Day 1: Big-O; Search

→ Binary Search

find "19" $[\underbrace{2, 3, 5, \underset{\text{mid}}{7}}_{\text{discard}}, 11, 13, 17, 19]$

7<13

$[\underbrace{11, \underset{\text{mid}}{13}}_{\text{discard}}, 17, 19]$

13<19

$[\underset{\text{mid}}{17}, 19]$

17<19

$[19]$

✓ found 19

pseudocode

```
1. calculate midpoint
           ↪ len(array)
   mid = $\frac{n}{2}$ if n is even
       = $\lfloor \frac{n}{2} \rfloor$ if n is odd
2. if array[mid] == target:
       terminate with "success"
3. if array[mid] < target:
       array <- array[mid+1: ]
4. if array[mid] > target:
       array <- array[:mid]
5. repeat 1 to 4 if
       len(array) > 0
```

Binary Search Patterns:

1) Rotated Sorted arrays:

pivot index (5) wouldn't be known beforehand

e.g. [4 , 5 , 6 , 7 , 8 , 1 , 2]
left — mid — right

target = 1

arr[left] ≤ array[mid]
but target not in the range!

[8 , 1 , 2]
left mid right
arr[mid] < arr[right]
target within this range.

[1 , 2]
mid right
left
arr[mid] = target

Takeaway:

Even when rotated, we know 3 things about a sorted array: leftmost, middle, rightmost elements
And hence we can find a target or pivot by comparisons

```
1. determine whether it's left-
   or right-rotated
2. if array[mid] ≥ array[left]:
3.     if array[left] ≤ target < array[mid]
           right ← mid-1
4.     else:
           left ← mid+1
5. else:
6.     if array[mid] < target ≤ array[right]
           left ← mid+1
7.     else:
           right ← mid-1
if array[mid]==target:
    return target/mid
```

2) Matrices: target = 10

$$\begin{bmatrix} \overset{\text{left}}{1} & 3 & 12 \\ 4 & 7 & 16 \\ 10 & 14 & \underset{\text{right}}{20} \end{bmatrix} \rightarrow \overset{\text{bin search}}{\begin{bmatrix} \boxed{1 \quad 3 \quad 12} \\ 4 \quad 7 \quad \cancel{16} \\ 10 \quad 14 \quad \cancel{20} \end{bmatrix}} \rightarrow \begin{bmatrix} 1 \quad 3 \quad 12 \\ \boxed{4 \quad 7} \quad \cancel{16} \\ 10 \quad 14 \quad \cancel{20} \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 1 & 3 & 12 \\ 4 & 7 & \cancel{16} \\ \boxed{10} & \cancel{14} & \cancel{20} \end{bmatrix}$$

target!

sorted row-wise & column-wise but search in which direction?

```
1. rows=[0,1,2], cols=[0,1,2]
2. for row in rows:
      perform bin search for
         row with cols element
      delete cols element
         where mat[row][col]
           > target.
   if mat[row][0] ≤ target
        ≤ mat[row][col]
```

Takeaway:

Treat each row as a 1D array and perform bin search

Can prune some colums & rows on the go

3) Data Streams:

[1, 3, 7, 2, 6, ...]

we do not know what's next

[ ]

1. 1 -> [1] (count of elements received / len: 1)
2. 3 -> [1, 3] (len: 2)
3. 7 -> [1, 3, 7]
4. 2 -> [1, 2, 3, 7]
5. 6 -> [1, 2, 3, 6, 7]

use binary search to find out where to insert a new element

Q. what's the right data structure for storing such a data stream?

Task: Given a data stream input of non-negative integers a1, a2, ..., an, summarize the numbers in a sorted manner.

4) Counting:

[5, 2, 6, 1] $\xrightarrow{\text{sort}}$ [1, 2, 5, 6]

find index (with what?)

how many element are smaller than this element? → count = index

order / input: [1, 6, 2, 5] <- backwards of the original input

use binary search to find correct index

count how many elements to the left

streams:

a. [ ] + 1	→ [1]	0
b. [1] + 6	→ [1, 6]	1
c. [1, 6] + 2	→ [1, 2, 6]	1
d. [1, 2, 6] + 5	→ [1, 2, 5, 6]	2

Task: You are given an integer array nums and you have to return a new counts array. The counts array has the property where counts[i] is the number of smaller elements to the right of nums[i].

Input: nums = [5,2,6,1] --> Output: [2,1,1,0]

Sorting:

Task: Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same.

[Remove Duplicates In-place]

Input: [1, 1, 2, 3, 4, 4, 5]

Expected Output: [1, 2, 3, 4, 5, _, _]

_ , _ can have any value

1) Duplicates

iter #1
```
i =  0 , j =  1
nums =  [1, 1, 2, 3, 4, 4, 5]
i =  0 , j =  1
nums =  [1, 1, 2, 3, 4, 4, 5]
```

iter #2
```
i =  0 , j =  2
nums =  [1, 1, 2, 3, 4, 4, 5]
i =  1 , j =  2
nums =  [1, 2, 2, 3, 4, 4, 5]
```

iter #3
```
i =  1 , j =  3
nums =  [1, 2, 2, 3, 4, 4, 5]
i =  2 , j =  3
nums =  [1, 2, 3, 3, 4, 4, 5]
```

iter #4
```
i =  2 , j =  4
nums =  [1, 2, 3, 3, 4, 4, 5]
i =  3 , j =  4
nums =  [1, 2, 3, 4, 4, 4, 5]
```

iter #
```
i =  3 , j =  5
nums =  [1, 2, 3, 4, 4, 4, 5]
i =  3 , j =  5
nums =  [1, 2, 3, 4, 4, 4, 5]
```

iter #
```
i =  3 , j =  6
nums =  [1, 2, 3, 4, 4, 4, 5]
i =  4 , j =  6
nums =  [1, 2, 3, 4, 5, 4, 5]
```

logic:

```
i = 0

for j in range(1, len(nums)):
    if nums[j] != nums[i]:
        i += 1
        nums[i] = nums[j]
```

