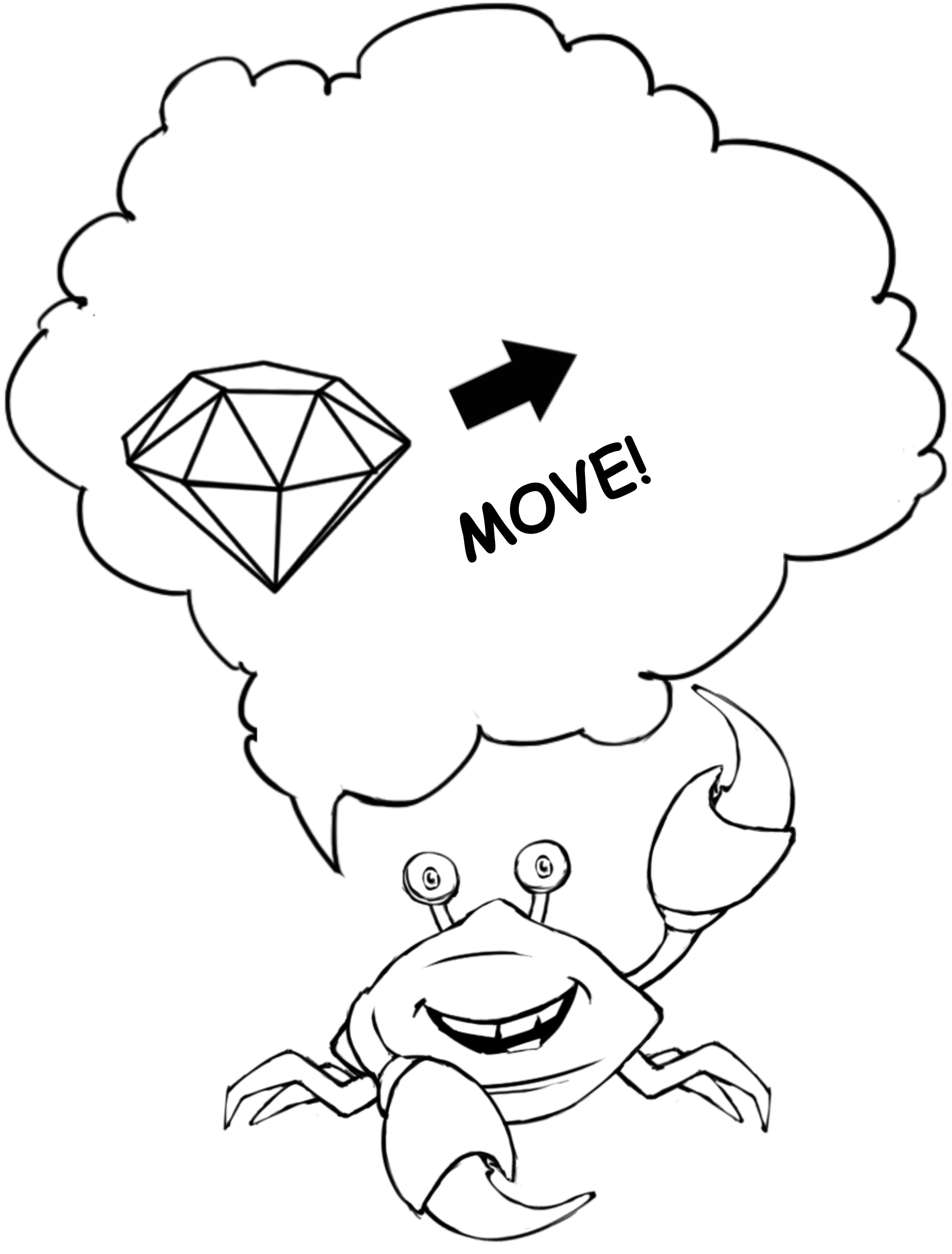

FREE CAVE

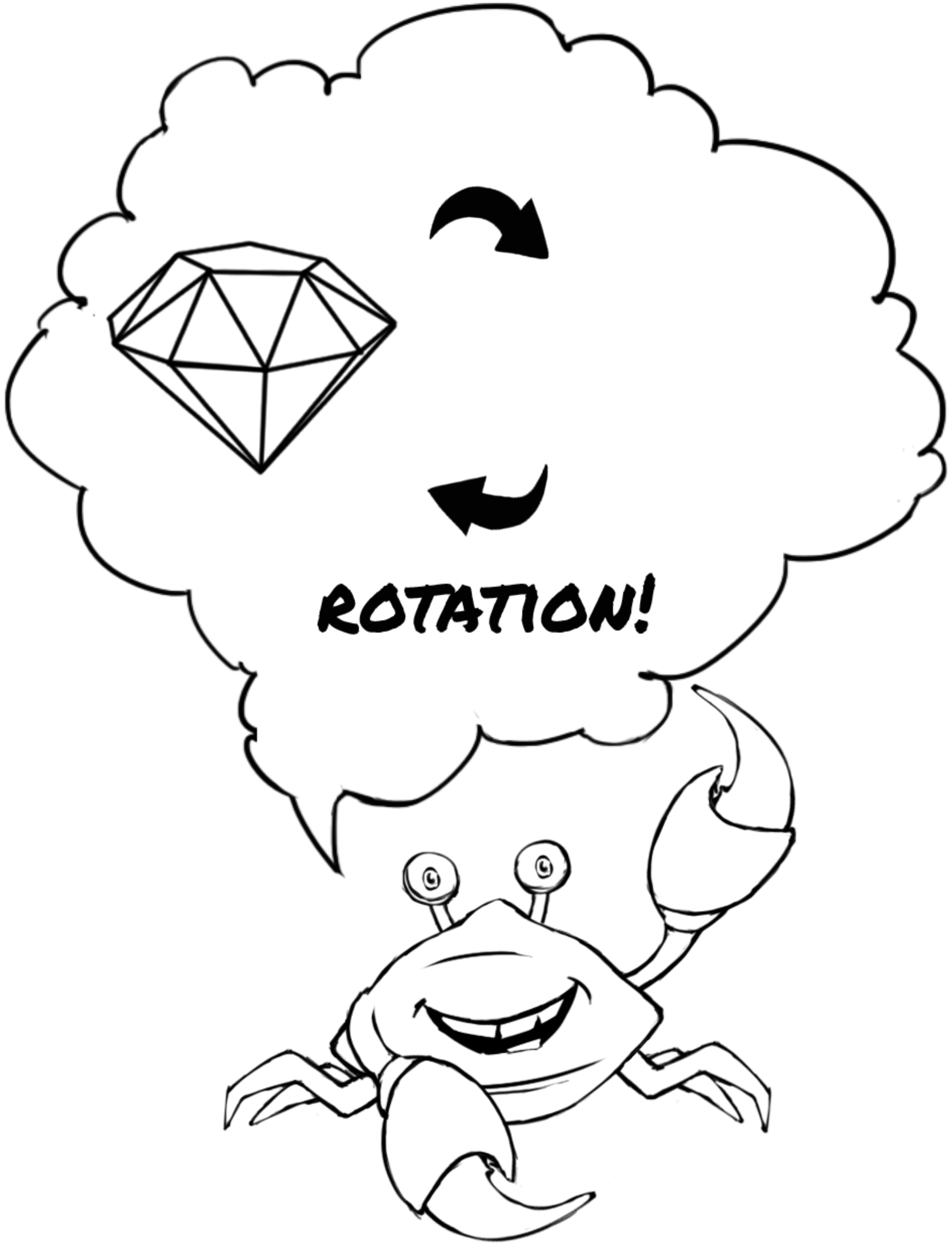


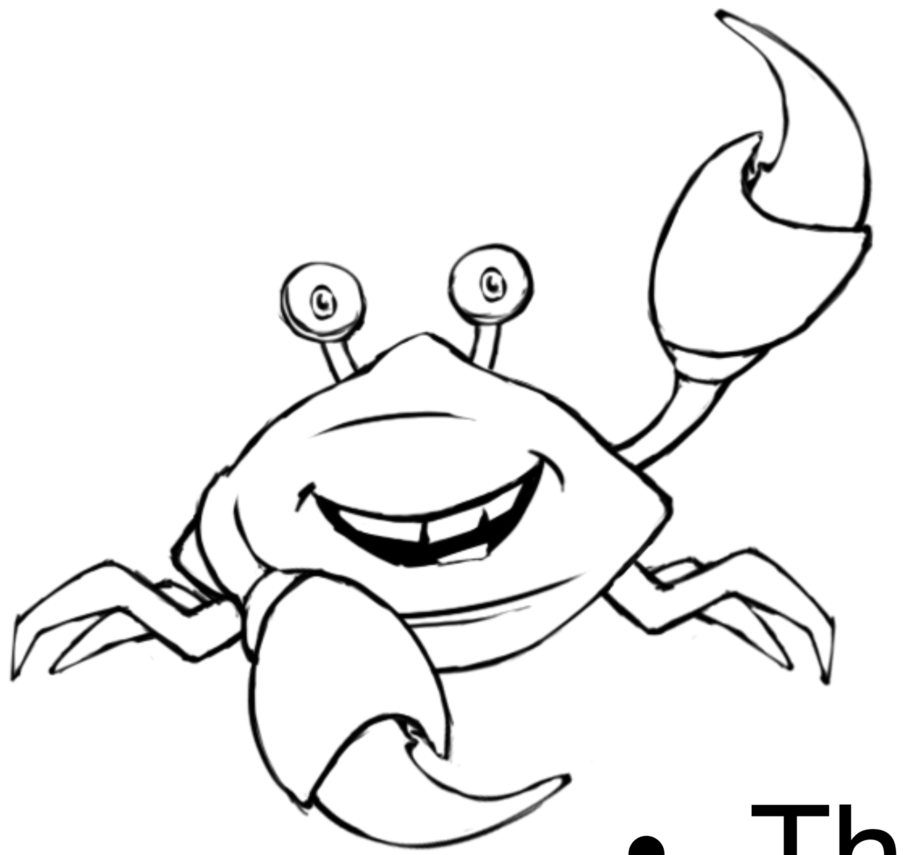



MAX $$$




MOVE!


ROTATION!

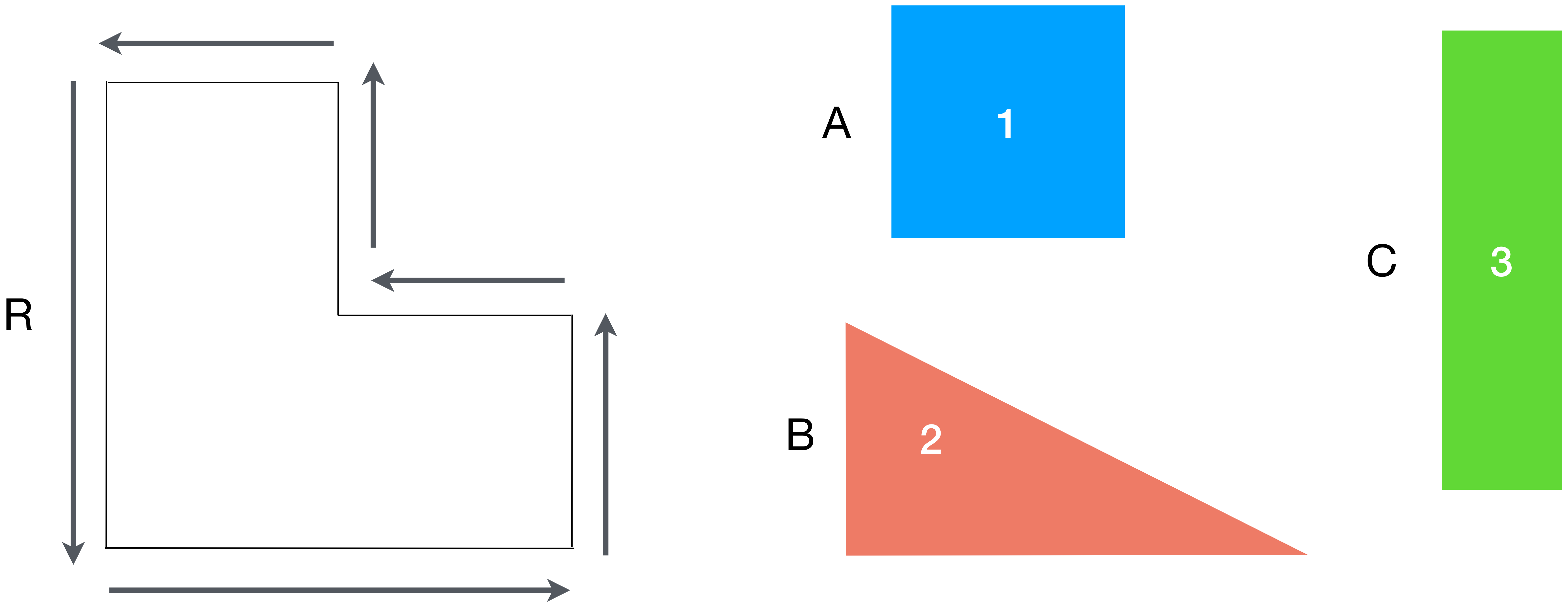


Rules of the Game

- **The problem**: pack Torpe's belongings into a cave (2D)
- **Requirements**:
  - No overlapping, all within the room, at least 30% covered
  - Try to find the best (maximal cost)
- **Available actions**:
  - Moving the furniture
  - Rotating the furniture

