# PYTHON PROGRAMMING
## GATE DA/DSA

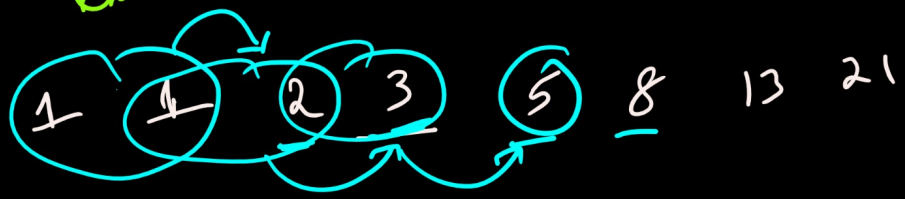
Agenda :

→ Recursion
- fibonacci series
- Gate DA 2024 problems
- few more problems

→ Advanced Functions
* Lambda function
* Filter function
* Map ''

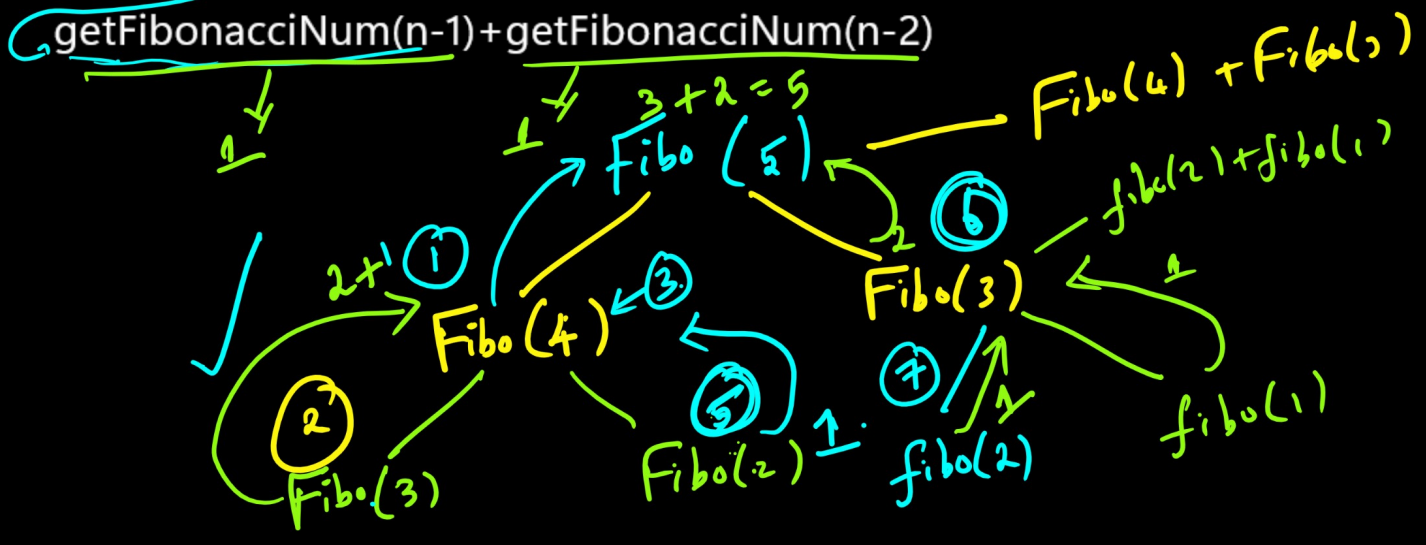
---

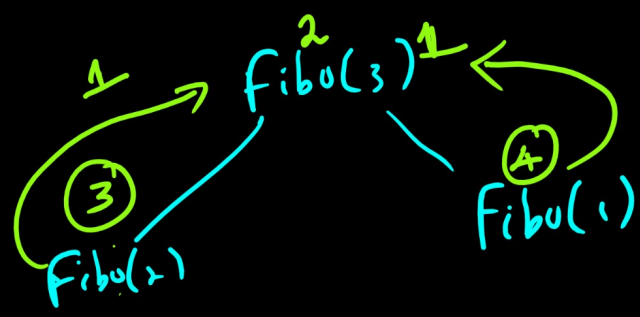
Fibonacci Series :

