* Binary Search:

- It can only be performed on a sorted Array.
- Array can be sorted in any manner. (Ascending/descending)

Algo:- (Ascending order sorted Array)

1) • Take the middle element of the array. (If middle = target, ✓)

2) • Compare middle element with target.

If (>) → search right else left.

(* The search gets halved / divided)

3) • If (>) the right side of Array becomes new search and takes the new middle element.

Repeat step 2 & 3 till element found.

Eg: S, E

2, 4, [6], 8, 10, 12 ; t = 12

mid = $\frac{0+5}{2}$ = 2.5/8

S, E

6 < 12 => 8, 10, 12 => 12 > 10 => 12 found.

mid

4) If (start > End) => Element not found.

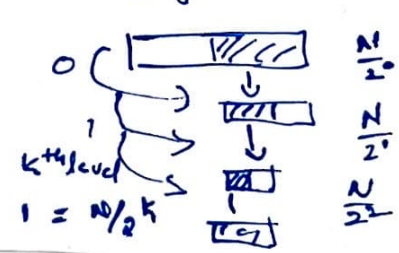
* Why Binary Search? much better than Linear Search.

- only 1 step in best case => O(1)

Best Case

Worst case: $k = \log_2(N)$

(Apply search max. comp.)



$N = 2^k$

$\log(N) = \log(2^k)$

$k \log 2$

$\Rightarrow \log_2(N)$

// better way to find Middle element

* $m = \frac{s+e}{2}$ (But, if large values, may exceed int range).

* $m = s + \frac{(e-s+e)}{2}$ or $\frac{(e-s)}{2}$

Pseudo Code:-

```
while (s <= e) :-
    if (mid < target) => s = m+1;
    if (mid > target) : e = m-1;
    if (mid = target) : return m;
```

* Order - Agnostic Binary Search :-

- when you dont know if array is sorted in ascending or descending order.

Algo to find if Array is ascending or descending :-

// Assume Array is sorted.

if (start > end) { descending

else Ascending.

- you can take any two elements but two elements can be same so to prevent such prob., above is best.

if (start = end) => Array contains only one number.

* For Any Sorted Array, Always Binary Search first //.

Few problems based on Binary Search :

a) find ceiling of a given no.

Ceiling = Smallest element in array greater than / equal to target.

Perform binary search

Just return start = end+1 & or start;

why?

```
 0   1
13  14  16  17  18 ,  target = 15
 S      m       e
```

m = 16

target < m {

so end = m-1;

}

```
 0   1
13  14
 S   e
```

start = mid+1

ans supposed to lie

while loop breaks here.

(S > e) { so start = ceiling

```
 S  (ans)  e
```

ans = S

return -1 if the target is last element or larger than last element.

Another example

```
 0  1  2  3  4  5  6
 2, 5, 9, 13, 16, 19, 23.
```

target = 11

m = 13

m > target {

end = m-1;

}

```
 2  5  9
 S  m  e
```

ceiling

target ≤ mid

start & end are just two pointers where among this answer:

$ (ans) ∈

But here,

∈ (ans) $ // condition violated

we need no. greater than target / equal

Hence, return start.

start = end + 1; or return start.

when while (start <= end) is violated {

return start = end + 1; //

Q2) Find floor number

floor = Greatest no. i.e smaller or equal to target.

0 1 2 3 4 5 6
2, 3, 5, 9, 14, 16, 18

$ (ans) ∈

∈ (ans) $

But we need greatest no. smaller than target,

Hence return end; //

∈ (ans) $
↳ calling
floor

// when loop violated,

d3) Smallest letter greater than target.

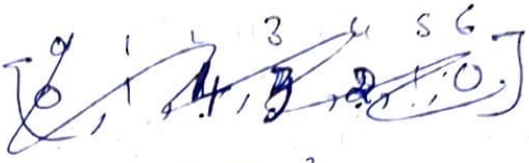
a b c d (e)

e

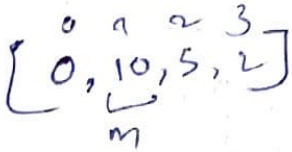
m < t

Same as ceiling floor but not equal to target.

