Binary Search

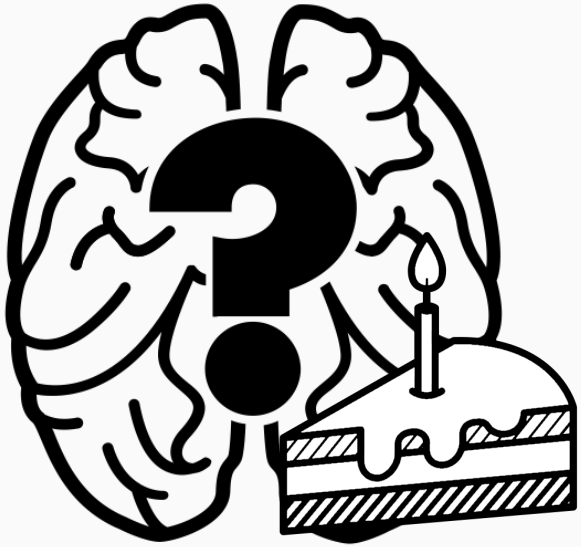
Joël Mathys

5. November 2016

Swiss Olympiad in Informatics

Introduction to Binary Search

Guessing Game



My date of birth

Guessing Game Reloaded

Numbers

$1, 2, 3, ..., 10'000$

Only 15 Questions

Guessing Game Strategies

- Go through every Element
  - $O(n)$

Guessing Game Strategies

- Go through every Element
  - $O(n)$
- Divide into Categories
  - $O(\sqrt{n})$

Guessing Game Strategies

- Go through every Element
  - $O(n)$
- Divide into Categories
  - $O(\sqrt{n})$
- Divide into two equal Halves
  - Binary Search
  - Much faster

Binary Search Theory

- Elements must have some kind of Order
  - Booleans, consecutive
    - 1110000

Binary Search Theory

- Elements must have some kind of Order
  - Booleans, consecutive
    - 1110000
  - Numbers, asc./desc.
    - 1,4,5,23,123
    - 5,2,-4,-23

Binary Search Theory

- Elements must have some kind of Order
  - Booleans, consecutive
    - 1110000
  - Numbers, asc./desc.
    - 1,4,5,23,123
    - 5,2,-4,-23
  - Curves
    - left or right

Binary Search Theory

- Elements must have some kind of Order
  - Booleans, consecutive
    - 1110000
  - Numbers, asc./desc.
    - 1,4,5,23,123
    - 5,2,-4,-23
  - Curves
    - left or right
- Next step must be clear

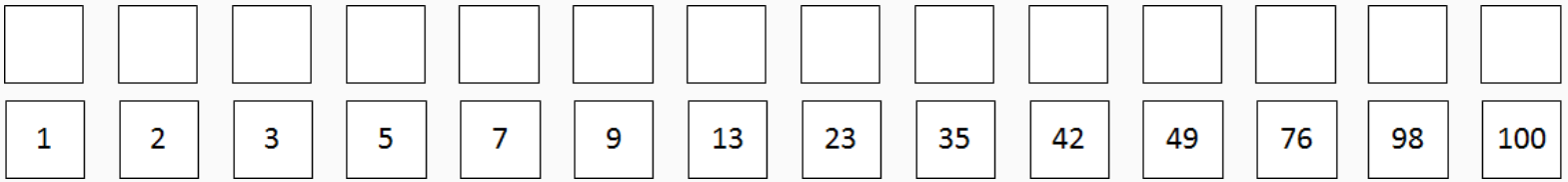
Binary Search Theory

- Elements must have some kind of Order
  - Booleans, consecutive
    - 1110000
  - Numbers, asc./desc.
    - 1,4,5,23,123
    - 5,2,-4,-23
  - Curves
    - left or right
- Next step must be clear
- End must be clear