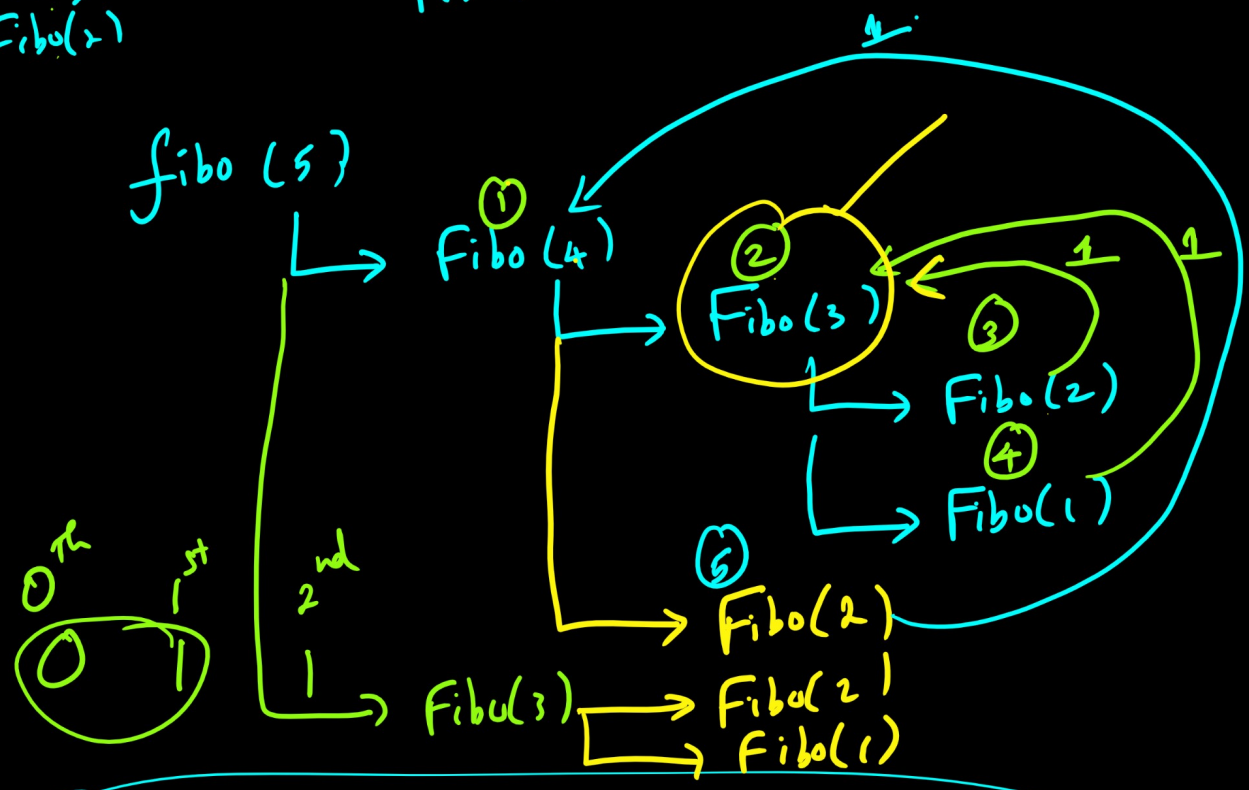



```
def getFibonacciNum(n):
  if n==1 or n==2:
    return 1
  return
getFibonacciNum(n-1)+getFibonacciNum(n-2)
```

