Binary Search:

* It is is performed in a sorted array like ascending or descending order.

* It searches the element by finding the middle element the range of indices.

Algorithm:

1. Find the middle element
2. Check if target > middle element ⇒ search in right
   else search left
3. If the middle element == target element // Then that's the answer
4. If start > end , element not found.

Eg:-

$$arr = \begin{array}{ccccccccccc} & \text{S} & & & & & \text{S} & & & \text{S} & \text{E} \\ [& 2, & 4, & 6, & 9, & \boxed{11} & 12, & 14, & \boxed{20}, & 36, & 48] \\ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$

$\boxed{\text{target} = 36}$

Step:- (1) $\frac{0+9}{2} = 4$ ⇒ middle.

$\downarrow$

$\boxed{11}$

(2) 36 > 11 so

start = middle + 1

end = end

$\frac{5+9}{2} = 7$ ⇒ middle

$\downarrow$

$\boxed{20}$

(3) 36 > 20

start = mid + 1 end = end $\frac{8+9}{2} = 8$ ⇒ mid

$\downarrow$

$\boxed{36}$

(4) 36 == 36 answer found,

If target < middle

start = start, end = middle - 1. and find check mid.

Why we need Binary Search:

* When we find the target as first middle element, then it is a best case O(1).

* For every comparision, the search range reduce by 1/2. For every level the formula for time complexity $N/2^k$

$k \rightarrow$ level, $N \rightarrow$ size.

* $\frac{N}{2^k} = 1 \Rightarrow N = 2^k$

$\Rightarrow \log N = k \log 2$

$\Rightarrow k = \frac{\log N}{\log 2}$

$\Rightarrow k =$ ~~log N~~ $\log_2 N$

* The worst case is $O(\log_2 N)$

* Lets take the array size is 1 million.

Linear Search	Binary Search
1million comparisions	20 composisons

Order agnostic Binary Search:

* To check whether the array is sorted in ascending or descending.

if start > end → descending

else start < end ⇒ ascending

* We are not checking first two elements because
both

some time the numbers will be same.