Peak index of mountain



arr[mid] > arr[mid+1]



arr[mid] < arr[mid+1]

---

d) Find first & last position

Algo:

Apply Binary Search.

- if mid == element required {

  (if searching for 1st occurence)

  end = mid - 1;

  else {

  (last occurence)

  start = mid + 1;

return ans;

initially ans = {-1, -1}, so func can be search(arr, target, boolean)

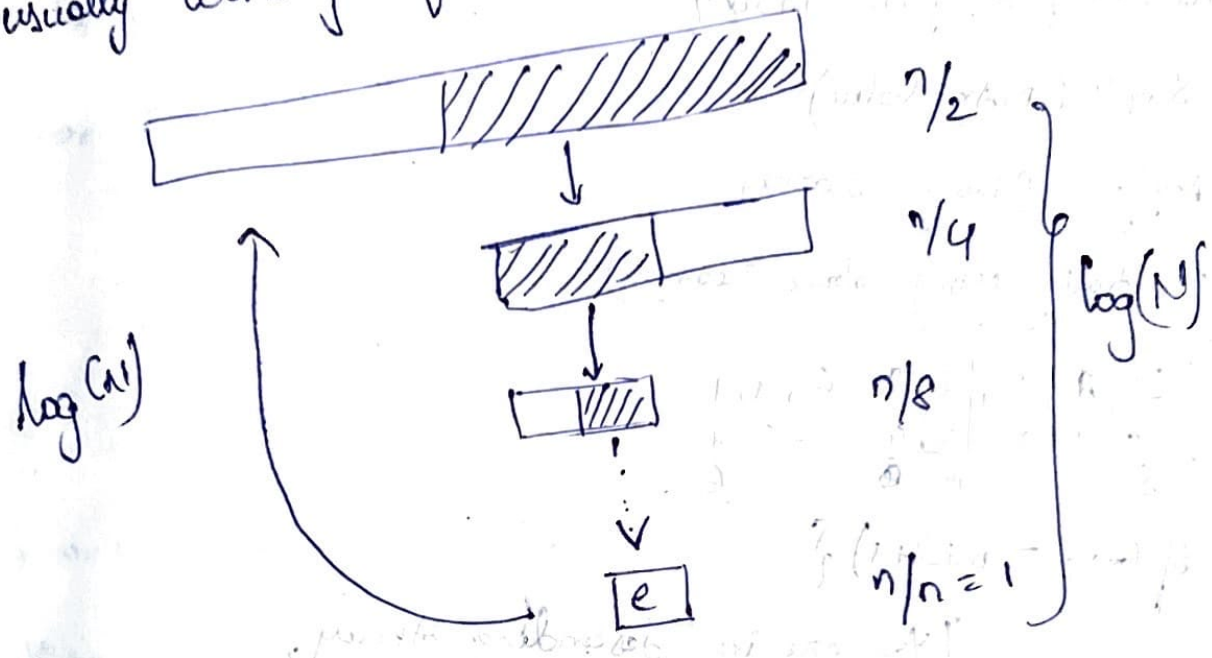
Q5) Search in infinite Array.

Q5.1 Solve without using 'var-name.length - 1'.

- So find the start and end of Array chunk.
- Then search that chunk of array for the target element.

How?

usually working of BS on Array.



Instead of approaching from Top, we can go from bottom to top. (bottom-up approach)

which means instead of dividing, we shall multiply the size of the box.

i.e take start = 0, end = 1

And keep doubling the size of chunk of array until target is less than end.

Then return BinarySearch(arr, target, start, end)

60) Peak index or peak value index in a mountain Array.

Ans:-

Bitonic / mountain Array means Array is ascending then descending.

0, 2, 3, 7, 8, 6, 4, 1

→ ←

How to find peak index?

Simple (MAx value)

Apply Binary Search.