base condition





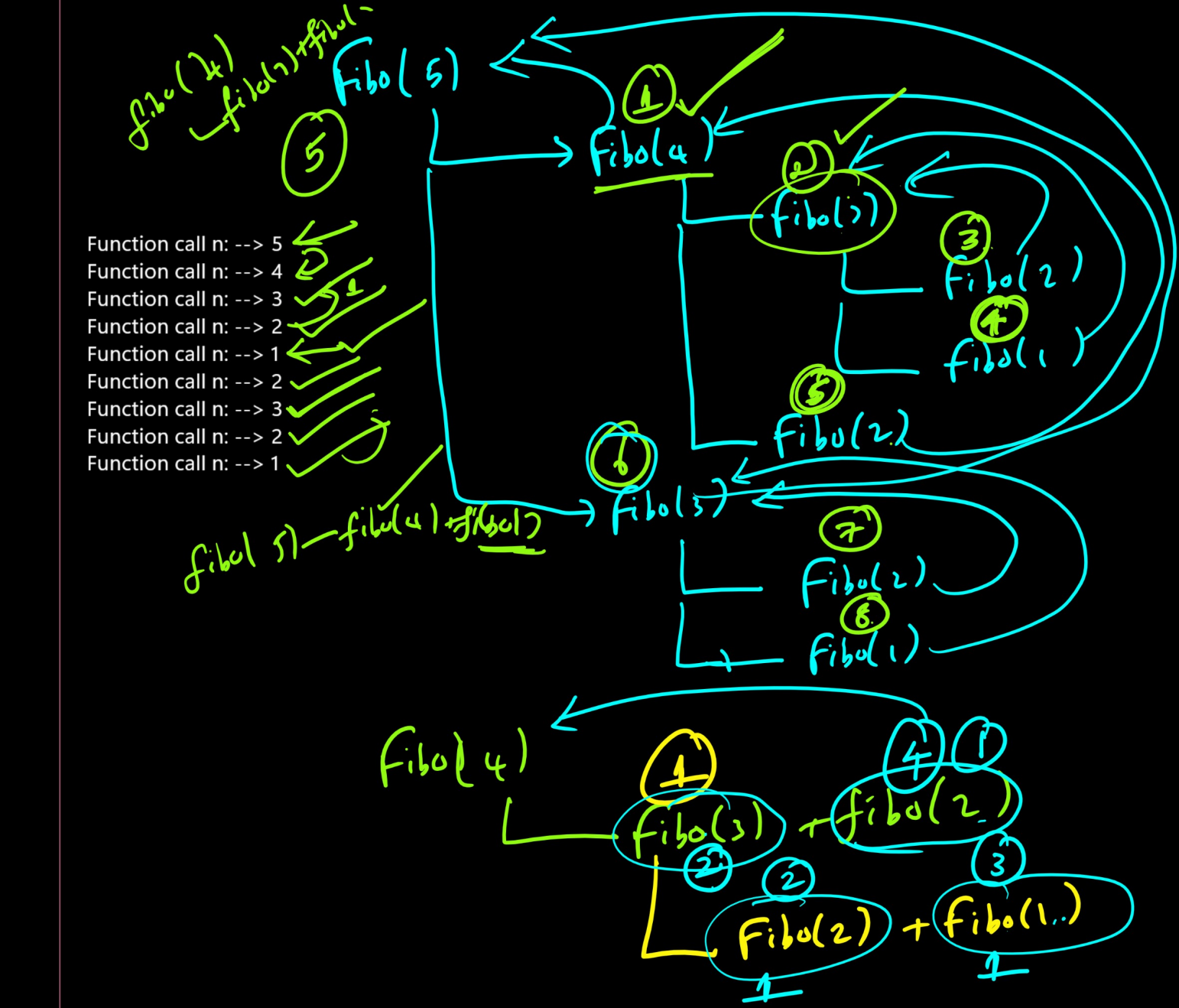

```
def fibo(n):
    if n==0:
        return 0
    dif n==1:
        return 1
    return fibo(n-1) + fibo(n-2)
```

n<=1: return n

fibo(1) → 1

fibo(0) → 0

1+0 = 1







GATE 2024:

child_dict , 0

function.

```
def count(child_dict, i):
    if i not in child_dict.keys():
        return 1
    ans = 1
    for j in child_dict[i]:
        ans += count(child_dict, j)
    return ans

child_dict = dict()
child_dict[0] = [1,2]
child_dict[1] = [3,4,5]
child_dict[2] = [6,7,8]
```

```
print(count(child_dict,0))
```

child_dict = { } ⟹ child_dict

{ 0: [1,2], 1: [3,4,5], 2: [6,7,8] }

1250

child_dict = { 0: [1,2], 1: [3,4,5], 2: [6,7,8] }

print ( count ( child_dict , 0 ) )

dict , 0 — 0 , 1 , 2

child_dict, 0

```
def count(child_dict, i):
 if i not in child_dict.keys():
  return 1
 ans = 1
 for j in child_dict[i]:
  ans += count(child_dict, j)
 return ans
```

ans = 1

ans = count dict (child_dict, 3) child_dict [0] = [1,2]

ans += count (child_dict, 1)

1 + 1 = 2 + 1 = 3 + 1 = 4

count (child_dict, 1)

