Thinking Efficiently

J.P. Pretti
jpretti@uwaterloo.ca

CENTRE for EDUCATION in MATHEMATICS and COMPUTING
University of Waterloo

August 13, 2019

Thinking Efficiently

Claim

Modern languages provide useful slick tools. However, if you are concerned about how long it takes your programs to run, then you should use these features with care and knowledge.

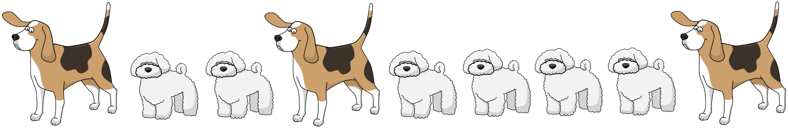
This Afternoon

1. warm-up problem(s)
2. case study: nine different algorithms solving the same problem
   - illustrated with Python 3.7
3. cool-down problem(s)

Problem A

Swapping Dogs

Two types of dogs are standing as shown below.



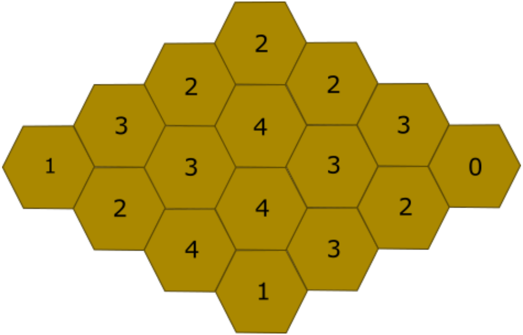
A *swap* occurs when two dogs that are beside each other exchange positions. After some swaps, the three large dogs end up in three consecutive positions.

What is the fewest number of swaps that could have occurred?

Problem B

Beehive

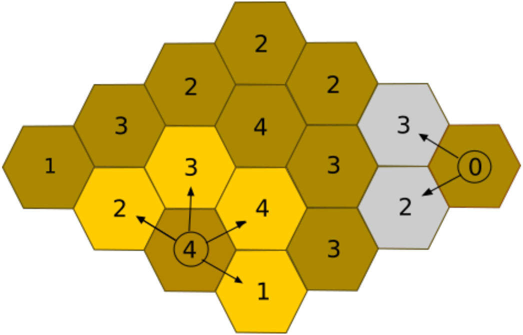
For each hexagon below, a bear records how many *other* hexagons touching this hexagon contain honey. So this number could be 0, 1, 2, 3, 4, 5 or 6. How many hexagons contain honey?



Problem B

Beehive Solution

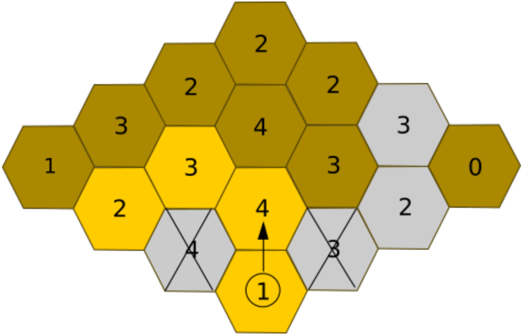
For each hexagon below, a bear records how many *other* hexagons touching this hexagon contain honey. So this number could be 0, 1, 2, 3, 4, 5 or 6. How many hexagons contain honey?



Problem B

Beehive Solution Continued

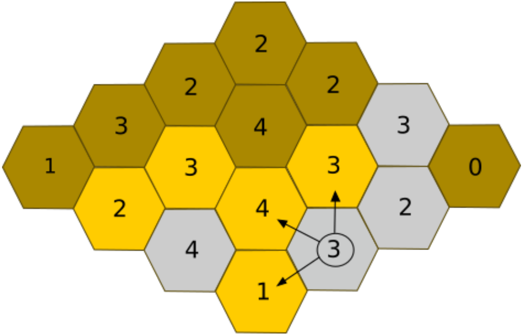
For each hexagon below, a bear records how many *other* hexagons touching this hexagon contain honey. So this number could be 0, 1, 2, 3, 4, 5 or 6. How many hexagons contain honey?



