New homework

Check your understanding of what we’ve done
• Focus on essential skills
• Know what to ask questions about
• Get practice for final
• Due 4/29

P1. Simulate an algorithm

Another beautiful algorithm from ancient Greece.
Problem: find largest square tile
that covers a rectangular room evenly.
Input: dimensions X and Y (positive integers)
Result: tile size T
Key step: taking remainder when one number is
divided by another – “mod” again
We’ll see why the algorithm works later.
It’s cool.
But you can understand what this does separately.
It’s good practice.
Everything you need is on the next slide.

repeat until $Y = 0$
- set $R$ to $X \mod Y$
- set $X$ to $Y$
- set $Y$ to $R$
set $T$ to $X$. 
**Problem is:**

Trace the loop:
- Give the values of X and Y each time we test whether to repeat the loop.

For examples:
- X = 32, Y = 12
- X = 27, Y = 8
- X = 13, Y = 21
- X = 60, Y = 36

---

**How to solve it**

Simulate the action of loop, arithmetic, assignment

You should know how this works
Heuristics

Assume bins have size 100.

What does FF give for this list of items:

\{14, 14, 14, 14, 14, 14, 34, 34, 34, 34, 34, 34, 52, 52, 52, 52, 52, 52\}

What does FFD give?

Heuristics

Assume bins have size 100.

What does FF give for this list of items:

\{52, 27, 21, 52, 27, 21, 29, 21, 29, 21, 29, 21, 29, 21, 29, 21\}

What does FFD give?

(Note: 6 21s, 2 each 27s, 29s, 52s)
How to solve it

Refer to the pseudocode or your intuitive understanding of the procedures
Carry them out intuitively

Note

You would need to do a lot more to write a computer program to do this

Like what?
Note

You would need to do a lot more to write a computer program to do this

Like what?

• A sorting algorithm
• Way of writing down how many bins (list?)
• Way of writing down what is in each bin (list?)

Note

You would need to do a lot more to write a computer program to do this

Computer scientists get very comfortable with ideas like “lists of lists”

• each list is a bin
• the list in each bin is the items packed there
• another case of abstraction/reduction
**Moral**

Heuristics go wrong in special cases

- You have to work harder and harder to avoid the special cases
- Eventually you reach diminishing returns

---

**Ways of writing loops**

For each of the following loops, say how often the cat says “hi”
V1

set i to 1
repeat until i > N
  • change i by 1
  • set j to 1
  • repeat until j > M
    • change j by 1
    • say “hi”

Aside:

How do you solve these problems?
  • Proceed from inside out, at “hi” (just like algebra)
  • Innermost “hi” is executed once each time it is reached
  • Decide how often next loop out runs – what is it counting? how many are there?
  • Keep multiplying counts
**V2**

set i to 1
repeat until i > K
  • set j to 1
  • repeat until j > M
    • change j by 1
    • say “hi”
  • change i by M

**V3**

set i to 1
set j to 1
repeat until i > N
  • change j by 1
  • if j > L
    • set j to 1
    • change i by 1
  • say “hi”
Moral

In computer programming, there are many ways, roughly speaking, to do the same thing.

Computer scientists talk and think about the algorithms, not the programs.

Songs, loops, programs

The House that Jack Built is what we call $O(N^2)$

- The length of each verse increases with the number of verses sung so far
- Like the 12 days of Christmas.
**Songs, loops, programs**

Verse 1 is

This is

the house that Jack built

Verse 11 is on the next slide. You can figure out all the other verses by looking at Verse 11.

---

**Songs, loops, programs**

This is

(11) the Farmer who sowed the corn that fed
(10) the Cock that crowed in the morn that waked
(9) the Priest all shaven and shorn that married
(8) the Man all tattered and torn that kissed
(7) the Maiden all forlorn that milked
(6) the Cow with the crumpled horn that tossed
(5) the Dog that worried
(4) the Cat that killed
(3) the Rat that ate
(2) the Malt that lay in
(1) the House that Jack built.
Design a Scratch program that uses “say” to print out all the words. It should use a variable $i$ that controls which verses (1 through $i$) will be said.