Takeaways:

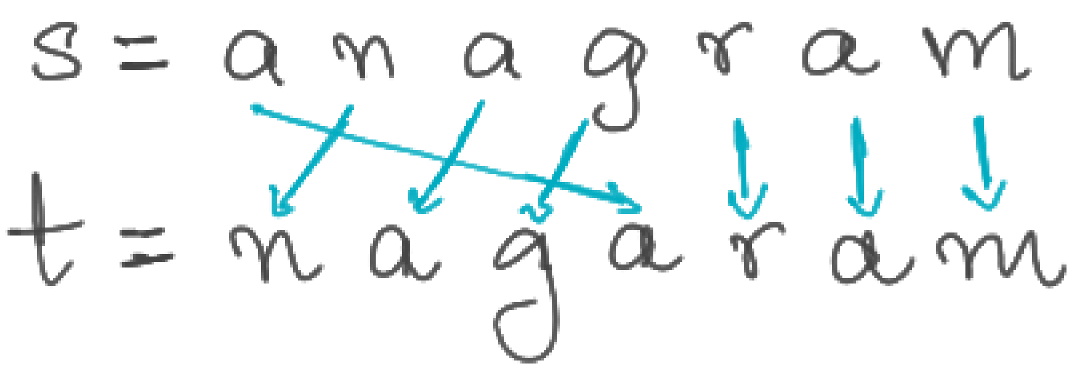
a. for in-place array operations, consider using more than one pointer.

b. compare and swap

② Anagrams:

Task: Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Input: s = "anagram", t = "nagaram" --> Output: true



s = a n a g r a m

t = n a g a r a m

different approaches:

1. use dictionary to keep char occurence counts
2. sort both strings and compare
3. remove a char from s when a char from t matches

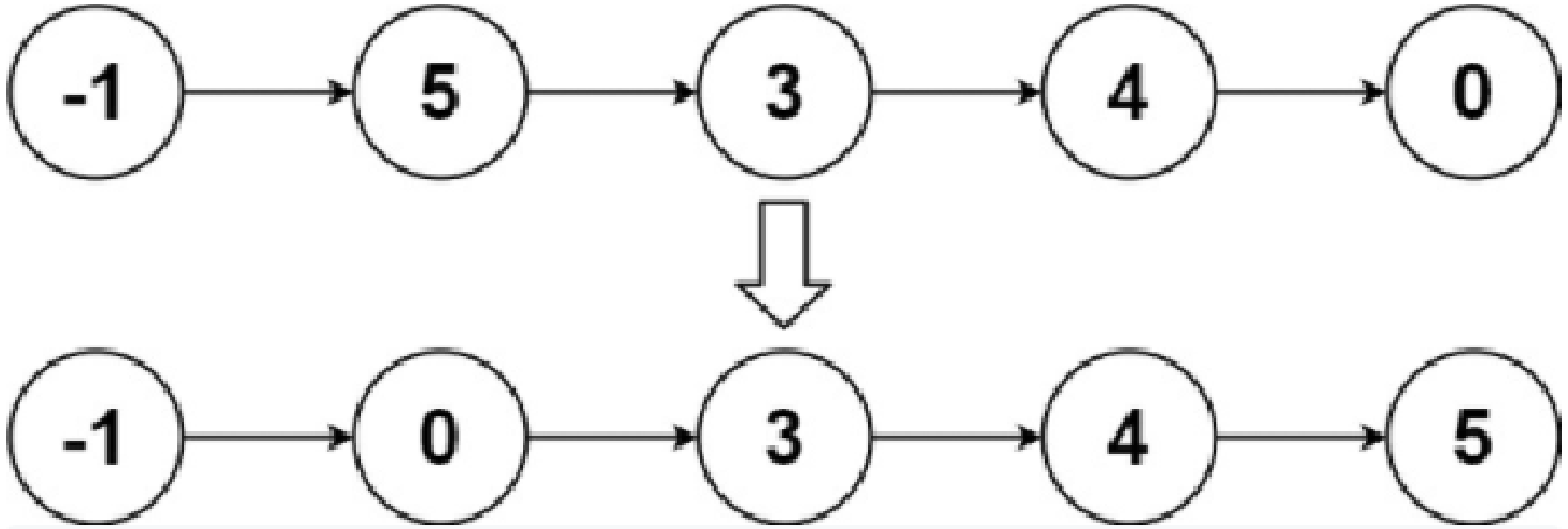
s_sorted = "aaagmnr"

t_sorted = "aaagmnr"

