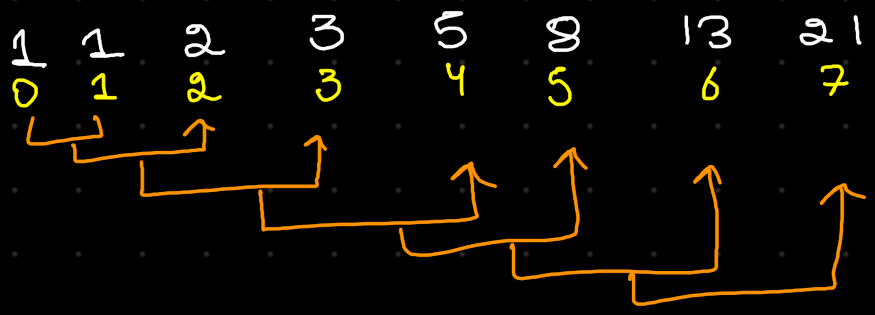
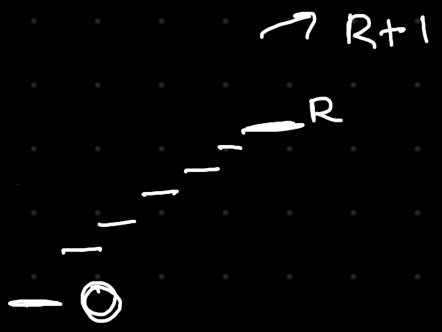
Fibonicci Series

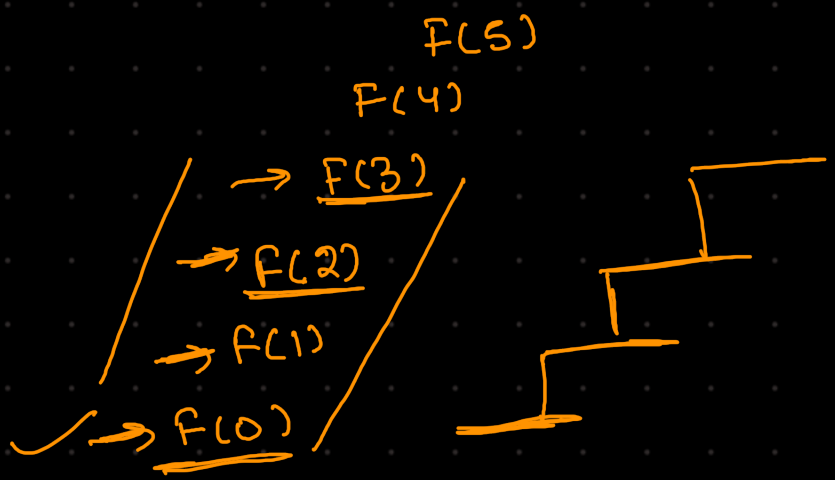


① Base case ⇒ F(0) = 1

② Assume F(R) ⇒ true



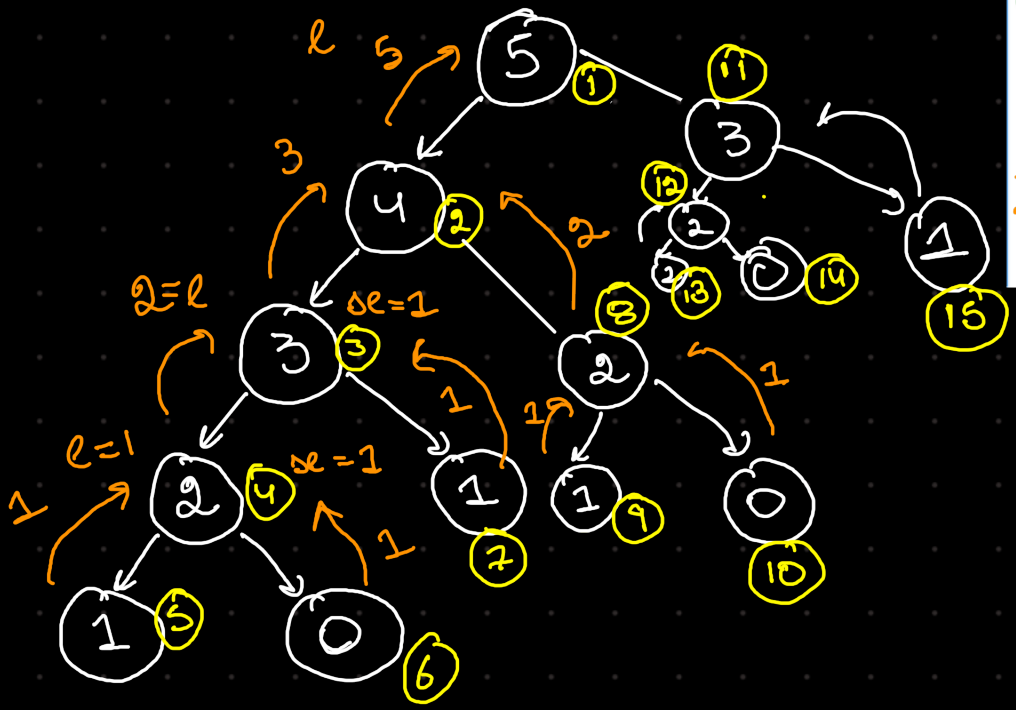
$0 \le i \le R$



last =

second_last =

$$F(\underbrace{n}_{R+1}) = F(\underbrace{n-1}_{R}) + F(\underbrace{n-2}_{R-1})$$

