DSML : Python Intermediate.

Recursion - 1

Recap: (a) Python syntax.
(b) Algorithms: Searching and Sorting.
(c) Time Complexity
(d) Big - Oh notation.

Today: (a) Recursion : Repeating computation.
(b) Steps : Assumption, Main logic, Base condition.
(c) Dry run : Function stack.
(d) Problem solving ; Summation, factorial, fibonacci, sorting.

(a) Recursion :

(Assumption). 1] Bigger dolls will contain smaller dolls.

(Subproblem) 2] Design of 4 dolls are the same. We can open the bigger doll from the middle to find a smaller doll inside.

(Base case.) 3] We cannot open any more dolls once we get a doll with a blue base.

(b) Recursion: Print all numbers from n to 1 in the descending order.

Let the name of the function which does this be called `print_n(n)`.

→ Assumption: (this is about the function we are implementing). print_n(k) will print all numbers from k to 1 in descending order.

→ Subproblem. (use the assumption to solve for smaller n) - I know how to print n. I will print n, and use print_n(n-1) to print the remaining.

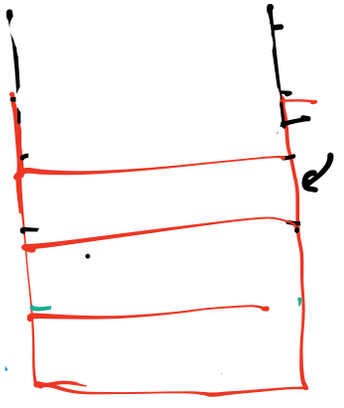
Base case: Handle corner cases.

I will stop printing when n == 0.

(c) Dry run: The function stack.

```
1] def print_n(n):
2]     if n == 0:
3]         return
4]     print(n)
5]     print_n(n-1)
```

>> print_n(5) ↵



> Function stack.

screen

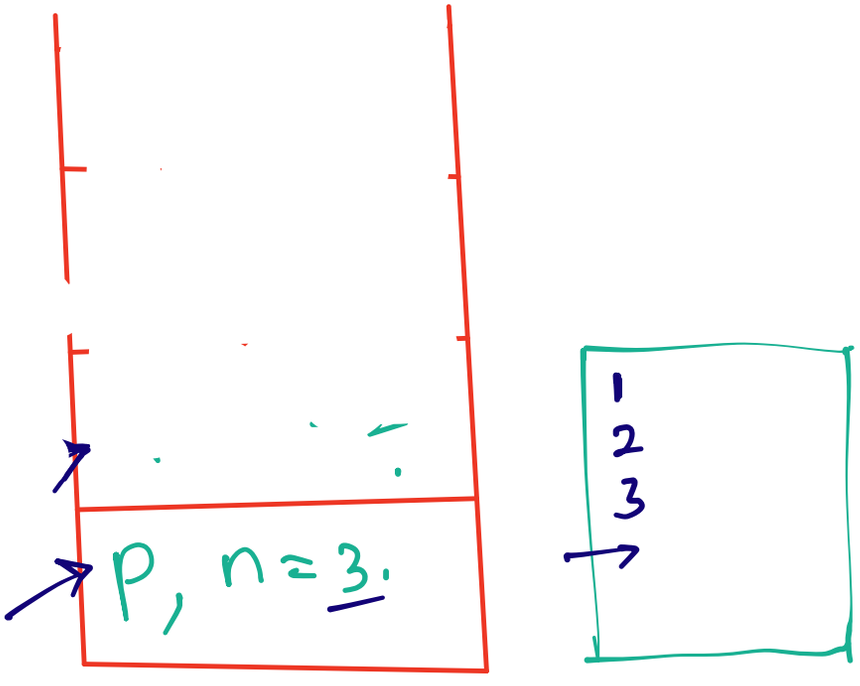
5
4
3
2
1

op:

line	Command Executed
2	5 == 0.
	False.
4	print(5)
2	4 == 0.
	False.
4	print(4)
2	3 == 0:
	F
4	print(3)
2	2 == 0
4	print(2)
2	1 == 0 F
4	print(1)
2	0 == 0 → T.

```
1  def print_n(n):
2      if n == 0:
3          return None
4      print_n(n-1)
5      print(n)
```

>> print_n(3)



Dry run:

line number. Command ex.

