find triplets (sum of 2 ints = other int)

i beginning →

j i →

k j →

merge 2 sorted arrays
BUT
do not use extra space
should execute in O((m+n) log(m+n))

input: A1[] = {10}
A2[] = {2, 3}

output: A1[] = {2}
A2[] = {3, 10}

input: A1 {1, 5, 9, 10, 15, 20}
A2 {2, 3, 8, 13}

output: A1 {1, 2, 3, 5, 8, 9}
A2 {10, 13, 15, 20}

largest number
from array

3 30 34 5 9

→ 9534330

54 546 548 60

→ 6054854654

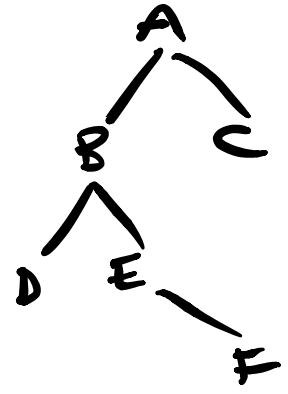
check for BST

just inorder?

D < B < E < F < A < C

↑

check as each is added



compare two trees equal

(mirror → compare root.left, root.right ?)

set up 2 queues q1, q2

run through level traversal (BFS)

for both q1 and q2 in one loop

```
q1.push(root1)
q2.push(root2)
diff = False
while q1 not empty AND q2 not empty AND diff == False:
    n1 = q1.pop()
    n2 = q2.pop()
    if n1.data != n2.data:
        diff = True
        break
    q1.push(n1.left)
    q1.push(n1.right)
```

→ only push if

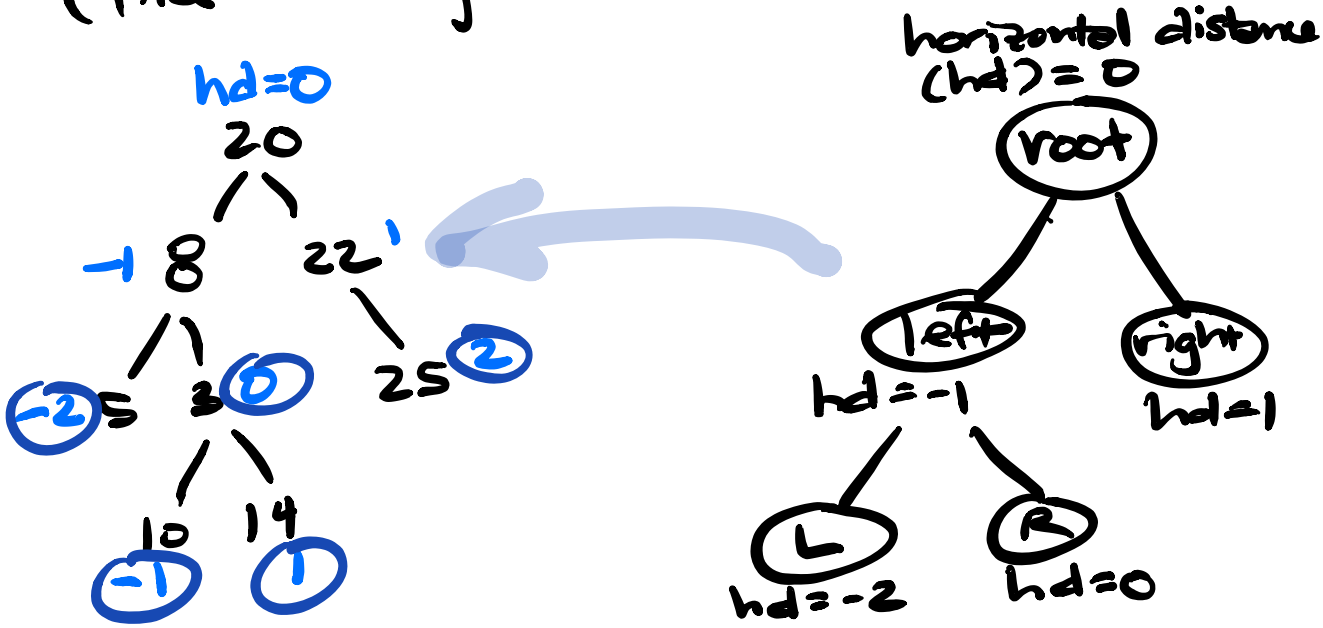
```
        q2.push(n2.left)
        q2.push(n2.right)
```

value is not None

```
if diff == True:
    return False
else:
    return q1.is_empty() == q2.is_empty()
```