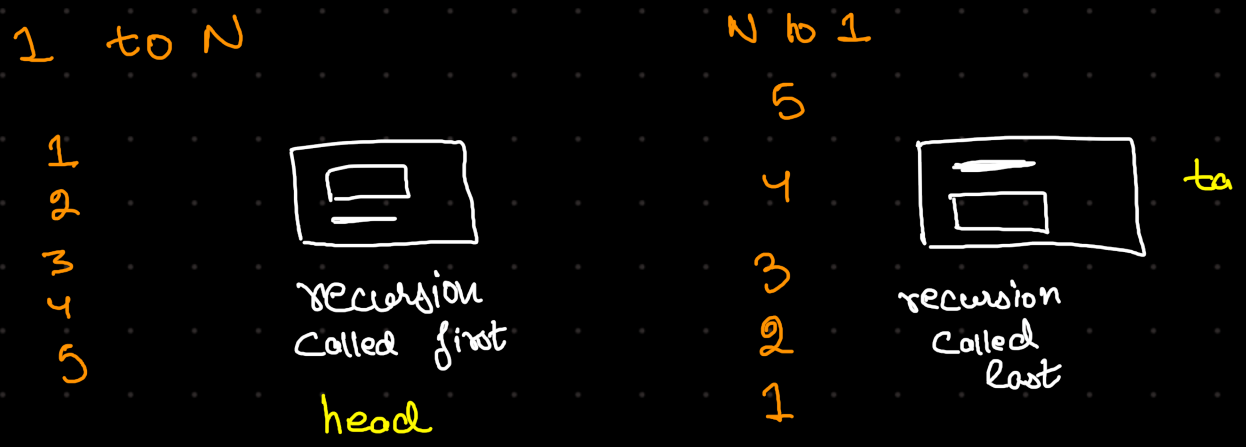



```python
def fib(n):
    # print(n)
    if(n==0):
        return 1
    if(n==1):
        return 1

    last = fib(n-1)
    secondLast = fib(n-2)

    ans = last + secondLast
    return ans
```

Q: Write a program for a given number n.

1] To print 1 to N.

2] To print N to 1.



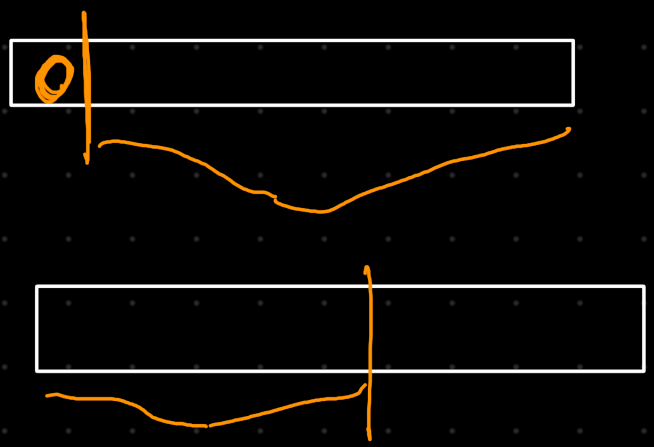
Assignment

Q-1 Sum of digits of a number

Q-2 Power of a number

1. Base case
2. Recursive call
3. Your calculation

Recursion with Arrays / List

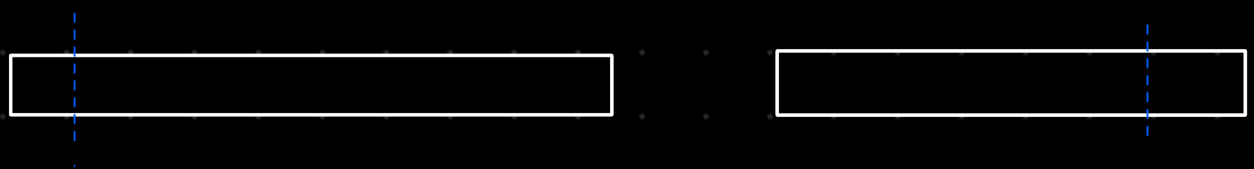


Recursion with Arrays/List

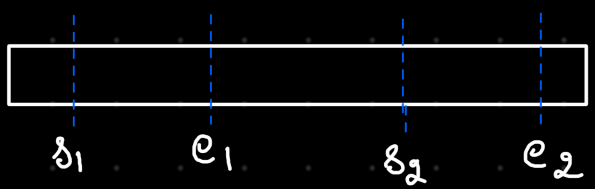
Whenever in arrays, we are dealing with a problem that can be broken & individually be applied to left over part, we can use recursion there.

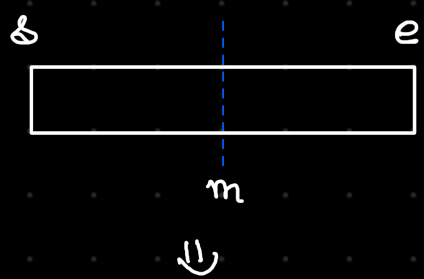
Below are major ways we break our array/list.

