finding half → $\frac{xyz}{2}$ → $xyz >> 1$

$low + (high - low)/2$

$low + (high - low) >> 1$

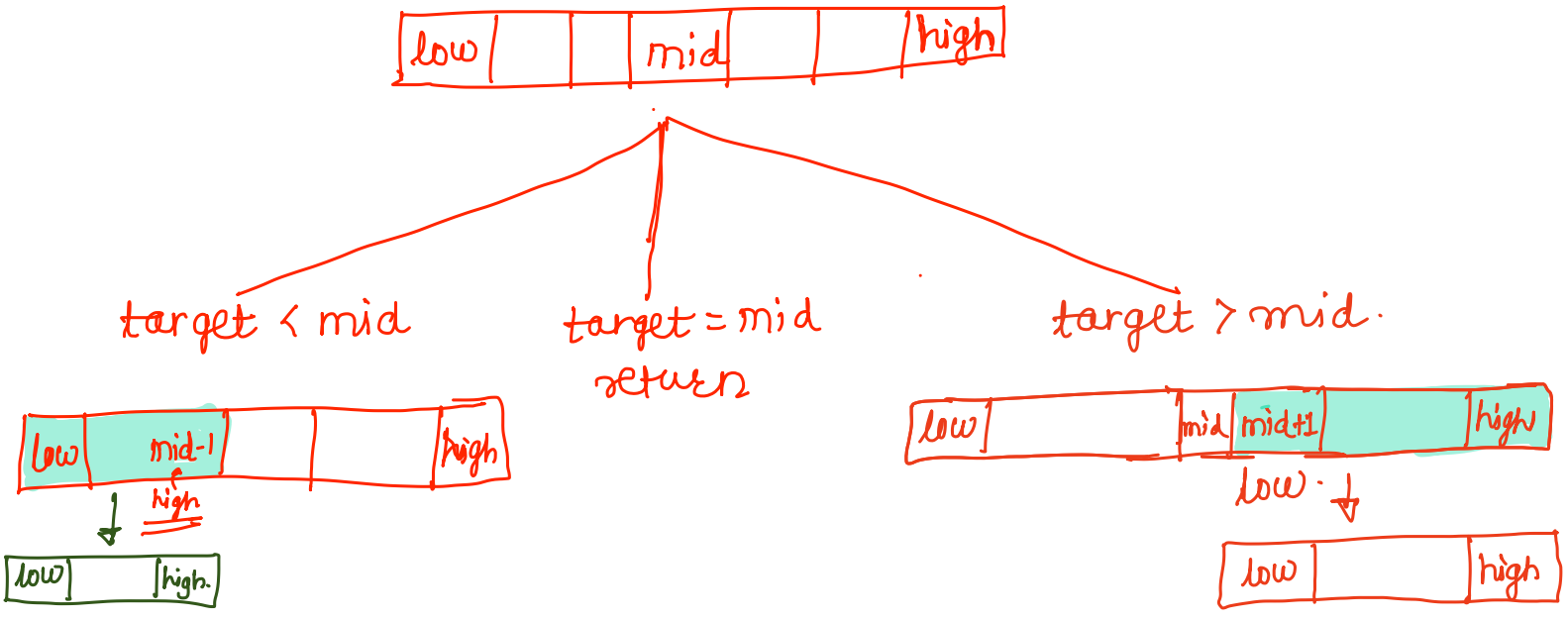
① Binary search -

Given → sorted array →

as the array is sorted.

lower indices → smaller numbers.

higher indices → greater number.



(2) Seach in 2D matrix

Given - 2D matrix -

1	2	4	8
10	11	12	13
15	16	17	18
22	24	26	28

Given → each row is sorted.

we traverse each row in matrix

pass each row as an array in binary search.

BinarySearch (row, target).

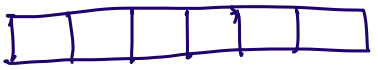
BinarySearch ([1,2,4,8], 17)

Binary Search ([10,11,12,13], 17)

Binary Search ([15,16,17,18], 17) → true

Binary Search ([22,24,26,28], 17)

③ Koko eating banana -

Given → array → 

hrs → x.

we have to decide

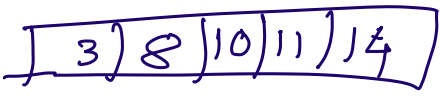
speed at koko to eat Banana

tricky thing

once koko starts eating a pile

it has to spend full hr

Suppose -



speed = 4

even if 3 < 4 → koko would spend full one hr here

for 10 → 10/4 → 2.5 → so koko would spend 3hrs

that way we take ceiling of division

As we know

once koko start with one pile

it has to spend 1 hr on it no matter what.

so maximum speed it could achive. is

max in array

lowest speed it could get is 1

so, we have to do binary search from 1 to max.

