Ruchika PSA Class 6

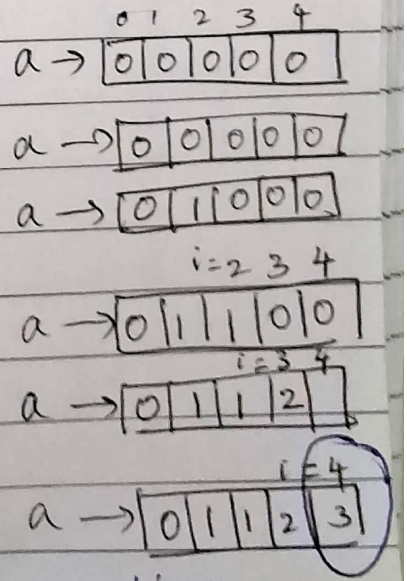
- All problems can be solved either by recursion or iteration
  Recursion involves stack - can overflow

- Fibonacii

n	0	1	2	3	4	5	6	7	8	...
a	0	1	1	2	3	5	8	13	21	...

- Dynamic Programming
  requires BASE case - acutlly knows answer
  memoization - stores prev. solutions
  provides optimal solution / sub structure
  requires $>\Theta(n)$ space, sometimes $\Theta(1)$ space

```
def fibonacii(n):
    a = [0 for i in range(n+1)]
    a[0] = 0
    a[1] = 1
    for i in range(2, n+1):
        a[i] = a[i-1] + a[i-2]
    return a[n]
```



Time complexity = $\Theta(n)$
Space complexity = $\Theta(n)$

But we can optimize space, we can have 2 var to store last last & last

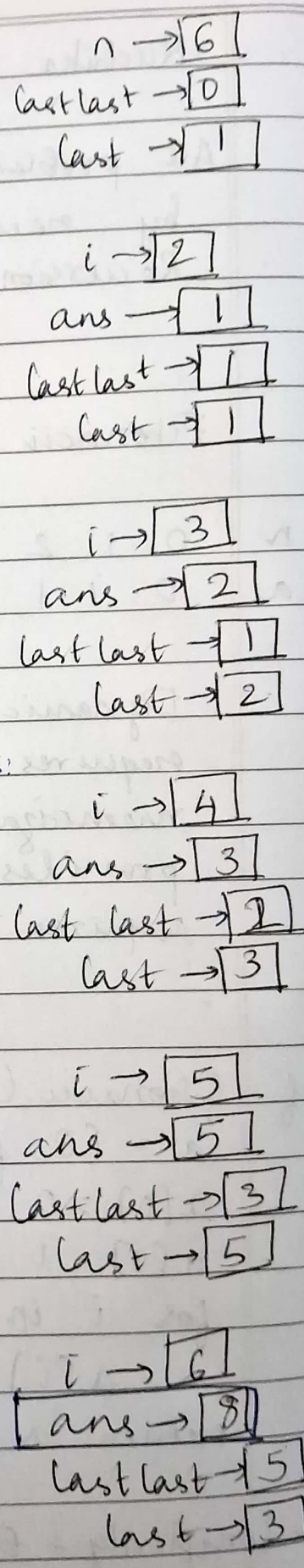
Fibonacii Number (Iterative)

n = 6

```
def FibI(self, n:'int'):
    if (n < 2)
        return n
    lastlast = 0
    last = 1
    for i in range(2, n+1):
        ans = lastlast + last
        lastlast = last
        last = ans
    return ans
```

Time complexity = $\theta(n)$

Space complexity = $\theta(1)$ → we only use 2 vars: lastlast & last



Fibonacii Number (Recursive)

n = 6

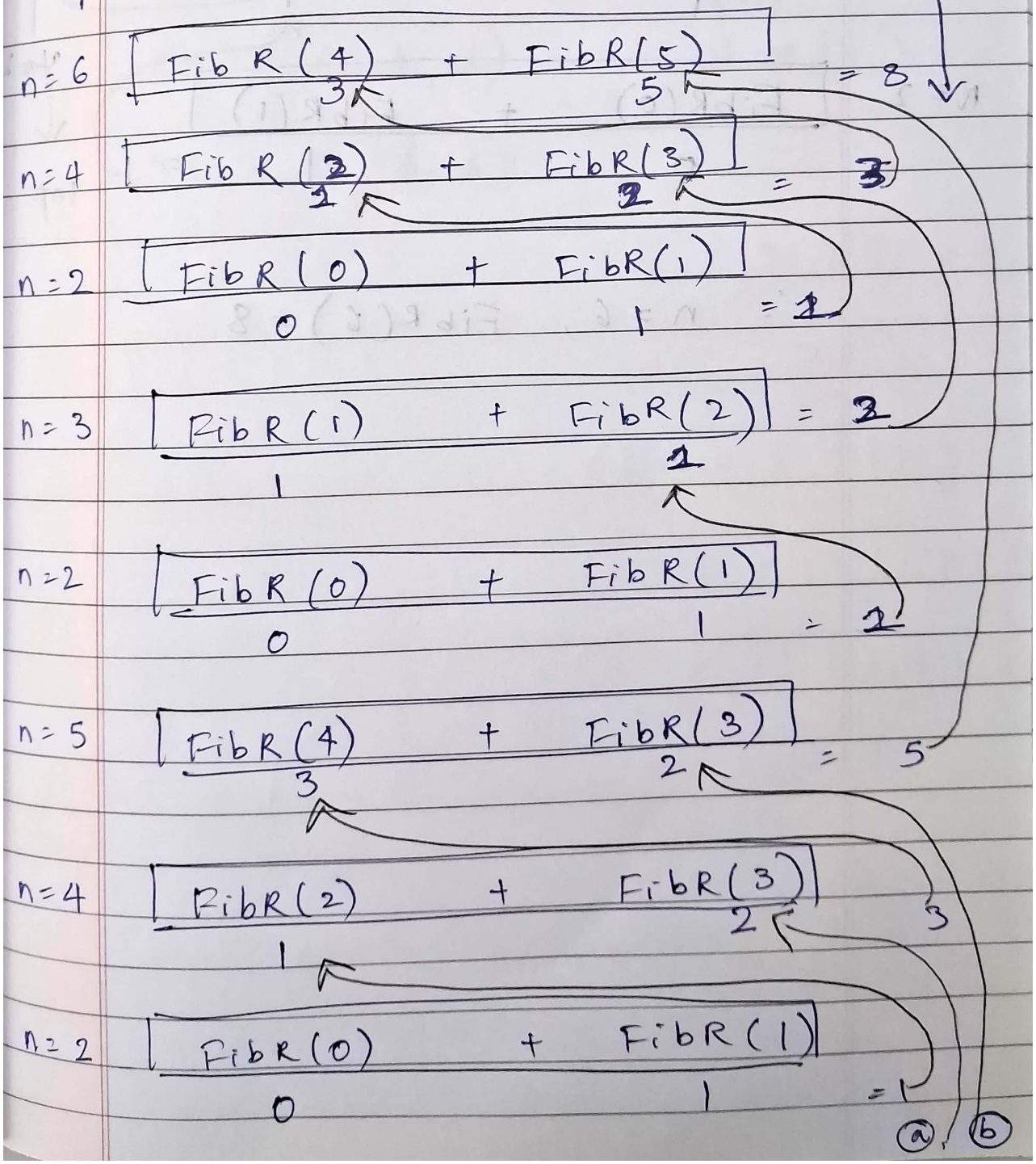
```
def FibR(n: 'int'):
    if (n<2)
        return n
    return FibR(n-2) + FibR(n-1)
```

