Happy 1 month & 1 day until Pi Day!

Edward Frenkel

Is the Universe a Simulation?

FEB. 14, 2014

Several algorithms...

Hw #3 due Monday, 10:00

Sound Lab!

Office Hrs.!

Today, 2:30-4,
HMC's LAC lab...

Hw #3 due Monday, 10:00

Sound Lab!
Sound + Starbucks!

We saw you in Lab3... heard
Take-away ~ Lab3

```python
def flipflop(filename):
    """flipflop swaps the halves of an audio file
    input: filename, the name of the original file
    output: no return value, but
        this creates the sound file 'out.wav'
        and plays it"
    
    print("Playing the original sound...")
    play(filename)

    print("Reading in the sound data...")
    sound_data = [0, 0]
    read_wav(filename, sound_data)
    samps = sound_data[0]
    sr = sound_data[1]

    print("Computing new sound...")
    # this gets the midpoint and calls it x
    x = len(samps)//2
    newsamps = samps[x:] + samps[:x]

    newsr = sr
    new_sound_data = [ newsamps, newsr ]

    print("Writing out the new sound data...")
    write_wav( new_sound_data, "out.wav" ) # write data to out.wav

    print("Playing new sound...")
    play('out.wav')
```

intro stuff – important, but less algorithmic

algorithmic stuff

"outro" stuff
BR 5 Snczx

**Algorithms**

*Englishness...*
*Classifying life*
*Removing/Sorting and Jotto!*

**HW 3**

Hw #3 due **Monday, 10:00**

*Sound Lab!*
*Several algorithms...*

---

Edward Frenkel

*Is the Universe a Simulation?*

FEB. 14, 2014
BR 5 Snczx

**Algorithms**

*Englishness...*
Classifying life
Removing/Sorting and *Jotto!*

---

**HW 3**

Hw #3 due **Monday, 11:59**

*Sound Lab!*

*Several algorithms.*
Caesar Cipher: **encipher**

encipher(s, n) should return the string s with each alphabetic character shifted/wrapped by n places in the alphabet.

- encipher( 'I <3 Latin' , 0 ) returns 'I <3 Latin'
- encipher( 'I <3 Latin' , 1 ) returns 'J <3 Mbujo'
- encipher( 'I <3 Latin' , 2 ) returns 'K <3 Ncvkp'
- encipher( 'I <3 Latin' , 3 ) returns 'L <3 Odwlq'
- encipher( 'I <3 Latin' , 4 ) returns 'M <3 Pexmr'
- encipher( 'I <3 Latin' , 5 ) returns 'N <3 Qfyns'
- ...
- ...
- encipher( 'I <3 Latin' , 25 ) returns 'H <3 Kzshm'
ASCII and Unicode

convert # to char

\texttt{chr}


downarrow

class \texttt{ord}

convert char to #

\texttt{This is why 'CS' < 'clear'!}

Julius spr'15
def rot13( c ):
    """ rotates c by 13 chars, "wrapping" as needed 
    NON-LETTERS don't change! 
    """
    if 'a' <= c <= 'z':
        if ord(c)+13 <= ord('z'):
            return chr( ord(c)+13 )
        else:
            return chr( ord(c)+13-26 )
    elif 'A' <= c <= 'Z':
        return c
    else:
        Extra: How would you rotate n places, instead of 13?
        # upper-case test!
        # same, but for 'Z'
        # upper-case test!
        # same, but for 'Z'

(0) What do these tests do?
(1) What code will "wrap" to the alphabet's other side?
(2) How will upper case change? Try noting only the code differences...
(3) What if c is not a letter at all?
Caesar Cipher: \texttt{encipher}

\begin{verbatim}
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',25)
'Aycqyp agnfcp? G npcdcp Aycqyp qyjyb.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',15)
'Qosgof qwdvsf? W dfstsf Qosgof gozor.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',4)
'Fdhvdu flskhu? L suhihu Fdhvdu vdodg.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',1)
'Caesar cipher? I prefer Caesar salad.'

An education is what remains after we forget everything we have learned.'
\end{verbatim}
Caesar Cipher: \texttt{decipher}

>>> \texttt{decipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.'})
'Cae\textsf{s}ar cipher? I prefer C\textsf{a}esar s\textsf{a}lad.'

>>> \texttt{decipher('Hu lkbjhapvu pz doha ylthpuz hmaly dl mvynla \textbackslash
c\textbackslash lclyfaopun dl ohcl slhyulk.'})
'An education is what remains after we forget everything we have learned.'