TREE BOTTOM VIEW

(like looking at tree from bottom?)



bottom view: 5 10 3 14 25

So for each horizontal distance, choose the LOWEST node in the tree

Can I just do DF traversal that keeps track of LEVEL and HORIZONTAL DISTANCE?

(recursive method takes 'level'

parent for that increment ... elements ?)

this should allow me to determine tree height, yes?

LOWEST COMMON ANCESTOR (LCA) IN BST

do Depth First Search (DFS) for each of the two node values

this gives us path to node1 and node2

push each node along the way into a stack (one stack for each path)

FINALLY, pop values off each stack, if value exists in 'bag', we have reached LCA; otherwise, add the values to the bag

NUMBER OF LEAF NODES

leaf node = NO child nodes

So just Depth First traversal and when we reach any childless node, add it to our 'Leaf' bag. Yes?

GIVE CHANGE AS COINS (YouTube Google int)

33¢ → 25¢ 5¢ 1¢ x3

quarter	dime	nickel	penny
25¢	10¢	5¢	1¢

```
def num_coins(cents):
    coins=[25,10,5,1]
    num_coins=0
    for coin in coins:
        num_coins += cents/coin
        cents = cents % coin
        if cents==0:
            break
```

int division? →

num_coins(31) → 3 (25+5+1)

NOW... if you didn't have nickels, this algo num_coins(31) would give 7 (25+1+1+1+1+1+1)

but minimum coin answer is 4 (10+10+10+1).

How to fix this?

So... coins = [25, 10, 1]

num_coins(31) ▶

← work backward

Subtract 1 until cents % 10 == 0

coins = [25, 10, 1]

index: 0 1 2

div = [0, 0, 0]

(whole number times divides into cents)

31/25 = 1 r 6

31/10 = 3 r 1

31/5 = 6 r 1

31/1 = 31 r 0

ATTENDING WORKSHOPS

```
struct Workshop
    startTime
    duration
    endTime

struct AvailableWorkshops
    int n  (# of workshops signed up for)
    A<Workshop>[n]
```

```
AvailableWorkshops initialize(int startTime[],
    int duration[], int n)

int CalculateMaxWorkshops(AvailableWorkshops w)
```

sample input:

```
6
130558
116241
```

6 — n

130558 — start times

116241 — durations

sample output:

```
4
```

4 — max workshops that can be attended

SOCK MERCHANT

- given array of ints where each int represents a sock color
- determine how many pairs exist (ignore left-over singles)

n = 7
ar = [1, 2, 1, 2, 1, 3, 2] } pairs = 2

int sockMerchant(n, ar)

sample input:

9

10 20 20 10 10 30 50 10 20

sample output:

3

Facebook interview YouTube

MESSAGE MAPPING

mapping {}

'a' → 1

'b' → 2

⋮

'z' → 26

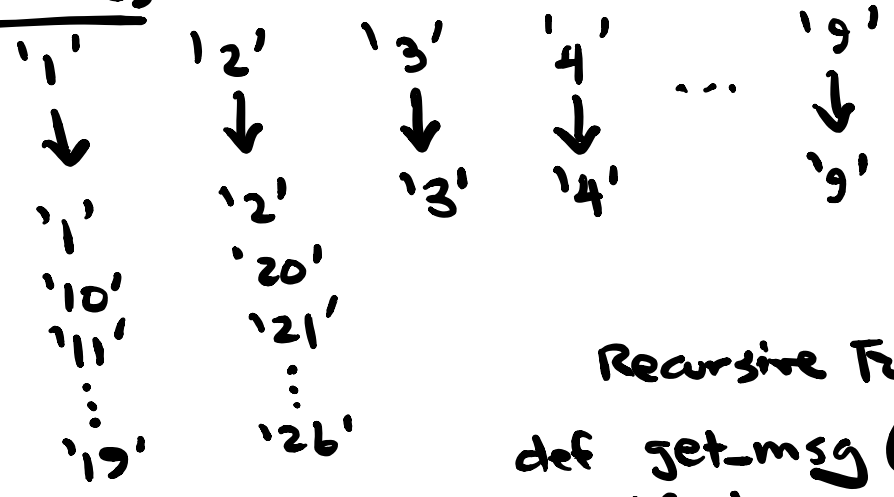
data = '12'

