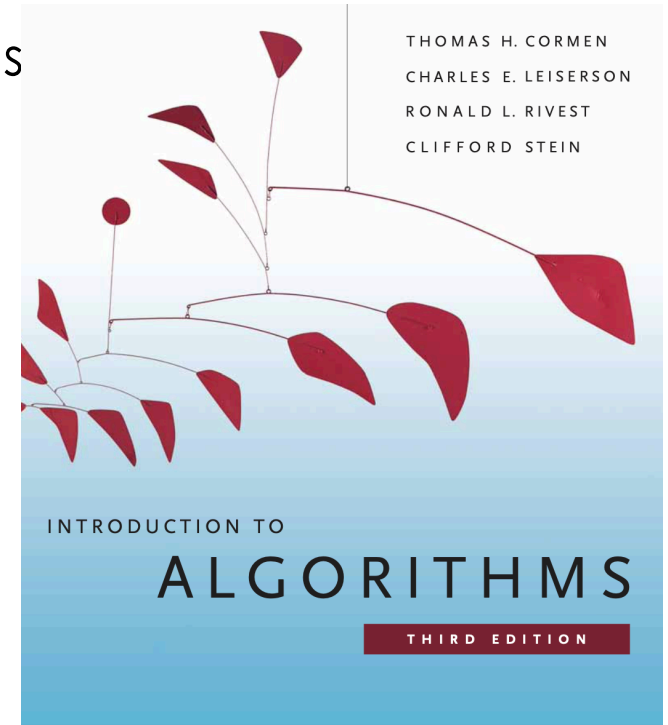
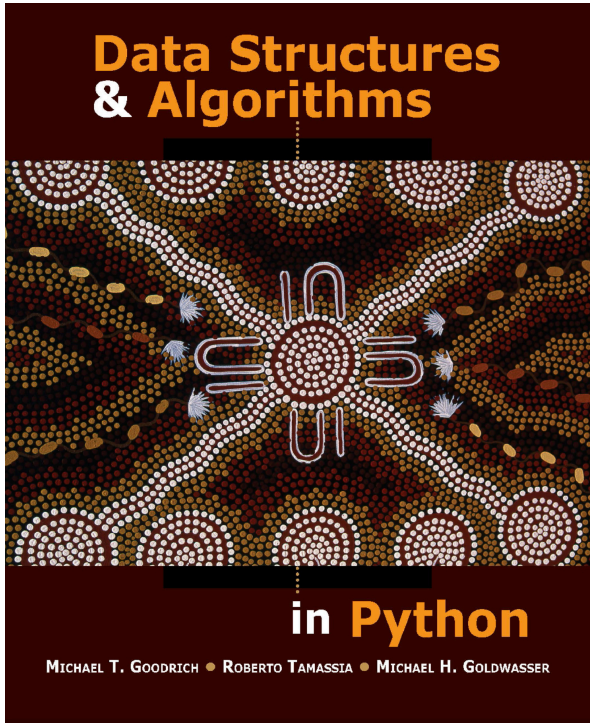
DATA STRUCTURES AND ALGORITHMS

School of Artificial Intelligence

THIS COURSE

- This course: Data Structure and Algorithms
- Text books
  - Data Structures & Algorithms in Python
  - Introduction to Algorithms
  - Documents containing core materials
- Use of Language
  - English (75%)
  - Mandarin (25%)
- Types of sessions
  - Lecture (60%)
  - Review (20%)
  - Practical (20%)





USE OF ENGLISH – WHY?

If you can't write it down in English, you can't code it - Peter Halpern

什么是“动态规划”？

Do you mean "Dynamic Programming"?
A term coined by Richard Bellman to hide the fact he was researching in mathematics?