Problem B

Beehive Solution Continued

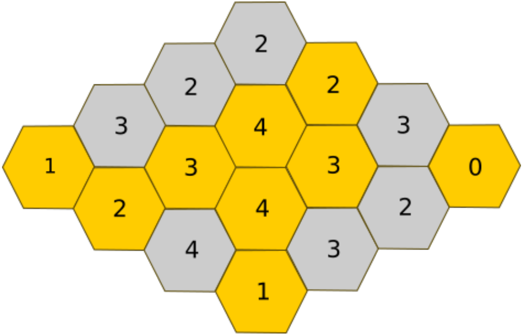
For each hexagon below, a bear records how many *other* hexagons touching this hexagon contain honey. So this number could be 0, 1, 2, 3, 4, 5 or 6. How many hexagons contain honey?



Problem B

Beehive Solution Continued

For each hexagon below, a bear records how many *other* hexagons touching this hexagon contain honey. So this number could be 0, 1, 2, 3, 4, 5 or 6. How many hexagons contain honey?



Our Case Study

The problem

Find the maximum among a collection of numbers.

Notes

- unbounded integers
- stored in a non-empty Python list
- duplicates allowed
- do not use the built-in function `max` on a list

Possible solutions

What different algorithms can you come up with?

Max So Far – Standard

Idea

Scan the list keeping track of the largest integer seen so far.

- `for v in L`

Max So Far – Standard

Idea

Scan the list keeping track of the largest integer seen so far.

- `for v in L`

Notes

- arguably natural
- simple and straightforward

Efficiency

Many ways to compare algorithms

memory usage, maintenance, time to program, runtime, . . .

Efficiency

Many ways to compare algorithms

memory usage, maintenance, time to program, runtime, . . .

Our focus

- only care about how time increases as input size increases
  - why?

Efficiency

Many ways to compare algorithms

memory usage, maintenance, time to program, runtime, . . .

Our focus

- only care about how time increases as input size increases
    - why?
- time it takes the algorithm to run on the worst possible input
- measured as a function of the length of the list

Efficiency

Many ways to compare algorithms

memory usage, maintenance, time to program, runtime, . . .

Our focus

- only care about how time increases as input size increases
  - why?
- time it takes the algorithm to run on the worst possible input
- measured as a function of the length of the list

Our model

- each basic operation takes constant time
- ignore constants and low-order terms using big-O notation
  - e.g $O(1), O(n), O(\log n), O(n^2), \ldots$

Big-O Notation

Let n be the length of the list.

$O(1)$

When the input size increases, the runtime does not increase.

Big-O Notation

Let n be the length of the list.

$O(1)$

When the input size increases, the runtime does not increase.

$O(n)$

When the input size is doubled, the runtime is doubled.

Big-O Notation

Let n be the length of the list.

$O(1)$

When the input size increases, the runtime does not increase.

$O(n)$

When the input size is doubled, the runtime is doubled.

$O(n^2)$

When the input size doubled, the runtime is quadrupled.

Big-O Notation

Let n be the length of the list.

$O(1)$

When the input size increases, the runtime does not increase.

$O(n)$

When the input size is doubled, the runtime is doubled.

$O(n^2)$

When the input size doubled, the runtime is quadrupled.

$O(2^n)$

When the input size doubled, the runtime is squared.

`max1`: Max So Far – Standard

Idea

Scan the list keeping track of the largest integer seen so far.

- `for v in L`

Notes

- arguably natural
- simple and straightforward

Analysis

- $O(n)$

Our Sample Input

Three Lists

```
L1 = list(range(10))
L2 = list(range(10**3))
L3 = list(range(10**6))
```

The items of each list are then scrambled using `random.shuffle`.

Our Sample Input

Three Lists

```
L1 = list(range(10))
L2 = list(range(10**3))
L3 = list(range(10**6))
```

The items of each list are then scrambled using `random.shuffle`.