③ Linked Lists

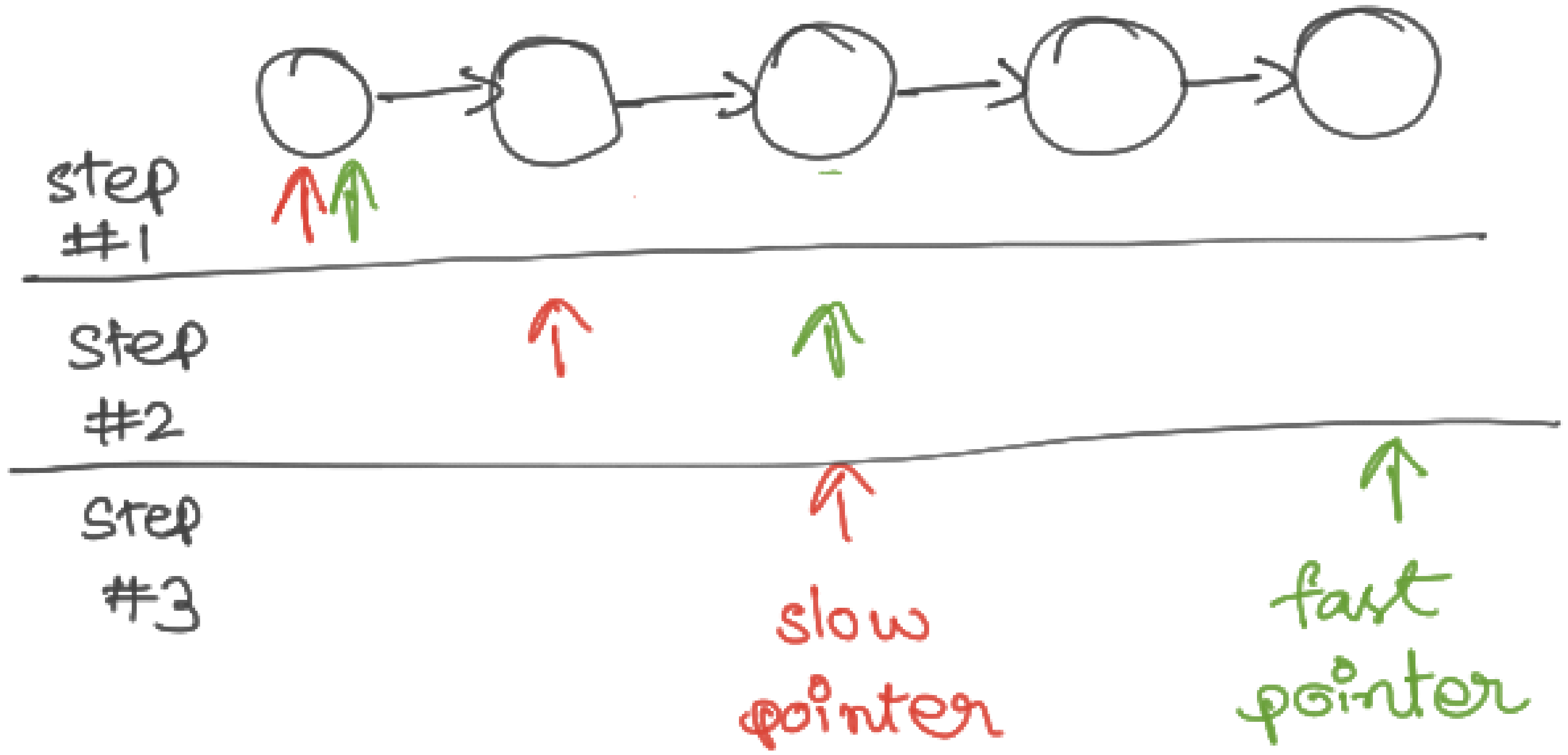
- Some of the "random access" assumptions for sorting an array wouldn't work here.

Task: Can you sort the linked list in O(n logn) time and O(1) memory (i.e. constant space)?



```
Input: head = [-1,5,3,4,0]
Output: [-1,0,3,4,5]
```

How do we find the middle element?



4 Inplace Sorting:

We cannot return a new array; although, you can use temporary variables.

which sorting algorithms are in-place?

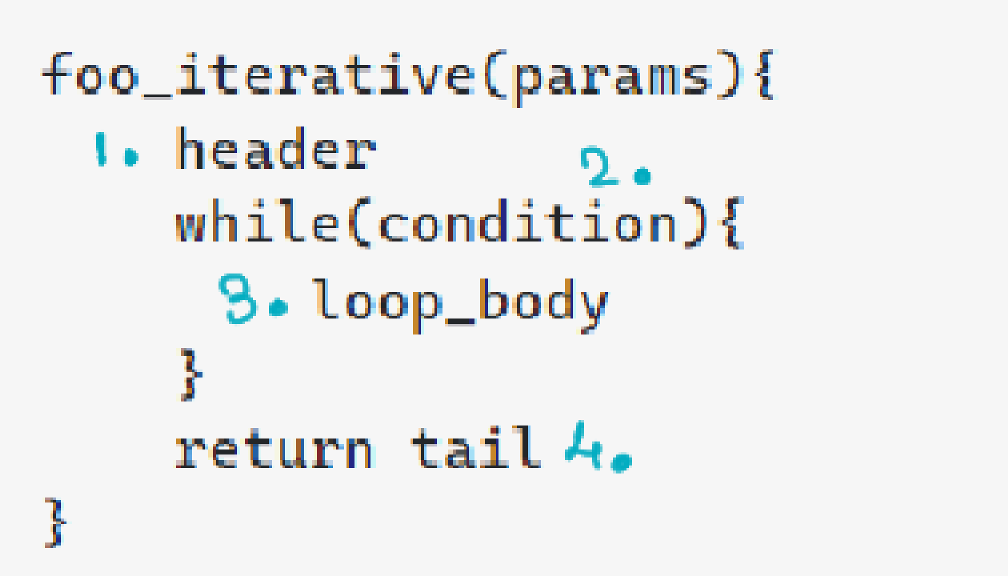
Also remember: Stable algorithms

					from 0x03	from 0x04	from 0x01	from 0x02
	[20	30	10	10] Sort →	[10	10	20	30]
Addresses:	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08