& our target would be given hrs

Also, if we dont get matched with exacts hrs

we have to return max speed.

(4) Find minimum in rotated ~~sorted~~ Array

Given → [3, 4, 5, 6, 1, 2]

3, 4, 5, 6, 1, 2
low mid high

6 1 2 3 4 5
low mid ~~high~~

mid > high. { low = mid + 1

3, 4, 5, 6, 1, 2
low high

6 1 2 3 4 5
low high

6 1 2
low high

we apply binary search
just a tweek

⑤ Search in rotated sorted array:

Binary search -

if we find element in middle → just return

check if left part of array is sorted →

we see if the number is between the left part of array

if yes → reduce the workspace

else

see if its in right part of an array.

if it is → reduce work space.

target = 6

3, 4, 5, 6, 1, 2.

low mid high

⑥ Time based key-value store -

key → Integer

value → strings.

we want → get (IF No match, give smaller close value)

↘ set.

suppose,

we are given '7' like get(7)

but we dont have '7'

we would search for any value less than 7

like 6, 5, 4,

for that we need our map to be sorted.

so, we make a treemap

foo → Treemap {

key

1 → "apple"

4 → "banana"

7 → "cherry"

}.

for every word we have treemap.

foo →

treeMap

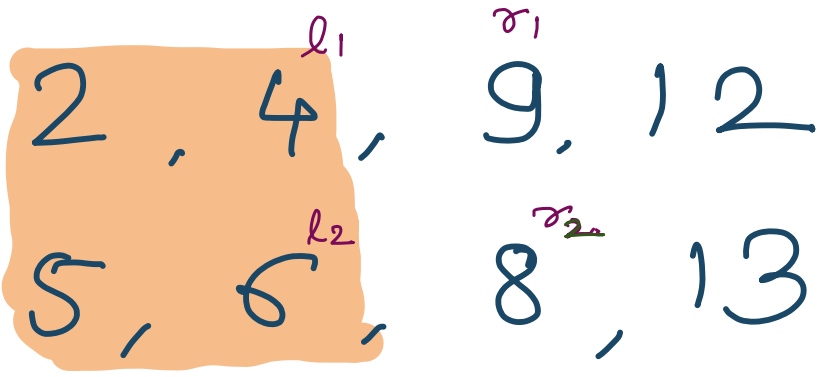
key	value
1	apple
4	Banana
5	kiwi

drink →

treeMap.

key	value
10	7up
12	Fanta
14	Maza

⑦ Median of two sorted arrays.



In order to have correct partition

$l_1 <= r_2$

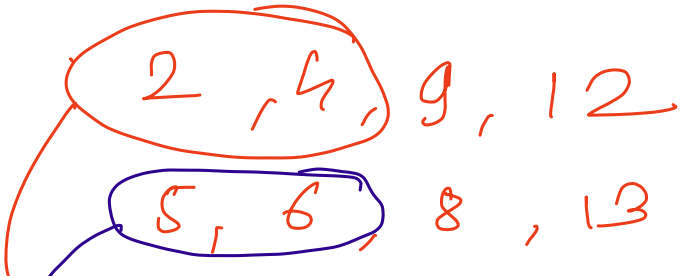
$l_2 <= r_1$

we find the smaller array & try to apply binary search on it.

for applying binary search on first array

mid1 = (low + high) / 2

& mid2 = would be the remaining element of that half.



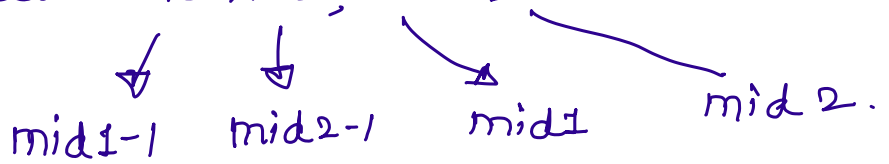
If we choose these two at first half

→ we have to choose these two

because partition should be 4-4

① find mid of both array

② decide l_1, l_2, r_1, r_2



③ Apply condition for perfect partition

$l_1 <= r_2$

$l_2 <= r_1$

If satisfied-

Odd elements → $max(l_1, l_2)$

even elements → $max(l_1, l_2) + max(r_1, r_2)$

④ If the partion is Not perfect

$l_1 > r_2$

high = mid - 1.

else

low = mid + 1.