Analysis

	$O(n)$	$O(n^2)$	$O(2^n)$
L1	✓	✓	✓
L2	✓	✓	×
L3	✓	×	×

Our Sample Input

Three Lists

```
L1 = list(range(10))
L2 = list(range(10**3))
L3 = list(range(10**6))
```

The items of each list are then scrambled using `random.shuffle`.

Analysis

	$O(n)$	$O(n^2)$	$O(2^n)$
L1	✓	✓	✓
L2	✓	✓	×
L3	✓	×	×

It is impossible to come up with a correct $O(1)$ algorithm.

`max2`: Max So Far – Shrinking List

Scan the list keeping track of the largest integer seen so far.

- `while len(L)>0`

`max2`: Max So Far – Shrinking List

Scan the list keeping track of the largest integer seen so far.

- `while len(L)>0`

Notes

- natural coming from Scheme or Racket
- uses handy slice operator (`L = L[1:]`)
- try `max2(range(10**6))`

`max2`: Max So Far – Shrinking List

Scan the list keeping track of the largest integer seen so far.

- `while len(L)>0`

Notes

- natural coming from Scheme or Racket
- uses handy slice operator (`L = L[1:]`)
- try `max2(range(10**6))`

Analysis

- $O(n^2)$

max3: Max So Far – Accumulative Recursion

Idea

Scan the list keeping track of the largest integer seen so far.

- `return max3acc(L, larger, i+1)`

`max3`: Max So Far – Accumulative Recursion

Idea

Scan the list keeping track of the largest integer seen so far.

- `return max3acc(L, larger, i+1)`

Notes

- information stored in parameters instead of variables
- limited by `sys.getrecursionlimit()`

`max3`: Max So Far – Accumulative Recursion

Idea

Scan the list keeping track of the largest integer seen so far.

- `return max3acc(L, larger, i+1)`

Notes

- information stored in parameters instead of variables
- limited by `sys.getrecursionlimit()`

Analysis

- $O(n)$

`max4`: Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

max4: Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

Notes

- try `max4(list(range(23)))`
- uses slicing but this inefficiency is overshadowed
- limited by `sys.getrecursionlimit()`

`max4`: Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

Notes

- try `max4(list(range(23)))`
- uses slicing but this inefficiency is overshadowed
- limited by `sys.getrecursionlimit()`

Analysis

- $O(2^n)$

max5: Faster Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

- restructure the if statement

`max5`: Faster Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

- restructure the `if` statement

Notes

- uses slicing
- limited by `sys.getrecursionlimit()`

`max5`: Faster Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

- restructure the `if` statement

Notes

- uses slicing
- limited by `sys.getrecursionlimit()`

Analysis

- $O(n^2)$

max6: Even Faster Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

- use an index instead of slicing

`max6`: Even Faster Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

- use an index instead of slicing

Notes

- limited by `sys.getrecursionlimit()`

`max6`: Even Faster Natural Recursion

Idea

Find the maximum of the first element and the maximum of the rest of the list.

- use an index instead of slicing

Notes

- limited by `sys.getrecursionlimit()`

Analysis

- $O(n)$

max7: Scan Until You Find It

Idea

Scan the list checking each item against all the others along the way.

`max7`: Scan Until You Find It

Idea

Scan the list checking each item against all the others along the way.

Notes

- maybe not as unnatural as we think
- nested loops

`max7`: Scan Until You Find It

Idea

Scan the list checking each item against all the others along the way.

Notes

- maybe not as unnatural as we think
- nested loops

Analysis

- $O(n^2)$

`max8`: Divide and Conquer

Idea

Find the maximums of the left and right halves and then the maximum of these two maximums.

max8: Divide and Conquer

Idea

Find the maximums of the left and right halves and then the maximum of these two maximums.

Notes

- try `max1(range(10**7))` and `max8(range(10**7))`
- `sys.getrecursionlimit()` does not seem to be a problem

`max8`: Divide and Conquer

Idea

Find the maximums of the left and right halves and then the maximum of these two maximums.