Day 3 : Recursion

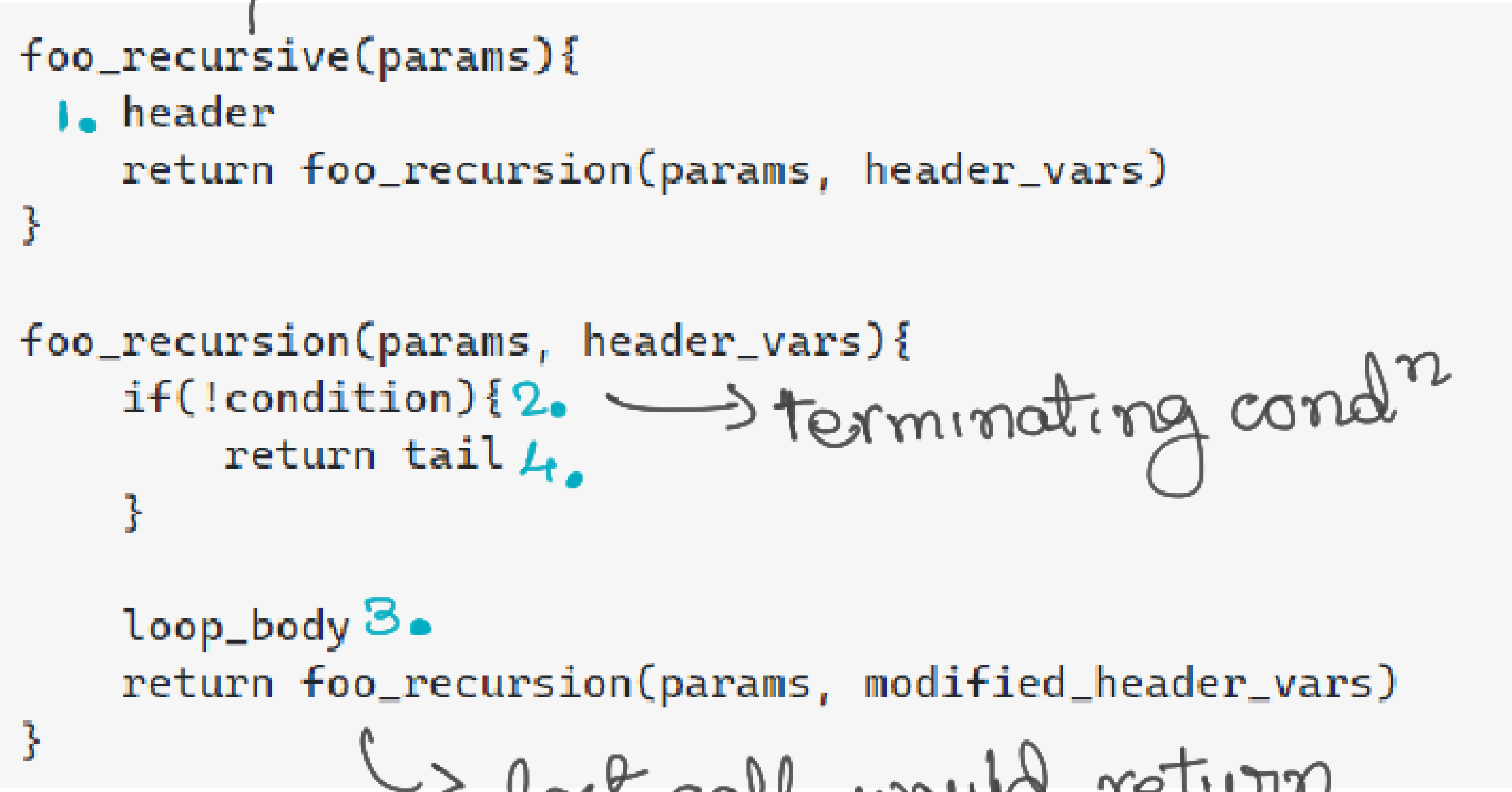
① Iterations ⟶ Recursions

4 parts



```
foo_iterative(params){
  1. header          2.
    while(condition){
     3. loop_body
    }
    return tail 4.
}
```

"init" function



```
foo_recursive(params){
 1. header
    return foo_recursion(params, header_vars)
}

foo_recursion(params, header_vars){
    if(!condition){2.
        return tail 4.
    }

    loop_body 3.
    return foo_recursion(params, modified_header_vars)
}
```

⟶ terminating cond^n

↳ last call would return tail too

4 Ordering:

s1 = "car"

s2 = "race" permutation?

more about memoization

s1_dict = {c: 1, a: 1, r: 1}

s2_dict = {r: 1, a: 1, c: 1, e: 1} is subset ✓

5 Divide & Conquer

See Merge Sort

② Subproblems:

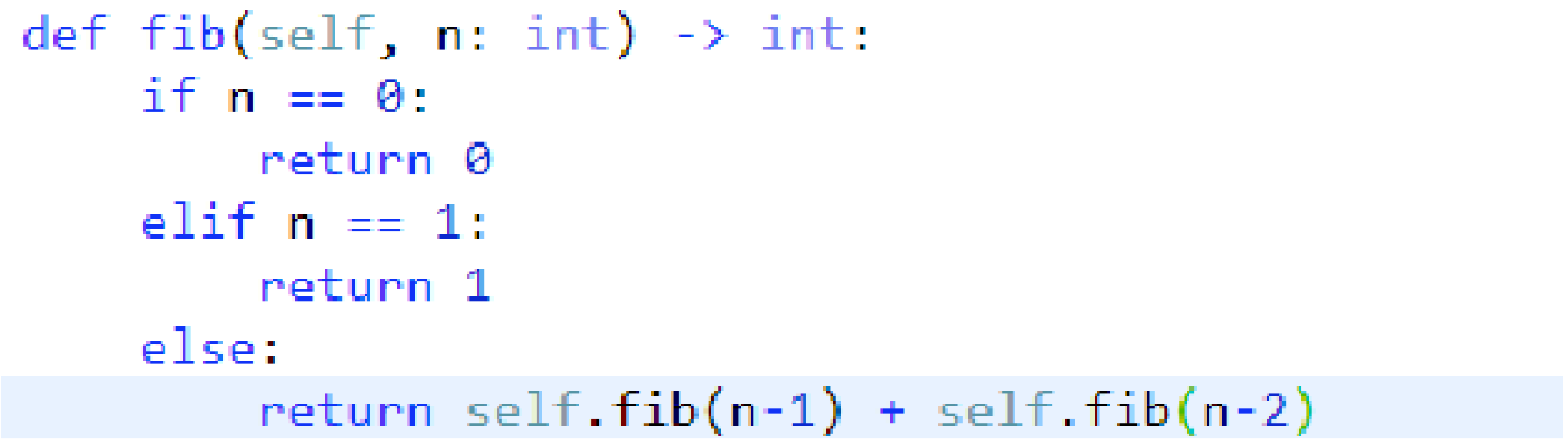
Fibonacci Numbers:

0, 1, 2, 3, 5, 8, 13, ...

$$Fib(0) = 0$$

$$Fib(1) = 1$$

$$Fib(n) = Fib(n-1) + Fib(n-2) \quad \text{if } n > 1$$



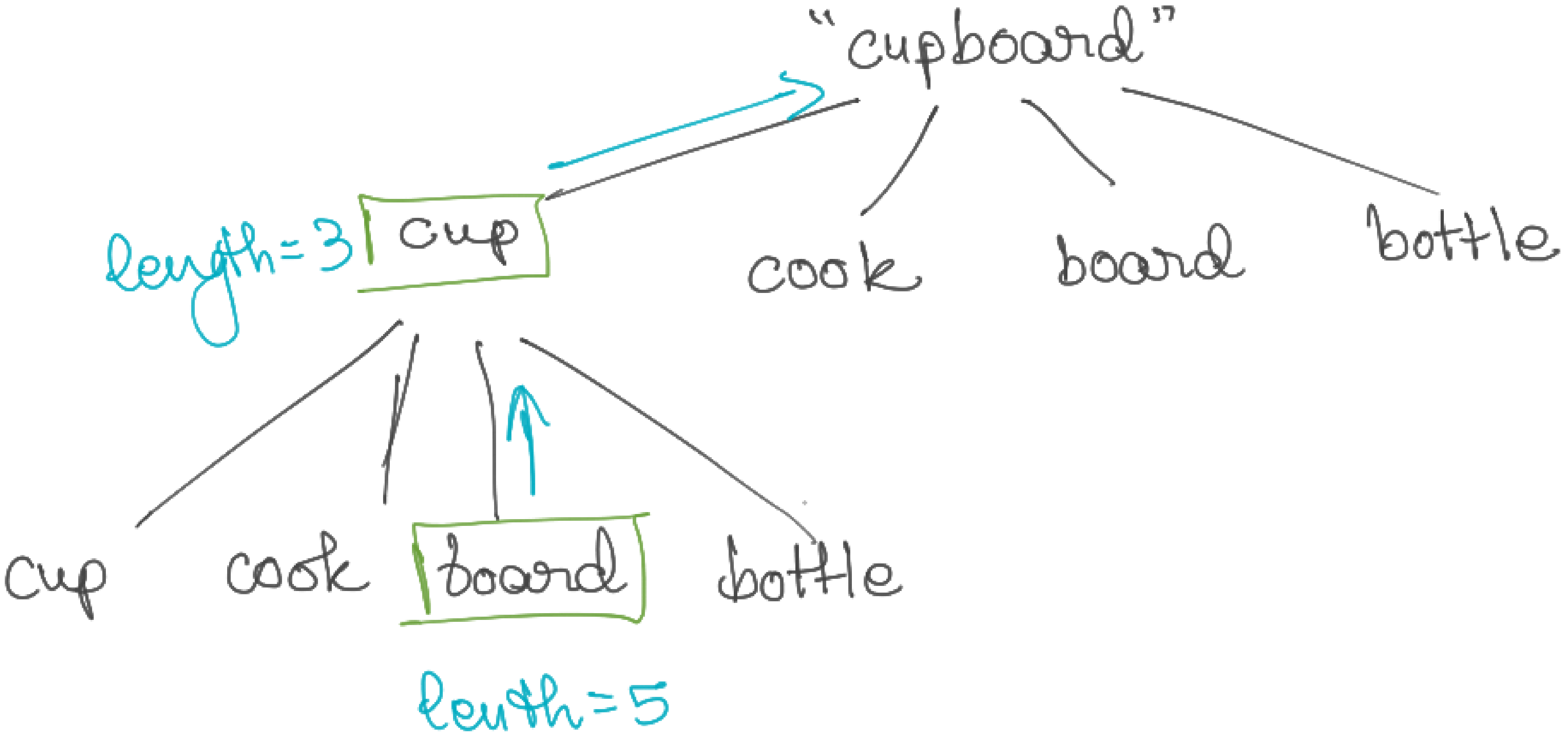
```python
def fib(self, n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return self.fib(n-1) + self.fib(n-2)
```

but... we are calculating this every time!

Memoization introduces space complexity.