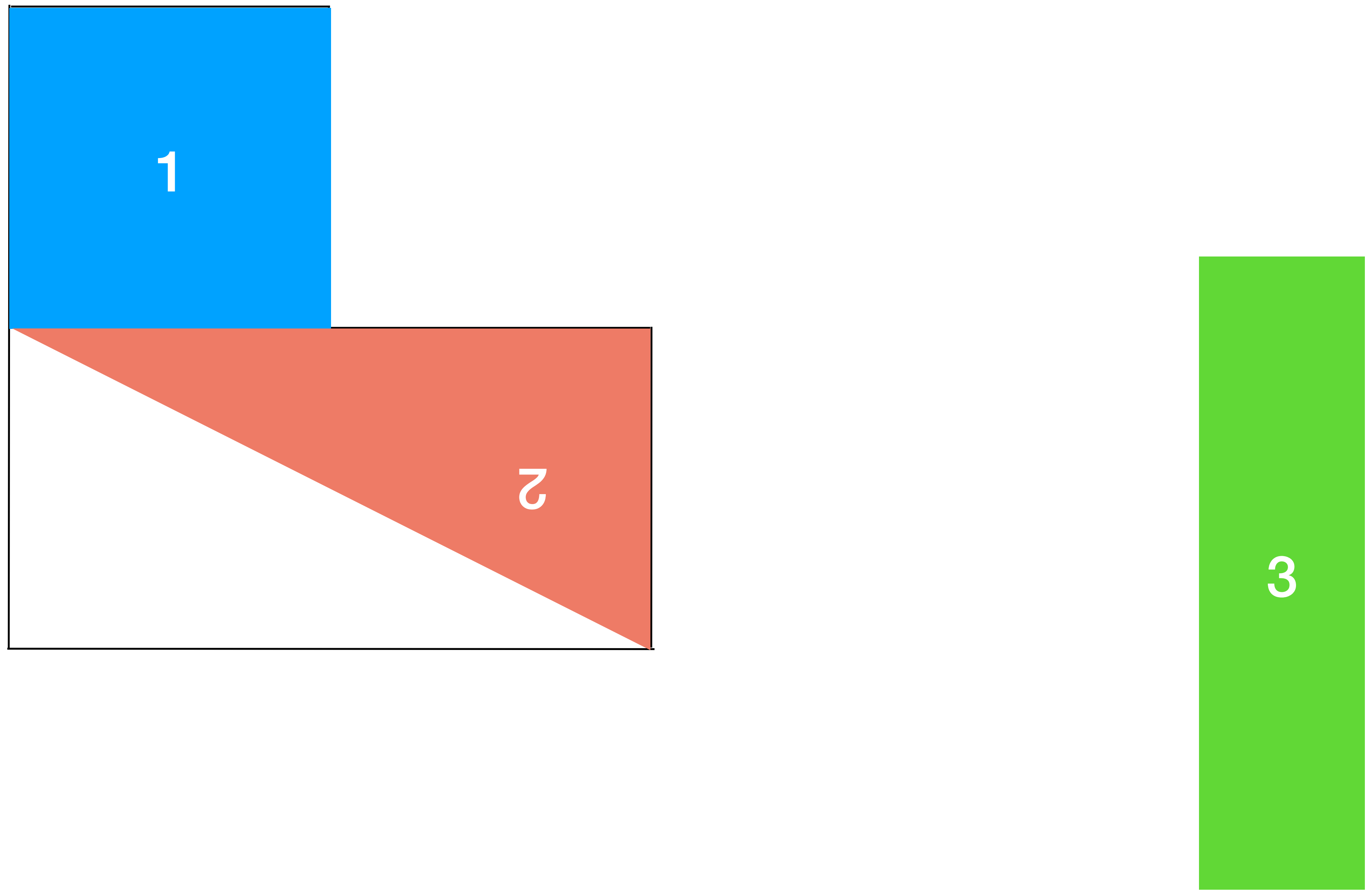
Data Structures

- Representing the cave
- Representing the furniture items
- Encoding the item costs
- Encoding the solutions



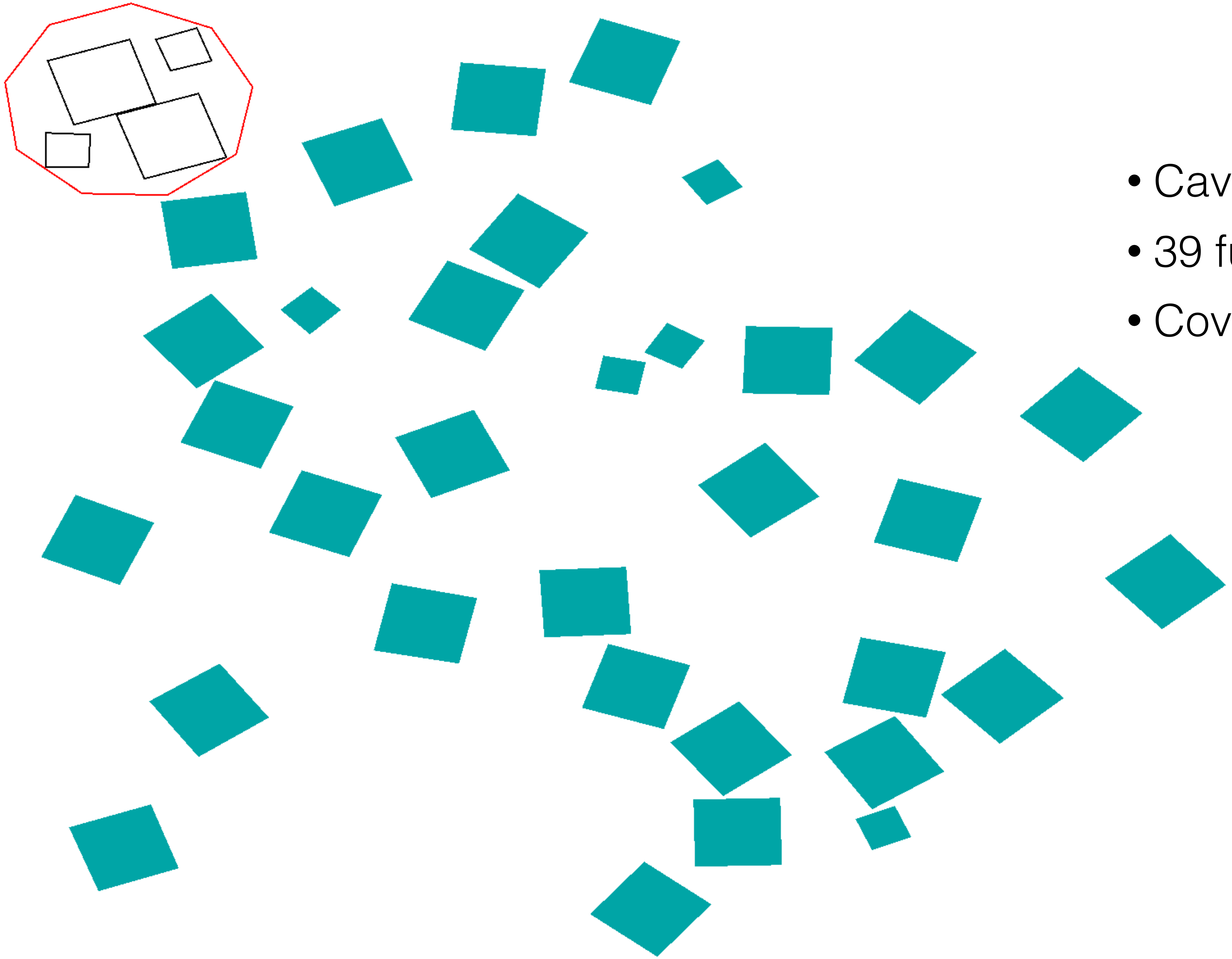
1: (0,0), (2,0), (2,1), (1,1), (1,2), (0,2) # 1:(0,0), (1,0), (1,1), (0,1); 2:(0,0), (2,0), (0,1); 3:(0,0), (0.5,0), (0.5,2), (0,2)