>>> \texttt{decipher('Uifz xpsl ju pvu xjui b qfodjm!')} 

>>> \texttt{decipher('gv vw dtwvg')} 

But \texttt{how}?
Decipher?

All possible decipherings

Strategies?

Algorithms?

gv vw dtwvg
hw wx euxwh
ix xy fvyxi
jy yz gwzyj
kz za hxazk
la ab iybal
mb bc jzcbm
nc cd kadc
od de lbedo
pe ef mcfep
qf fg ndgfq
rg gh oehgr
sh hi pfihs
ti ij qgjit
uj jk rhkju
vk kl silkv
wl lm tjmlw
xm mn uknmx
yn no vlony
zo op wmpoz
ap pq xnqpa
bq qr yorqb
cr rs zpsrc
ds st aqtsd
et tu brute
fu uv csvuf
Measuring *Englishness*

**Very English-y**
- "Call me Ishmael."
- "Attack at dawn!"
- "rainbow, table, candle"
- "Yow! Legally-imposed CULTURE-reduction is CABBAGE-BRAINED!"
- "quadruplicity drinks procrastination"
- "Hold the newsreader's nose squarely, waiter, or friendly milk will countermand my trousers."
- "the gostak distims the doses"
- "hension, framble, bardle"
- "jufict, stofwus, lictpub"
- "itehbs, rsnevtr, khbsota"
- "epadxo, nojarpn, gdxokpw"
- "hoq dedqBzdrzqrzkzc"

**Not English-y**

**higher scores**

**quantifying "Englishness"?**

**lower scores**

All of these sound good to me!
Decipher?

<table>
<thead>
<tr>
<th>All possible decipherings</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 'gv vw dtwvg'],</td>
</tr>
<tr>
<td>[2, 'hw wx euxwh'],</td>
</tr>
<tr>
<td>[2, 'ix xy fvyxi'],</td>
</tr>
<tr>
<td>[0, 'jy yz gwzyj'],</td>
</tr>
<tr>
<td>[2, 'kz za hxazk'],</td>
</tr>
<tr>
<td>[1, 'la ab iybal'],</td>
</tr>
<tr>
<td>[0, 'mb bc jzcbm'],</td>
</tr>
<tr>
<td>[1, 'nc cd kadcn'],</td>
</tr>
<tr>
<td>[4, 'od de lbedo'],</td>
</tr>
<tr>
<td>[3, 'pe ef mcfep'],</td>
</tr>
<tr>
<td>[0, 'qf fg ndgfq'],</td>
</tr>
<tr>
<td>[2, 'rg gh oehgr'],</td>
</tr>
<tr>
<td>[2, 'sh hi pfihs'],</td>
</tr>
<tr>
<td>[3, 'ti ij qgjit'],</td>
</tr>
<tr>
<td>[2, 'uj jk rhkju'],</td>
</tr>
<tr>
<td>[1, 'vk kl slikv'],</td>
</tr>
<tr>
<td>[0, 'wl lm tjmlw'],</td>
</tr>
<tr>
<td>[1, 'xm mn uknmx'],</td>
</tr>
<tr>
<td>[2, 'yn no vlony'],</td>
</tr>
<tr>
<td>[3, 'zo op wmpoz'],</td>
</tr>
<tr>
<td>[2, 'ap pq xnqpa'],</td>
</tr>
<tr>
<td>[1, 'bq qr yorqb'],</td>
</tr>
<tr>
<td>[0, 'cr rs zpsrc'],</td>
</tr>
<tr>
<td>[1, 'ds st aqtsd'],</td>
</tr>
<tr>
<td>[4, 'et tu brute'],</td>
</tr>
<tr>
<td>[3, 'fu uv csvuf']</td>
</tr>
</tbody>
</table>

Score them all max!

"Englishness" score ~ the #-of-vowels
### Decipher?

- **All possible decipherings**
  - `gv vw dtwvg`
  - `hw wx euxwh`
  - `ix xy fvyxi`
  - `jy yz gwzyj`
  - `kz za hxazk`
  - `la ab iybal`
  - `mb bc jzcbm`
  - `nc cd kadcn`
  - `od de lbedo`
  - `pe ef mcfep`
  - `qf fg ndgfq`
  - `rg gh oehgr`
  - `sh hi pfihs`
  - `ti ij qgjit`
  - `uj jk rhkju`
  - `vk kl silkv`
  - `wl lm tjmlw`

### Strategies?

#### Algorithms?

- "Englishness" based on letter-probabilities
  - `et tu brute`
  - `fu uv csvuf`
Decipher?