SOME AGREEMENTS (1)

- Difficulty: moderate
  - We want you to understand how data structures/algorithms work
  - We DO NOT want you to get out of course with a high score but fail your job interviews because you cannot write the pseudo code for some sorting algorithm
- Questions:
  - In-session: encouraged
    - English – preferred
    - Chinese – also good
  - Out-session: avoid Q&A twice, avoid not knowing what others have asked
    - Online forum: preferred
    - Email: moderate choice
    - After-session: maybe a poor choice

SOME AGREEMENTS (2)

- Programming language: no preference, really
  - Python
    - Interpreted language
    - Weakly typed
    - Weak Object-Oriented
  - C/C++: strongly recommended by my colleague
    - Compiled language
    - Strong typed (maybe?)
    - Pure procedure/Object-Oriented
  - Java
    - Compiled language
    - Strong typed
    - Object-Oriented

SOME AGREEMENTS (3)

- Integrated Development Environment (IDE)
  - PyCharm
  - Visual Studio
  - Eclipse
- Practice
  - Makes perfection
  - Will try to give out some quiz every week
  - Completely voluntary
- Use of Language
  - Speed of speech
    - Let me know how quick/slow should I go in English
  - English
    - American spelling vs. UK spelling
    - Some terms only makes sense in English
    - Some are poorly translated
    - Prep you for your future careers/studies
  - Chinese
    - OK, but try to use English

SOME AGREEMENTS (4)

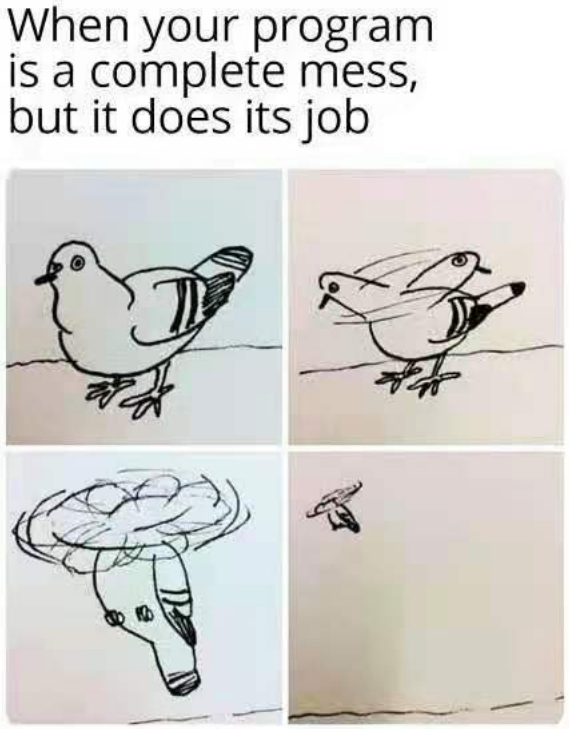
- Terminologies
  - English -> Chinese mapping
  - Online forum
  - You will be asked to write them on the blackboard in lecturers

ABOUT ME

- Me
  - Ran Wei, Or “Will” if you want
  - BSc, MSc, PhD in the UK
  - Intel, Rolls Royce, Siemens, Apple, BAE Systems
  - Research interests:
    - Model-Based System Engineering
    - High-Integrity Systems Engineering
    - Electromagnetism Metrology
- Guests
  - Dr. Athanasios Zolotas: prefer people to call him ”Thanos”
    - University of York
  - Dr. Haitao Mei
    - Apple research, Cambridge
    - Soon to be in Google, Islington
- Contact
  - ranwei@dlut.edu.cn
  - B1605

ABOUT THE COURSE

- Introduction to data structures and algorithms
  - Advanced topics are not covered here
- Design, analysis and implementation
  - Get to know your requirements
  - Design your data structure and algorithms based on requirements
  - Implement – code your algorithm with your choice or language
- From the view point of Object-Oriented design principle
  - Object-Oriented thinking
  - We will cover Model-Driven Engineering briefly
- Abstract Data Types (ADTs)
  - Data structures are not just .py files!
- Pseudo Code
  - Algorithms are not just .py files!



QUESTION

DO Algorithms only exist in computers?