0 , 1 , 2

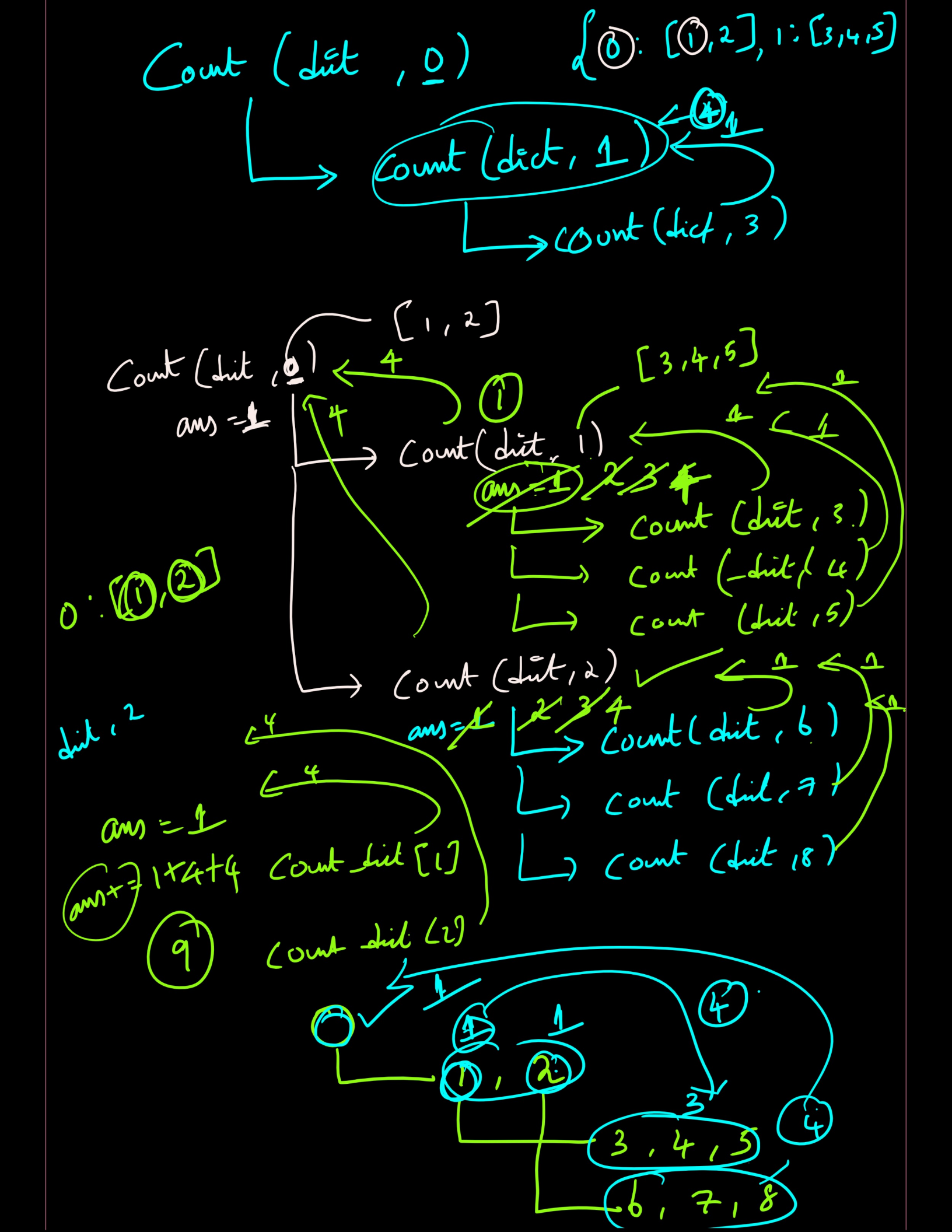
[3,4,5]

ans = 1

for j in child_dict [1]:

ans += count (child_dict

ans += count (child, 3)

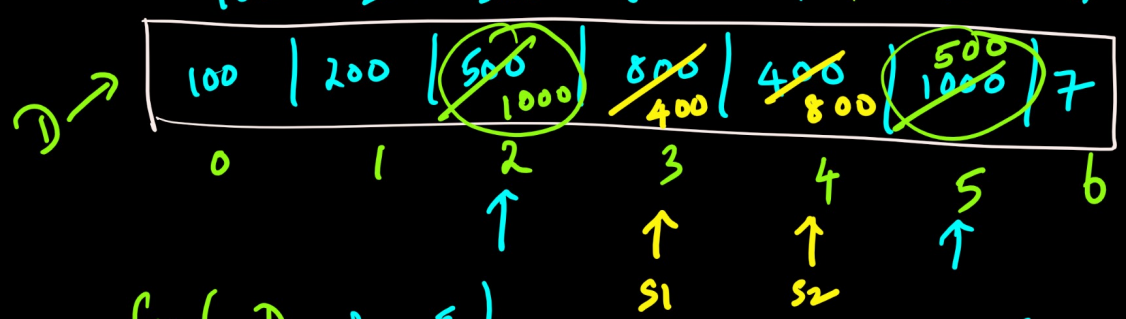

Count (dict, 0)
{0: [1,2], 1: [3,4,5]
Count (dict, 1)
Count (dict, 3)
[1, 2]
Count (dict, 0)
ans = 1
Count (dict, 1)
[3,4,5]
Count (dict, 3)
Count (dict, 4)
Count (dict, 5)
0: [1,2]
Count (dict, 2)
Count (dict, 6)
Count (dict, 7)
Count (dict, 8)
dict, 2
ans = 1
ans += 1+4+4
Count dict [1]
Count dict [2]
1,2
3,4,5
6,7,8