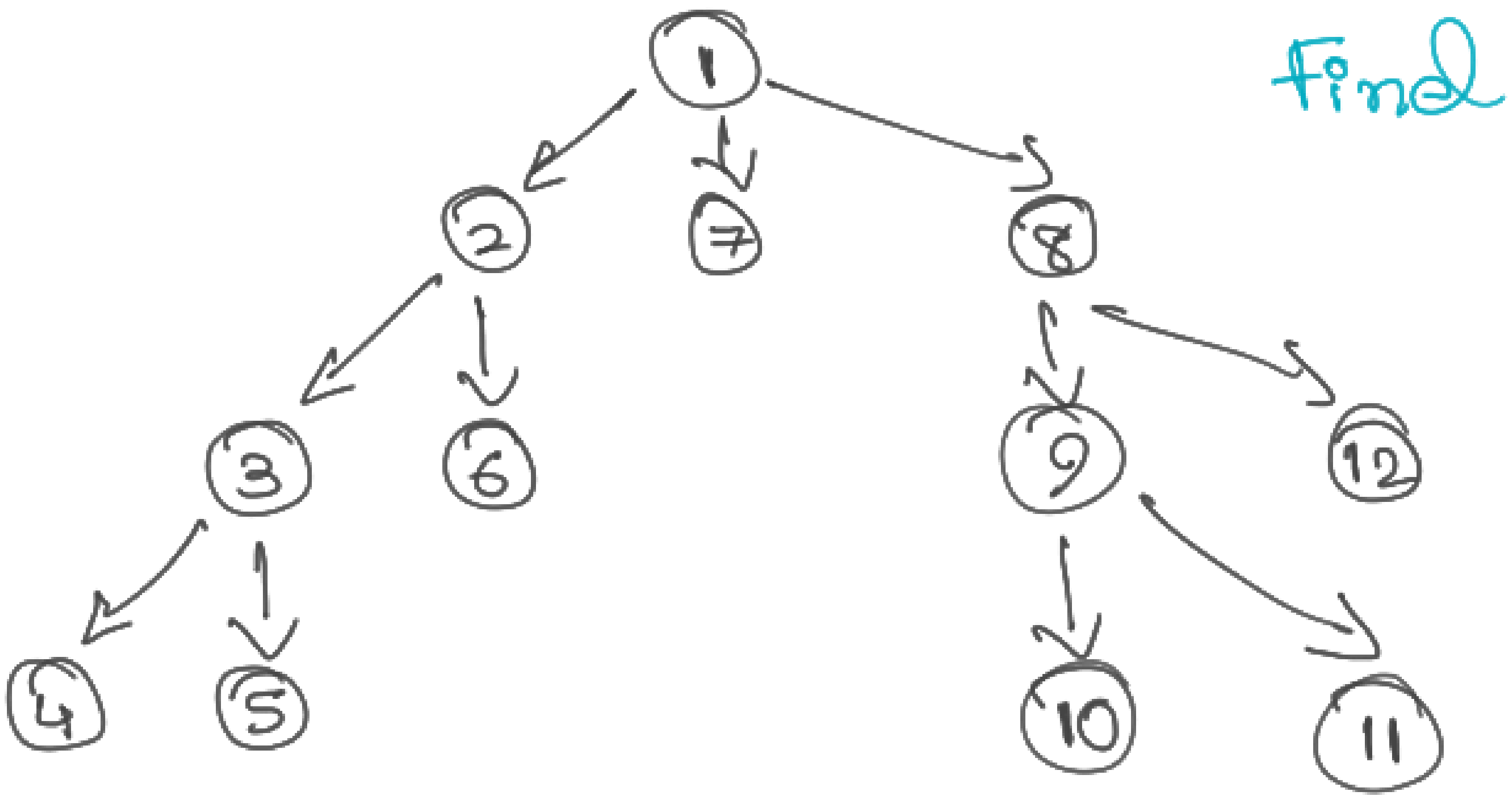
③ Backtracking

dict = {cup, cook, board, bottle}



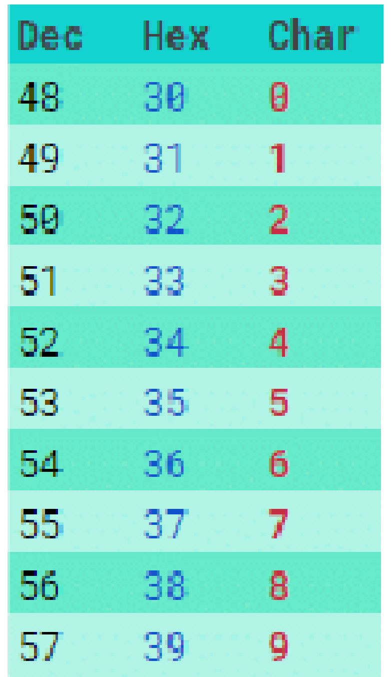
memo = {
0: True
1: False
2: False
3: True
4: False
5: False
6: false
7: false
}

⑥ Depth First Search : Recursion + Backtracking

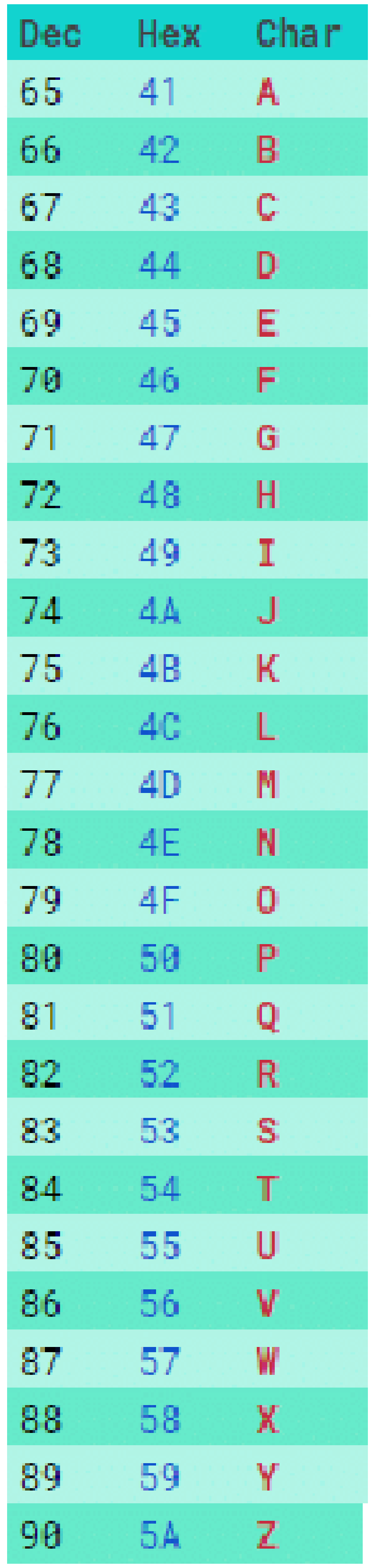


Find "12"