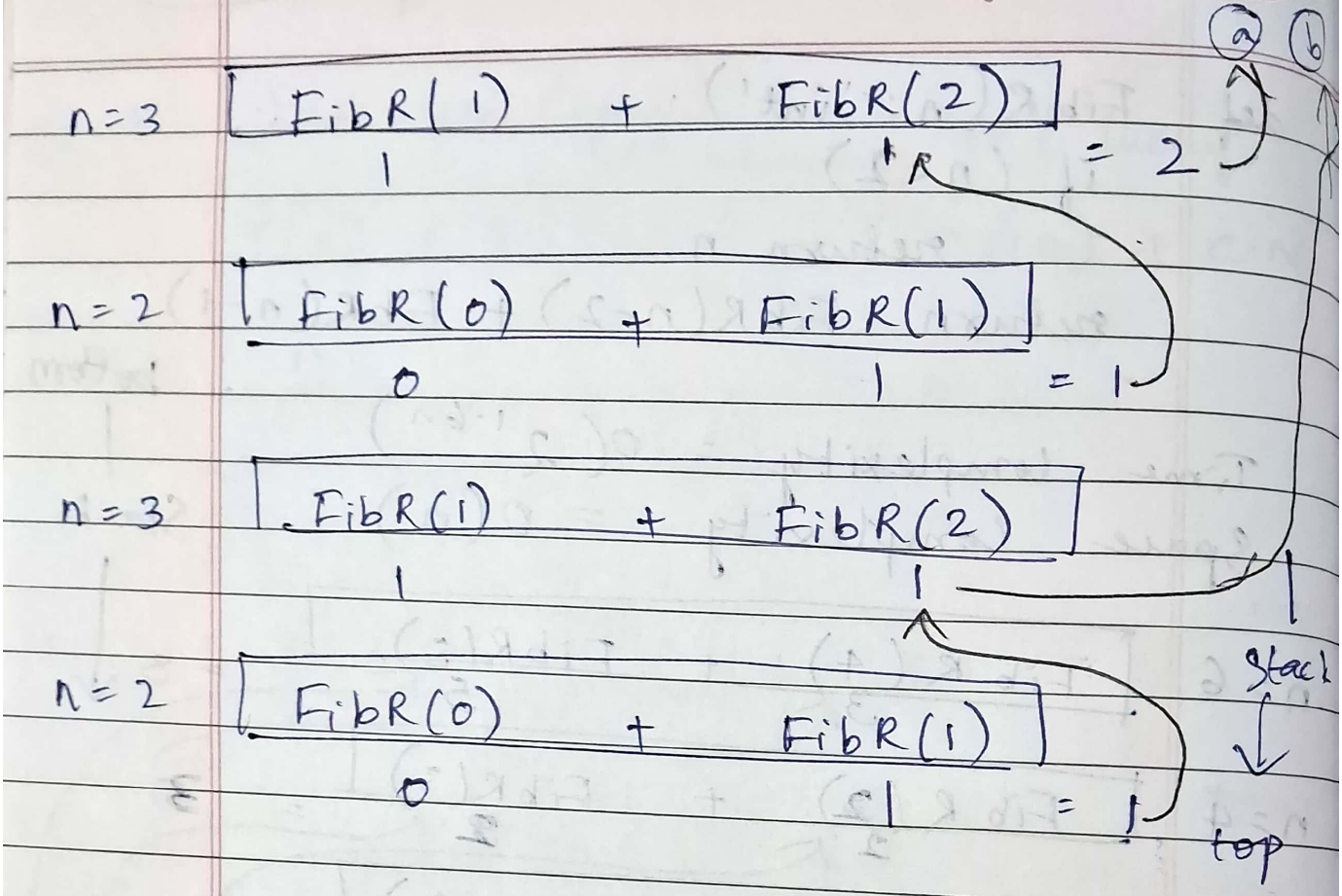
Time complexity = $O(2^{1.6n})$

Space complexity = $O(n)$

bottom

Stack




n=3
FibR(1) + FibR(2)
1 1
= 2
n=2
FibR(0) + FibR(1)
0 1
= 1
n=3
FibR(1) + FibR(2)
1 1
n=2
FibR(0) + FibR(1)
0 1
= 1
a
b
stack
top

∴ n = 6, FibR(6) = 8

$T(n) = T(n-1) + C$

$T(1) = 1$

n	C	
↓		
$n-1$	C	$T(n) = nC$
↓		
$n-2$	C	$\therefore T(n) = \Theta(n)$
↓		
$n-3$	C	
↓		
⋮		
1		

n

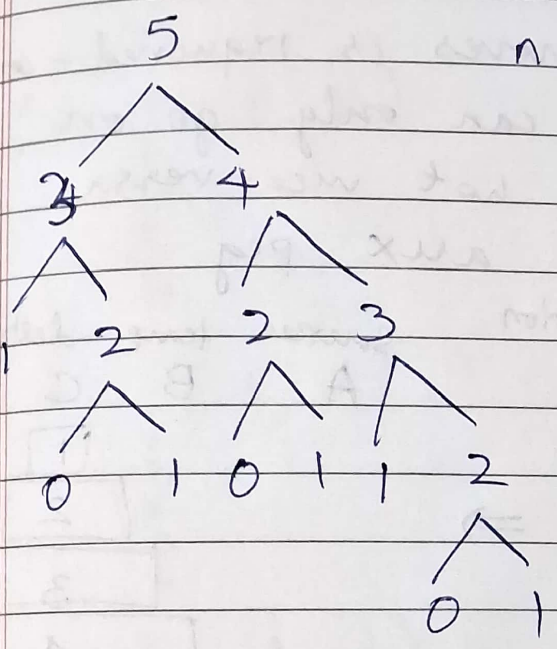
$T(n) = T(n/2) + C$

$T(1) = 1$

n	C	$n/2^0$	$\frac{n}{2^k} = 1$
↓			
$n/2$	C	$n/2^1$	$2^k = n$
↓			
$n/4$	C	$n/2^2$	$k = \log_2 n$
↓		↓	
....		....	$\therefore T(n) = \Theta(\log_2 n)$
↓			
1		$n/2^k$	