- write function that generates messages that could have been original message

(my 1st idea)

TAKE mappings and break it up into dictionary (or tree?) by FIRST char:

firstchar{}

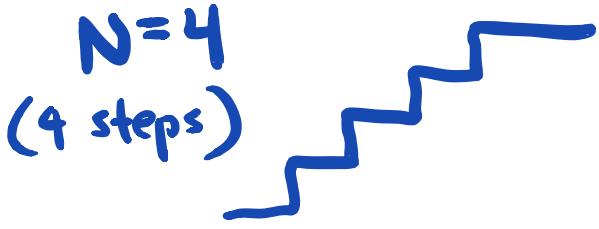


Recursive Function:

```
def get_msg(code, results=[])
    if len(code)==0:
        return ''
    ch = code[0]
    li = firstchar[ch]
    for xc in li:
        if len(code) >= len(xc):
            get_msg(
```

RECURSIVE STAIRCASE PROBLEM

Amazon interview YouTube



- N = # of stairs
- can only take 1 or 2 steps each "move"
- write function num_ways(N) which returns # of ways you can get from bottom to top

Base Case

- reached top step (add zero) or reached step N-1 (add one)

current step ↓ # of ways accumulator ↓

```
def num_ways(N, current=0, ways_acc=0):
    if current == N:
        return ways_acc
    elif current == N-1:
        return ways_acc + 1
    else:
        total += num_ways(N, current+1, ways_acc+1)
        total += num_ways(N
    return 2 + num_ways(N, current+1) +
               num_ways(N, current+2)
```

FIRST RECURRING CHARACTER

Google interview YouTube

- given a string, return the first recurring character in that string

"ABCA" → 'A'

"BCABA" → 'B'

"ABC" → None

(my idea)

- start at first and last chars and work outside-in, storing found chars in a dictionary

```
def first_recur(s):
    bag = {}
    N = len(s)
    for i in range(N/2):
```

```
if s[i] in bag: return s[i]
bag[s[i]] = True
if s[N-i] in bag: return s[N-i]
bag[s[N-i]] = True
```

ROW-WISE AND COLUMN-WISE SORTED MATRIX: COUNT NEG INTS

Amazon interview YouTube

```
[-3, -2, -1, 1]
[-2,  2,  3, 4]
[ 4,  5,  7, 8]
```

PURCHASE AND SALE OF STOCK

maximize profit for ONE purchase and ONE sale

array index is minutes after 9:30 open

```
stock_prices = [10, 7, 5, 8, 11, 9]
get_max_profit(stock_prices)
```

1st note: looking at ~~subarrays~~

not really subarrays, just 2 indexes

i1, i2 = start/end array index

maximize: A[i2] − A[i1]

moving i1 →

[10,7,5,8,11,9] → [−1, 2, 4, 1, −2, 0]

← moving i2

[0, −3, −5, −2, 1, −1]

[1, 5, 4, 3, 8, 2, 20] → [19, 15, 16, 17, 12, 18, 0]

[0, −4, −3, −2, −7, 1, 19]

Correct solution: O(n) and O(1) storage

- maintain min_price and max_profit as we move through the price list
- compare max_profit to current−min_price to check for new max_profit
- update min_price if current is < min_price