DESIRED OUTCOME OF THE COURSE

- Knowledge of the most common abstractions for data structures
- Understanding of algorithmic strategies for efficient implementation of common data structures
  - We want algorithms that can finish before we die
  - Or even better in milliseconds
- Being able to analyse algorithmic performance, and recognise common trade-offs between competing strategies
- Being able to recognise and make use of, existing data structures and algorithms found in modern programming language libraries
  - Lists, trees, heaps, etc.
- Being able to apply what you have learnt to solve complex problems, in your future studies/careers
  - My interview in Intel asked me to write a quick-sort algorithm with a pen and a paper

COVERED TOPICS

- Python and Object-Oriented Programming
- Algorithmic thinking and Algorithm Analysis
- Models of Computation and Recursion
- Array-Based Sequences
  - Hashing with chaining
  - Table doubling, Karp Rabin
- Stacks, Queues, and Deques
- Trees
- Sorting Algorithms
- Graph Algorithms
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)
  - Single-Source Shortest Paths Problem
  - Dijskstra
  - Bellman-Ford
- Dynamic Programming
- Some topics in Algorithms Research

ASSESSMENTS

- To be finalized
- Exams
  - Mid-term
  - End-term
- Assignments
  - 4 tiny ones
- Assessment weightings
  - 50% on exams
  - 50% on assignments

PYTHON

- Named after the TV Series “Monty Python’s Flying Circus”
- Compiled language
  - Source code -> compiler -> assembly code -> execution
  - Fast
  - Long turn-around time
- Interpreted language
  - Commands (one at a time) -> machine code -> execution
  - Slow
  - Quick interactive working style
- Strong typed vs. weak typed
  - “duck typing”

PYTHON

- Objects in Python
  - Object-Oriented language
  - Class vs. Objects?
  - Built-in classes

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

PYTHON

- List (array-based, zero-indexed)
  - Most used container type
  - Mutable
  - Central to our study of data structures
  - Delimiters: [and]
  - Prime = [2, 3, 5, 7, 11, 13, 17, 19]
  - What's the underlying mechanism for a List?
  - How does a list grow?
- Tuple
  - Immutable version of a sequence
  - Delimiters: (and)
    - (17,)

PYTHON

- The str class
  - String
  - Inmutable
  - Delimiters: ‘ and ‘, or “ and “
  - Escape character: \xx
  - Document distancing
    - How a search engine finds a document
- The set and frozenset class
  - Collection of unique elements
  - Mutable vs. inmutable
  - Delimiters: { and }
  - Based on hash table
  - Unordered
  - Only immutable types can be added to a Python set

PYTHON

- The dict class
  - Dictionary, mapping: key-value pairs
  - Delimiters: { and }
    - {‘gb’: ‘British’, ‘de’: ‘German’ }
  - Hashing

PYTHON

- The dict class
  - Dictionary, mapping: key-value pairs
  - Delimiters: { and }
    - {‘gb’: ‘British’, ‘de’: ‘German’ }
  - Hashing

- Expressions
  - Logical operators: no, and or
    - Short-circuit
  - Equality operators: is, is not, ==, !=
    - a is b evaluates to True: a and b are aliases for the same object
    - a == b evaluates to True: a and b refer to the same object, or a and b have same values

PYTHON

- Expressions
  - Comparison operators: <, <=, >, >=
    - Numeric types
  - Arithmetic operators: +, -, *, /, //, %
    - True division: 27/4 = 6.75 (not supported in C, C++ and Java)
    - Integer division: 27//4 = 6 (mathematical floor of the quotient)
      - -27//4 = -7
    - Modulo: 27/4 = 3
      - -27%4 = -1
  - Bitwise operators:
    - ~ bitwise complement
    - & bitwise and
    - | bitwise or
    - ^ bitwise exclusive-or
    - << and >> shift left and shift right

PYTHON

- Expressions
  - Sequence operators
    - S[j]
    - S[start:stop]
    - S[start:stop: step]
    - S+t
    - K*s
    - Val in s
    - Val not in s