R A B C

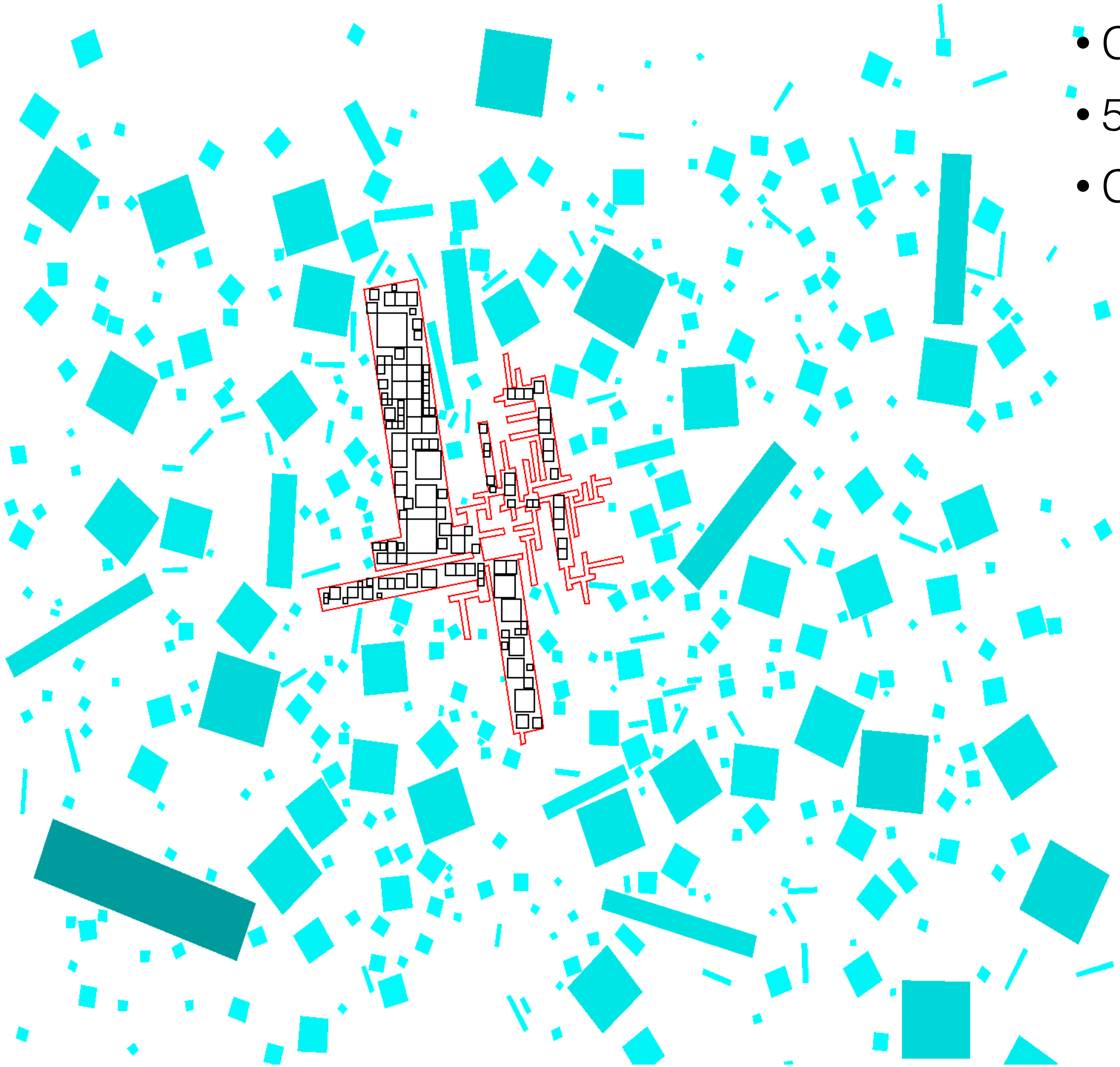

1
2
3

Checking a Solution

- What is an acceptable solution?
- How to check it using the data types we already have?