| | |
|---|---|
| (A) | It finds the smallest element in D from index s1 to s2, both inclusive. |
| (B) | It performs a merge sort in-place on this list D between indices s1 and s2, both inclusive. |
| (C) | It reverses the list D between indices s1 and s2, both inclusive. |
| (D) | It swaps the elements in D at indices s1 and s2, and leaves the remaining elements unchanged. |

```
def fun(D, s1, s2):
 if s1 < s2:
  D[s1], D[s2] = D[s2], D[s1]
  fun(D, s1+1, s2-1)
```

D, s1, s2 ← indices.

D[s1], D[s2] = D[s2], D[s1]




fun(D, 2, 5)

fun(D, s1=2, s2=5)  s1 < s2

D

↳ fun(D, s1+1, s2-1)

fun(D, 3, 4)

↳ fun(D, s1+1, s2-1)

fun(D, 4, 3)

---

```
def recurfun(a, b, c):
    if a >= b and c < b:
        return b
    elif a >= b:
        return recurfun(a, c, b)
    else:
        return recurfun(b, a, c)
```

recurfun(12, 8, 16)

8 12 16

8.28

a) Finds max of a, b, c

b) Finds min of a, b, c

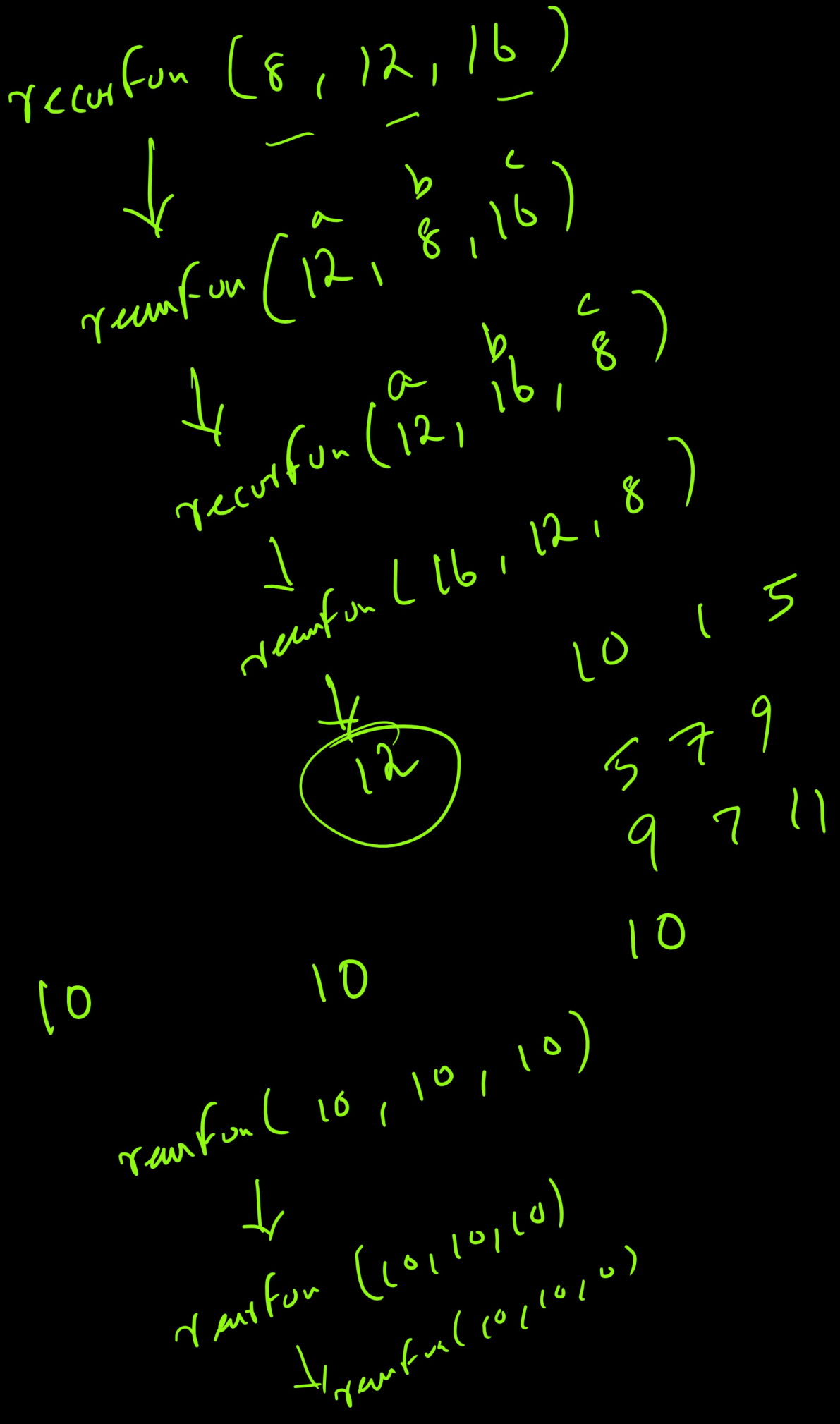
c) Finds the middle number of a, b, c

d) ✓ None of these (

a, b, c ← numbers

a <= b , b > c

| a | b | c | FR |
|---|---|---|---|
| 10 | 5 | 3 | 5 ✓ |
| 8 | 12 | 16 | 12 ✓ |


recurFun (8, 12, 16)
recurFun (12, 8, 16)
recurFun (12, 16, 8)
recurFun (16, 12, 8)
12
10 | 5
5 7 9
9 7 11
10
10
10
recurFun (10, 10, 10)
recurFun (10, 10, 10)
recurFun (10, 10, 10)

Consider the following program:

```
def myFun(n, sum):
    k = j = 0
    if n > 0:
        k = n % 10
        j = n // 10
        sum = sum + k
        myFun(j, sum)
    print(k)

a = 2048
Sum = 0
myFun(a, sum)
print(sum)
```