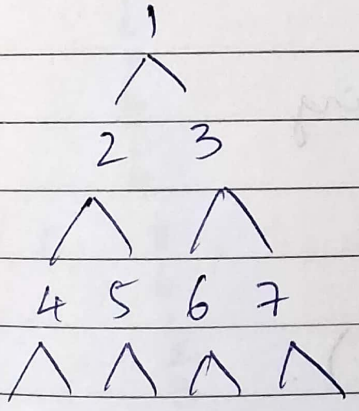
$T(n) = T(n-1) + T(n-2)$ $\approx O(1.6^n)$

$T(1) = 1$, $T(0) =$



$n = 5$

In Fibonacci, if it is not a complete binary tree $\approx 2^{1.6n}$

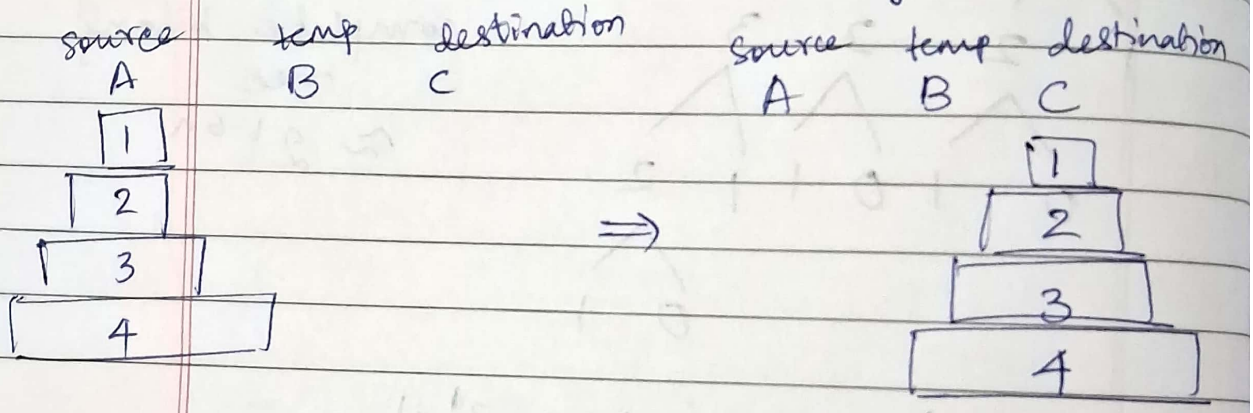


$\therefore T(n) = 2^n - 1$ (no. of nodes)

All elements filled in the binary tree = complete Binary Tree

Tower of Hanoi

- Move one disk at a time
- How many moves is required - complexity
- Small disk can only go on larger disk not vice-versa
- You can use aux peg



- no disk ⇒ do nothing
- 1 disk ⇒
- 2 disk ⇒

```
def TH(n, s, t, d):
    if (n):
        TH(n-1, s, d, t)
        s → d
        TH(n-1, t, s, d)
```

Time Complexity = O(2 ^ n)
Space Complexity = O(n)

start :

A → 4 3 2 1
B →
C →

Tower of Hanoi
n = 4

stack (bottom)

TH(n, s, t, d)

① TH(4, A, B, C)

② TH(3, A, C, B)

③ TH(2, A, B, C)

④ TH(1, A, C, B)

⑤ TH(0, A, B, C)

⑦ TH(0, C, A, B)

⑨ TH(1, B, A, C)

⑩ TH(0, B, C, A)

⑫ TH(0, A, B, C)

(24) s → A → 3 2 1
t → B →
d → C → 4

(13) s → A → 4
t → C → 2 1
d → B → 3

(8) s → A → 4 3
t → B → 1
d → C → 2

(6) s → A → 4 3 2
t → C
d → B → 1

(11) s → B →
t → A → 4 3
d → C → 2 1

TH(n, s, t, d)

(14) TH(2, C, A, B)

(15) TH(1, C, B, A)

(16) TH(0, C, A, B)

(18) TH(0, B, C, A)

(20) TH(1, A, C, B)

(21) TH(0, A, B, C)

(23) TH(0, C, A, B)

(25) TH(3, B, A, C)

(26) TH(2, B, C, A)

(19) s → C → ~~2~~
t → A → 4 1
d → B → 3 2

(17) s → C → 2 ~~1~~
t → B → 3
d → A → 4 1

(22) s → A → 4 ~~3~~
t → C →
d → B → 3 2 1

(24) s → B → ~~3~~
t → A → 2 1
d → C → 4 3

(31) s → B → 3 ~~2~~
t → C → 4 1
d → A → 2

TH(n, s, t, d)

(28) TH(1, B, A, C)

(29) TH(0, B, C, A)

(30) TH(0, A, B, C)

(32) TH(1, C, B, A)

(33) TH(0, C, A, B)

(35) TH(0, B, C, A)

(37) TH(2, A, B, C)

(39) TH(1, A, C, B)

(39) TH(0, A, B, C)

(41) TH(0, C, A, B)

(29) s → B → 3 2 X
t → A →
d → C → 4 1

(34) s → C → 4 X
t → B → 3
d → A → 2 1

(42) s → A → X
t → B → 1
d → C → 4 3 2

(40) s → A → 2 X
t → C → 4 3
d → B → 1

TH (n, s, t, d)

43 TH (1, B, A, C)

45 s → B → *
t → A →
d → C → 4 3 2 1

44 TH (0, B, C, A)

46 TH (0, A, B, C)

Stack (Top)

end :
A →
B →
C → 4 3 2 1

Tower of Hanoi

$T(n) = T(n-1) + C + T(n-1)$

$T(n) = 2T(n-1) + 1$

$T(0) = 0$

$T(0) = 0$

$T(1) = 2T(0) + 1 = 1 \quad = 2^1 - 1$

$T(2) = 2T(1) + 1 = 3 \quad = 2^2 - 1$

$T(3) = 2T(2) + 1 = 7 \quad = 2^3 - 1$

$T(4) = 2T(3) + 1 = 15 \quad = 2^4 - 1$

$T(n) = 2^n - 1$

Time complexity = $O(2^n)$

Space complexity = $O(n)$

Linear Algorithms:

$\theta(1)$ Sum of first n natural no.s
$n*(n-1)/2$

$O(\log n)$ Find a number in an unsorted list using binary search

$O(n)$ Find the minimum in an unsorted list

$O(n \log n)$ Merge sort

$O(n^2)$ Insertion sort

$O(n^3)$ Matrix 3D print

Exponential Algorithms:

$O(2^n)$ Print truth table (binary),
Tower of Hanoi

$O(n!)$ Permutation print

Sorting

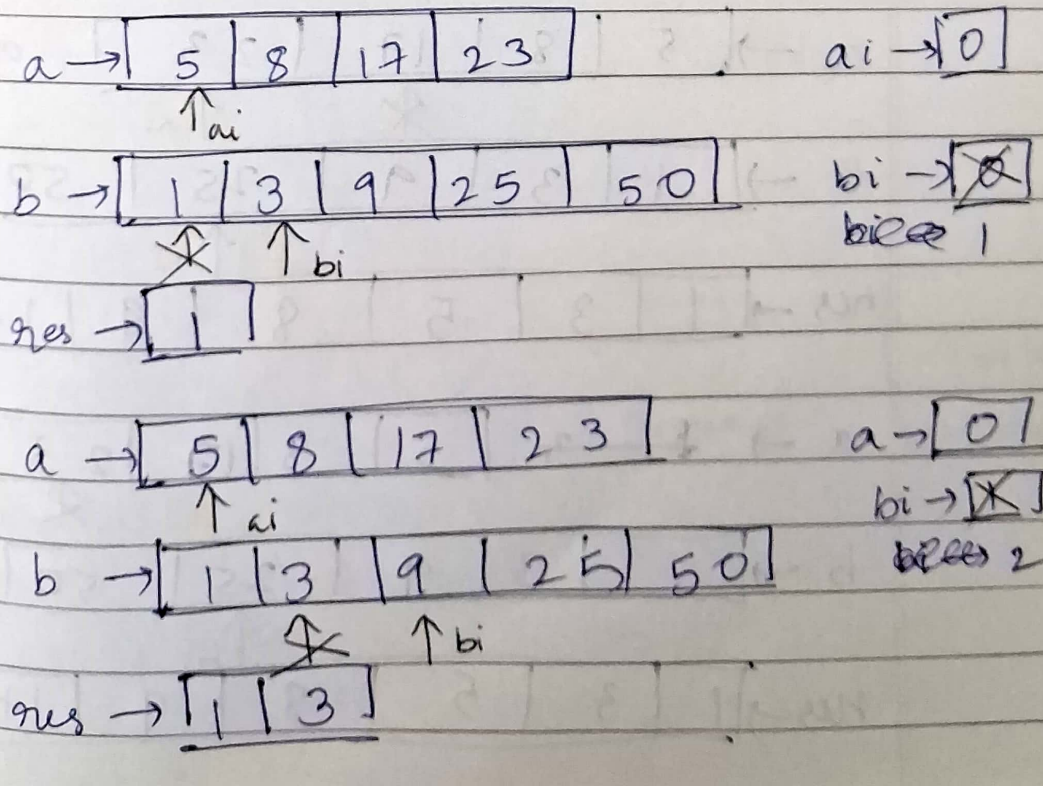
Selection, Insertion $\Rightarrow n^2$

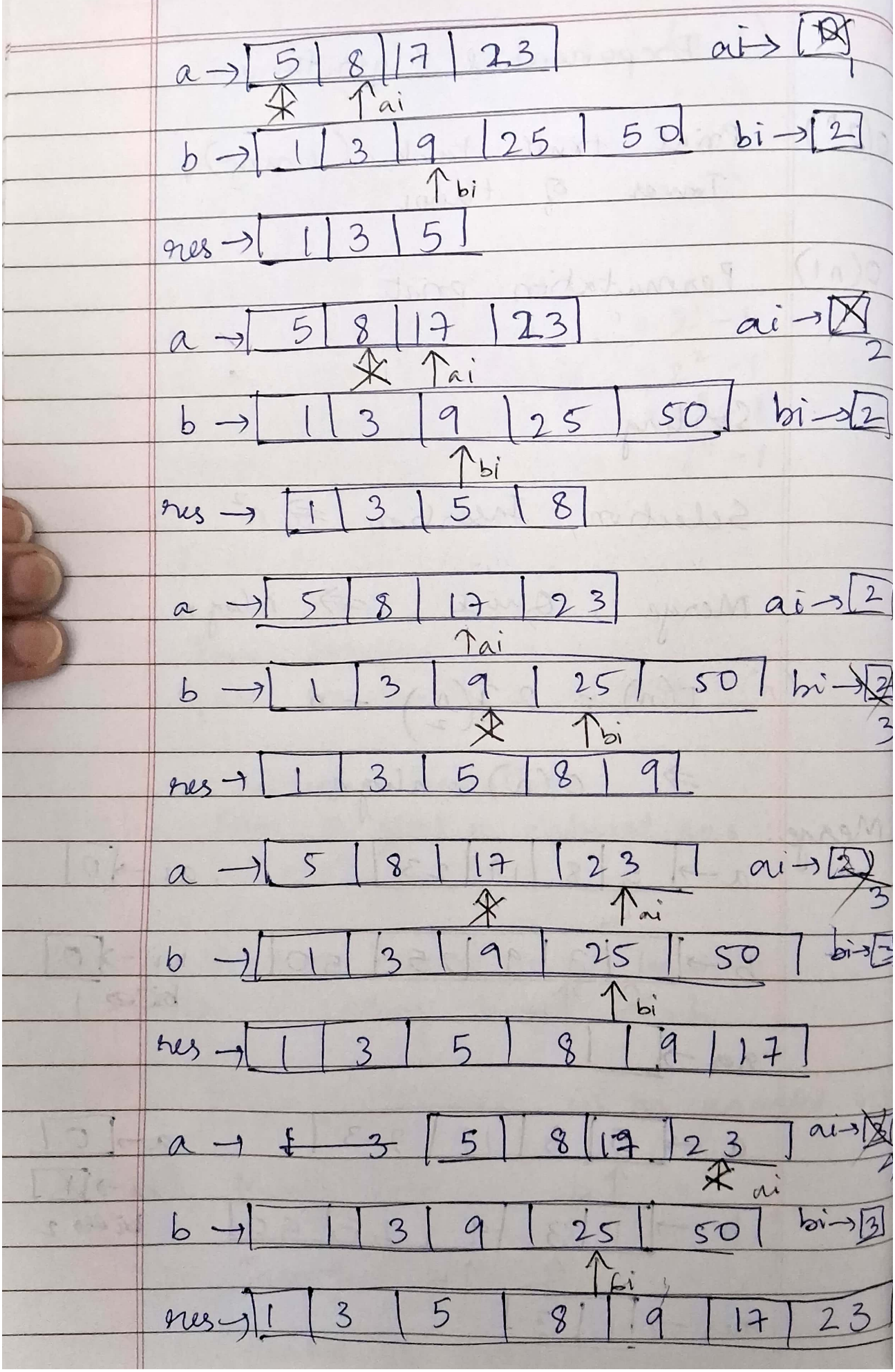
Merge, Quick $\Rightarrow n \log n$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