1. First or last element removed



2. Using start and end index (eg. binary search)





divide from middle

3. Copy part into a new array and send to further calls.

Q:- Sum of an array using Recursion

Find the sum of an array using recursion, specify the recurrence relation and recursion tree.

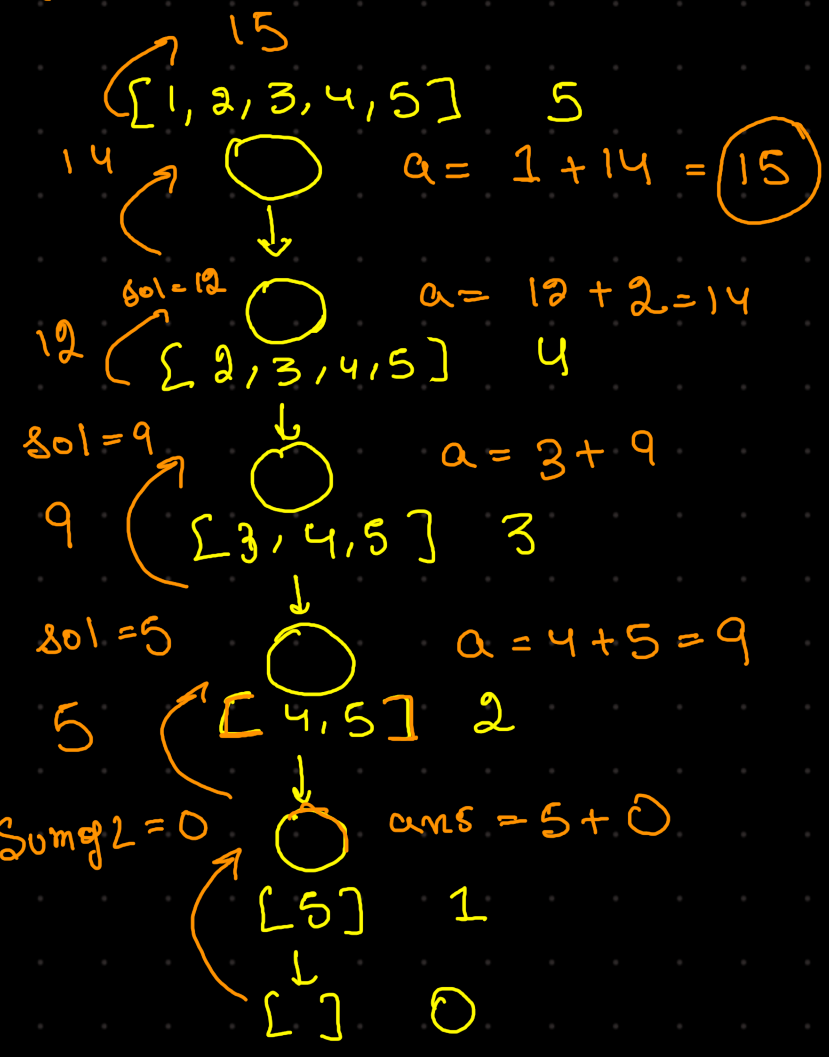
Give code solution via head and tail recursion.

1. Base case.

0 1

2. Recursive call

3. Our calcution



Q: Check if array is sorted or not using recursion.

Draw recurrence tree.