$\frac{n(n+1)}{2}$

(d) [Summation] Print the sum of all integers from 1 to n.

Let the name of the function which does this be summation(n).

Assumption: The function summation(k) will give me the correct summation from 1 to k. $1+2+3+\cdots+k$.

Subproblem: Let us say I have summation(n-1). Then, I will return:

summation(n-1) + n

Base Case → n == 1, return 1

(e) (Factorial) Print the product of all numbers from 1 to n.

Let the name of the function which does this be factorial(n).

Assumption factorial(k) gives me $1 \cdot 2 \cdot 3 \cdots k$ correctly.

Subproblem If I know factorial(n-1), I will return:

return. n * factorial(n-1)

Base Cases

if n == 2: ✓
return 2.

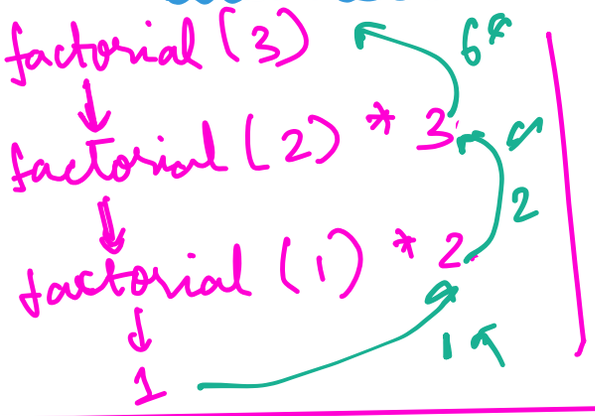
① n == 1
② n == 0
③ n == 2.

0! = 1

(f) Factorial : Tree structure analysis.

Ver.1

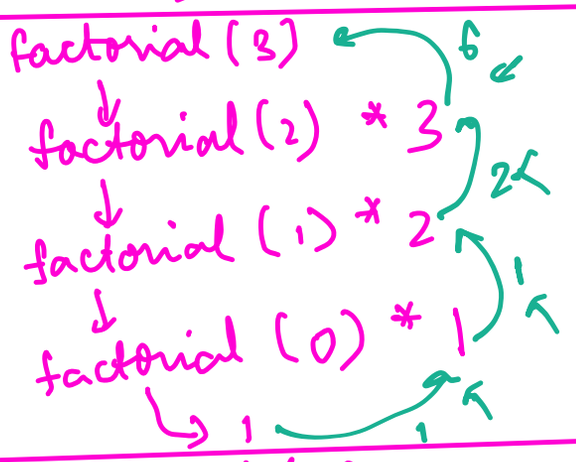
n == 1
↓
1



factorial (0)
↓
∞ loop.

Ver.2

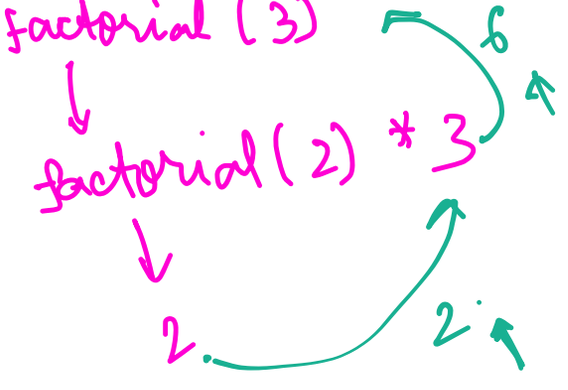
n == 0
↓
1



factorial (0), factorial (1)
↓
correct answer.

✓Ver.3

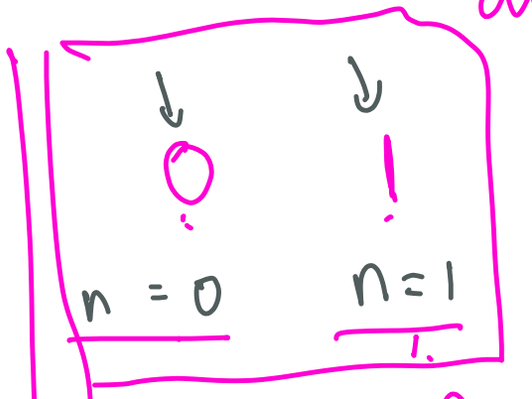
n == 2
↓
2.



factorial (1)
↓
∞ loop.

