if a sorted array is given, try binary search first.

-: Binary Search Questions :-

Problem 1:

Ceiling of a no.

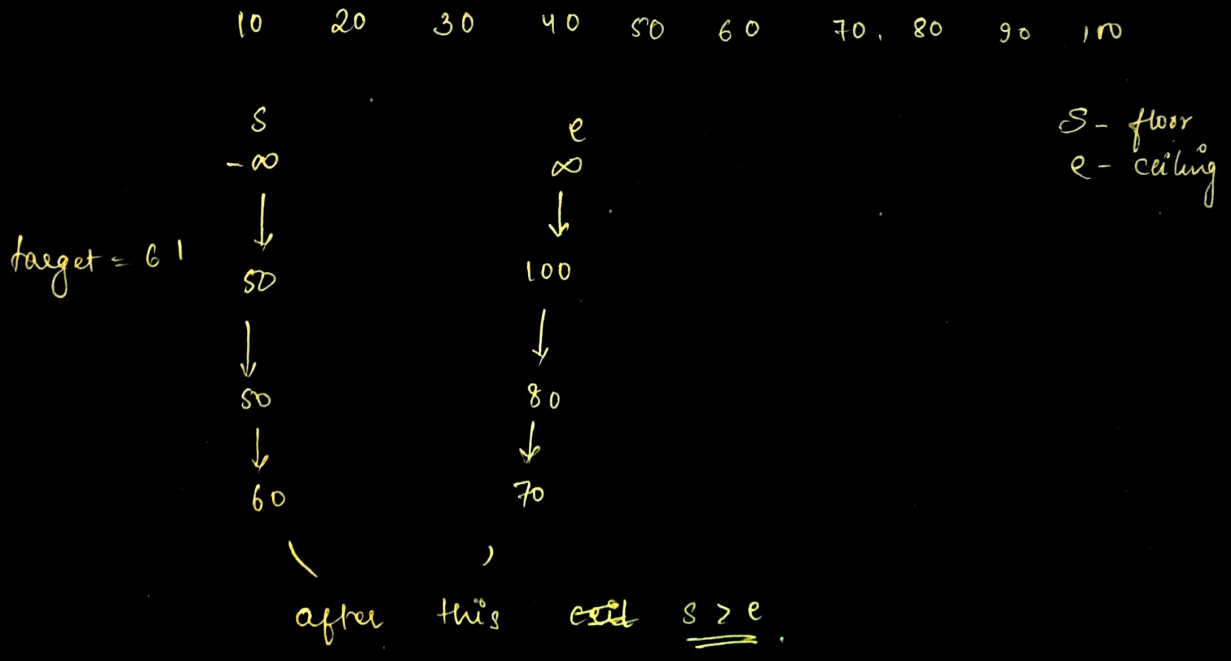
arr = [2, 3, 5, 9, 14, 16, 18]

target = 15. here ceiling = 16

Ceiling : smallest element greater or equal to target element.

ceiling (arr, target = 14) = 14
ceiling (arr, target = 15) = 16,
target = 4 ⇒ 5.

* if element is not found, return smallest greater element.



Algorithm.

1) start helps us locate the ~~ceil~~ floor
2) end helps us locate the ~~floor~~ ceil
3) we first assume both ceil and floor at $+\infty$ and $-\infty$.
4) when given the array range adjust between s and e.
5) while searching 61,
   range adjust from 50 (mid) to 100 (end)
6) further range adjusts from 50 to 80
7) ultimately we have 60 (↑ floor) and 70 (↑ ceil),

after which $\underline{\underline{s \geq e}}$.

Problem 2:

Floor of a number.

floor: largest element smaller than or equal to target.

* algorithm is already written above.

Problem 3

Find smallest letter greater than given letter.

i) exact same approach as ceiling problem.

ii) the equal to case gets ignored.

iii) arr = ['c', 'd', 'f', 'j']

target = j

start = end + 1 → condition violated.

↓

length of array.

return <u>s % N</u> → because in case the element is the last one then the first element is returned

else s only remains since it is < N.

Problem 4.

Return 1st and last index of a given element in sorted array.

