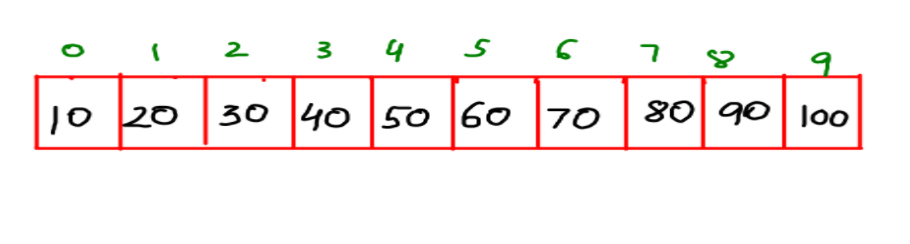
We are given an array arr[ ] of n elements and a number "d" that we are supposed to search in this array.

Let's assume the array is arr[ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ] of 10 elements and we are required to search element 20 in it.



What do you think should be the crudest way of searching for the required element 20 be?

Maybe you say, by comparing each element of the array with 20. You should know this method actually has a name! It is called the **Linear Search method**.

**HOWEVER.**Since it compares all the elements in an array, if we were to search an element which doesn't exist in the array, say 200, then we would have to iterate through all the elements till the end and this makes the performance of this method poor especially if the length of the array were larger , like 1024. The loop would have had to run 1024 times in that case with a time complexity of O (n).

Reader, to counter this problem we use the **Binary Search method**.

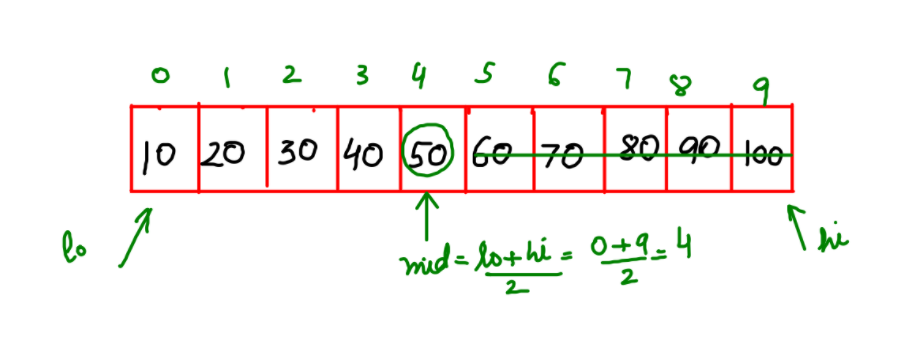
You should know that the Binary Search is applied only on sorted arrays.

WHAT & WHY

If you know how to search for a word in a dictionary, this will be a piece of cake for you. If you want to search the word cheese" in the dictionary and you land on a "m" page then you search for the word in pages with alphabets before "m" , right?

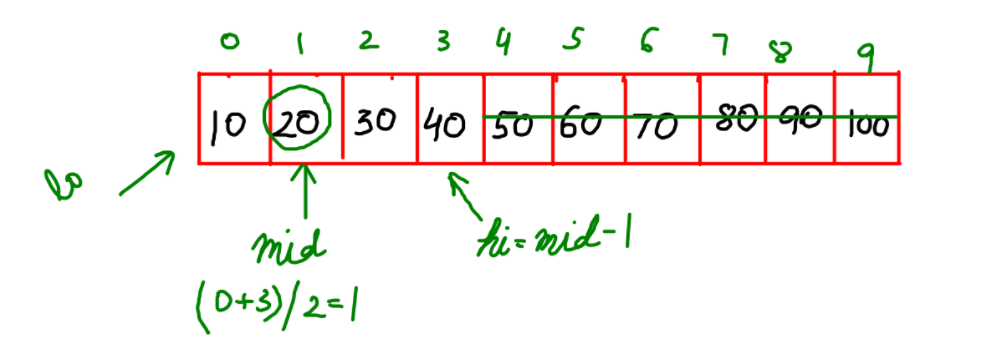
The Binary Search method is the same. Initially, we mark a "lo" and a "hi" pointer at the start and end indices of the array and find a "mid" index by taking the average of "lo" and "hi" i.e. mid=(lo +hi)/2. If we find the required element at the mid index then hooray!, we terminate the program else we accordingly change the values of lo and hi .