- Cave size: 9
- 39 furniture pieces
- Coverage: 40%

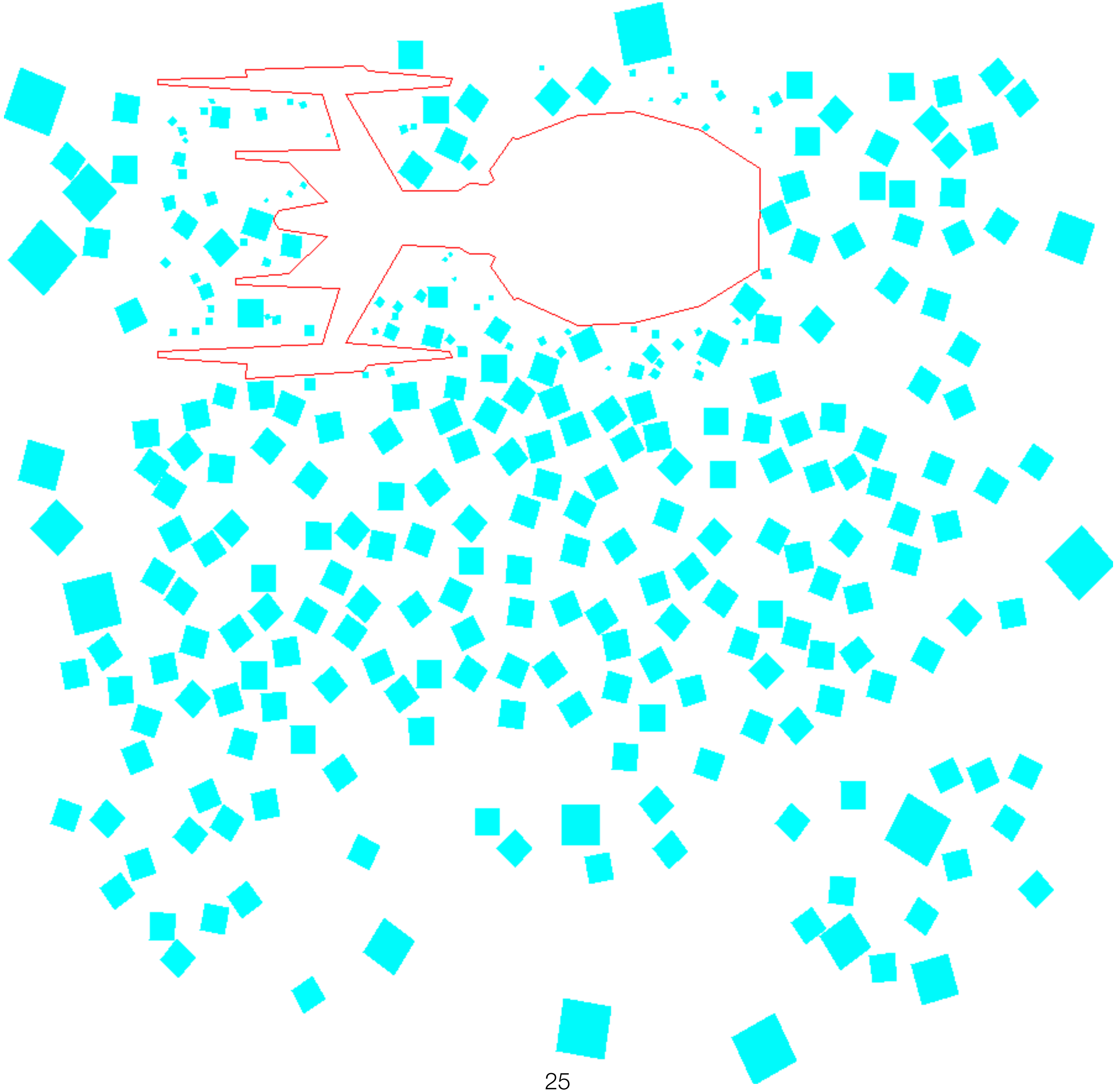


- Cave size: 180
- 500 furniture pieces
- Coverage: 46%

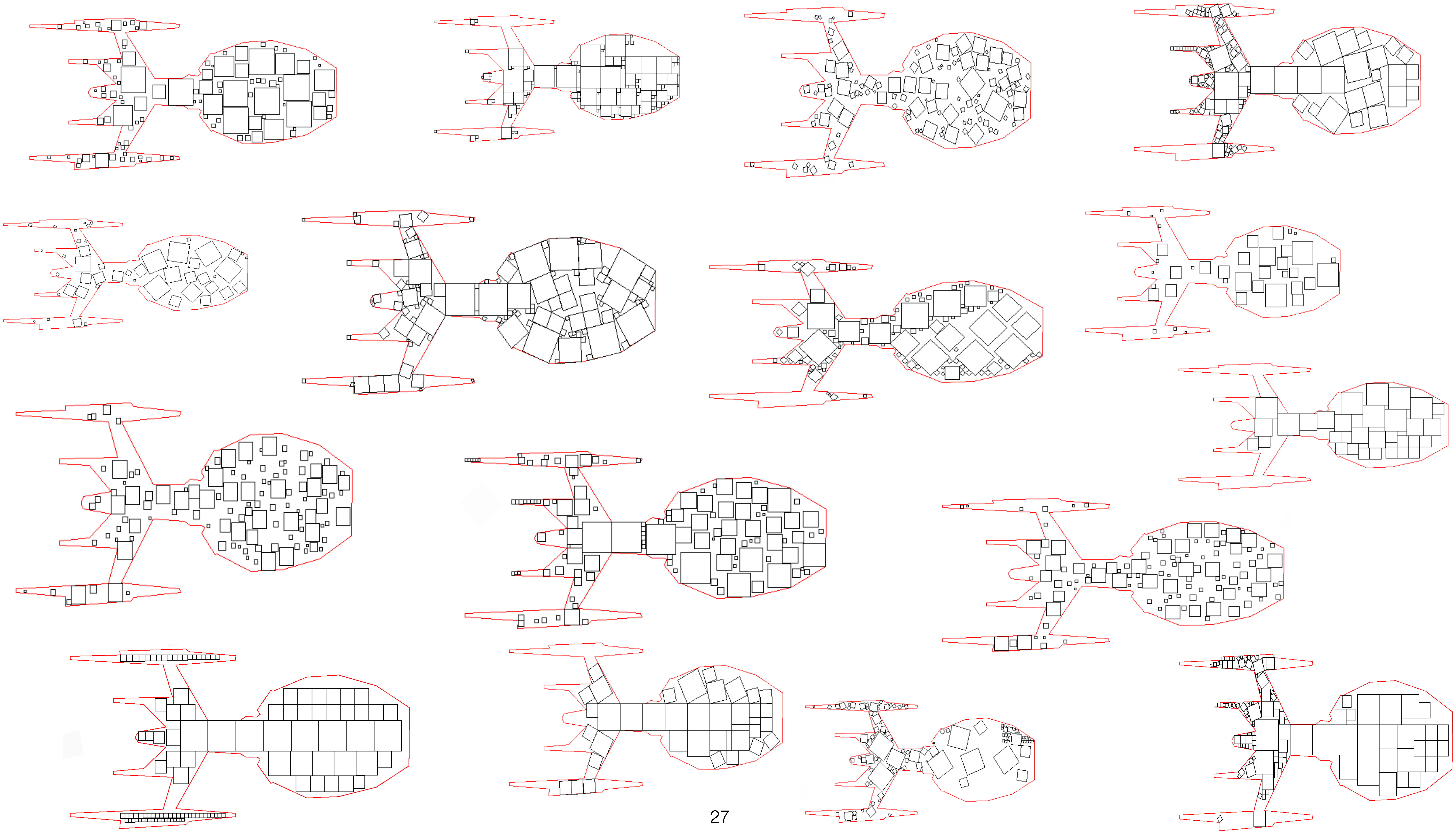
Towards an Algorithm

- What are the main steps?
- How to produce an acceptable solution?
- When should we stop?