Notes

- try `max1(range(10**7))` and `max8(range(10**7))`
- `sys.getrecursionlimit()` does not seem to be a problem

Analysis

- $O(n \log n)$

max9: Sort

Idea

Put the integers in ascending order and grab the last one.

`max9`: Sort

Idea

Put the integers in ascending order and grab the last one.

Notes

- try `max9(range(10**8))` then shuffle and try again
- using built-in functions is great but dangerous

`max9`: Sort

Idea

Put the integers in ascending order and grab the last one.

Notes

- try `max9(range(10**8))` then shuffle and try again
- using built-in functions is great but dangerous

Analysis

- $O(n \log n)$

Variants

How might these variations change things?

Variants

How might these variations change things?

1. no duplicates
   - ???

Variants

How might these variations change things?

1. no duplicates
   - ???
2. items all bounded by 10^6
   - search for an item in `range(10**6,-1,-1)`

Variants

How might these variations change things?

1. no duplicates
   - ???
2. items all bounded by 10^6
   - search for an item in `range(10**6,-1,-1)`
3. goal is to minimize the number of comparisons
   - all but `max7`, and `max9` are (asymptotically) optimal

Variants

How might these variations change things?

1. no duplicates
   - ???
2. items all bounded by 10^6
   - search for an item in `range(10**6,-1,-1)`
3. goal is to minimize the number of comparisons
   - all but `max7`, and `max9` are (asymptotically) optimal

Challenge

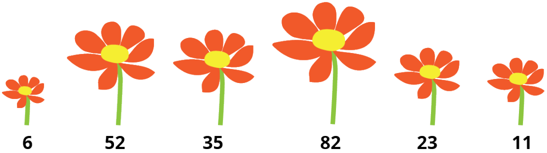
Find the *second* largest item among 100 integers using at most 105 comparisons.

Problem C

Collecting Pollen

Beever the bee makes 20 flights to a field of flowers. On each flight, it visits only one flower and can collect up to 10mg of pollen. It can return to the same flower more than once.

The initial amount of pollen in each flower (in mg) is shown below.



What is the maximum total amount of pollen Beever can collect?

Problem D

Find the Prize

Your friend is thinking of an integer between 1 and 63 (inclusive). They offer to give you money if you guess the integer.

If you guess the number on your first guess, you win \$1000. Every time you guess incorrectly, your friend will take \$10 away from the prize money, but also tell you whether your guess was above or below the integer they were thinking of.

You find a strategy that *guarantees* you win at least \$N, regardless of the number your friend is thinking of.

What is the largest possible value of N?

Beaver Computing Challenge

A CEMC fall contest you may not have heard of

- gentle introduction to logical and algorithmic thinking
- appropriate for students with no background in computer science
- intended for students in Grades 5 to 10
- weeks of November 4 and 11, 2019
- 45 minute online multiple-choice contest
- no programming required
- held in schools and supervised by teachers

We publish answers, explanations and connections to CS.

CEMC Courseware and Online Resources

CS Circles

- free interactive lessons teaching the basics of writing computer programs in Python requiring only a browser

Courseware

1. gentle introduction to programming designed with the beginner in mind
2. collection of videos teaching basic programming concepts in a language-independent manner
3. introduces the main ideas behind the specification of a web page in HTML5 and CSS3
4. presents the use of basic programming concepts as applied to web pages, using the language JavaScript

Canadian Computing Competition

A programming contest

- online grader for training and competition
- tests algorithmic thinking and implementation
- can begin practicing early with real-time feedback
- Pascal, C/C++, Python, Java are permitted
- offered for high school students at Junior and Senior levels
- top performers invited to the Canadian Computing Olympiad at UWaterloo

Sample problem

Compute the second largest item in a list of integers.

Thank you for Listening and Participating

Questions

- Now?
- Later: `jpretti@uwaterloo.ca` (don't be shy!)