Strategies?

Algorithms?

decPR(LAT)
decPR2(LAT)
decPR3(LAT)

All possible decipherings

Using the **LoL** technique to score each rotation's "Englishness"

"Englishness" based on scrabble-scoring!

[27, 'et tu brute']

min!
Design...

The of CS (and CSers...)

Algorithms!
Design...

Design of what?

Code?

syntax
The Economist explains

What is code?

Sep 8th 2015, 23:50 BY T.S.

FROM lifts to cars to airliners to smartphones, modern civilisation is powered by software, the digital instructions that allow computers, and the devices they control, to perform calculations and respond to their surroundings. How did that software get there?

Someone had to write it. But code, the sequences of symbols painstakingly created by programmers, is not quite the same as software, the sequences of instructions that computers execute. So what exactly is it?

Coding, or programming, is a way of writing instructions for computers that bridges the gap between how humans like to express themselves and how computers actually work. Programming languages, of which there are hundreds, cannot generally be executed by computers directly. Instead, programs written in a particular “high level” language such as C++, Python or Java are translated by a special piece of software (a compiler or an interpreter) into low-level instructions which a computer can actually run. In some cases programmers write software in low-level instructions directly, but this is fiddly. It is usually much easier to use a high-level programming language, because such languages make it
Design...

Code?

Algorithm!
Algorithm Design...

\[
\text{remAll}(e, L)
\]

*remove all e’s from L*
Design...

**Top-down design**

- Visualize
- Split into parts
- Build each part
- Combine
- Test

remAll(e, L)  
*remove all e's from L*

remAll(42, [5, 7, 42, 8, 42])  
\[5, 7, 8\]

remAll('q', 'qaqqlqqiqliqiqeqqnqs')  
'aliiiiens'
Design...

Top-down design

Visualize

Split into parts

Build each part

Combine

Test

remAll(e, L)
remove all e's from L

remAll(42, [5, 7, 42, 8, 42])

[5, 7, 8]

remAll('q', 'qaqqlqqiqqqiiqeqqqnqqs')

'aliiiiens'

Use it!

Lose it!
**Design...**

**Top-down design**
- Visualize
- Split into parts
- Build each part
- Combine
- Test

**Removal Functions**
- `remAll(e, L)`
  - remove all `e`'s from `L`

**Examples**
1. `remAll(42, [5, 7, 42, 8, 42]`)
   - keep `L[0]`
   - + remove `e` from the rest
   - `[5, 7, 8]`

2. `remAll('q', 'qaqqlqqiqqqiiqiqqeqqqqngs'`)
   - drop `L[0]`
   - + remove `e` from the rest
   - `'aliiiiens'`
Design...

Top-down design
- Visualize
- Split into parts
- Build each part
- Combine
- Test

remAll(e,L)
remove all e's from L

keep L[0]
+ remove e from the rest

remAll('q','qqqlqqiqlqqiiqlqqqnnqs
aliiiens
remAll(e,L)
remove all e's from L

Use it!
- or -
Lose it.
Allie Russell, '12 speaking of roadside church signs...
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])

If there are no elements or characters in L, we're done – return L itself!
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
def remAll(e, L):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])

If it is e, lose it (don't keep it in the return value)

AND still remove all of the e's from the rest of L!
Design ~ code

That's it. Algorithmic expression ~ it's what CSers (think they) do.

```python
def remAll(e, L):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:]
    else:
        return remAll(e, L[1:])
```

That's it.
def remAll( e, L ):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])

def subseq( s, sbig ):
    """
    returns True if s is a subseq. of sbig,
    False otherwise. Both are strings.
    ""
    if s == '':
        return True
    elif

Don't start yet...

Algorithm design

Change remAll so that it removes only one e from L.
(We could call it remOne.)

remOne(8,[7,8,9,8]) → [7,9,8]

Make more changes to remAll so that it removes all of the elements up to and including the first e in L.
(We could call it remUpto.)

remUpto( 'd', 'coded') → 'ed'

If e is not in L, remUpto should remove everything...

Write the other cases needed for subseq...

subseq('alg','magical')
False

subseq('alg','twasbrillig')
True

cs5 hrs last week
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:]),
    else:
        return remAll(e,L[1:]),

remOne(8,[7,8,9,8]) ➞ [7,9,8]

Algorithm design

Change remAll so that it removes only one e from L. (We could call it remOne.)