1)

	s			m.					e.
	↓			↓					↙
	10	10	10	20	20	30	30	30	40	40
	0	1	2	3	4	5	6	7	8	9.

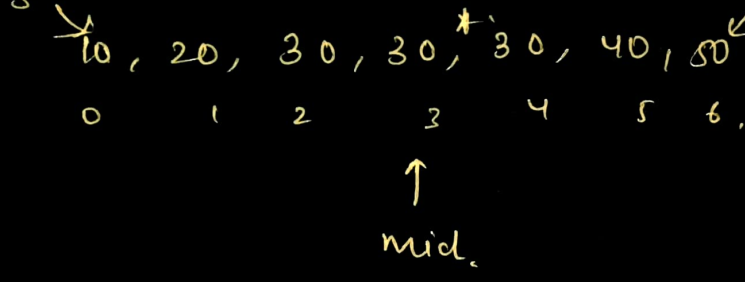
target = 30

2)

s		m.		e
↓		↓		↙
30	30	30	40	40
5	6	7	8	9.

↑

can be a potential ans array but not guaranteed.



target = 30

if target == mid.

↓

potential (not guaranteed) answer.

```
{
    fi = mid;   // save the value for first index
    e = mid - 1;
}
```

↳ move left to look if there is an occurence in the left.

for last index

everything remains same.

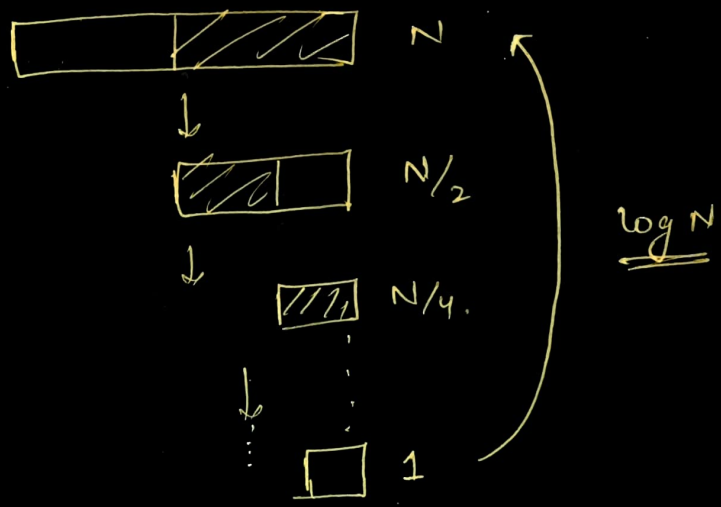
start = mid + 1

↳ move right and check for last occurence.

Problem 5:

Position of element in infinite sorted array.

→ move in chunks and then find start and end.



* after every step, double the size of array and check if it lies inside or not.

* just do opposite of what is done in binary search and then apply binary search

2, 3, 5, 6, 7, 8, 10, 11, 12, 15, 20, 23, 30

target = 15.

1) s e
   2 3

2) s e
   5 8

3) s e
   10 30

↓

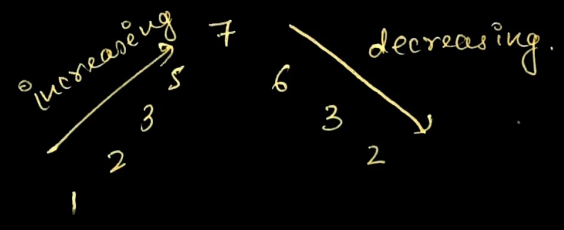
apply binary search.

* code yourself.

Problem 6 :

Peak Index in Mountain Array.

Mountain Array → Biatonic Array.



arr = [0:1, 1:2, 2:3, 3:5, 4:6, 5:4, 6:3, 7:2]

s m e

Possibility

el → element.

i) if el[mid] > el[mid+1]

↳ decreasing part of array.

e = mid → because we aren't sure if an answer exists in the left.

// checking L.H.S.

ii) el[mid] < el[mid+1]

↳ ascending part of array.

s = mid + 1

↳ because we are sure that there is an answer on the right.

iii) when will loop break

↳ when s and e point to the largest no..