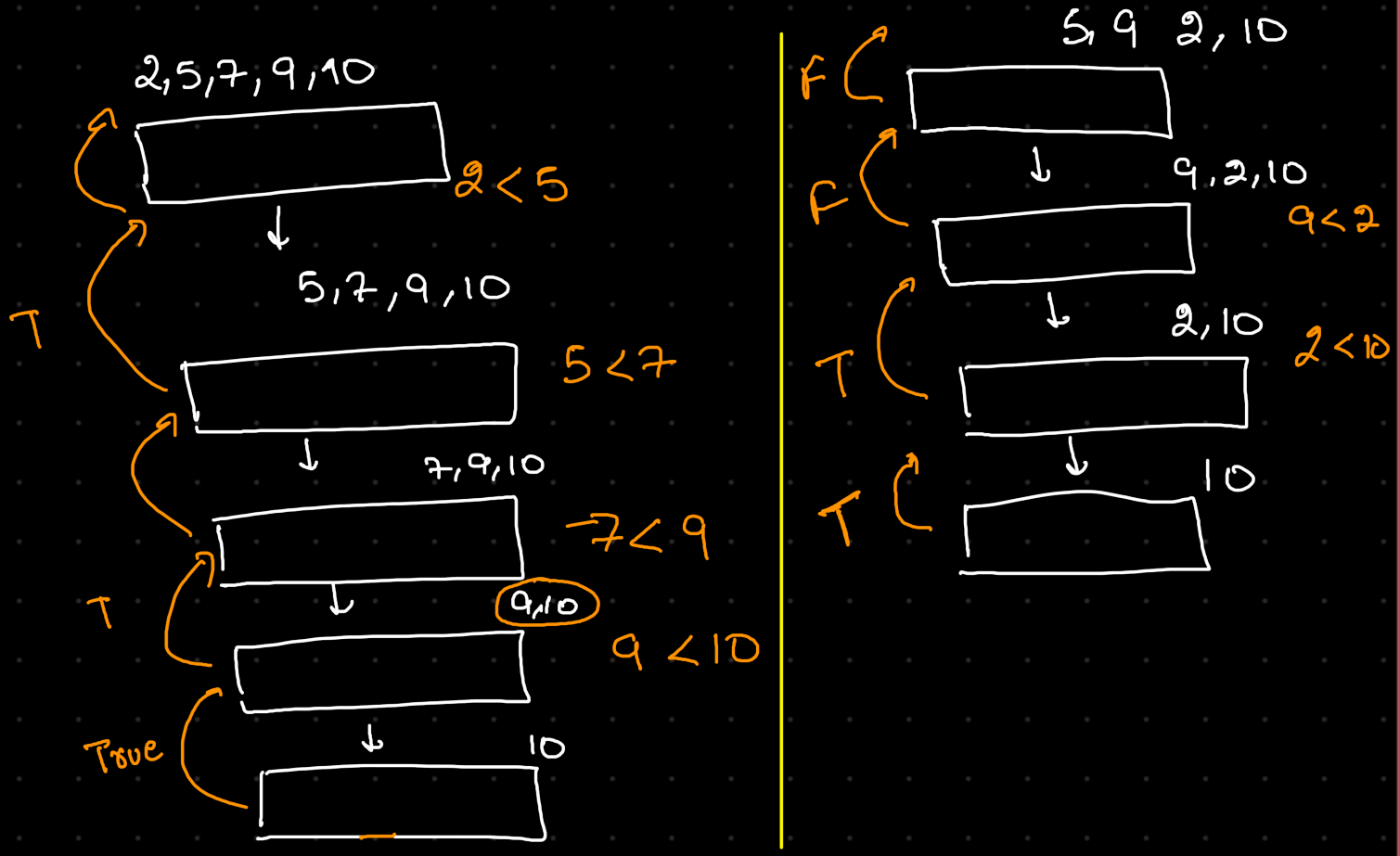
1. Base case if len == 0, 1 then true.
2. Recursive call
3. Our calculation.

So it normally doesn't matter if we are doing our work first or not in these kind of problems.

1] Base Case

2] Recursive call

3] Our Calculation.


2,5,7,9,10
2<5
5,7,9,10
T
5<7
7,9,10
7<9
T
9,10
9<10
True
10
5,9 2,10
F
9,2,10
F
9<2
2,10
T
2<10
10
T

Q: Given an array find the first, last and all indices of a number.

Use recursion.

[3, 2, 5, 2, 8, 2, 1]
  0 1 2 3 4 5 6

2

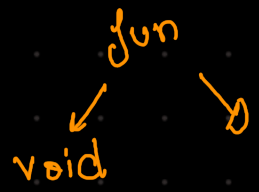
first = 1
last = 5
allIndex = [1, 3, 5]

[3, 2, 5, 2, 8, 2, 1]
  0 1 2 3 4 5 6
  ↑ ↑

All Indices of a number in an array

3	2	5	2	8	2	1
0	1	2	3	4	5	6

2 , 10

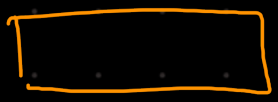


1. Print
2. List (all indices)
3. return a list to me with ans.

Print

1. Base case (l1).size == 0
2. I.H. ask recursion to do the work
3. Will check for one of the element

Send a list and get that populated.

