Binary Search

① Implement lower bound
- youre basically finding the lowest index with value greater than or equal to X
- So just divide the whole array 2 for mid if mid value is greater than or equal to X store that as ans for now
- then decrease the search space to left since it can only be less than mid
- only if X > mid then decrease search space only for righ half

② upper bound - same as lower bound but index value should be greater than x

③ Search insert position of X
basically you just have to insert X in lower bound position.

__/__/__

(5) Floor - Find the largest number in the array
i.e less or equal to x
- So check with midpoint if mid is greater than x there might be elements ahead of mid that is smaller or equal to x.

(5.1) Ceil - Find the smallest number in the array i.e greater or equal to x.
- Check with midpoint if mid is <= x then there might be smaller elements on left so check left.

(6) Find first & last occurence of X
(i.e starting index of X & ending index)
- Binary search
- if arr[mid] = X
  - first index will be on left of mid
    So reduce search space to 1st half
    if not = mid then look for in either halves
  - last index will be on right of mid
    so reduce search space to second half

(7) find total occurence
- find first & last occurence & do
  last - first + 1 will give total count.

__/__/____

⑧ Search element in rotated sorted array

- basically first divide the array and check if mid is the element target
- if not now check if left half array is sorted if yes then ~~check~~ check if target lies in left half & reduce search space to left half
- else if right half is sorted check if target lies in right half & decrease search space
- if element doesn't exist in both halves return -1.

⑨ search element in rotate sorted array but it has Duplicates

- Same logic as previous one but
- but just skip over Duplicates by checking if low mid & high are equal.

__/__/__

(10) Find minimum in rotated sorted array

\# - first check which half is sorted
then get the minimum from that half
- then check for the other ~~half~~ half & get
the minimum there
- finally return the minimum.

(10) & (10).1 use this minimum & how many times array rotated.

\# - start by keeping min index = 0

\# - Basically 1st check which half is sorted
- then in that sorted half compare the min
value i.e arr[low] < arr[ind] or arr[mid] < arr[ind]
- if you find the minimum then update the
index
- then discard that half
- then finally return the index or arr[index];

1 2 3 4 5

~~0 1 2 3 4~~
3 4 5 1 2   this has been rotate
3 times which is min index

__/__/__

(18) Search Single element in Sorted array

- basically using binary search you have to check which half the missing element lies on
- i.e by checking the pattern
- when mid is odd reduce it to even & check even odd pair if its equal then elements before mid are in Even odd pattern so reduce space to ~~left~~ righ half
- if its not equal then ~~[illegible]~~ even odd pattern has been broken & single ele lies on left so high = mid
- & finally return arr[low].

E	O	E	O	E	O	E
0	1	2	3	4	5	6
1,	1,	2,	3,	3,	4,	4

(10) Single element in a sorted array

- While runs from low to high (low<high)
cuz when array[mid] = low or high
i.e if a[mid] = 0 or n-1 the execution will stop
or like if low = high execution stops
cuz that point will be the single element.

- you find the mid point
- now if mid is odd you reduce mid to prev index (mid--) making it even & you check if even = odd place - if this is true that means the even odd pairing hasnt been disrupted till mid point so single element will be on right part of mid (so do low = mid+2)

- else if when mid is odd & you reduce mid to even & check if even = mid & if its not equal that means the even odd pattern is disrupted already & the single element is on left half

- & finally when low = high return arr[low].

E O E O E O E O E
1 1 2 3 3 4 4 5 5

Pattern break

E O E O E O E
2 2 3 3 4 5 5

Pattern break