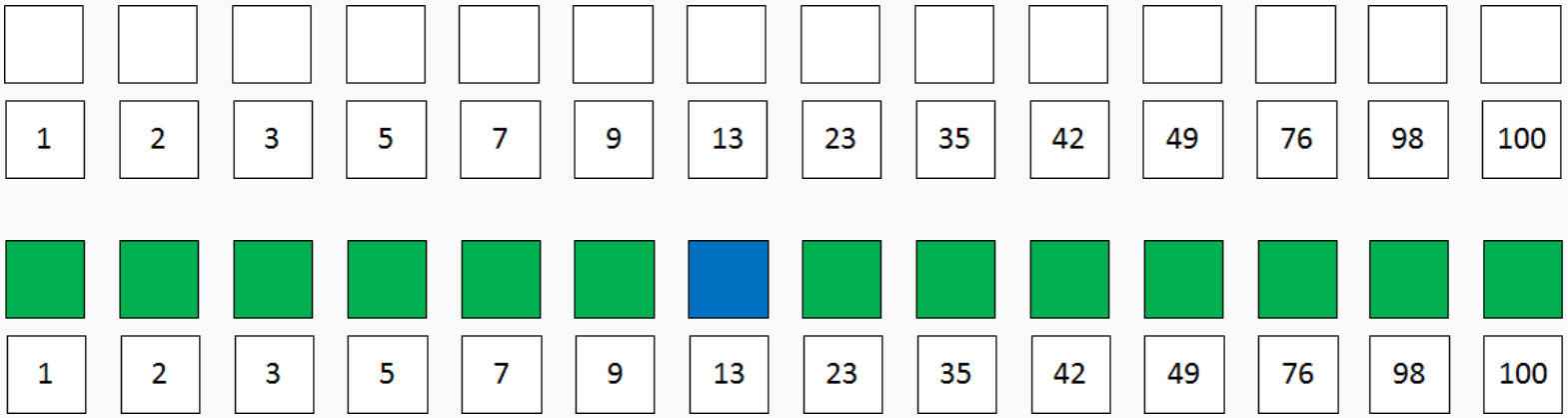
Binary Search Theory

- Elements must have some kind of Order
  - Booleans, consecutive
    - 1110000
  - Numbers, asc./desc.
    - 1,4,5,23,123
    - 5,2,-4,-23
  - Curves
    - left or right
- Next step must be clear
- End must be clear
- Runtime of $O(log(n)) \approx 10$
- almost constant time