- Use loops or subroutines so you don’t repeat the same words in the script.

You can do this by hand. You might find it easier to check your work in Scratch though.

**How to solve it - 1**

- Talk through the song. You can’t write a program until you know what it has to do.
- Decide how you will store the words. Perhaps use a list where item $i$ holds what first follows “this is” in verse $i$. Then write the code to do the last verse.
- Take the code for the last verse and make it general. Add control so that later stuff is only said when needed.
How to solve it - 2

- Take the code for the last verse and make it general. Add control so that later stuff is only said when needed.
- Use the general code, and a loop, to print out all the verses.
- Test each part as you go along, by hand and if you want also on the computer

Moral

All this stuff about loops fits together.
You need to nest loops and break problems down.
Ultimately, that’s what’s needed for “lists of lists” examples like the bin packing example.
1. **Paradox**

Alice has written a program she calls TRPL. TRPL takes a program P and its input I.

Alice claims:

- TRPL(P,I) is true if you get 3 by running P on I
- TRPL(P,I) is false otherwise.

Alice is wrong.

Use pseudocode to write a paradoxical program using TRPL that shows that Alice cannot be right.
Answer

Paradoxical program X(P)

    if TRPL(P,P) return 4 else return 3

How to solve it

Paradoxical program will look like this:

    If TRPL(P,P) then return __ else return __

Alice says either TRPL(X,X) returns true or TRPL(X,X) returns false. We want fill in the blanks and show she is wrong.
How to solve it
Suppose TRPL(X,X) is true
• then, according to the code, X(X) = ___
• but according to Alice, X(X) = 3.
• that’s a contradiction, because ___ ≠ 3.

What do we do: put a value in the code here that’s different from 3, like 4.

How to solve it
Suppose TRPL(X,X) is false
• then, according to the code, X(X) = ___
• but according to Alice, X(X) ≠ 3.
• that’s a contradiction, because ___ = 3.

What do we do: put a value in the code here that’s equal to 3, like 3.
**Answer**

Paradoxical program $X(P)$

- if $TRPL(P,P)$ return 4 else return 3

---

**Halting**

Suppose $L$ can be any integer (zero, positive, negative). For which values of $L$ does this scratch pseudocode halt:

- repeat until $L = 0$
  - set $L$ to $L - 1$

Answer: $L = 0$, $L > 0$ : halts; $L < 0$ does not halt.
Halting, continued

Suppose L can be any integer (zero, positive, negative). For which values of L does this scratch pseudocode halt:

repeat until L < 1
  • set L to L - 1

Answer: always halts.
**Writing Pseudocode**

Suppose you define a rectangle on the screen with four numbers: T, L, B, R.
- T is the Y of the top of the rectangle
- L is the X of the left of the rectangle
- B is the Y of the bottom of the rectangle
- R is the X of the right of the rectangle

Following Scratch, (0,0) is the center of the screen, with bigger X to the right and bigger Y up.

---

**Writing Pseudocode, continued**

Scratch gives you values (in the sensing menu):
- mouse x – x coordinate of the mouse pointer
- mouse y – y coordinate of the mouse pointer

Write pseudocode to tell whether the mouse pointer is inside the rectangle.
**Answer**

Mouse pointer is inside rectangle exactly when
mouse x > L and mouse x < R and
mouse y > B and mouse y < T

---

**Writing Pseudocode**

You are given a list of numbers, such as these two:

- 1, 2, 2, 4, 9, 9, 15
- 1, 4, 6, 3, 8

Consider how you can tell, step by step, whether the list of numbers proceeds in order from smallest to largest.

- You should get “yes” for the first list.
- You should get “no” for the second.
**Answer**

Lists of length 1 or less are in order.

Assuming the list has length 2 or more,

- we will keep track of our position in the list - i
- the last number seen - n
- whether anything is out of order - wrong

start with i at 2, n as item 1 of the list, wrong = false

repeat until i exceeds the length of the list

- if n is larger than item i set wrong to true
- set n to item i
- change i by 1

---

**Understanding pseudocode**

If As = (s, l, o, g, s) and Bs = (t, w, i, c, e)?

