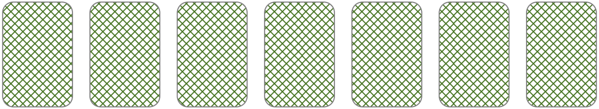
A Problem

You are give a set of 7 cards where each card contains a number between 0 and 100.

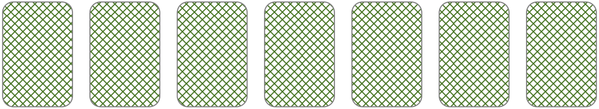


What would you do in order to find the card with number 57?

- You are allowed to ask *one* question!

A Problem

You are give a set of 7 cards where each card contains a number between 0 and 100.



Now, supposed that this card the number in the cards are sorted.

- Will that affect your strategy of finding the card with number 57?
- You are allowed to ask *one* question!

CPT108 Data Structures and Algorithms

Lecture 3

Analysis of Algorithms

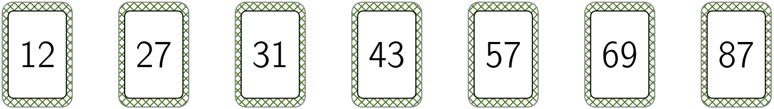
Outline

Algorithm Analysis

Algorithm formulation

- Linear search
- Binary search

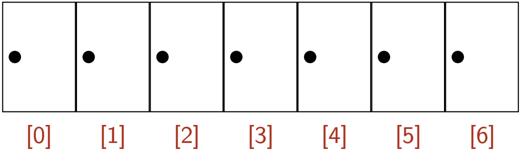
- Suppose the 7 cards before are the cards as shown below.
- Can you tell me which card has the number 57 in it?



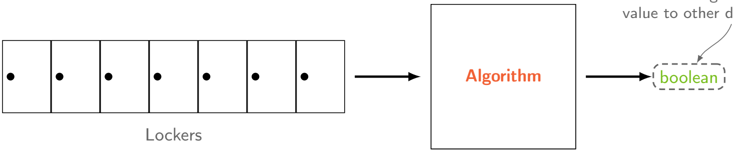
12 27 31 43 57 69 87

- It is easy! Because all people in the lecture hall has the "*bird's eye view*" of the cards!
- Do computers have such "bird's eye view" of their own memory?
  - . . . it can only look at each (memory) location one at a time!
  - . . . you have to shield your eye and only look at one number at a time, in any order, in order to find out is the number 57 actually there . . . if you pretend to be a computer!

- That is, we can view the cards as an set of lockers with the number inside, but the door is closed; and
- Each lockers has an label from 0 to 6 (i.e., from 0 to $n - 1$), just like the index of an array



- Hence, to search for a number behind these doors is just the same as to find the actual number inside the locker



Can we change the return value to other data type?

So, what should we put inside the “black box”, i.e., the algorithm?

- Depending on the situation (i.e., the *context*)
  - What is the best way to find a number, or the data we cared about?

> Or, in a more general terms, it is the best solution that we have to achieve our goal in a particular context.
>
> ***Note:*** Other considerations include *data size*, *memory requirement*, *execution platform*, *communication bandwidth*, *development cost*, etc.

Formalization of the 1st Algorithm – Linear search

Function: `linear_search`

Input : `num` : Value to be found

`lockers[n]` : An array with size `n`

Output : `true` if `num` is in the `lockers`; `false` otherwise

Pseudocode

```
For each locker from left to right
    If the number that we look for is inside the locker
        Return true
Return false
```

Notice the indentation here!

What will happen if the algorithm is now changed to this?

```
For each locker from left to right
    If the number that we look for is inside the locker
        Return true
    Return false
```

Formalization of the 1st Algorithm – Linear search

Function: `linear_search`

Input : `num` : Value to be found

`lockers[n]` : An array with size `n`

Output : `true` if `num` is in the `lockers`; `false` otherwise

Pseudocode

```
For each locker from left to right
    If the number that we look for is insider the locker
        Return true
Return false
```

In a bit technical way, this will become:

```
for $i = 0$ to $n - 1$
    if $num == lockers[i]$
        return TRUE
return FALSE
```

Formalization of the 2$^{\text{nd}}$ Algorithm – Binary search

Function: `binary_search`
Input : `num` : Value to be found
`lockers[n]` : An array with size `n`
Output : `true` if `num` is in the `lockers`; `false` otherwise

How many scenarios we need to consider when comparing two numbers, say *a* and *b*?

$$a \left\{ \begin{matrix} > \\ = \\ < \end{matrix} \right\} b$$

Formalization of the 2nd Algorithm – Binary search

Function: `binary_search`

Input : `num` : Value to be found

`lockers[n]` : An array with size `n`

Output : `true` if `num` is in the `lockers`; `false` otherwise

Pseudocode

```
If no locker left
    Return false
If num is inside middle locker
    Return true
Else if num < middle locker
    Search left half
Else if num > middle locker
    Search right half
```

```
if no locker left
    return FALSE
if $num == lockers[middle]$
    return TRUE
elseif $num < lockers[middle]$
    return SEARCH $lockers[0]$
            through $lockers[middle - 1]$
elseif $num > lockers[middle]$
    return SEARCH $lockers[middle + 1]$
            through $lockers[n - 1]$
```

The rest of the changes will leave as an exercise to you!

Some tips on Debugging an Algorithm

Similar to debug a program, you can:

- Check whether all variables has been *defined* and *initialized* correctly (if necessary)
- Check whether all the `null` value has been handled correctly (if necessary)
- Check whether all *cases* of a variable, or value returned by a function, are handled correctly
- Do the *signatures* of functions defined and used consistently?
- . . .

Selection of Algorithms

Choice of Algorithms

- Criteria:
  - Time efficiency – how fast?
  - Space efficiency – memory requirement?
  - Development cost – reuse existing / proven ones?
- May also include:
  - Communication methods and bandwidth
    - synchronous or asynchronous communications?
    - any issues due to communication overhead?
  - + **ALL** other things that need to be considered within the context of the applications

Reading

- Chapter 2, Cormen (2022)