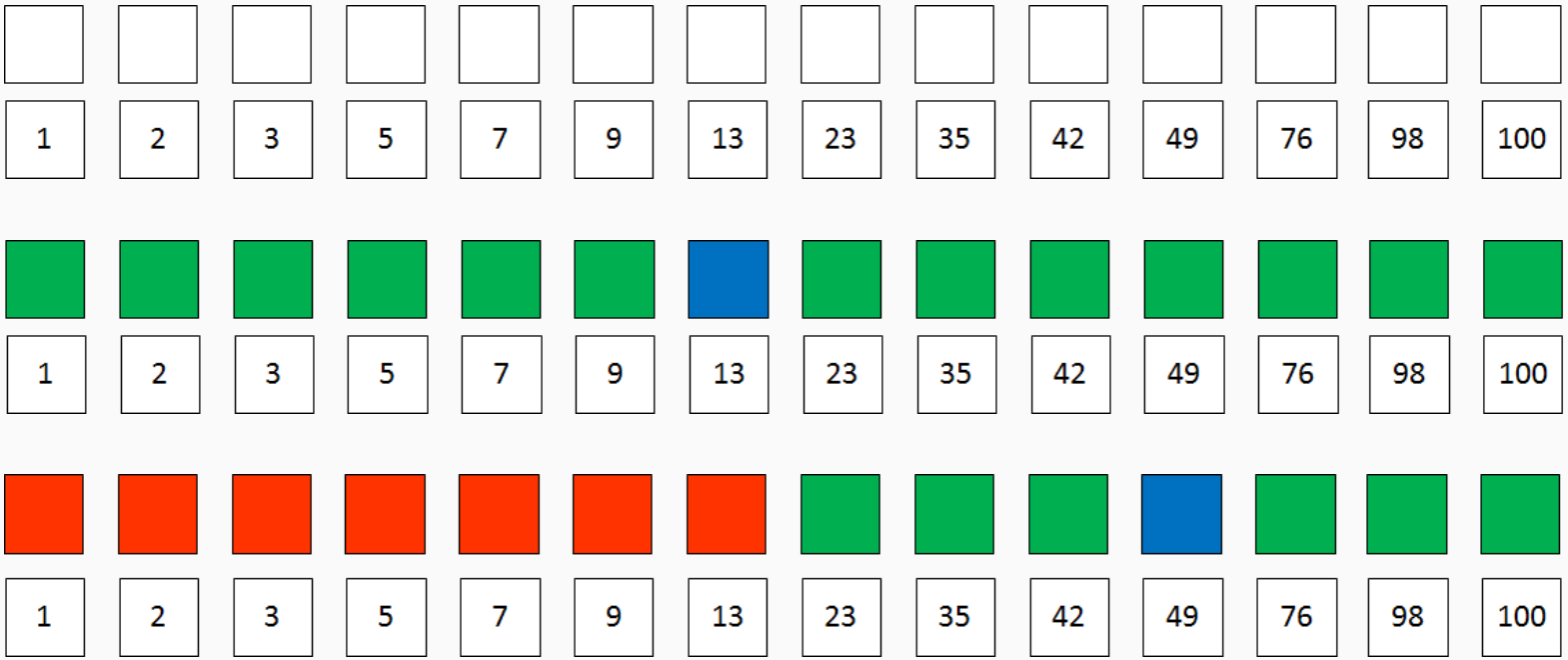
Visualisation



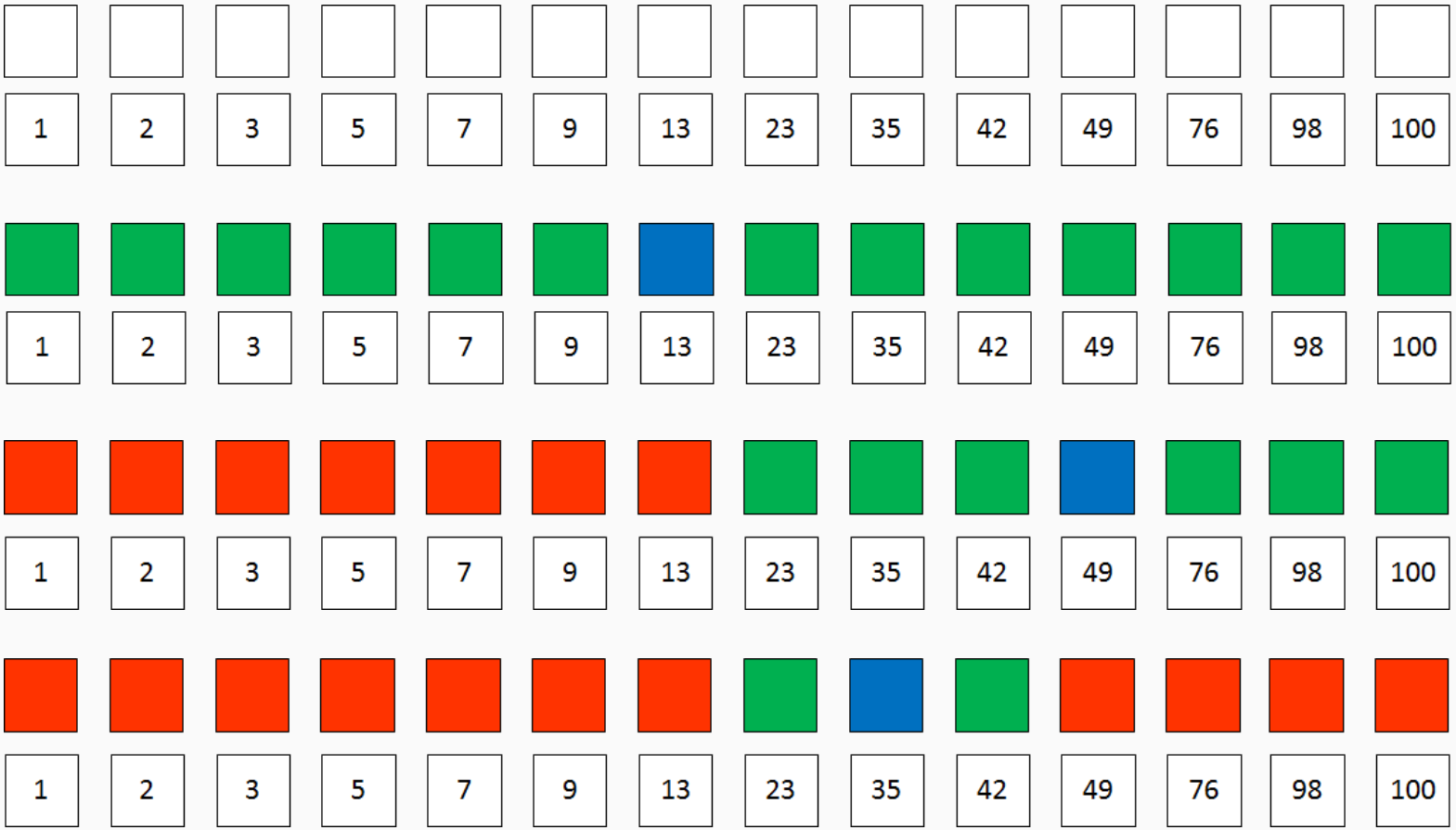
Visualisation



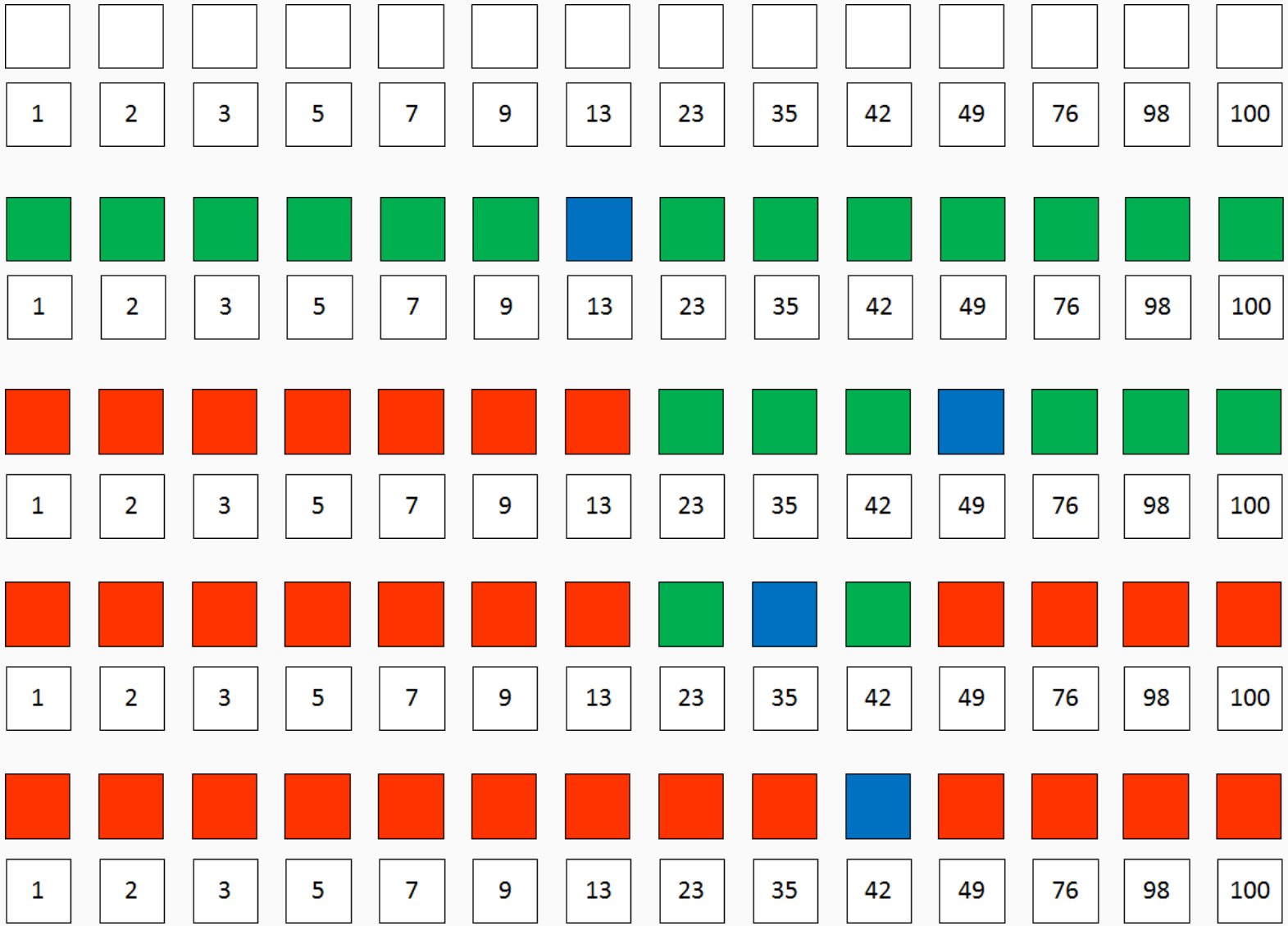
Visualisation



Visualisation



Visualisation



Implementation

```cpp
int bsearch(vector<int> &A, int e){
    int down = 0;
    int up = (int)A.size();
    int middle;
    // down is always <= e
    while(up - down > 1){
        middle = (up+down)/2;
        if(A[middle] <= e){
            down = middle;
        }else{