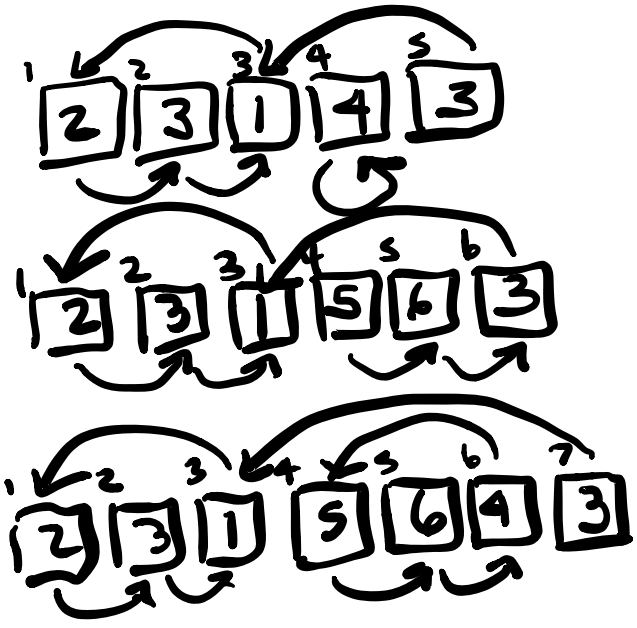
FIND A DUPLICATE
(optimize for space)

Interview Cake
google

list [2, 3, 1, 3]

position 1 2 3 4

linked list



TWO EGG PROBLEM

InterviewCake
google

100 story building (N=100)

F = max floor from which egg can drop without breaking

dropping from $f > F$ causes egg to break

Given 2 eggs, find F.

100 — don't test because $F \leq 100$

so... range to drop eggs is [2, 99]

1 — don't test because $F \geq 1$

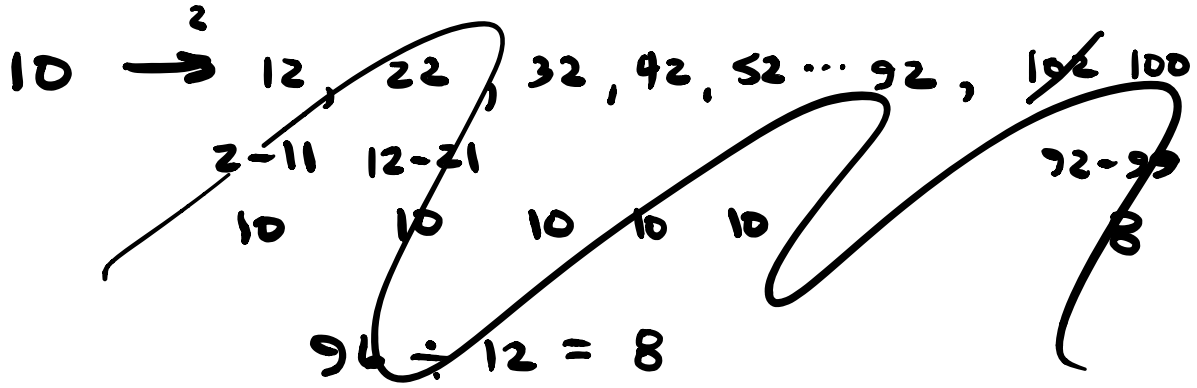
IF first egg breaks, we have to iterate one floor at a time with 2nd egg.

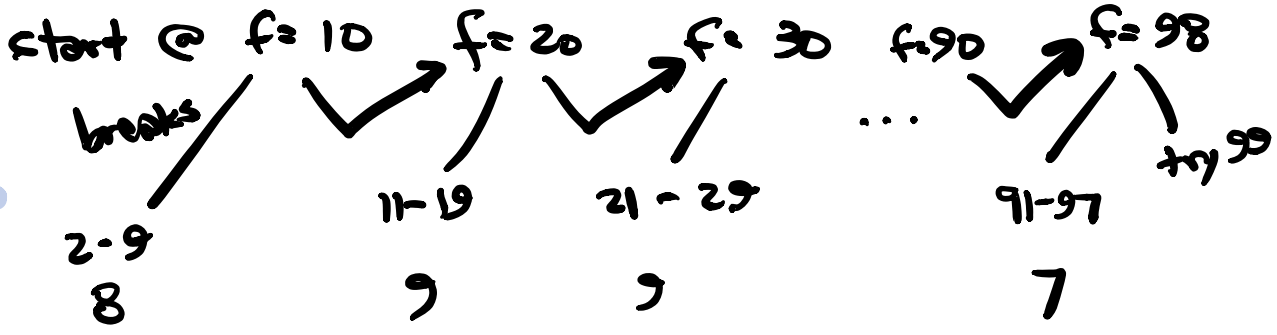
f = 2 : if egg breaks, F = 1

Scratch

[2, 99]