Some solutions

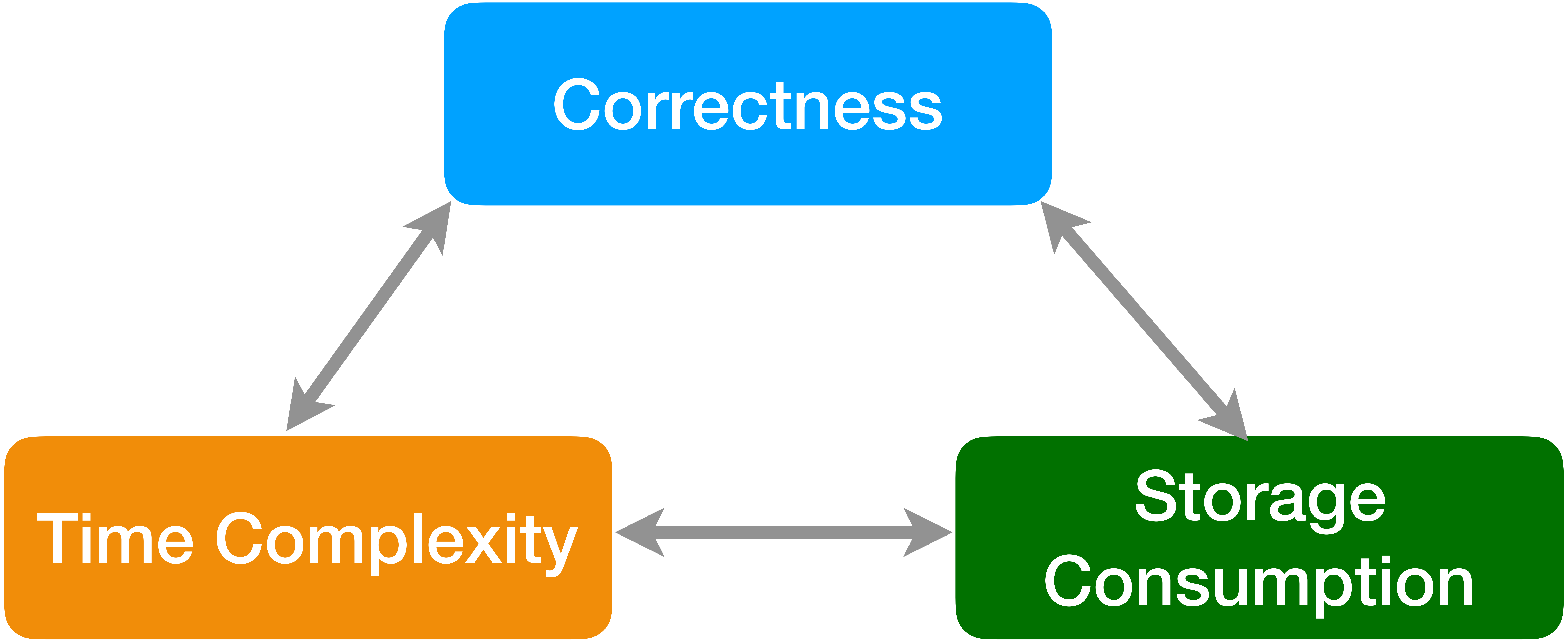



USS ENTERPRISE
NCC-1701



What else this course is about

Analysis of Algorithms


Correctness
Time Complexity
Storage Consumption

Algorithmic problems and Time Complexity

- **tractable problems** — admit solutions that run in “reasonable” time (e.g., sorting, searching, compression/decompression)
- **possibly intractable** — probably don’t have reasonable-time algorithmic solutions (e.g., SAT, graph isomorphism)
- **practically intractable** — definitely don’t have such solutions (e.g. the Towers of Hanoi)
- **non-computable** — can’t be solved algorithmically at all (e.g., the halting problem)

Aspects that we will study

- Algorithm Correctness
- Algorithm Termination
- Time complexity
  - Worst case
  - Average case
  - Best Worst case

Aspects that we will study

- Algorithm Correctness — *Does my algorithm really do what it's supposed to do?*
- Algorithm Termination — *Does my algorithm always complete its work?*
- Time complexity — *How slow is my algorithm…*
  - Worst case — *… in the worst possible case?*
  - Average case — *… in an average case?*
  - Best Worst case — *… if I do my best to optimise it?*

Example: Determinant of a matrix

Laplace expansion: $$|M| = \sum_{i=1}^{n} (-1)^{i-1} \overbrace{M_{1,i}}^{M\text{'s element at row 1, column i}} \overbrace{|M^{1,i}|}^{\text{(1,i)-minor of M}}$$

For a 3x3 matrix: $$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} =$$

$$a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} =$$

$$a_{11}(a_{22} \cdot a_{33} - a_{23} \cdot a_{32}) - a_{12}(a_{21} \cdot a_{33} - a_{23} \cdot a_{31}) + a_{13}(a_{21} \cdot a_{32} - a_{22} \cdot a_{31})$$

Example: Determinant of a matrix

Laplace expansion: $$|M| = \sum_{i=1}^{n} (-1)^{i-1} M_{1,i} |M^{1,i}|$$

(in Haskell)

```
detLaplace :: Num a => Matrix a -> a

detLaplace m
    | size m == 1 = m ! (1,1)
    | otherwise    =
       sum [ (-1)^(i-1) * m ! (1,i) * detLaplace (minorMatrix 1 i m) |
             i <- [1 .. ncols m] ]
```

(demo)

Triangular matrices

$$L = \begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \qquad U = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix}$$

Determinant of a triangular matrix
is a *product* of its diagonal elements.

For a 3x3 matrix:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{vmatrix} =$$

$$a_{11}(a_{22} \cdot a_{33} - \cancel{a_{23} \cdot 0}) - a_{12}(\cancel{0 \cdot a_{33}} - \cancel{a_{23} \cdot 0}) + a_{13}(\cancel{0 \cdot a_{32}} - \cancel{a_{22} 0}) = a_{11} \cdot a_{22} \cdot a_{31}$$