            up = middle;
        }
    }
    if(A[down] == e){
        return down;
    }
    return -1;
}
```

```python
def bsearch(A, e):
    down = 0
    up = len(A)
    middle = 0
    while up-down > 1:
        middle = int((down+up)/2)
        if A[middle] <= e:
            down = middle
        else:
            up = middle

    if A[down] == e:
        return down
    return -1
```

Implementation

Danger

- down ! = up
- off by one
- false rounding

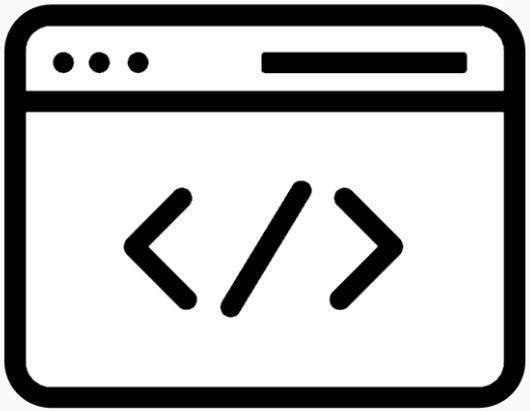
Good Practise

- size of intervall, up-down
- make sure either left or right are always true
- always round the same

```
int bsearch(vector<int> &A, int e){
    int down = 0;
    int up = (int)A.size();
    int middle;
    // down is always <= e
    while(up - down > 1){
        middle = (up+down)/2;
        if(A[middle] <= e){
            down = middle;
        }else{
            up = middle;
        }
    }
    if(A[down] == e){
        return down;
    }
    return -1;
}
```

