Binary Search

Binary Search

- a search algorithm to find a target value in a sorted array
- With each iteration, half the values are eliminated
- time complexity: $O(\log(n))$

Steps: We want to find the value 12

① Sort the array (if not already sorted) and assign left pointer to the start & right pointer to the end

arr = [1 4 5 12 15 16 19 24 30 33]

left, mid (4), right

② Find the middle index of the left & right pointers
- Can round up or down but stay consistent (rounding down seems most common)

$$(start\_idx + end\_idx) // 2 = (0+9) // 2 = 4$$

int division rounds down when arr length is even

4 = middle index

arr = [1 4 5 12 15 16 19 24 30 33]

left, mid (4), right

③ if the middle index value is equal to the target 12, return the index, if the value is greater than the target, exclude that value and everything greater in the array, if it's less than the target value exclude itself and everything less than in the array and adjust pointers

arr = [1 4 5 12 ~~15 16 19 24 30 33~~]

left, right, mid (4)

④ repeat the process until the target value is found OR until L ≤ R and the element is not in list

mid_idx = (0+3)//2 = 1

arr = [~~1~~ ~~4~~ 5 12 ~~15 16 19 24 30 33~~]

mid, left, right

mid_idx = (2+3)//2 = 2

arr = [~~1~~ ~~4~~ ~~5~~ 12 ~~15 16 19 24 30 33~~]

mid, right & left

mid_idx = (3+3)//2 = 3

arr = [1 4 5 12 15 16 19 24 30 33]

↑ left & mid

↑ right

Since the value of mid = our target, we can return the index of 3

Answer = index 3

time complexity = O(log(n)) because each iteration we're dividing the search space in half