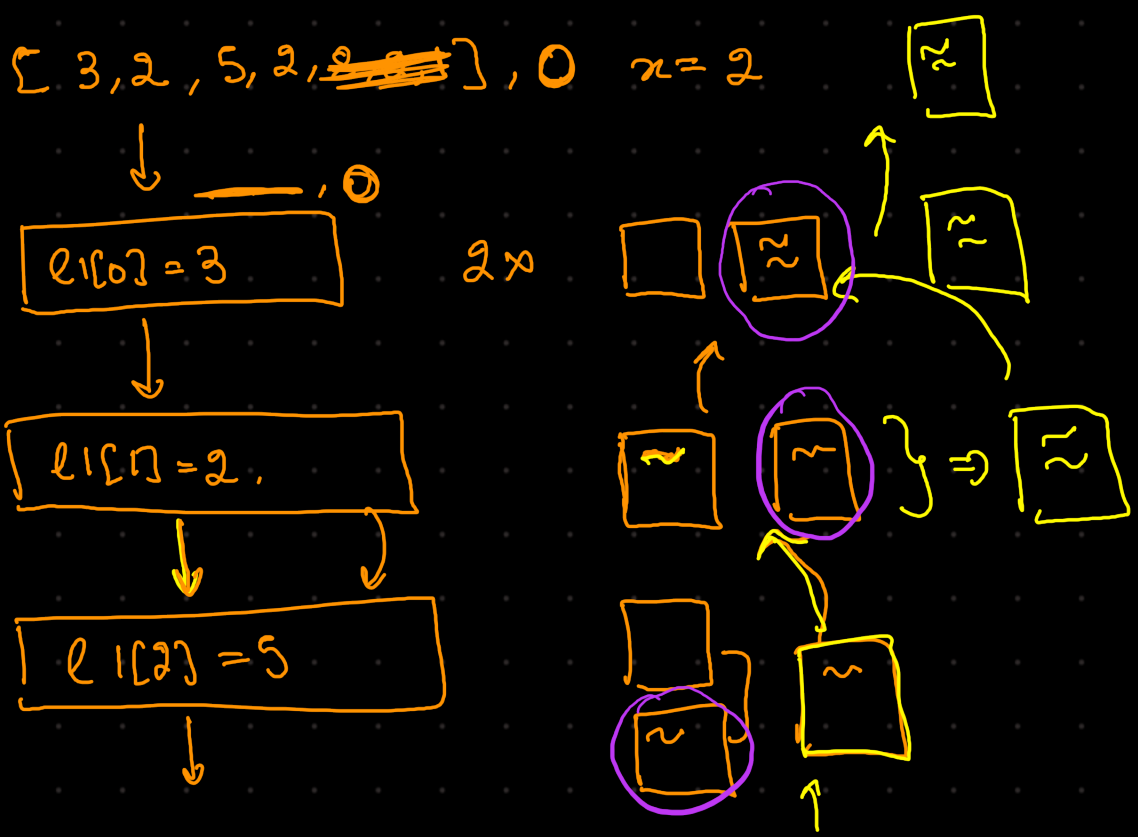


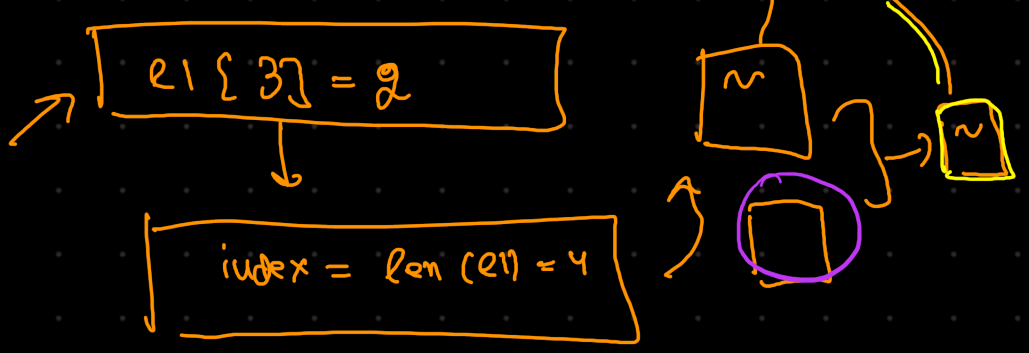
, index , x , ansList

↓

+1

Return list of all Indices of a number.




el[3] = a
index = len(el) = 4

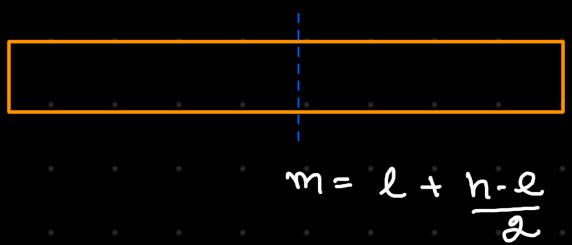
Searching using recursion

→ Linear Search

[3, 2, 5, 8, 9, 10, 2], 2

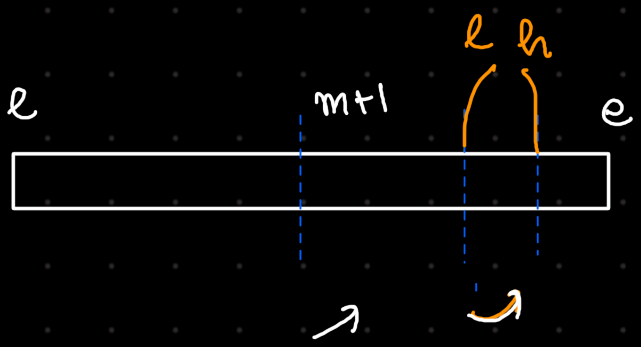
1. Base case if len(l) == 0
   len(l) == index
2. Recursive Call ↻
3. Your work

→ Binary search
(sorted)