Let's understand it through the previous example of finding 20 in the array.



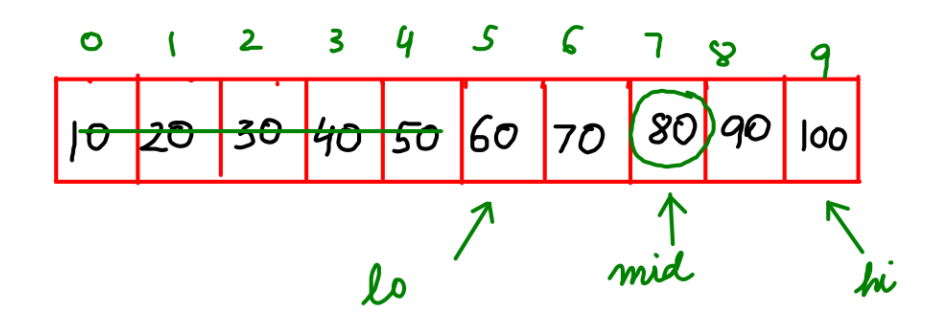
Here, when "lo" is at the 0th index and "hi" is at the 9th index then their mid is (lo+hi)/2= (0+9)/2 =4. Hence, "mid" points at the 4th index. On comparing 20 with the value at 4th index i.e. 50 we find that 20<50 so it's no use searching elements larger than 50.

Now, we find a new mid in the area less than 50 by putting hi=mid-1=4-1=3 . Then , mid=(lo+hi)/2 =(0+3)/2=1.



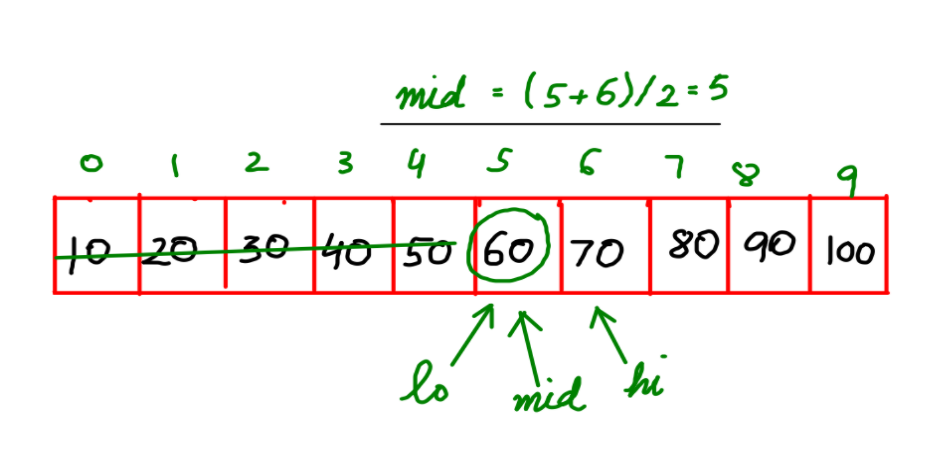
Since, 20 matches the value at mid=1 , therefore we return "true" and terminate the program. Had 20 not matched with the mid we would have continued the loop as long as an existing index of array is found by changing either the "hi" or the "lo".

Let's take another example for better understanding. This time we need to find 60.



Here, as in the previous case, our first mid was 50. BUT , in this case the element to be searched, 60 ,is greater than mid=50 ,so we need to constrict our search in elements after 50. For that we take the new value of "lo" as mid+1= 4+1=6 and "hi" remains the same.

Now the new mid is (6+9)/2=7. The value at mid=7 i.e. 80 is greater than the value to be found i.e. 70. So our new value of hi=mid-1 = 7-1= 6.



Here, the mid value comes out to be 5 as shown in the figure. Since 60 matches with the value at mid=5 , therefore we have found our required number.

HOW

Now that we have conquered our "WHAT" and "WHY" , **"HOW"** isn't going to be much of a bother.

In conclusion, if the value to be searched is "d" ,

Then three cases arise.

**Case I :** d< mid ?? hi=mid-1 , lo remains same

**Case II :**d>mid ??lo=mid+1, hi remains same

**Case III :**d=mid ?? return true and terminate the program.

Analysis

**Time Complexity:**

**O(log2n)**

The time complexity of the binary search algorithm is O(log2n).