Make more changes to remAll so that it removes all of the elements up to and including the first e in L. (We could call it remUpto.)

remUpto('d','coded') ➞ 'ed'

Challenge...

Write the other cases needed for subseq...

def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig, False otherwise. Both are strings. """
    if s == '':
        return True
    elif

subseq('alg','magical')
False

subseq('alg','twasbrillig')
True

Hint: In both cases, what's needed is mostly crossing stuff out! What stuff?
remAll insight

```python
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

remAll(8, [7,8,9,8]) → [7,9]

sharpening our model for where + how actions happen...
other `rem` examples...

```plaintext
remAll(8, [7, 8, 9, 8]) → [7, 9]
remAll('d', 'coded') → 'coe'
remOne(8, [7, 8, 9, 8]) → [7, 9, 8]
remOne('d', 'coded') → 'coed'
remUpto(8, [7, 8, 9, 8]) → [9, 8]
remUpto('d', 'coded') → 'ed'
```
Subsequences

```python
def subseq(s, sbig):
    # s is the subsequence to find (or not)
    # sbig is the bigger string in which we are looking for s
```

subseq('', 'cataga') ➞ True
subseq('ctg', 'cataga') ➞ True
subseq('ctg', 'tacggta') ➞ False
subseq('aliens', 'always frighten dragons') ➞ True
subseq('trogdor', 'that dragon is gone for good') ➞ False

Why Are these True? or False?

In order, but not necessarily adjacent...
def remAll(e, L):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])

remOne(8, [7, 8, 9, 8]) ➞ [7, 9, 8]

remUpto('d', 'coded') ➞ 'ed'

def subseq(s, sbig):
    """ returns True if s is a subseq. of sbig, False otherwise. Both are strings.
    ""
    if s == '':
        return True
    elif:

subseq('alg', 'magical')
subseq('alg', 'twasbrillig')

Challenge...
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])

def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig, False otherwise. Both are strings. """
    if s == '':
        return True
    elif
def remAll( e, L ):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])

remOne(8,[7,8,9,8]) ➞ [7,9,8]

Hint: remove one thing for remOne!

Hint: remove one more thing for remUpto!

Hint: In both cases, what's needed is mostly crossing stuff out! What stuff?

def subseq( s, sbig ) :
    """ returns True if s is a subseq. of sbig, False otherwise. Both are strings. ""
    if s == '':
        return True
    elif s[0] in sbig:
        return subseq(s[1:], sbig)
    else:
        return False

subseq('alg','magical')
subseq('alg','twasbrillig')

Hint: you'll need 3-4 cases total for subseq.
from `remAll` to `remOne`