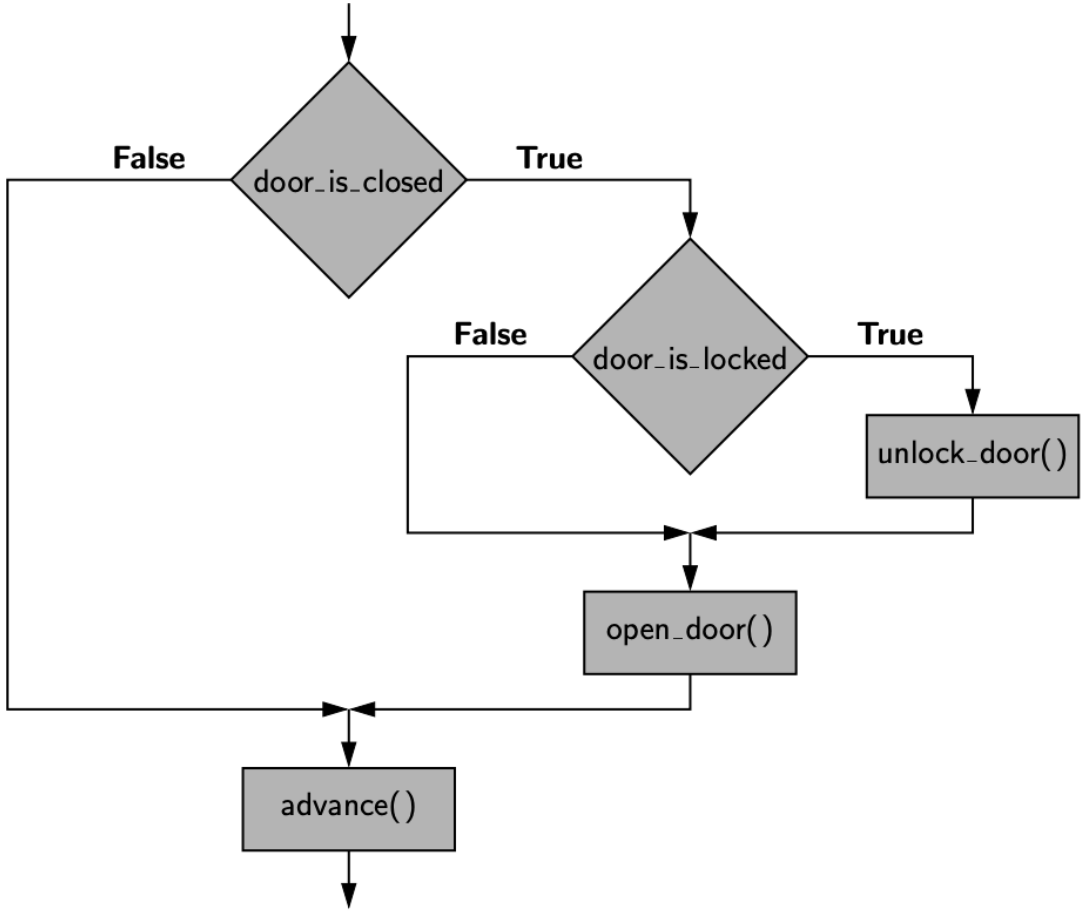
PYTHON

- Expressions
  - Sequence operators:
    - S[j]
    - S[start:stop]
    - S[start:stop: step]
    - S+t
    - K*s
    - Val in s
    - Val not in s
- What about the time it takes to do above? For a S with size N?
  - Models of Computation
  - Random Access Memory
  - Asymptotic analysis

PYTHON

- Control Flow
  - Conditionals

```
if first_condition:
    first_body
elif second_condition:
    second_body
elif third_condition:
    third_body
else:
    fourth_body
```



PYTHON

- Control Flow
  - While loops, for loops, break and continue

```
while condition:
    body
```

```
for element in iterable:
    body
```

```
found = False
for item in data:
    if item == target:
        found = True
        break
```