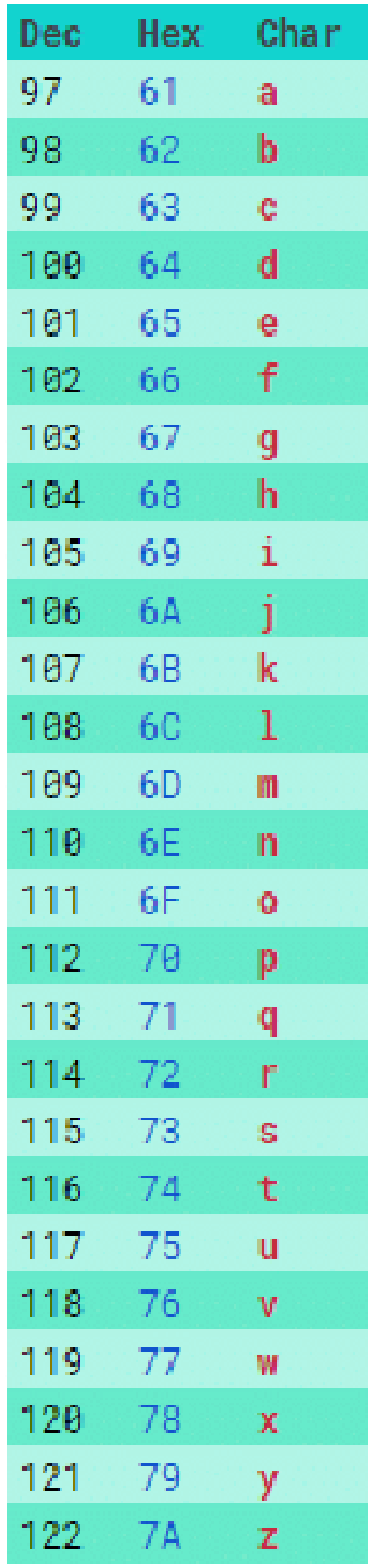
Day 4 : Strings



Dec	Hex	Char
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9



Dec	Hex	Char
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z



Dec	Hex	Char
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z

int(num, base)

char(num)

ord (char)

str (num)

b' . . . '

hex(num)

④ Two pointers:

"the sky is blue"

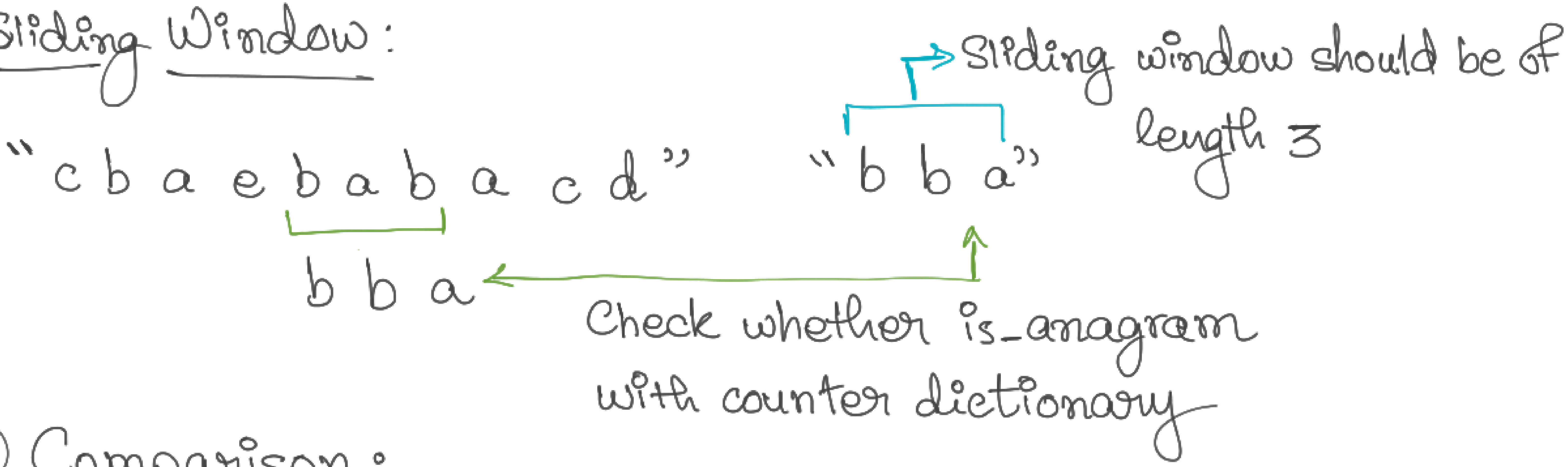
dealing with word ?
convert them into a
list of strings

["the", "sky", "is", "blue"]

↑ left ptr

↑ right ptr

⑤ Sliding Window:



"c b a e b a b a c d"

b b a

"b b a"

Sliding window should be of length 3

Check whether is_anagram with counter dictionary

⑥ Comparison:

Longest Common Subsequence

	a	b	e
a	✓		
b		✓	
c			
d			
e			✓

→

	a	b	e
a	1	1	1
b	1	2	2
c	1	2	2
d	1	2	2
e	1	2	3

```
if c1 == c2:
    tbl[i+1][j+1] = tbl[i][j]
                          +1
else:
    tbl[i+1][j+1] =
    max(tbl[i+1][j], tbl[i][j+1])
```

Day 5 : Hash Tables

① Numbers

Task -

You are given an integer array nums with the following properties:

* nums.length == 2 * n.
* nums contains n + 1 unique elements.
* Exactly one element of nums is repeated n times.

conclude → If number x is getting repeated, it will be getting repeated $x-1$ times.