Os = (s, l, i, c, e)

If As = (o, l, d, w, i, n) and Bs = (h, i, n, d, e, r)?

Os = (h, i, d, d, e, n)
Bugs in loops

The length of a list counts the number of elements currently in it. For which lengths of the list “numbers” does this scratch pseudocode halt:

set i = 1
repeat until i = length of numbers
  • replace item i of numbers with i
  • change i by 1

Careful: A list with no elements will have length 0!
Replacing an item past the end of the list does nothing!

Answer

set i = 1
repeat until i = length of numbers
  • replace item i of numbers with i
  • change i by 1

Halts for any length but 0.
Does not halt for 0.
Bugs in loops, continued

For which lengths of the list “numbers” does this scratch pseudocode halt:

set i = 1
repeat until i = length of numbers
• delete item i of numbers
• change i by 1

Deleting an item beyond the end of the list does nothing!

Answer

set i = 1
repeat until i = length of numbers
• delete item i of numbers
• change i by 1

If length = 0 or 1 – does not halt
If length = 2 – does not halt, and same for bigger even nos
If length = 3 – halts, and same for bigger odd nos
**Algorithms**

Does value $V$ occur in sorted list $L$?

---

**Simple scan**

set found = false  
set $i = 1$  
repeat until found or $i >$ length of list  
  • if item $i$ of list = value  
    • set found = true  
  • else  
    • change $i$ by 1
What if never found?

- set found = false
- set i = 1
repeat until found or i > length of list
  - if item i of list = value
    • set found = true
  - else
    • change i by 1

How many times does this loop run if list contains N things?

A. 1
B. 10
C. N
D. N²
E. N!

Better algorithm

Keep track of interval where item might be
Test the middle of the interval
Cut the interval in half if not found
Pseudocode...

set low = 1
set high = length of list
set pivot = \lfloor (low + high)/2 \rfloor

Visualization

| Low | Pivot | High |

Pseudocode...

repeat until low > high (ie., not found)
or value = item pivot of list (ie., found)

Visualization

| Low | Pivot | High |
Pseudocode...

- if value > item pivot of list
  set low to pivot + 1

Pseudocode...

- else (ie., value < item pivot of list)
  set high to pivot - 1
Pseudocode...

- finally set pivot = ⌊(low + high)/2⌋

Visualization

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Pivot</td>
</tr>
</tbody>
</table>

Summary

set low = 1
set high = length of list
set pivot = ⌊(low + high)/2⌋
repeat until low > high or value = item pivot of list
  • if value > item pivot of list
    set low to pivot + 1
  • else
    set high to pivot – 1
  • set pivot = ⌊(low + high)/2⌋
How big a list?

If done after 1 iteration?

• 1 item

How big a list?

If done after 2 iterations?

• 3 items
  • check 1 in first iteration
  • divide remainder 2 in halves of size 1
  • check 1 in last iteration
How big a list?

If done after 3 iterations

• 7 items
  • check 1 in first iteration
  • divide remainder 6 in halves of size 3
  • check 3 in 2 more iterations

How big a list?

If done after 4 iterations

• 15 items
  • check 1 in first iteration
  • divide remainder 14 in halves of size 7
  • check 7 in 3 more iterations
How big a list?

If done after 5 iterations

- 31 items
  - check 1 in first iteration
  - divide remainder 30 in halves of size 15
  - check 15 in 4 more iterations

Summary

Handle list of size $2^{N-1}$ in $N$ iterations.

“Logarithmic complexity”

If list has length $L$ need log $L$ time.

$$2^{\log L} = L$$
**Motivation: Sorting**

It’s a big win to put stuff in order

- Alex has 30 items jumbled in a bag
- Bo has 1 Billion items in sorted order
- Ask them to give you something.
- Who tells you first if they don’t have it?

A. Alex  
B. Bo

---

**Motivation: Sorting**

It’s also surprisingly easy

- Alex has inherited grandpa’s accounting firm. It has a billion pieces of paper out of order.
- For what N would you say: if Alex expects to need N of those pieces of paper, it’s worth it to sort the whole lot.

A. 1 Million  
B. 10,000  
C. 1000  
D. 100