$m = l + \frac{h-l}{2}$

1. middle element



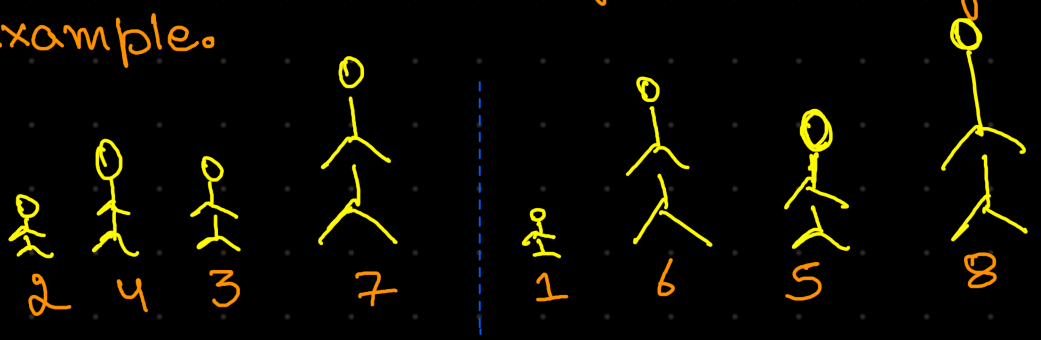
Merge Sort

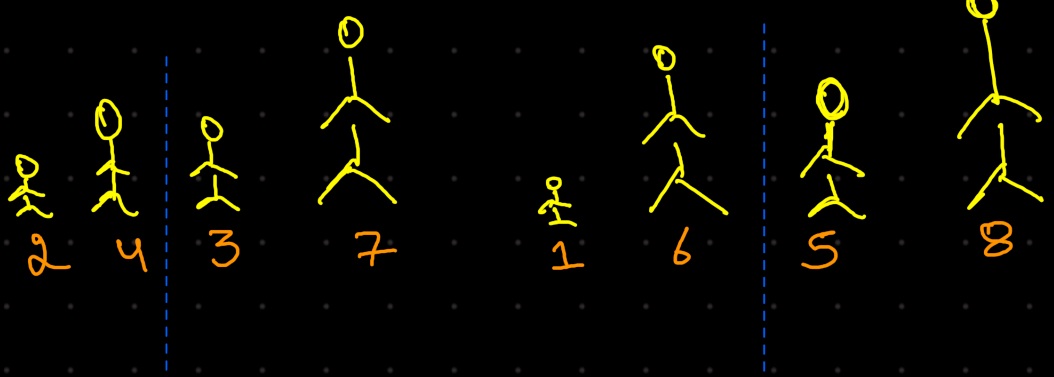
Merge Sort

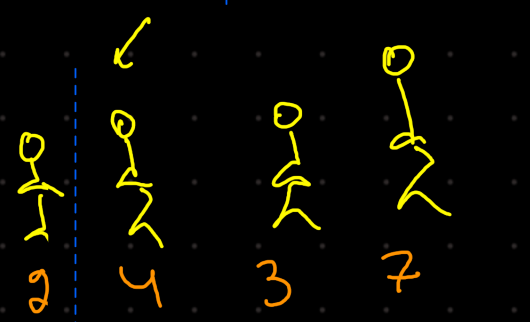
⇒ recursive sorting algorithm

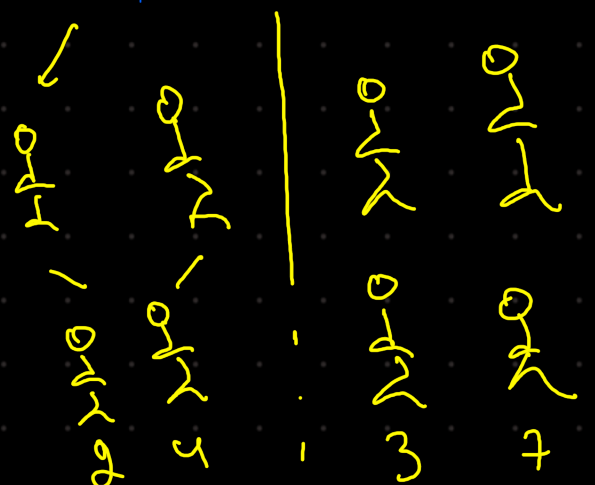
⇒ fast as compared to other sorting (will learn why)

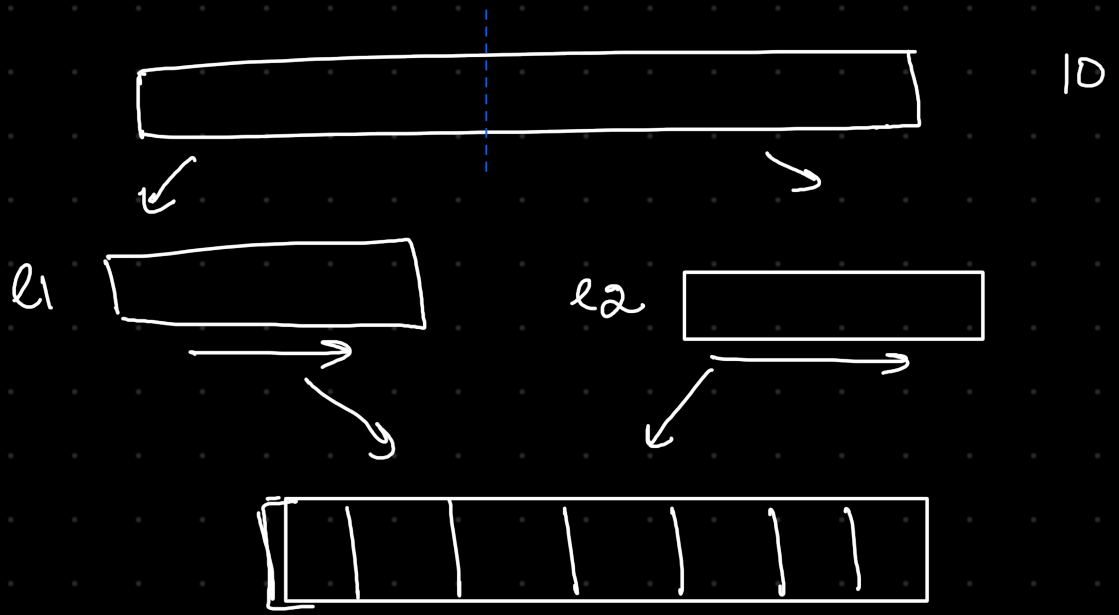
Let's understand merge sort using an example.