```python
def remAll(e, L):
    """ returns seq. L with all e's rmoved """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])
```

`remOne(8,[7,8,9,8])` ➞ `[7,9,8]`  
`remOne('d','coded')` ➞ 'coed'

**Hint:** remove one thing for `remOne`!
from remOne to remUpto

def remOne( e, L ):
    """ returns seq. L with one e removed """
    if len(L) == 0:
        return L

    elif L[0] != e:
        return L[0:1] + remOne( e, L[1:]

    else:
        return L[1:]  # done!

remUpto(8,[7,8,9,8]) ➞ [9,8]  
remUpto('d','coded') ➞ 'ed'
```python
def subseq(s, sbig):
    """ returns True if s is a subseq. of sbig; False otherwise. Both are strings. """
    if s == '':
        return True
    elif s[0] in sbig:
        rest = remupto(s[1:], sbig)
        return subseq(s[1:], rest)
    else:  # s[0] is not in sbig
        return False
```

subseq('alg', 'magical')
False
subseq('alg', 'twasbrillig')
True
I ❤️ NY
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig; False otherwise. Both are strings. """
    if s == '':
        return True

subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
What is a small (initial) piece of the problem?
How would we describe it in terms of the inputs?

What is left after handling this piece?
Are there other functions we will need?

Use it!
- or -
Lose it!
def subseq(s, sbig):
    """ returns True if s is a subseq. of sbig; False otherwise. Both are strings. """
    if s == '':
        return True
    elif s[0] not in sbig:
        return False
    elif s[0] == sbig[0]:
        return subseq(s[1:], sbig[1:])
    else:
        return subseq(s[0:], sbig[1:])
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:]):
    else:
        return remAll(e,L[1:])

def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig, False otherwise. Both are strings. """
    if s == '':
        return True
    elif

Algorithm design

Pass those in and up...

Challenge...

Write the other cases needed for subseq...

subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
Design ~ (code)

That's it. *Algorithmic expression ~ it's what CSers (think they) do.*

wrench!
What's the problem?!

Top-down design

- Visualize
- Split into parts
- Build each part
- Combine
- Test

Which one of these steps is the most important?
What's the problem?!

Top-down design

Visualize

Split into parts

Build each part

Combine

Test

understanding what the problem demands!!

I want some examples!
hw3pr2: use it or lose it

Longest Common Subsequence

Jotto Score counting

binary list and general list sorting

exact_change making
Longest Common Subsequence

LCS( S, T )

'HUMAN'

'CHIMPANZEE'

'CGCTGAGCTAGGCA...'

'ATCCTAGGTAACTG...'

Changes from subsequence:
- return type
- we can skip 360J or T30J

Eye oneder if this haz other aplications?
Why LCS?

Screenshot from the ClustalX multiple subsequence alignment tool...

| 1 | Metridium | AATACCCAAATCC CTAGTCAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 2 | A.sulcata  | AATACCCAAATCTCTAGTCAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 3 | Hematodinium| AATACCCAAATCTCTAGTCAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 4 | S.raphanus | AATACCCACTCTCTAGTCAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 5 | N.virens  | AATACCCACTCTCTAGTCAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 6 | L.latreillii | AATACCCACTCTCTAGTCAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 7 | Modiolus  | AATACCCACTCTCTAGTCAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 8 | S.solidissima | AATACCCATTCCCAGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 9 | Pagurus   | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 10 | Emerita   | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 11 | Coelotes  | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 12 | F.lateroclitus | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 13 | Chrysocopa| AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 14 | D.simulans | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 15 | S.purpuratus | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 16 | A.forbesi | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 17 | G.rhodei  | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 18 | A.crucifera| AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| 19 | M.portucalensis | AATACCCACTCTCGGAGGAGGTGATGCAAAATCAATACAGGGCTCTTT-
| ruler       | ..........490..........500..... |
What was gained (or lost) here?
What was gained (or lost) here?

What was gained (or lost) here?

Mourning species...?

Night-loving species!

Caffeine cinema chocolate

Phylogeny
Subsequences @ 5Cs

host: figs

parasites: wasps

matching two phylogenies together together!
Jane's source data: 100s of species, 6 continents ...

Largest co-phylogeny ever computed (in 2012)
also in hw3pr2: *Jotto*!

*a word-guessing game...*