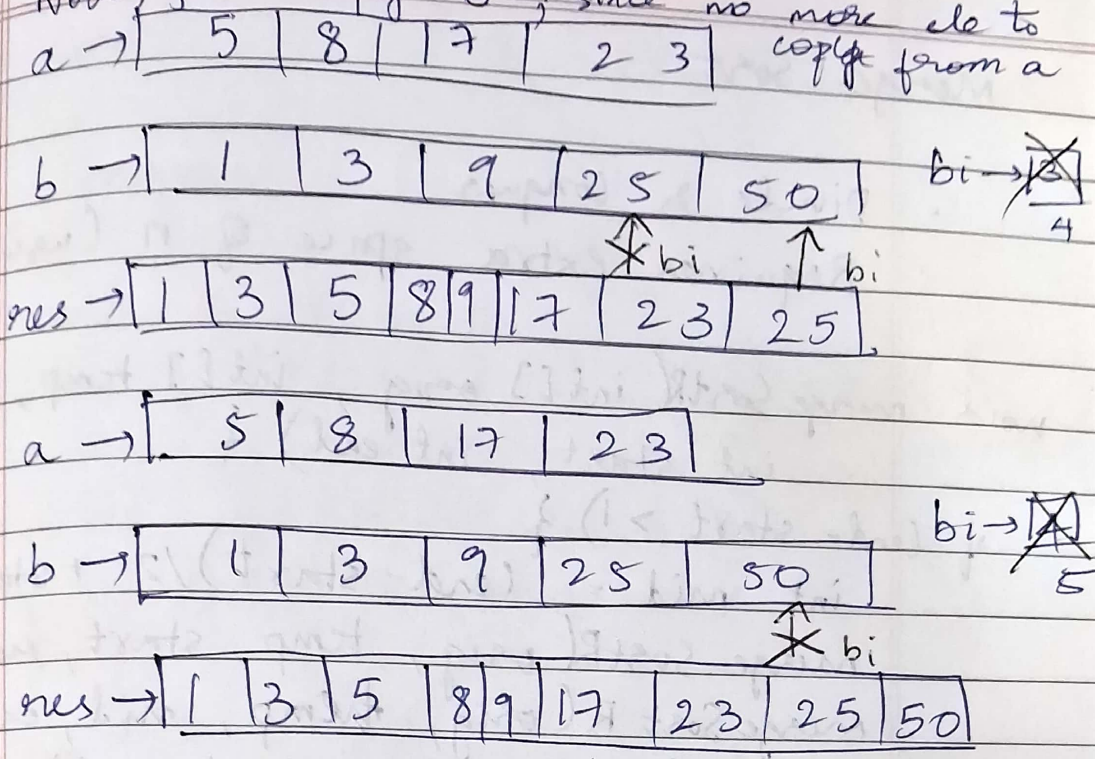
$$\Rightarrow O(n) = n \log n$$

Merge:





Now, just copy b, since no more ele to copy from a



Merging 2 sorted lists:

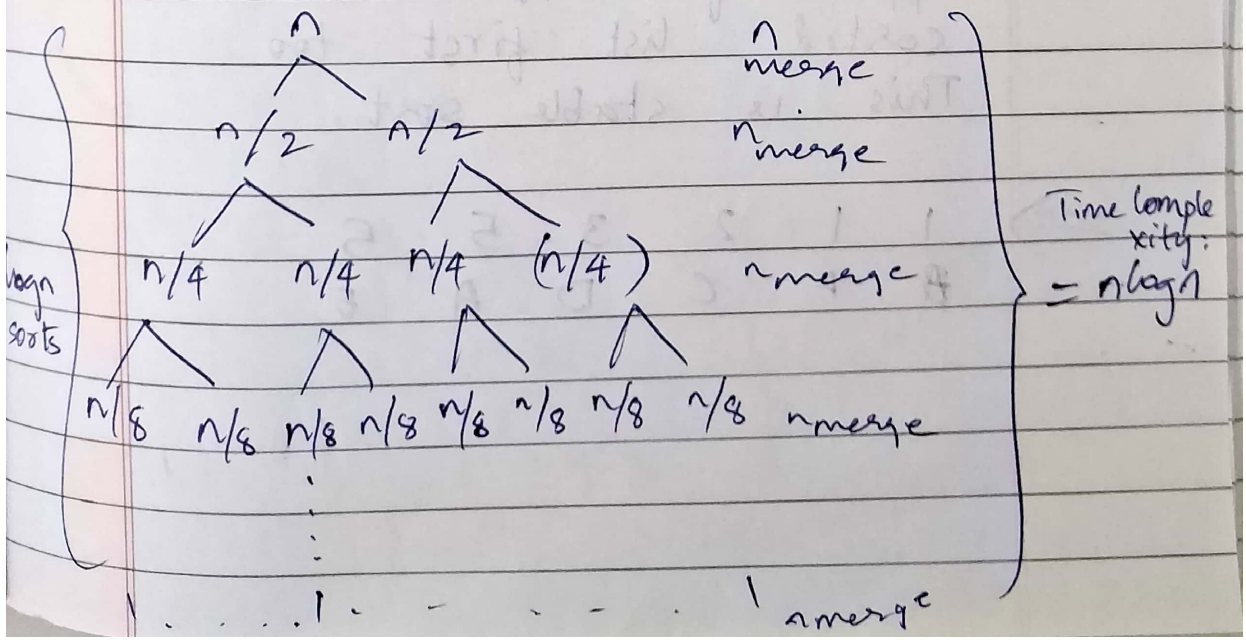
Time Complexity = ~~$O(n \log n)$~~ $O(n)$

Space Complexity = $O(n)$ } space can be improved to $O(1)$ if inplace

∴ For Merge Sort:

$$T(n) = T(n/2) + T(n/2) + O(n)$$

$$T(n) = 2T(n/2) + O(n)$$



Merge Sort

- Divide & Conquer
- Requires extra space of n (using

```
void mergeSortR( int[] orig, int[] temp,
        int start, int end) {
  if (end - start > 1) {
    int mid = (end-start)/2 + start
    mergeSortR(orig, temp, start, mid)
    mergeSortR(orig, temp, mid, end)
    merge (orig, temp, start, end)
  }
}
```

- Inplace, but uses temporary space

5	1	2	3	5	1
A	B	C	D	E	F

- If no. is identical, the no. which appears first must be in the sorted list first too.
- This is stable sort.