SO... 98 floors possible





max drops = 10 + 9 = 19

---

f = 1 - 100 N = 100 floors

divide this up: 10 20

1-9 11-19 ...

when n=10 : $100/n + (n-1)$

10 + 9 = 19

when n=9 : $100/9 + (9-1) = 11 + 8 = 19$

AND only gets to f=99

when n=11 : $100/11 + (11-1) = 9 + 10 = 19$

when n=

$100/n = n-1$

$100 = n(n-1)$

$100 = n^2 - n$

$100 + n = n^2$

n=10: 110 = 100

n=9: 109 = 81

n=11: 111 = 121

BUT...

After f=10, there are only 90 floors to test...

doesn't add up to 100!

1	2	3	4	5	6	7	8	9
= 10	+ 10	+ 9	+ 9	+ 8	+ 7	+ 6	+ 5	+ 4
max tries: 11	12	12	13	13	13	13	13	13

f= 10 20 30 40 50 ... 100

$90+n \quad n^2 \rightarrow$ 10: 100 100; 9: 99 81; 8: 98 64

$80+n \quad n^2 \rightarrow$ 10: 90 100; 9: 89 81

$70+n \quad n^2 \rightarrow$ 9: 79 81; 8: 78 81

$60+n \quad n^2 \rightarrow$ 9: 69 81; 8: 68 64; 7: 67 49

$50+n \quad n^2 \rightarrow$ 8: 58 64; 7: 57 49

$40+n \quad n^2 \rightarrow$ 8: 48 64; 7: 47 49; 6: 46 36

$30+n \quad n^2 \rightarrow$ 7: 37 49; 6: 36 36

$20+n \quad n^2 \rightarrow$ 6: 26 36; 5: 25 25

$10+n \quad n^2 \rightarrow$ 5: 15 25; 4: 14 16

CAKE THIEF

should be: O(n*k) time
O(k) space
n = # of cakes
K = duffel capacity

kilograms
shillings

cake_tuples = [(7, 160), (3, 90), (2, 15)]
capacity = 20

max_duffel_bag_values (cake_tuples, capacity)

capacity : 1 ⟶ 20

ALL SUBSETS OF A SET

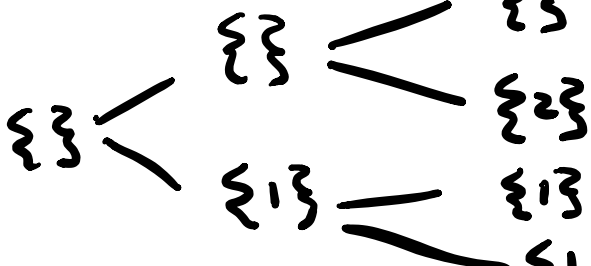
Facebook interview YouTube

[1, 2] → {}, {1, 2}, {1}, {2}

output :

1,
2,
1,2,

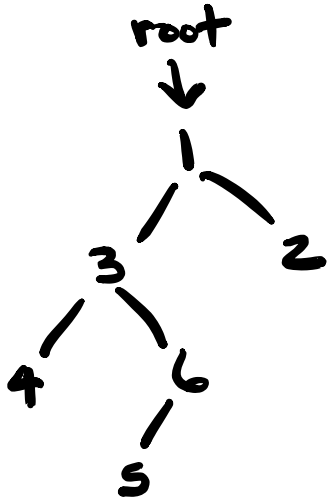
# of subsets = 2^n
= 4 if n = 2



{1, 3}

LOWEST COMMON ANCESTOR

Microsoft interview YouTube



LCA (root, 4, 5)

- find paths to both elements:
  1, 3, 4
  1, 3, 6, 5

function pathToX (root, 5)
(returns a stack)

1
3
6
5