```
jscore( S, T )
```
also in hw3pr2: !

These are two cute

Let's try it!

'robot'  'otter'

\( \text{jscore}( \text{'robot'}, \text{'otter'} ) \rightarrow \)

\( \text{jscore}( S, T ) \)  \( \text{in general...} \)
also in hw3pr2: \texttt{sort + exact\_change}

\begin{align*}
\text{sort( [42,5,7] )} & \quad \rightarrow \quad [3,7,42] \\
\text{sort( [42,7] )} & \quad \rightarrow \quad [7,42] \\
\text{sort( [42] )} & \quad \rightarrow \quad [42] \\
\text{returns an ascending list} \\
\text{returns True or False}
\end{align*}

\begin{align*}
\text{exact\_change( 42, [25,30,2,5] )} & \quad \rightarrow \quad False \\
\text{exact\_change( 42, [25,30,2,15] )} & \quad \rightarrow \quad True
\end{align*}
The diagram outlines several algorithms and their implementations:

1. **jscore** function for computing jotto scores for strings:
   - `jscore('robot', 'otter')` → 3
   - `jscore('geese', 'seems')` → 3
   - `jscore('fluff', 'lulls')` → 2
   - `jscore('pears', 'diner')` → 3
   - `jscore('xylyl', 'slyly')` → 0

2. **sort** function for sorting a list:
   - `sort([42,5,7])` → [5,7,42]
   - `sort([42,7])` → [7,42]
   - `sort([42])` → [42]
   - `sort([])` → []

3. **blsort** function for sorting binary lists:
   - `blsort([1,0,1])` → []

4. **LCS** function for finding the Longest Common Subsequence:
   - `LCS('ctga', 'tagca')` → 'tga'
   - `LCS('tga', 'taacg')` → 'ta' (or 'tg')
   - `LCS('tga', 'a')` → 'a'
   - `LCS('gattaca', 'ctctgcgat')` →

5. **exact_change** function for checking if any subset of elements in a list add up to a target:
   - `exact_change(42, [25,30,2,5])` → False
   - `exact_change(42, [22,16,3,2,17])` → True
   - `exact_change(42, [18,21,22])` → False
   - `exact_change(42, [40,17,1,7])` → True
   - `exact_change(20, [16,3,2,17])` → True

The diagram also includes notes on brainstorming algorithms for these problems and considering helper functions that might be useful for each.

**Extra!** Which of these 10 is the cruellest hidden jotto word?
jscore(s1,s2)

<table>
<thead>
<tr>
<th>Input</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>'robot', 'otter'</td>
<td>3</td>
</tr>
<tr>
<td>'geese', 'seems'</td>
<td>3</td>
</tr>
<tr>
<td>'fluff', 'lulls'</td>
<td>2</td>
</tr>
<tr>
<td>'pears', 'diner'</td>
<td>2</td>
</tr>
<tr>
<td>'xylyl', 'slyly'</td>
<td>4</td>
</tr>
</tbody>
</table>

Extra! Which of these 10 is the crueller hidden jotto word?

should return the jotto score for any strings s1 and s2

sort(L)

<table>
<thead>
<tr>
<th>Input</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>[42,5,7]</td>
<td>[5,7,42]</td>
</tr>
<tr>
<td>[42,7]</td>
<td>[7,42]</td>
</tr>
<tr>
<td>[42]</td>
<td>[42]</td>
</tr>
<tr>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>[1,0,1]</td>
<td>[0,1,1]</td>
</tr>
</tbody>
</table>

LCS(S,T)

<table>
<thead>
<tr>
<th>Input</th>
<th>LCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ctga', 'tagca'</td>
<td>'tga'</td>
</tr>
<tr>
<td>'tga', 'taacg'</td>
<td>'ta' (or 'tg')</td>
</tr>
<tr>
<td>'tga', 'a'</td>
<td>'a'</td>
</tr>
<tr>
<td>'gattaca', 'ctctgcgat'</td>
<td>'ttca'</td>
</tr>
</tbody>
</table>

exact_change(t,L)

<table>
<thead>
<tr>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>42, [25,30,2,5]</td>
<td>False</td>
</tr>
<tr>
<td>42, [22,16,3,2,17]</td>
<td>True</td>
</tr>
<tr>
<td>42, [18,21,22]</td>
<td>False</td>
</tr>
<tr>
<td>42, [40,17,1,7]</td>
<td>False</td>
</tr>
<tr>
<td>20, [16,3,2,17]</td>
<td>True</td>
</tr>
</tbody>
</table>
should return the `jotto` score for any strings `s1` and `s2`:

\[
\text{jscore}(s1, s2)
\]

- `jscore('robot', 'otter')` → 3
- `jscore('geese', 'seems')` → 3
- `jscore('fluff', 'lulls')` → 2
- `jscore('pears', 'diner')` → 2
- `jscore('xylyl', 'slyly')` → 4

\[
\text{exact_change}(t, L)
\]

returns True if any subset of elements in `L` add up to `t`; returns False otherwise:

- `exact_change(42, [25,30,2,5])` → False
- `exact_change(42, [22,16,3,2,17])` → True
- `exact_change(42, [18,21,22])` → False
- `exact_change(42, [40,17,1,7])` → False
- `exact_change(20, [16,3,2,17])` → True

should return a new list that is the sorted version of the input `L`:

\[
\text{sort}(L)
\]

- `sort([42,5,7])` → [5,7,42]
- `sort([42,7])` → [7,42]
- `sort([42])` → [42]
- `sort([])` → []
- `blsort([1,0,1])` → [0,1,1]

should return the **Longest Common Subsequence** of strings `S` and `T`:

\[
\text{LCS}(S, T)
\]

- `LCS('ctga', 'tagca')` → 'tga'
- `LCS('tga', 'taacg')` → 'ta' (or 'tg')
- `LCS('tga', 'a')` → 'a'
- `LCS('gattaca', 'ctctgcgat')` → 'ttca'

\[
\text{remOne}
\]

**Extra!** Which of these 10 is the cruellest hidden jotto word?

\[
\text{min}
\]

\[
\text{Brainstorm algorithms} \text{ for these problems -- what helper functions?? might help for each?}
\]

Returns True if any code for these...

- `use it!
- lose it!
- don't write any code for these...
- use it!
- lose it!
- lose it!
- lose it!
- use it!
- lose it!

Only recursion here...

- ... and here
decipher( 'Weet bksa ed xecumeha 3!' )
decipher( 'Weet bksa ed xecumeha 3!' )

Good luck on homework 3!

and have a great weekend ...

kxn rkfo k qbokd goouoxn ...