(9) (Fibonacci Sequence) Print the n^{th} term in the fibonacci sequence.

Let the name of the function that does this be fib(n).



1 2 3 5 8 13 21

n=2 n=3 n=4 n=5 n=6 n=7 n=8.

→ fib(n) = fib(n-1) + fib(n-2).

* **Assumption**

fib(k) and fib(k-1) will give me the correct fibonacci numbers.

Subproblem

Assume 9 know fib(n-1) and fib(n-2)

9 will give fib(n) as follows:

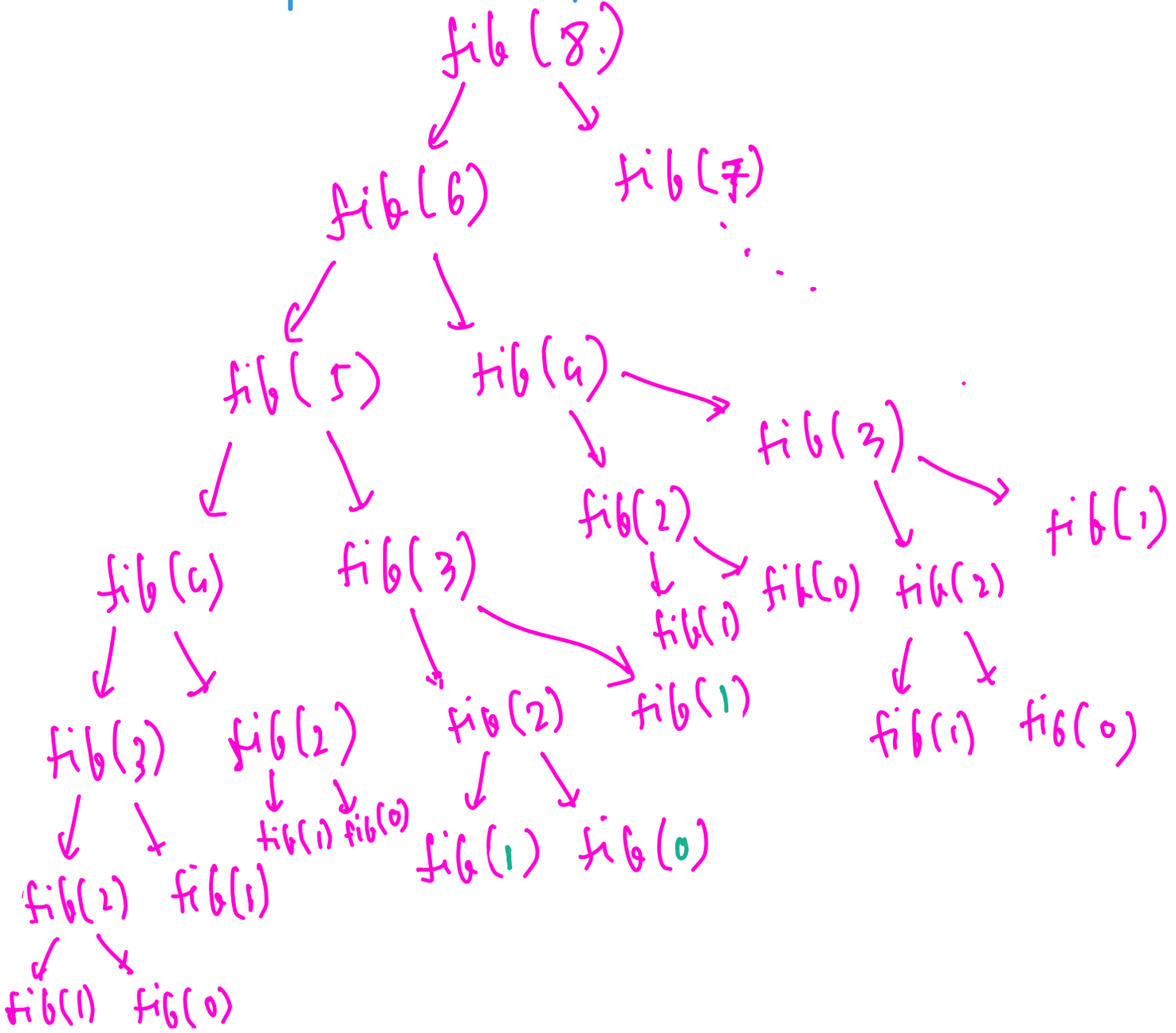
return fib(n-1) + fib(n-2)

Base Case

if n == 0 return 0

if n == 1 return 1

(h) Fibonacci sequence: Tree structure analysis.