⇒

1	1	2	3	5	5
B	F	C	D	A	E

Time Complexity = O(nlogn)
Space Complexity = O(n)

Merge Sort
n = 9

	0	1	2	3	4	5	6	7	8
data →	1	15	5	64	8	12	11	4	35

Date
Page

(1)

mergeSortR(0, 9) ① (31)

mergeSortR(0, 4) ② (12)

mergeSortR(0, 2) ③ (7)

mergeSortR(0, 1) ④

mergeSortR(1, 2) ⑤

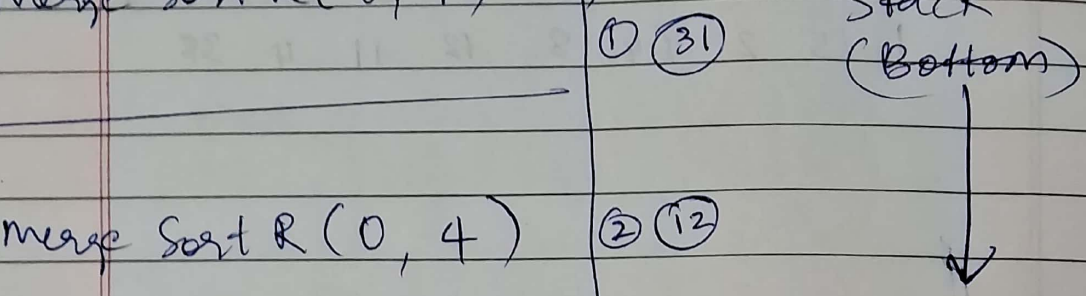
merge(0, 1, 2) ⑥

	0	1	2	3	4	5	6	7	8
data →	1	15	5	64	8	12	11	4	35

temp → | 1 | 15 |

mergeSortR(2, 4)

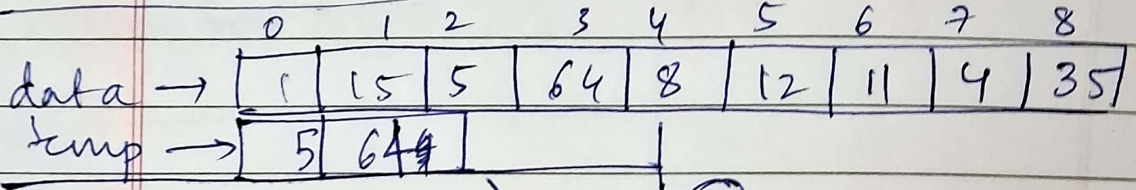
Stack (Bottom)



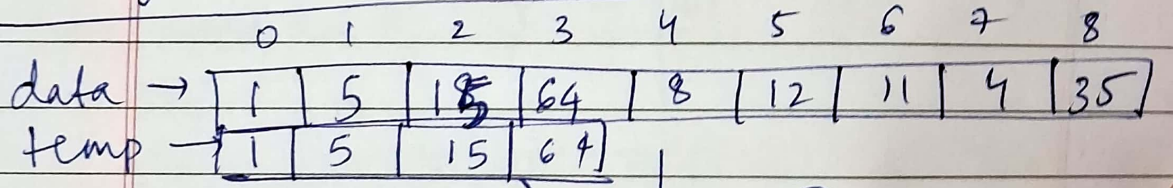
mergeSortR (2, 3) (8)

mergeSortR (3, 4) (9)

merge (2, 3, 4) (10)



merge (0, 2, 4) (11)



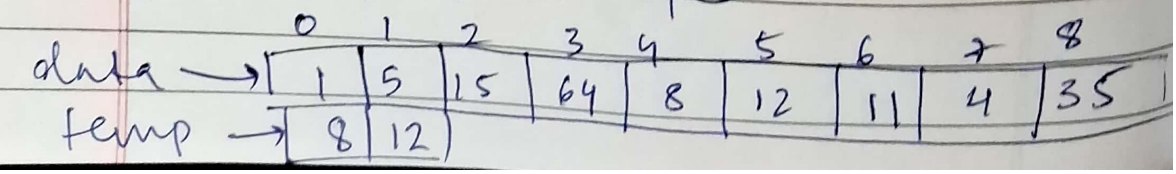
mergeSortR (4, 9) (13) (29)

mergeSortR (4, 6) (14) (18)

mergeSortR (4, 5) (15)

mergeSortR (5, 6) (16)

merge (4, 5, 6) (17)



mergeSortR(6, 9) (19) (27)

mergeSortR(6, 8) (20) (24)

mergeSortR(6, 7) (21)

mergeSortR(7, 8) (22)

merge(6, 7, 8) (23) (25) mergeSortR(8, 9)

	0	1	2	3	4	5	6	7	8
data →	1	5	15	64	8	12	4	11	35
temp →	4	11							

mergeSortR(6, 8, 9) (26)

	0	1	2	3	4	5	6	7	8
data →	1	5	15	64	8	12	4	11	35
temp →	4	11	35						

merge(4, 6, 9) (28)

	0	1	2	3	4	5	6	7	8
data →	1	5	15	64	4	8	11	12	35
temp →	4	8	11	12	35				

stack (TOP) ↓

merge(0, 4, 9) (30)

sorted

	0	1	2	3	4	5	6	7	8
data →	1	4	5	8	11	12	15	35	64
temp →	1	4	5	8	11	12	15	35	64

Quick Sort

Conquer

```
void qs_r(a, p, r) {
    if (p < r) {
        j = partition(a)
        qs_r(a, p, j-1)
        qs_r(a, j, r)
}
```

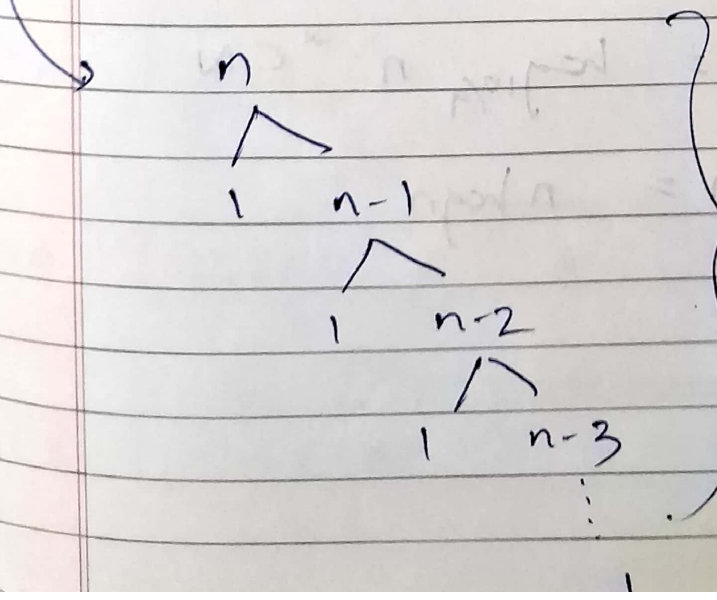
Time → O(n)
Inplace

$$T(n) = T(n/2) + \theta(n)$$

If partition is not correctly choosen, then partition = O(n)
conquering = O(n)