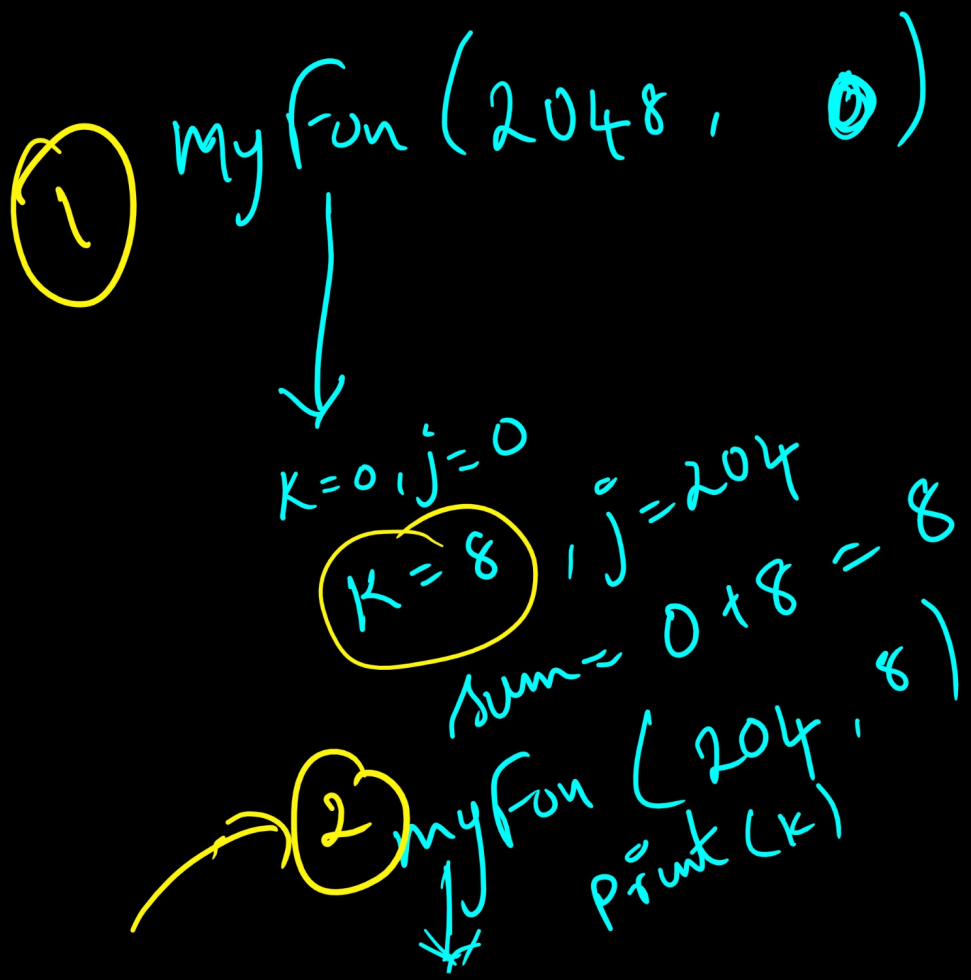
a) 8 4 0 2 14

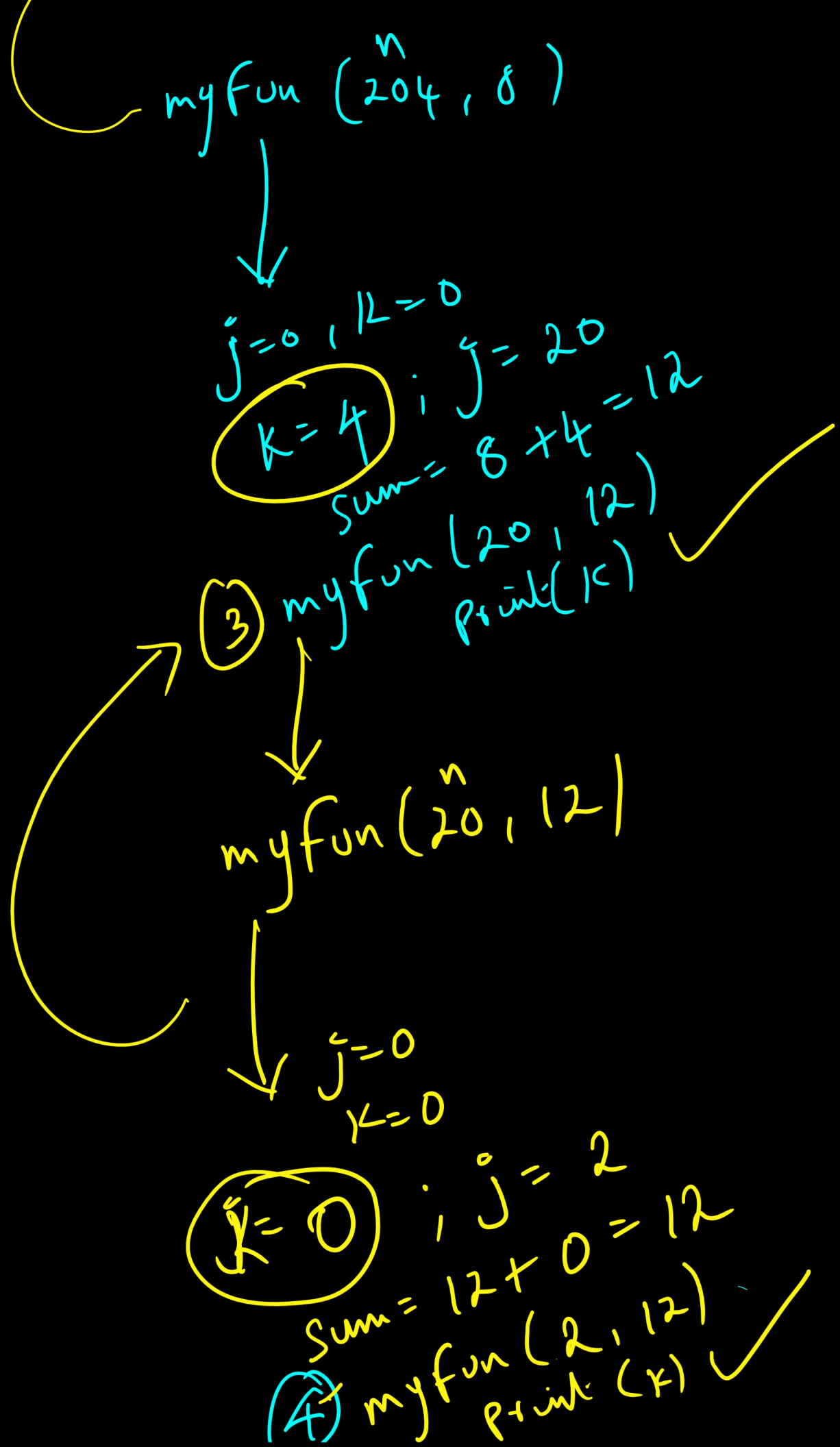
b) 2 0 4 8 0

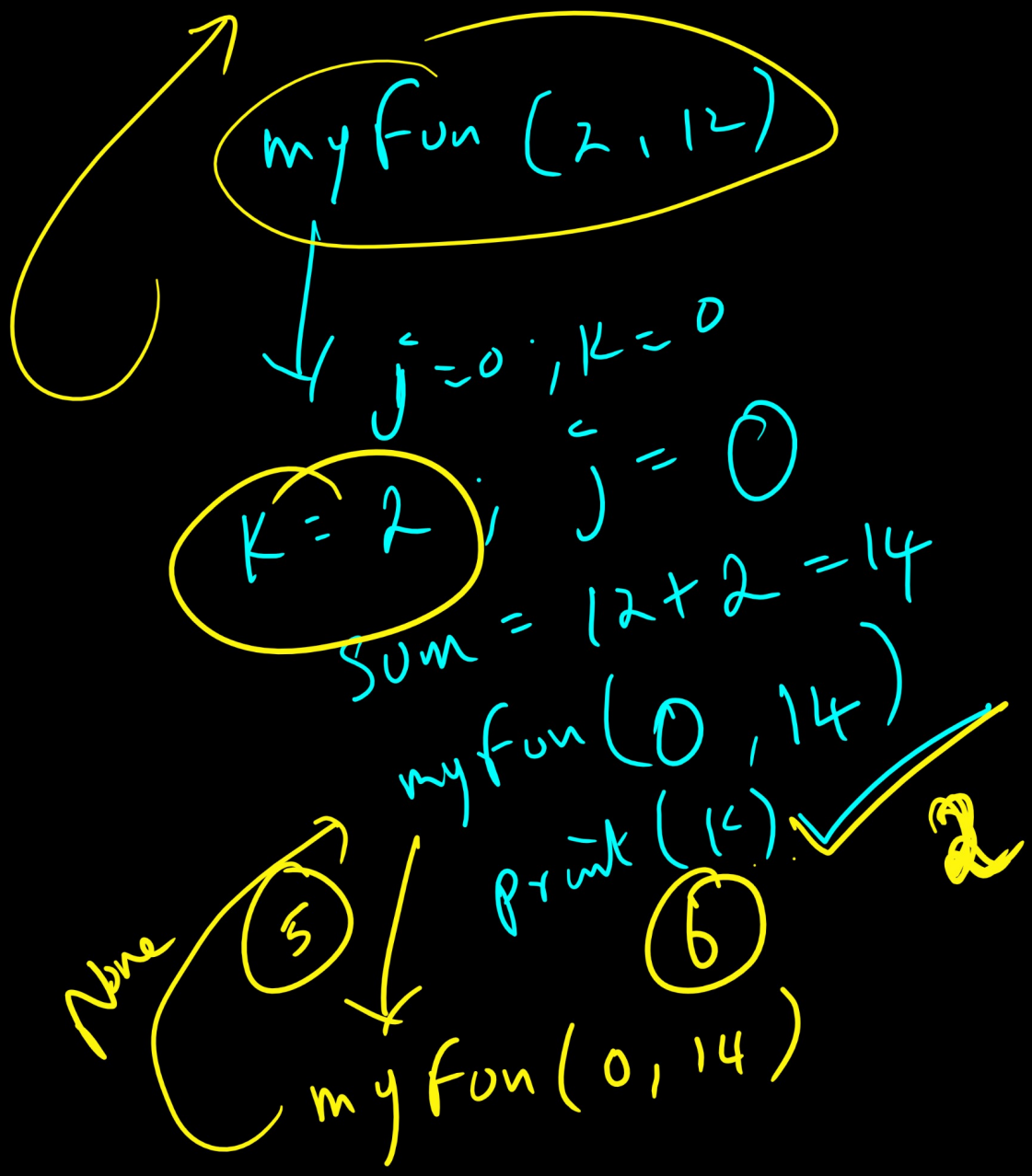
c) 8 4 0 2 0

d) 2 0 4 8 14

---




myFun (204, 8)
n
j=0, k=0
k=4 ; j=20
sum = 8+4 = 12
③ myfun(20, 12)
Print(k)
myfun(20, 12)
n
j=0
k=0
k=0 ; j=2
sum = 12+0 = 12
④ myfun(2, 12)
Print(k)


myFun(2, 12)
j=0, k=0
k=2; j=0
sum = 12+2 = 14
myFun(0, 14)
print(14)
2
None
5
6
myFun(0, 14)

2 0 4 8 0