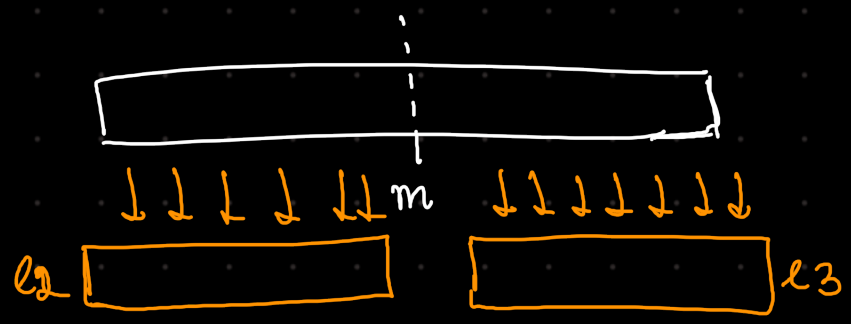








def mergeS(l1)

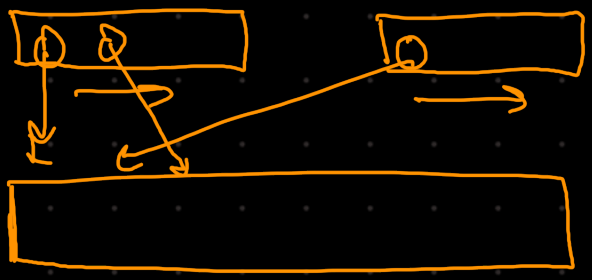


1) copy

mergeS(l2)
mergeS(l3)

merge two Sorted List (l2, l3)

merge two Sorted List

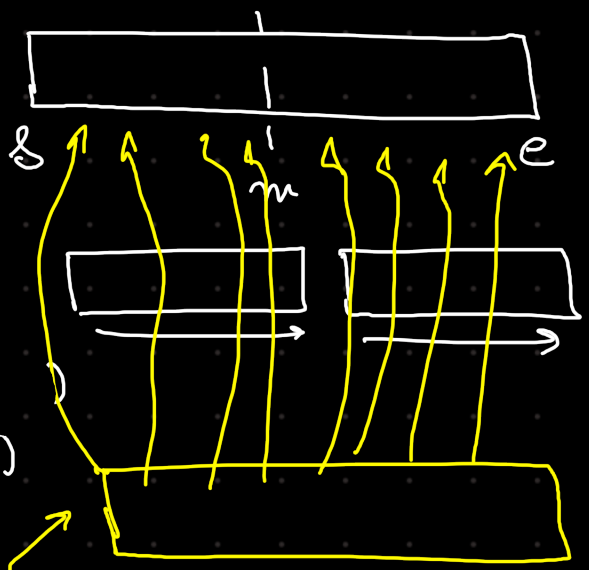


2] merge (l1, s , e)

if (s >= e)

return

m = s + (e - s) // 2

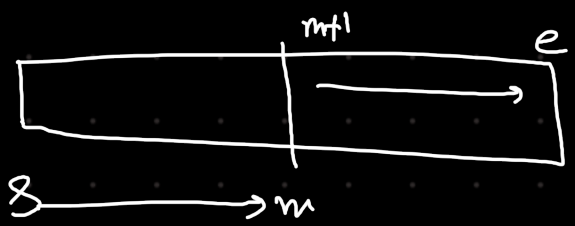


merge (l1 , s , m)

merge (l1 , m+1 , e)

merge (l1, s, m , e)