Determinants via LU-decomposition

LU-decomposition: any square matrix M, such that its top-left element is non-zero can be represented in a form

$$M = LU$$

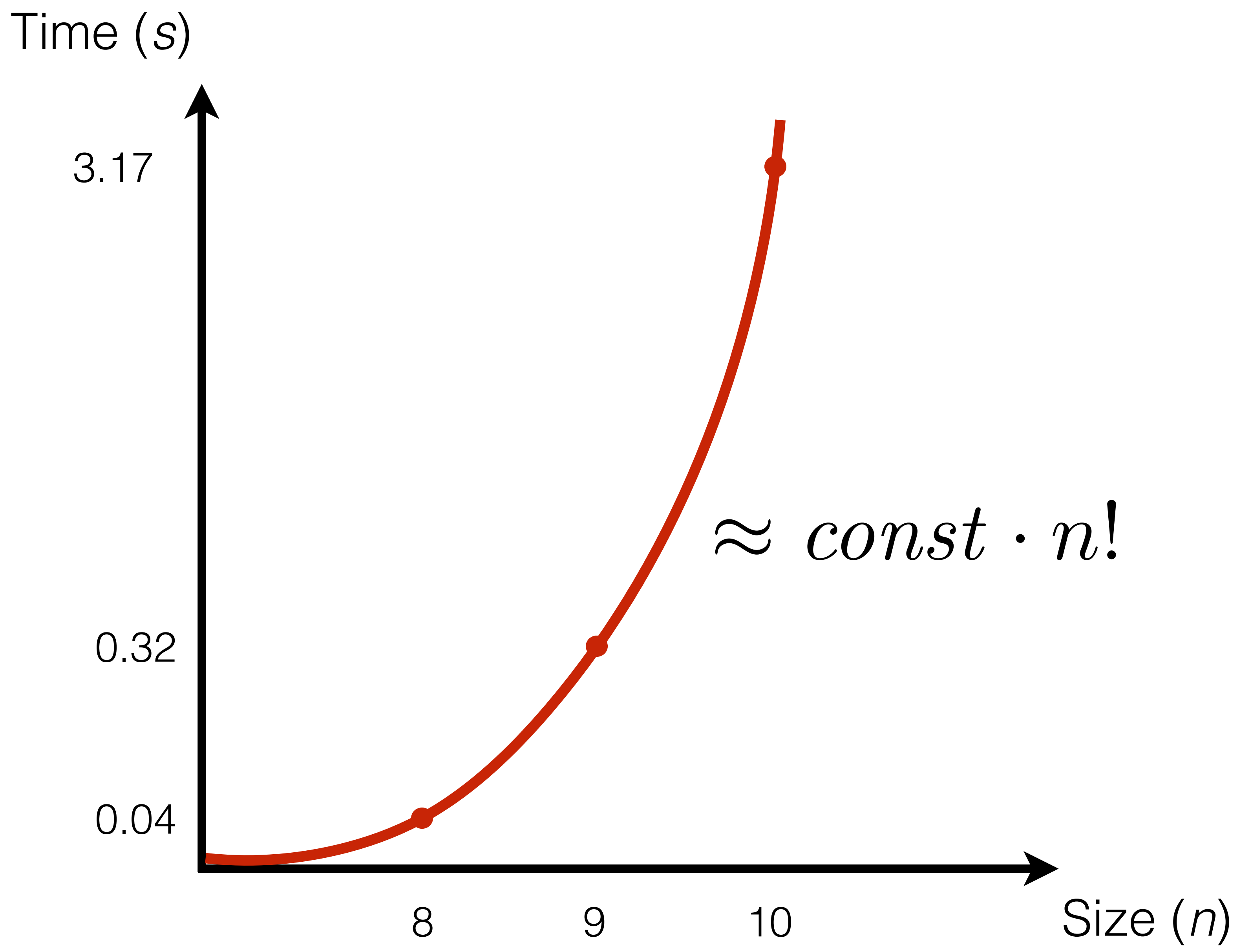
where L and U are lower- and upper-triangular matrices.

Therefore, $|M| = |L| \cdot |U|$

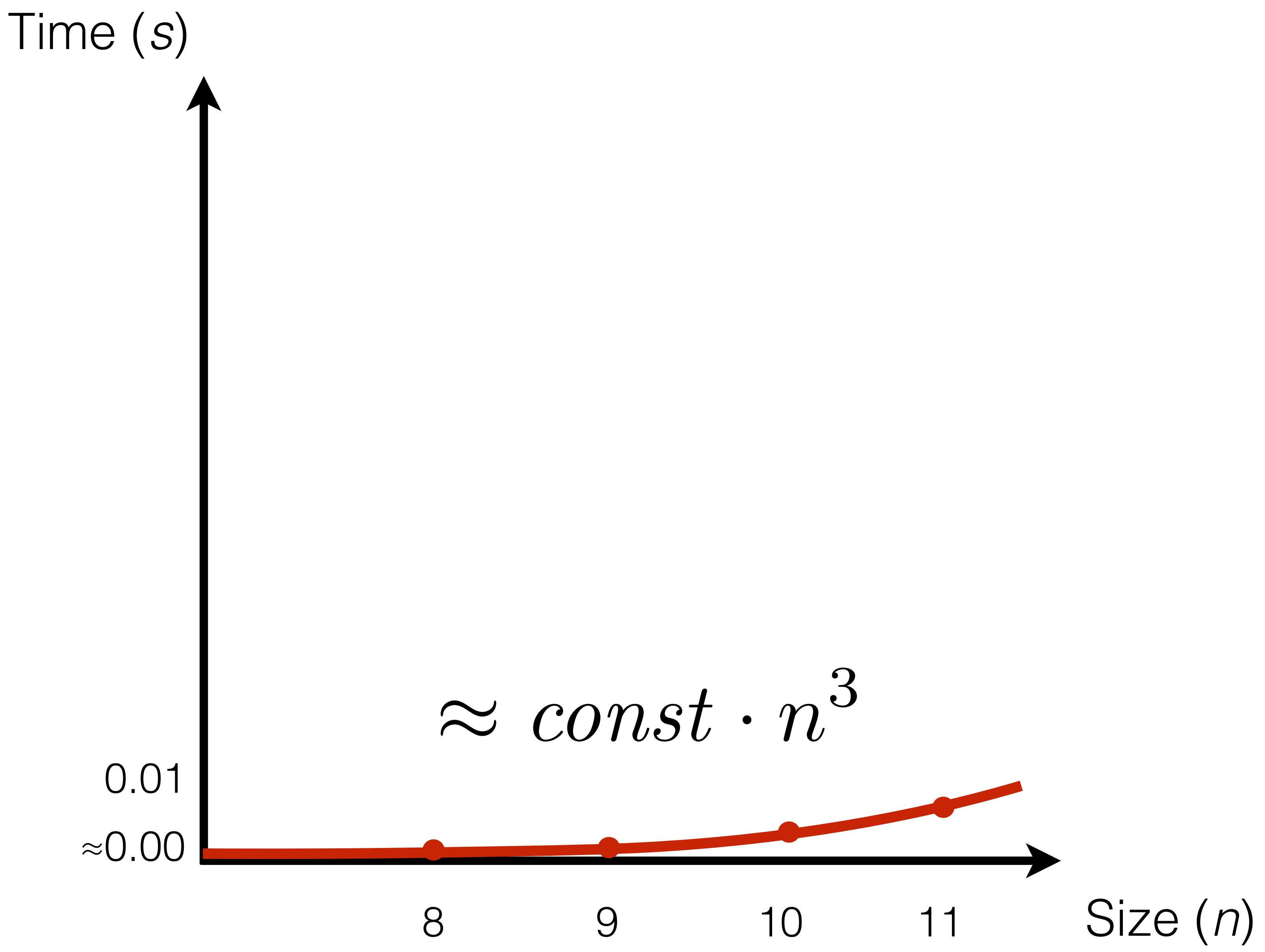
```
detLU :: Num a => Matrix a -> a
detLU m = case luDecomp m of
            (l, u) -> diagProd l * diagProd u
```