(Illustrating using above example)

0, 2, 3, 7, 8, 6, 4, 1

s m e

```
if (mid > mid+1) {
    // You are in descending Array,
    end = mid;
    if (mid < mid+1) {
        //you are in Ascending order
        start = mid+1;
```

8, 6, 4, 1

4 5 6 7

s m e

s e

8, 6 m = 4

8 e

m

=) Start / End point o same element

=) peak index element.

d) Find in mountain Array.

Ans. Since mountain Array

Algo:

- Find the peak element.
- write function to search (BS) in order-Agnostic manner.

peak element allows us to determine which BS to perform and among which indexes.

for example, if (peak) > target {

target lies on left (perform BS from start to peak)

else {

otherwise //.

Q) write a function to get peak, then binary search then a final function that uses them both to get the answer.

---

28) Search in Rotated Array.

Ans) what is rotated Array?

If 2, 4, 6, 8, 9, 13, 17

After 1st rotation => 17, 2, 4, 6, 8, 9, 13

After 2nd rotation => 13, 17, 2, 4, 6, 8, 9

This is called rotation of Array,

How to search in rotated Array?

- Find pivot (usually the largest no. & from where the succeeding nos are in ascending)

How?

case 1:- mid > mid + 1 {return mid as pivot}

mid - 1 > mid { " mid - 1 " }

if (start < mid) { end = mid - 1 }

else { start = mid + 1 }

return pivot.

Call BS function to search the two halves.

BS (arr, target, start, pivot)

&

BS (arr, target, pivot + 1, end)

---

qn) what if duplicate elements are present?

Then start ++; } skip the duplicates.
end --; }

But check if the duplicates are pivot.

arr(start) > (start + 1)

Similarly for end --;

iii) Rotation Count of Array. Simply

Find pivot.

pivot + 1 = Rotation Count //

---

iv) HARD

Split Array, minimize largest Sum

The array given is not sorted.

Largest Sum ⇒ Sum of all elements

minimized largest Sum ⇒ max. value of Array.

This right here gives a range.

↳ [min.LS, LS]

Apply Binary Search

Get mid.

Now, Split the Array such that

Eg: 7, 2, 8, 10, 5

[10, 32]

n = 21

The sum of sub Array is not more than the mid element.

then, we get pieces.

Now, we check, if (pieces $\leq$ m) {

perform BS again; mid again.

}

In the end, the start, end & middle all point to the same element (Ans) //.

★ Binary Search for 2D Arrays :

- Basically, 2D Array are matrix / matrices

If matrix is sorted row-wise, col-wise, then how to perform BS

	0	1	2	3
0	10	20	30	40
1	11	21	31	41
2	12	22	32	42
3	13	23	33	43

* for search probs, always try to minimise the search area/space. ***

- Find start & end (Eliminate rows & columns)

In 2D Array,

- take [0][0] as lower bound and [0][3] (last column of row) as the upper bound.
- using this eliminate rows/columns & reduce search space.

Suppose target > arr[0][3] {

column--;

}

target > arr[0][3] {

row++;

}

* traversal, & complexity will only be $O(2n) \Rightarrow O(n)$ //.

---

d) what if the matrix is strictly sorted?

Eg:

10	12	14	16
18	20	22	24
26	28	30	34
40	44	49	50

- You can convert it into 1D Array and perform Binary Search.

(or)

- Take either middle row / middle column and perform binary search.

```
if (element == target) {
    return element;
}
if (element > target) {
    everything after the element will also
    be greater, so, ignore
    // ignore succeeding elements
}
if (element < target) {
    everything preceding the element is
    less than target, so ignore them.
    // ignore prev. row/col.
}
```

- This leaves only 2 rows/columns remaining.

→ Take mid of row/col again and perform BS.

→ if mid == target & return

or search across columns remaining. // find element (BS) or →

- In such question, it is possible matrix is of 1D,

  so be cautious ☆☆☆

```
r = matrix.length;
c = matrix[0].length; // Be cautious //.
```