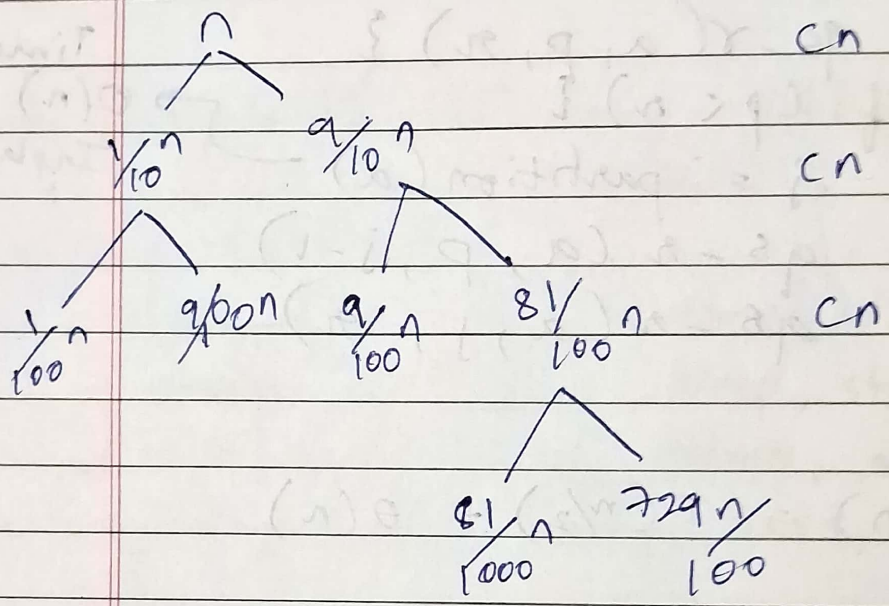
∴ Time Complexity = $O(n^2)$
worst case

but, best case (avg) = $O(n \log n)$



best case (avg):
90% unlucky 10% lucky

$T(n) = T(9n/10) + T(n/10) + c \cdot n$



largest depth = $\log_{10/9} n$

$T(n) = \log_{10/9} n * cn$

$O(n) = n \log n$

Time Complexity = Average: O(nlogn), Worst: O(n^2)
Space Complexity = Average: O(logn), Worst: O(n)

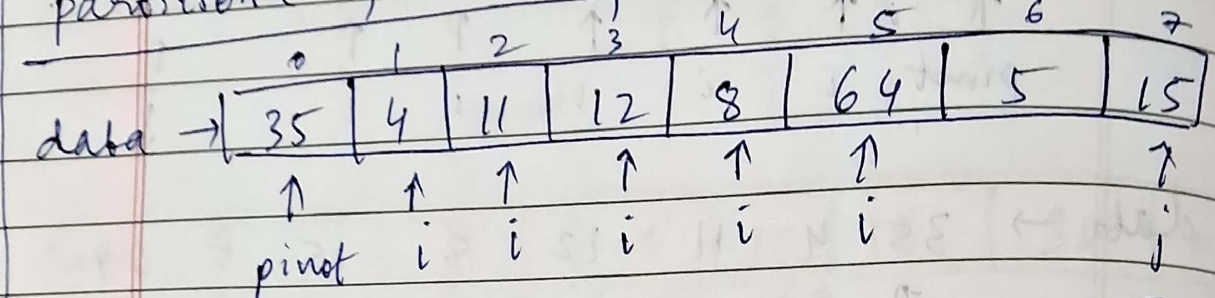
Quick Sort
n = 8

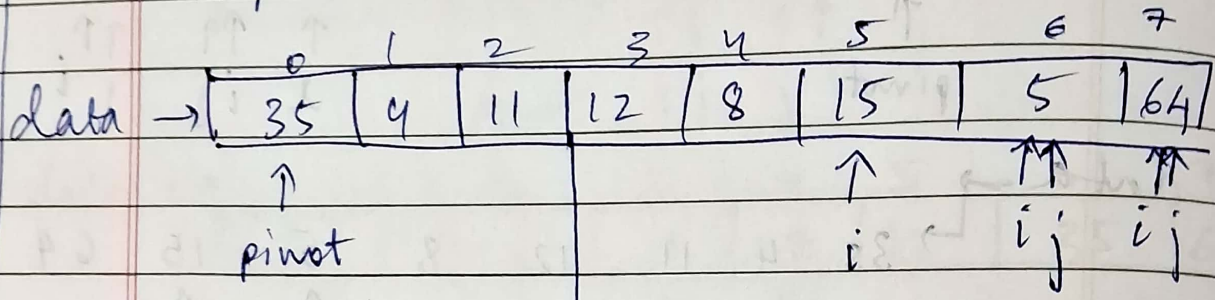
	0	1	2	3	4	5	6	7
data →	35	4	11	12	8	64	5	15

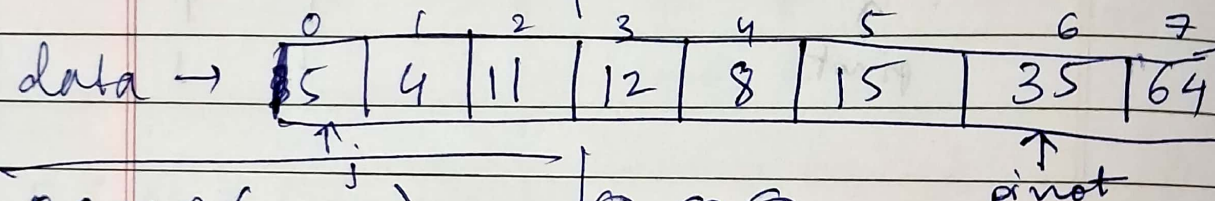
Stack (Bottom)