(demo)

Running time as a function of size



Determinant via
Laplace expansion



Determinant via
LU-decomposition

Time demand depends on problem size

Function	Problem size: 10	10^2	10^3	10^4
$\log_2 n$	3.3	6.6	10	13.3
n	10	100	1000	10^4
$n \log_2 n$	33	700	10^4	1.3×10^5
n^2	100	10^4	10^6	10^8
n^3	1000	10^6	10^9	10^{12}
2^n	1024	1.3×10^{30}	$> 10^{100}$	$> 10^{100}$
$n!$	3×10^6	$> 10^{100}$	$> 10^{100}$	$> 10^{100}$

http://en.wikipedia.org/wiki/Googol

“Sizes” of different problems

Problem	**Input size, *n***
sorting	number of items to be sorted
searching	size of the set to query
determinant calculation	number of rows and columns in the matrix
finding a shortest path	number of “checkpoints” to choose from

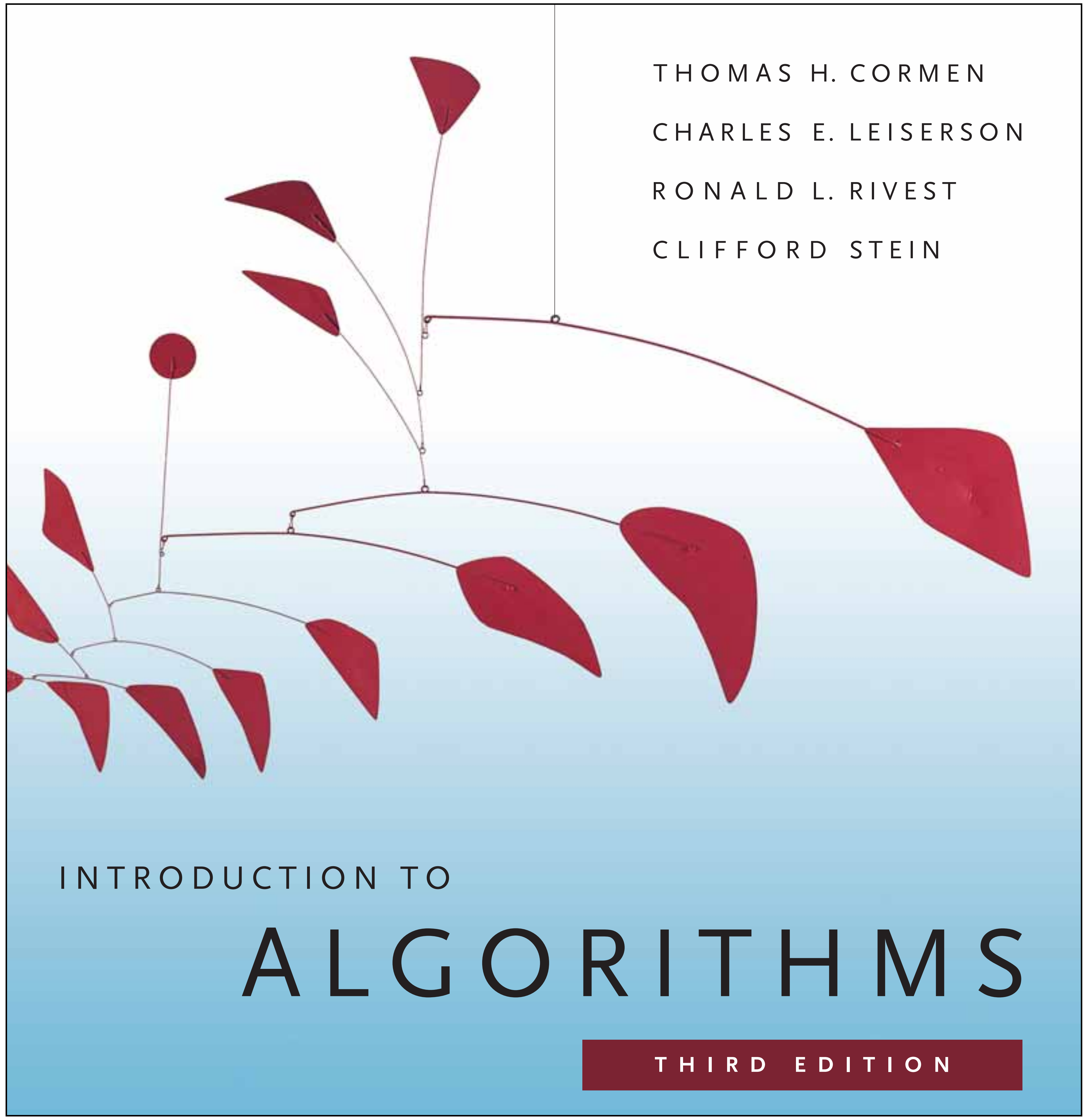
Two ways to analyse algorithms

- **Empirical** — repeatedly run algorithm with different inputs to get some idea of behaviour on different inputs
  - was our selection of inputs representative?
  - this consumes the very resource (time) we are trying to conserve!
- **Theoretical** — analysis of a “paper” version of the algorithm
  - can deal with all cases (even impractically large input instances);
  - machine-independent.

Working Tools

- OCaml
- Emacs/Aquamacs
- Toolbox
  - **Tuareg mode** for syntax highlighting and REPL
  - **Merlin mode** for type information
  - **Company mode** auto-completion and types
  - **ocp-indent** for smart indentation
  - https://github.com/ocaml/merlin/wiki/emacs-from-scratch

The Textbook



Lecture Notes (WIP)

ilyasergey.net/YSC2229

Assessment

- 30% homework exercises
- 30% mid-term project
- 35% final project
- 5% class participation

Homework

- Done in groups of 3-4 people (3 is optimal)
- Deliverables:
  - an OCaml file with the solutions
  - a PDF with explanations of what has been done
- Graded out of 20
- No extra points for recommended exercises (sorry)

Before the Break

- Tell a bit about yourself:
  - Your name (optionally)
  - What is your programming background?
  - Why are you interested in Computer Science?
  - Which computing problems did you deal with?
  - What are your expectations from this course?