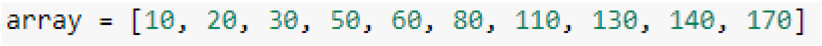
BINARY SEARCH ALGO

Note: If you want to find the index position of a given array in a highly optimized way, you can try using binary search.

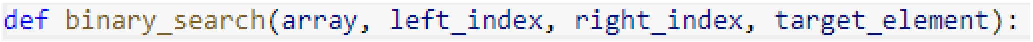
So we are going to follow the algorithm and technique and understand the flow.



```
array = [10, 20, 30, 50, 60, 80, 110, 130, 140, 170]
```

Suppose you have an array and you want to find the index of 110. How are we going to approach this problem? Let's take a closer look.

So basically, in binary search, we use four parameters. The first parameter is the array, followed by the left index, right index, and target element.



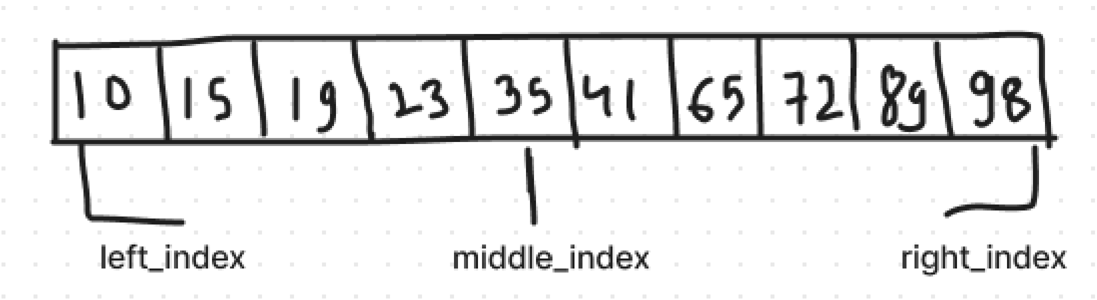
```
def binary_search(array, left_index, right_index, target_element):
```

Note: But always remember Binary search is not performed efficiently when the array is not sorted.

But if we are given a sorted array, then binary search works well and is even better than linear search.

in this algorithm we have to consider m also ie; middle index so that we have to calculate but inside the loop.

So this while loop will execute until left_index < right_index.

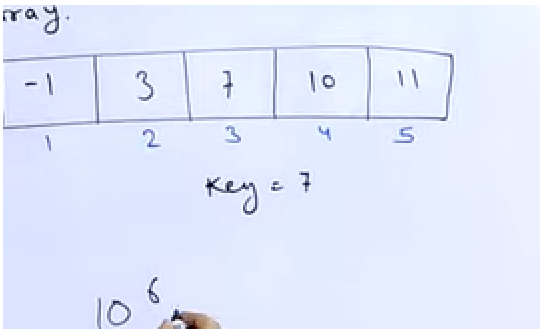


So it should not be overlapped with the `left_index` on the right side.

So, `left_index` can be less than or equal to `right_index`.

Note: `why binary search is better than normal looping concept` ?

because suppose if u have large set of array , if suppose you have

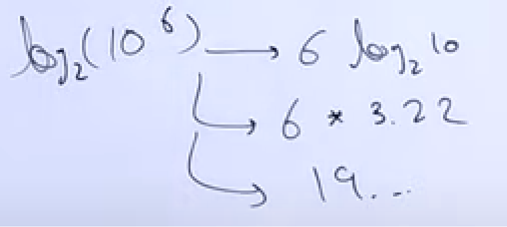


-1	3	7	10	11
1	2	3	4	5

Key = 7

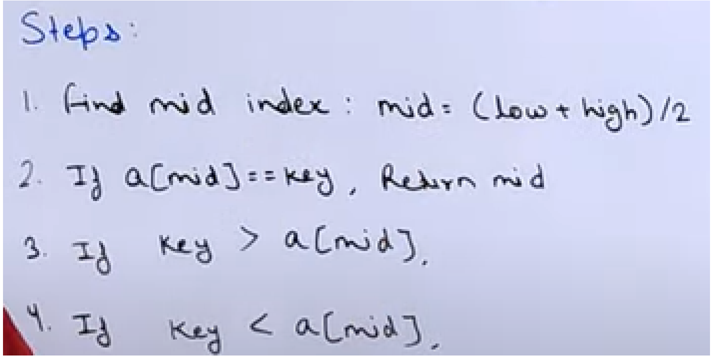
10^6

10^6. Suppose you need to find the largest index; in this scenario, it will iterate hundreds of thousands of times and then return the answer. However, if we use the concept of binary search…