Q SortR(0,7) — ① ⑱ ⑳

partition(0,7) — ②

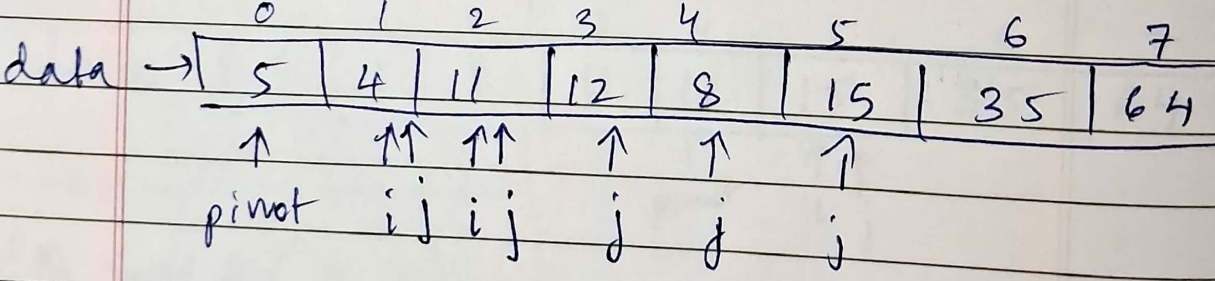


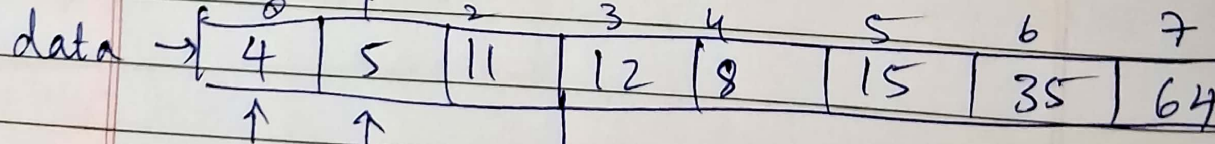




Q SortR(0,5) — ③ ⑤ ⑦

partition(0,5) — ④

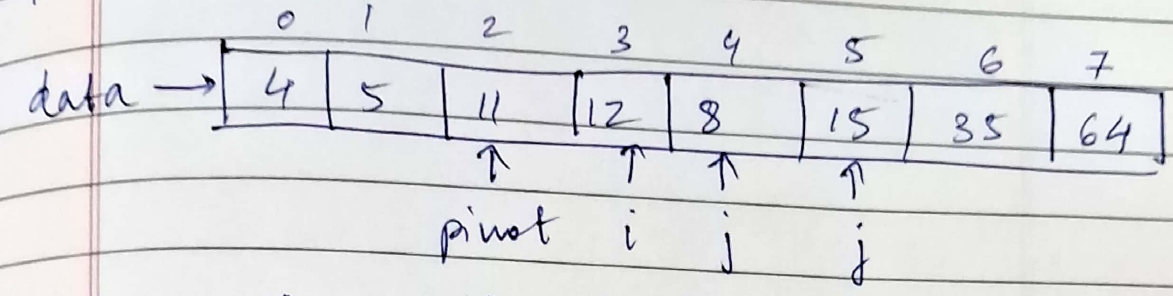


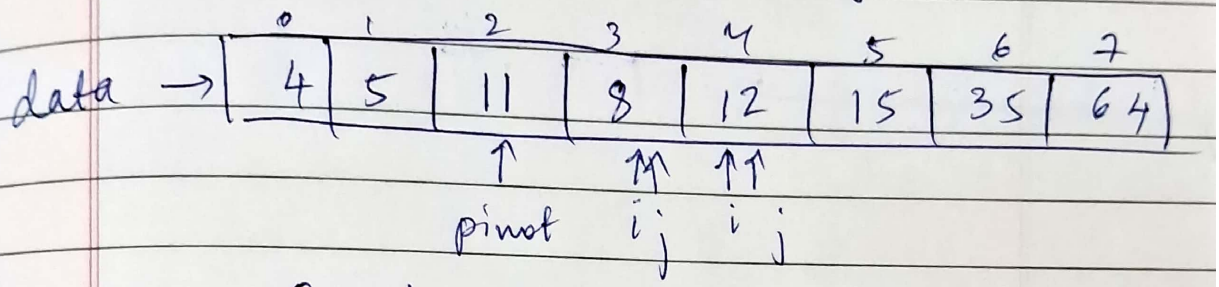


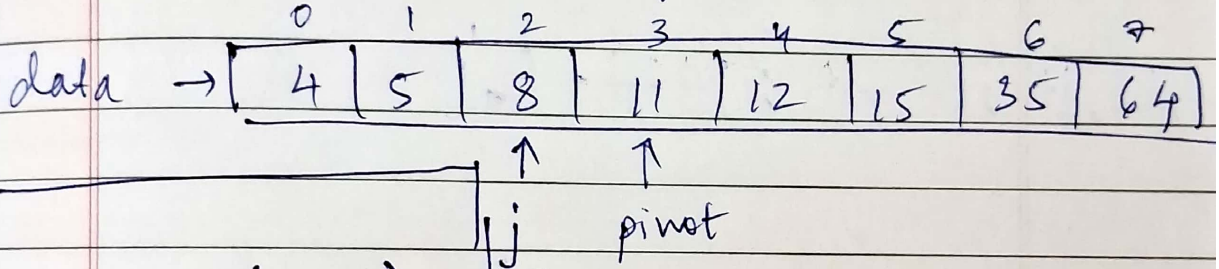
Q SortR(0,0) — ⑥

Q sort R (2, 5) (8) (11) (17)

partition (2, 5) (9) (15)



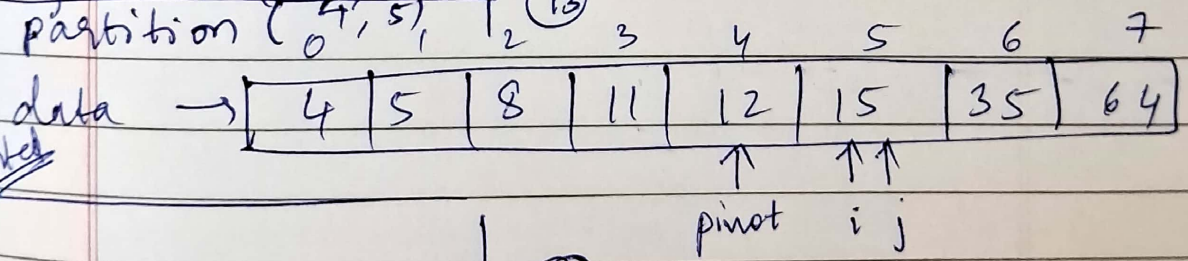




Q sort R (2, 2) (10)

Q sort R (4, 5) (12)

partition (4, 5) (13)



Sorted

Q sort R (5, 5) (16)

Q sort R (7, 7) (19)

Stack (Top)