return



Always debug the code on uppermost

ans / step

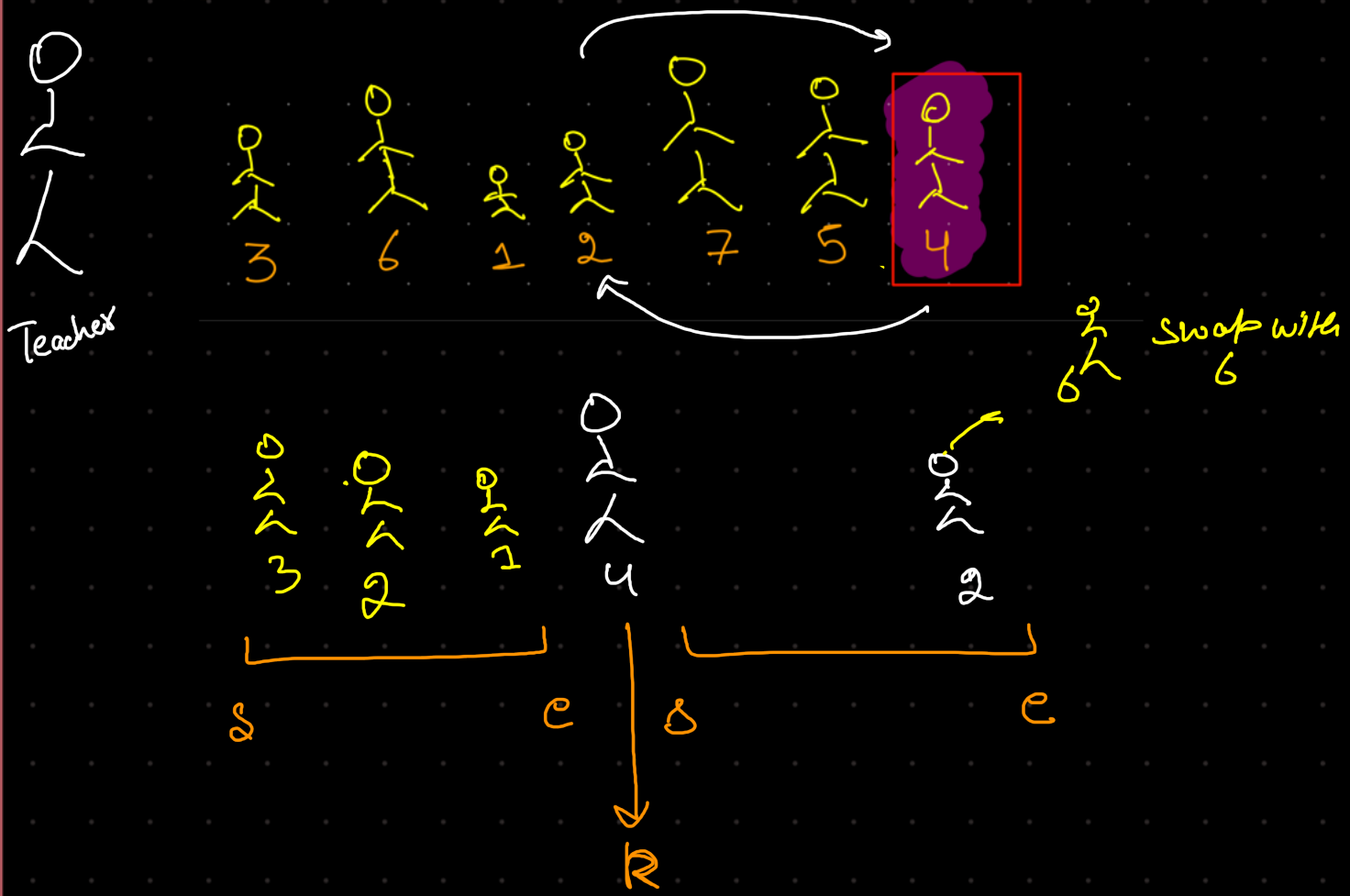
very imp

Quick sort

Quick Sort

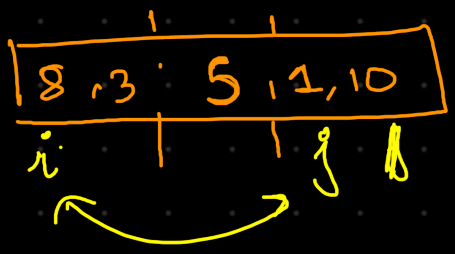
⇒ In quick sort, unlike merge sort.

we will first do our work and then let recursion handle the rest.



partition around pivot

8, 3, 10, 1, 5



equal

5 = 5

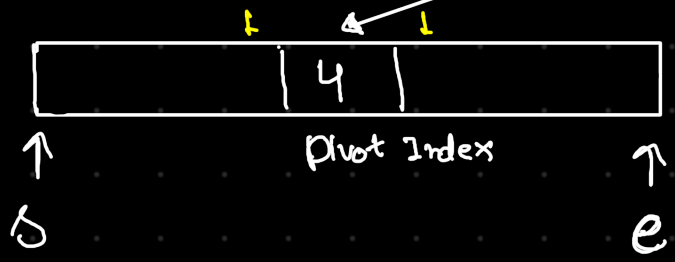
1, 3, 5, 8, 10

3 6 7 2 1 5 4

s e

p = e1[e]

s+c e



< 4 > 4

i < p1 j > p1

i

j

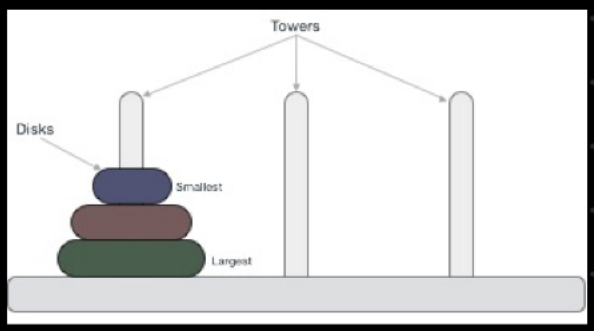
Recursion with Strings

→ Just like arrays, for string too we work on them after piching one of the element and then asking recursion to work on rest.

Subsequences of a string (Print, return, list)

Print and return subset sum to R

Tower of Hanoi



Letter Combination Of a Keypad

Print and return all codes

Framework to solve any recursion question.

1] Identify if problem can be broken into a smaller same problem.

2] Write recurrence relation.

3] Code

a) Identify and write base cases.

b) Assume (don't Question) the Induction hypothesis.

c]