PYTHON

- Functions
  - Signature `def count(data, target):`
  - Activation record
    - Each time a function is called
    - Namespace: manage identifiers that have local scope
    - Local scope vs. global scope
      - Execution frames
      - Often implemented as stacks
  - Return statement
  - Information passing
    - Formal parameters: expected parameters in the function signature
    - Actual parameters: objects sent by the caller to the function

PYTHON

- Functions
  - Information passing
    - Formal parameters: expected parameters in the function signature
    - Actual parameters: objects sent by the caller to the function

```
def count(data, target):
  n = 0
  for item in data:
    if item == target:
      n += 1
  return n
```

```
prizes = count(grades, 'A')
```

PYTHON

- Exception handling
  - Important for debugging, especially in concurrent programming
    - Asynchronised transfer of control
  - Errors (Exception in C++, Java)
    - Raised (thrown) by the code that encounters an unexpected circumstance
    - Caught by a surrounding context that "handles" the exception
  - Exception types
    - Exception
    - AttributeError
    - EOFError
    - IOError
    - IndexError
    - KeyError
    - etc.

```
def sqrt(x):
  if not isinstance(x, (int, float)):
    raise TypeError('x must be numeric')
  elif x < 0:
    raise ValueError('x cannot be negative')
  # do the real work here...
```

PYTHON

- Iterators
  - An object that manages an iteration through a series of values
  - next(i): produces a subsequent element from the underlying series
    - StopIteration exception when there are no further elements
  - Iterable: an object that produces an iterator via iter(obj)

  data = [1, 2, 4, 8]
  next(data)?
  i = iter(data)
  - Implicit iterable
    - range(1000000) does not return a list of numbers
    - Returns a range object that is iterable
  - Lazy evaluation: no need to allocate memory for 1000000 elements
    - Keys(), values(), items() in the dictionary class

PYTHON

- Generators
  - Similar syntax with iterators
  - ''yield' statement
  - Determining all factors of a positive integer
    - 100: 1, 2, 4, 5, 10, 20, 25, 50, 100

```
def factors(n):
  results = [ ]
  for k in range(1,n+1):
    if n % k == 0:
      results.append(k)
  return results
```

```
def factors(n):
  for k in range(1,n+1):
    if n % k == 0:
      yield k
```

PYTHON

- Scopes and Namespaces
  - Name resolution
  - Global scope vs. local scope
  - More on how these are done at the end of the course
- Modules and the import statement
  - Object Oriented programming
  - Next lecture

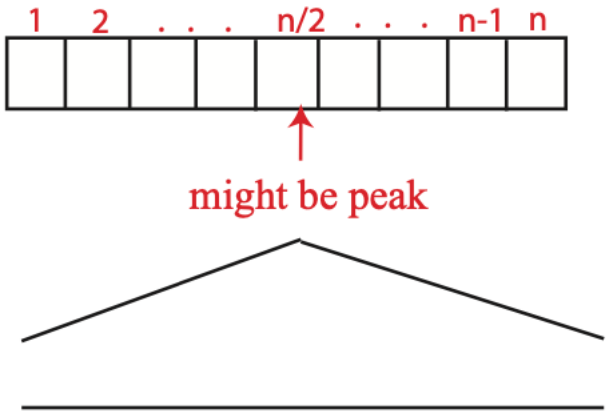
ENOUGH PYTHON! A TEASER

- Algorithmic thinking: peak finder
  - Array of integers
    - Position 2 is a peak if and only if (iff) b>=a and b>=c
    - Position 9 is a peak if i>=h
  - Problem: find a peak, if it exists
    - Does it always exist?
  - Means of measure the efficiency of an algorithm: CPU time
    - 1 unit of time to access an element
    - 1 unit of time to compare two elements
    - More on asymptotic complexity later