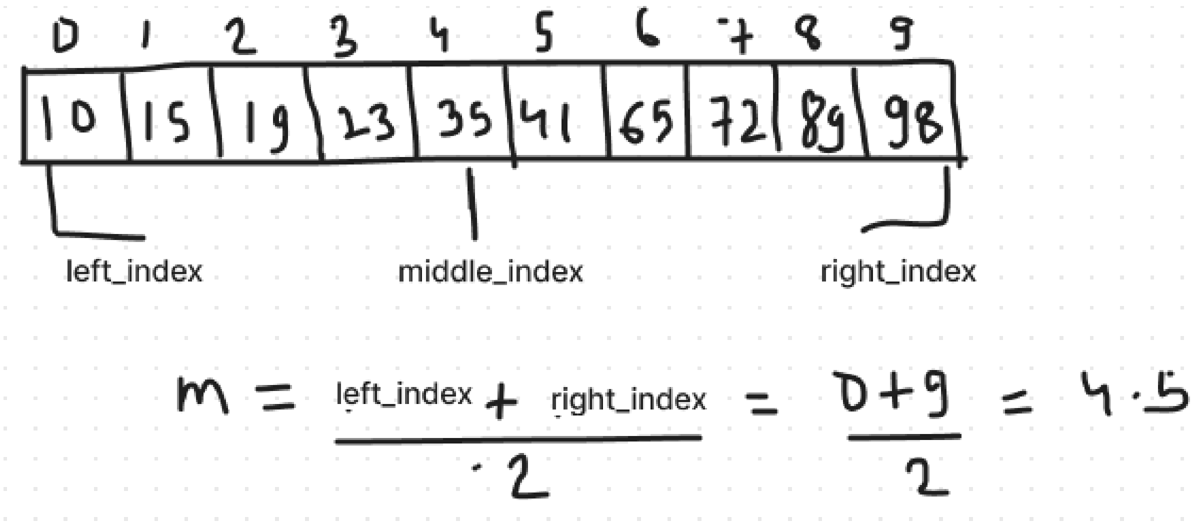
$$\log_2(10^6) \longrightarrow 6 \log_2 10$$
$$\hookrightarrow 6 * 3.22$$
$$\hookrightarrow 19...$$

Then we can fetch the exact value within approximately `20 operations`, but using a looping concept, it will take `100000 operations`.



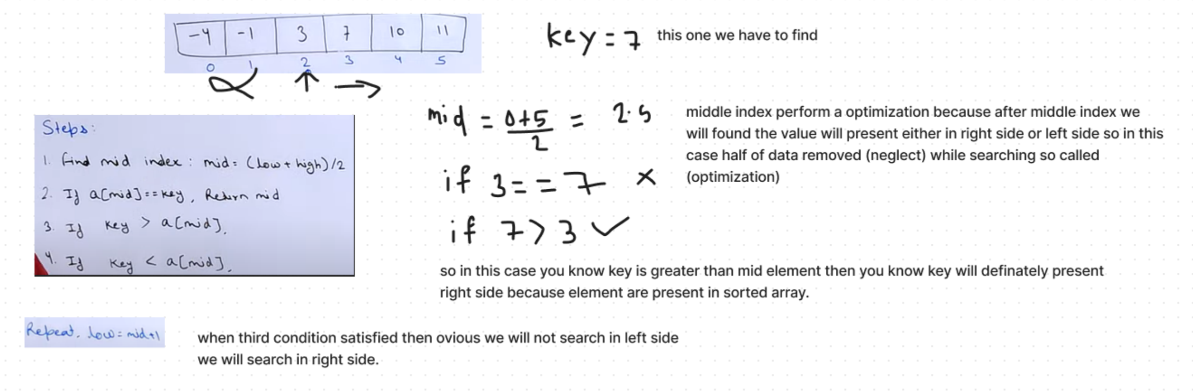
Steps:

1. Find mid index : mid = (low + high)/2
2. If a[mid] == key , Return mid
3. If key > a[mid].
4. If key < a[mid].



$$m = \frac{\text{left\_index} + \text{right\_index}}{2} = \frac{0+9}{2} = 4.5$$

Note: The middle index is always rounded down, meaning that anything after the decimal point is not considered. For example, `4.5 = 4`.



key = 7 this one we have to find

Steps:

1. find mid index : mid = (low + high)/2
2. If a[mid]==key, Return mid
3. If key > a[mid],
4. If key < a[mid],

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

if 3==7 ×

if 7>3 ✓

middle index perform a optimization because after middle index we will found the value will present either in right side or left side so in this case half of data removed (neglect) while searching so called (optimization)

so in this case you know key is greater than mid element then you know key will definately present right side because element are present in sorted array.

Repeat. low = mid+1

when third condition satisfied then ovious we will not search in left side we will search in right side.