Merge Sort ↓ $O(n \log_2 n)$

(i) (Sorting). Sort an array a using recursion

Let the name of the function which does this be "sort(a)".

Assumption: sort(arr) will sort the array in the ascending order.

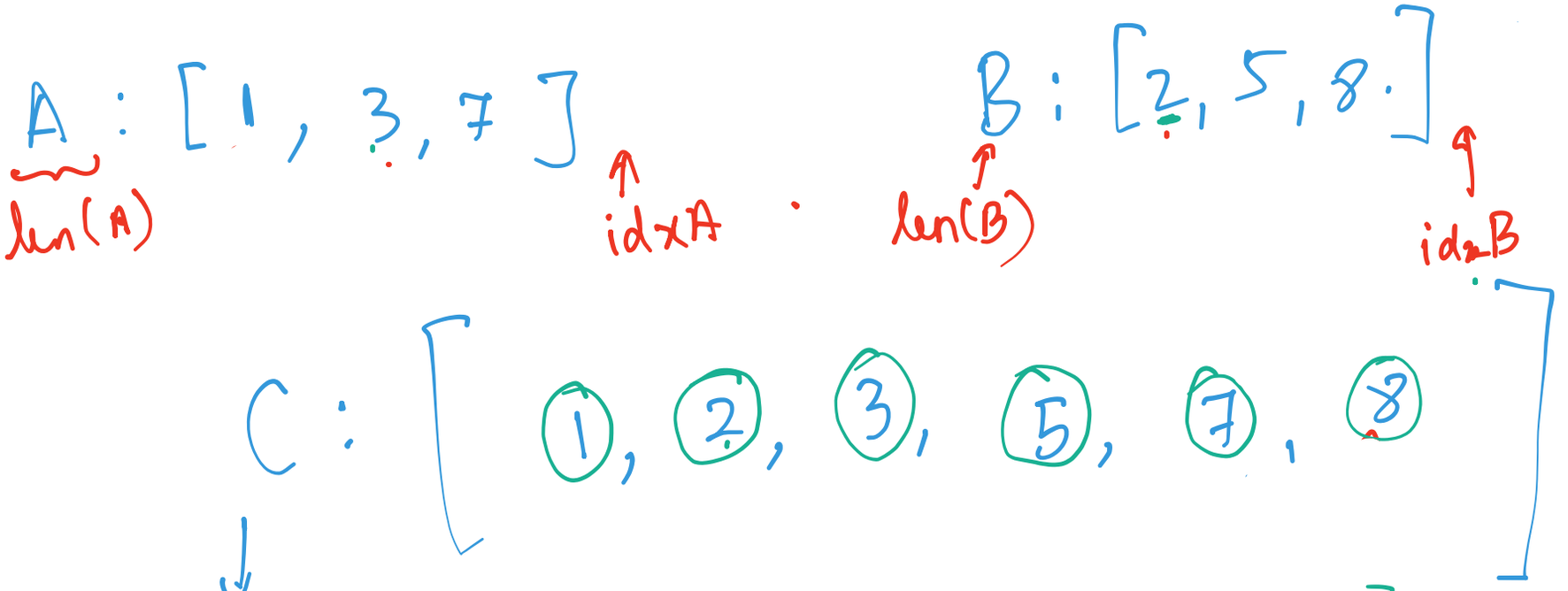
Subproblem: I have two sub-array of arr: A and B. To get C from A and B, I will use the 'merge' operation.

Base Case:

return → [ ], [1] → Already sorted

↳ already sorted.

def merge(A, B):
# A and B are both sorted.



should be sorted.

C : [1, 2, 3, 5, 7, 8.].

a → n.

O (len(A) + len(B)) → algorithm to do this!