1	2	3	4	5	6	7	8	9
a	b	c	d	e	f	g	h	i

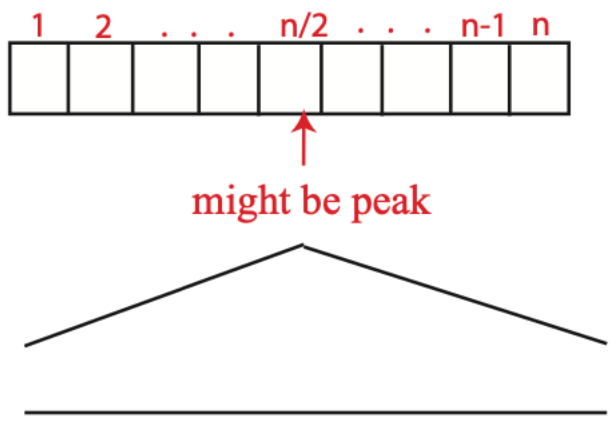
PEAK FINDING

- Straightforward approach: start from the left
- Worst case for a array of size n?



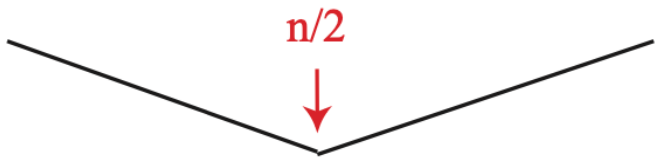
PEAK FINDING

- Straightforward approach: start from the left
- Worst case for a array of size n?
- Complexity?
  - c*n, c is some constant factor
  - We will call it O(n) for now
  - More on this later



PEAK FINDING

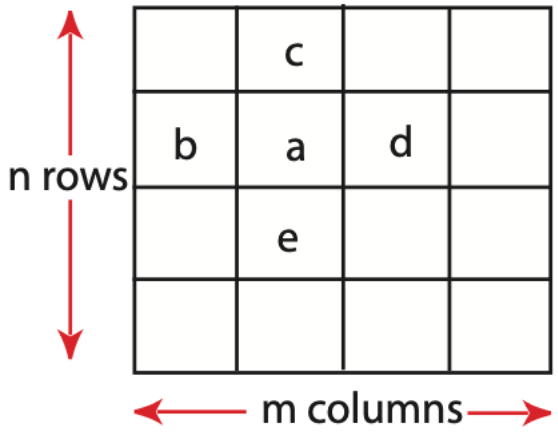
- What if we start in the middle?
  - We would look at n/2 elements
  - Would we have to ever look at more than n/2 elements?
  - If a[n/2] < a[n/2-1] then only look at left half 1..n/2-1 to look for a peak
  - Else if a[n/2] < a[n/2+1] then only look at right half n/2 +1 .. N to look for a peak
  - Else n/2 position is a peak: why?
    - a[n/2] >= a[n/2-1]
    - a[n/2] >= a[n/2+1]
  - Complexity?
    - c*log(n), for some constant c
    - We call it O(log n) for now
- If n = 1,000,000
  - O(n) algorithm finishes in 13 seconds in Python
  - O(log n) inishes in 0.001 sec



PEAK FINDING

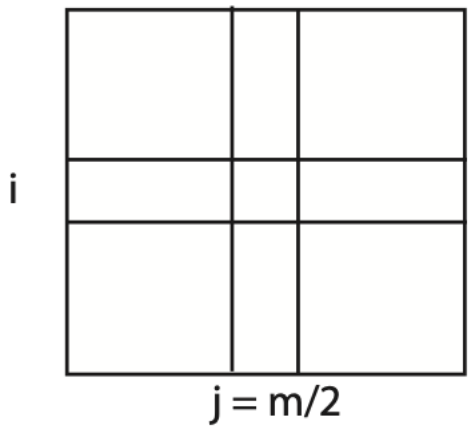
- Two-dimensional version
  - A is a 2D peak iff a>=b, a>=d, a>=c and a>=e
  - 20 is a peak
  - Greedy ascent algorithm
    - Going to the direction where the element is bigger
    - Complexity: $O(n^2)$ if m = n