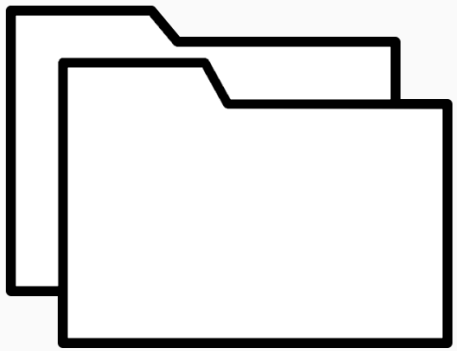
Practical Tipps

- If you only search Element or Bounds
  - Built in functions of STL Library
  - SOI-Cheatsheet
- Why implement Binary Search ?
  - normally very task dependant
  - not that hard



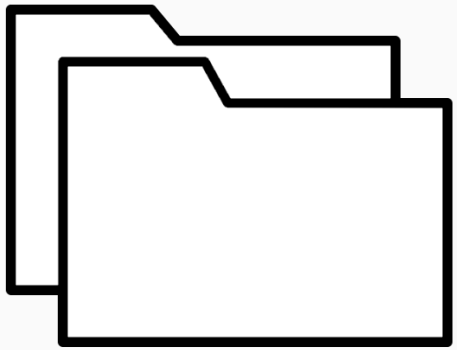
SOI-Tasks

- Treasure
  - Search treasure on a map
  - get directions
  - 1 Dimension



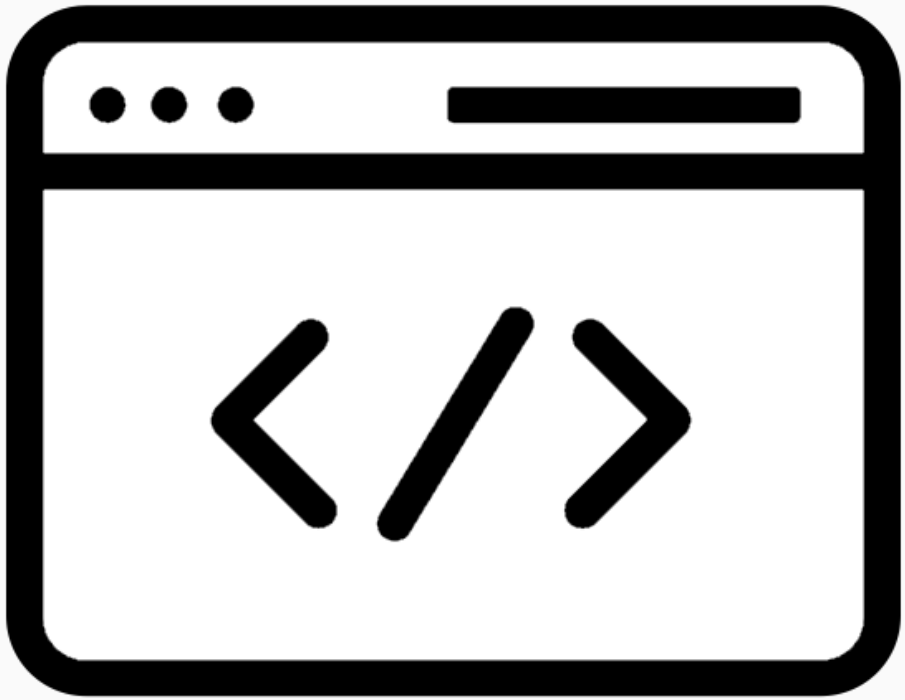
SOI-Tasks

- Treasure
  - Search treasure on a map
  - get directions
  - 1 Dimension
- Loan
  - split tasks upon co-workers
  - which tasksizes make sense?
  - how would you distribute the work?
  - how would the first worker begin?



Questions

Let's Code



Valar Codelis