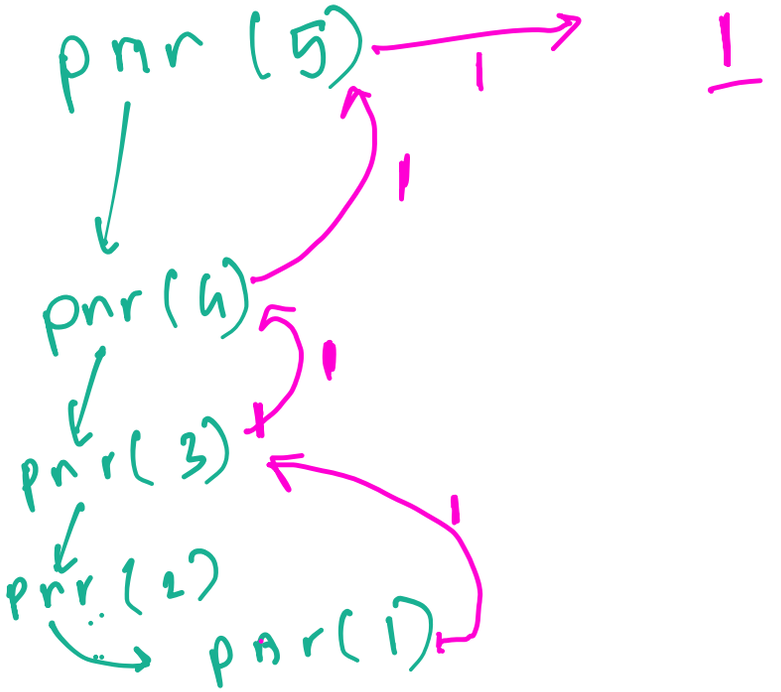
O(n)

```
def merge(A,B): # A and B are sorted.
    idx_A = 0
    idx_B = 0.
    C = []
    while idx_A < len(A) and idx_B < len(B):
        if A[idx_A] < B[idx_B]:
            C.append(A[idx_A])
            idx_A += 1
        else:
            C.append(B[idx_B])
            idx_B += 1.
    if idx_A != len(A):
        C += A[idx_A:]
    if idx_B != len(B):
        C += B[idx_B:].
    return C.
```

```
return C + A[idx_A:] \
       + B[idx_B:]
```

```
1 def print_n_roshan(k):
2     if k > 1:
3         return print_n_roshan(k-1)
4         print(k)
5     return 1

print_n_roshan(5)
```



```
for i in range (5):
    for j in range (i, -1, -1):
        print (j, end = " ")
    print().
```

i
0
1
2
3
⋮
n

j
[0]
[1, 0]
[2, 1, 0]
[3, 2, 1, 0]

count
1
2
3
⋮
n

$$\frac{n(n+1)}{2}$$

summation().

```
for i in range(5):
    for j in range(i, -1, -2):
        print(j, end=" ")
    print()
```

Venkatesh : Bubble Sort .

```python
def bubble_sort(a):
  count = 0
  for i in range(len(a) - 2, -1, -1):
    for j in range(0, i+1):
      count += 1
      if a[j] > a[j+1]:
        a[j], a[j+1] = a[j+1], a[j]
  print(f"Num times the loop was run: {count}")
  return a
```

len(a) - 1 times.

i+1.

	i	j
1]	len(a) - 2	len(a) - 1
2]	len(a) - 3	len(a) - 2
	k = 2	len(a) - k - 1

$$\frac{n(n-1)}{2} - \frac{(n-k-1)(n-k)}{2}$$

$1 \text{ to } n - \frac{n(n+1)}{2}$

$1 \text{ to } (n-k-1) - \frac{(n-k-1)(n-k)}{2}$

$$\frac{n(n-1)}{2} - \frac{(n-k-1)(n-k)}{2}$$

$$\frac{n^2 - n - (n^2 - kn - kn + k^2 - n + k)}{2}$$

$$2kn - k^2 - k$$

$$\frac{k(2n-k-1)}{2}$$

def merge_sort(arr):

→ 2 recursive calls to merge_sort.

→

And use the merge function to do the sort.

```
i,j,k,p,q = 0
for i in range(n):
  p = 0
  j = n
  while(j>1):
    p+=1
    k = 1
    while(k<p):
      q+=1
      k *=2
    j = j/2
```