14	13	12	
15	9	11	17
16	17	19	20



PEAK FINDING

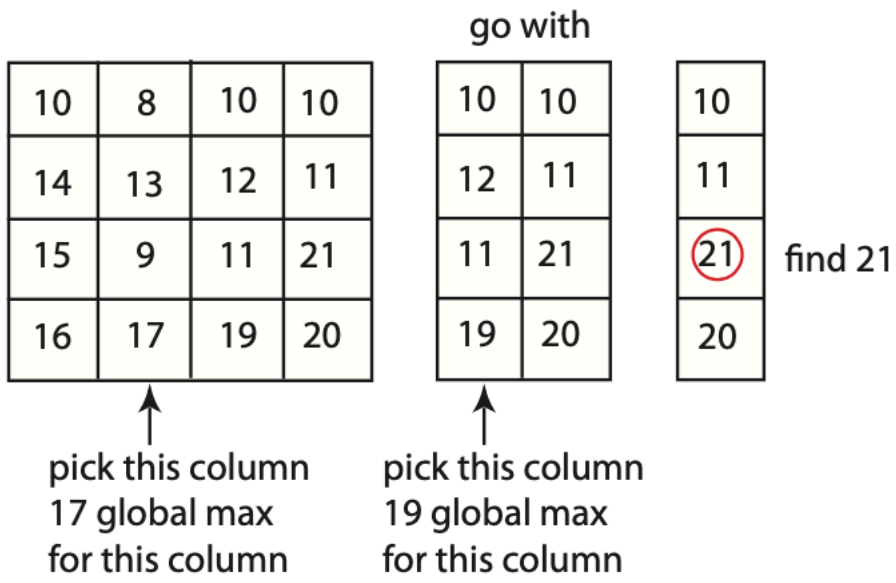
- 2D peak finding
  - Attempt #1: extend 1D divide and conquer to 2D
    - Pick middle column j = m/2
    - Find a 1D-peak at i, j
    - Use (i,j) as a start point on row I to find 1D-peak on row i
  - Attempt # 1 fails
    - Problem?
    - 2D peak may not exist on row i



		10	
14	13	12	
15	9	11	
16	17	19	20

PEAK FINDING

- Attempt #2
  - Pick middle column j = m/2
  - Find global maximum on column j at (i, j)
  - Compare (i, j-1), (i, j), (i, j+1)
  - Pick left columns if (i, j-1) > (i, j)
  - Similarly for the right
  - (i, j) is a 2D peak if neither condition holds -> why?
  - Solve the new problem with half the number of columns
  - When you have a single column
    - find global maximum and you are done



PEAK FINDING

- Attempt #2
  - Complexity?
  - If T(n, m) denotes work required to solve problmes

  With n rows and m columns

  T(n,m) = T(n, m/2) + O(n) – global maximum on m

  T(n,m) = O(n) + O(n) … log m times

  = O(n log m)

  = O(n log n) if m = n

go with

10	8	10	10
14	13	12	11
15	9	11	21
16	17	19	20

pick this column
17 global max
for this column

10	10
12	11
11	21
19	20

pick this column
19 global max
for this column

10
11
21
20

find 21

PREVIOUSLY WE ASKED:

DO Algorithms only exist in computers?

No, it is a way of thinking

Computers help us to solve the same problem

But with a very much larger input

Millions of element at my time

Trillions of element at yours

SMALL QUIZ FOR THIS WEEK:

- Problem settings:
  - 2 players
  - 1 table
  - Infinite amount of coins
- Rule:
  - Each turn: one player places a coin on the table
    - Anywhere
  - The player that places the last coin (i.e. table if full of coins) wins
  - Coins cannot overlay with each other
- If you are to win, would you choose to:
  - Take turns first
  - Take turns second
- What's the winning strategy?

THANKS

See you in the next session!