Return the element that is repeated n times.

Input: nums = [2,1,2,5,3,2]
Output: 2

e.g. [2, 1, 2, 5, 3, 2]

hash table = {2 : 1}

{2 : 1, 1 : 1}

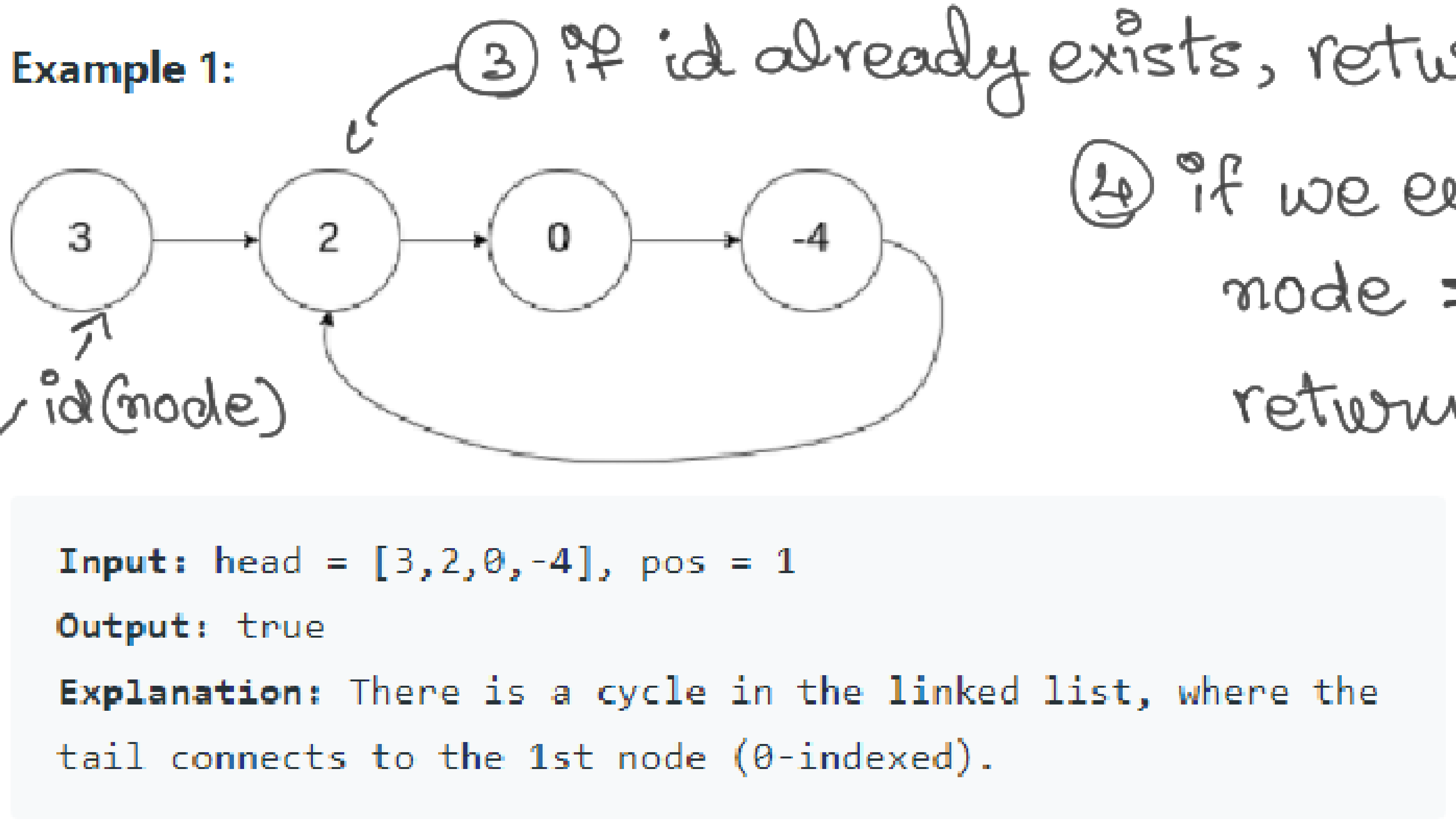
{2 : 2, 1 : 1}

!!!

② Linked Lists:

Task:

Given head, the head of a linked list, determine if the linked list has a cycle in it.



Example 1:

Input: head = [3,2,0,-4], pos = 1

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

① iterate through all the nodes

② store their id's / pointers in a list

id(node)

③ if id already exists, return true

④ if we encounter node == None; return false

③ Arrays :

Task:

Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must be unique and you may return the result in any order.

Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]

num1 = [1, 2, 2, 1]

num2 = [2, 2]

a. dictionary from num1 = {1 : 1} $\xrightarrow{\text{next}}$ {1 : 1, 2 : 1} $\xrightarrow{\text{next}}$ {1 : 1, 2 : 2} $\xrightarrow{\text{next}}$ {1 : 2, 2 : 2}

b. remove from dictionary with num2 = {1 : 2, 2 : 2} $\xrightarrow{\text{next}}$ {1 : 2, 2 : 1} $\xrightarrow{\text{next}}$ {1 : 2, 2 : 0}