Search

Linear Search

You can perform a search for an array element based on its value or its index.

Algorithm

Consider **LA** is a linear array with **N** elements and **K** is a positive integer such that **K<=N**. Following is the algorithm to find an element with a value of ITEM using sequential search.

```
Search (LA,N, ITEM)
1. Start
2. Set J = 0
3. Repeat steps 4 and 5 while J < N (Repeat step 4 and 5 For j = 0 to N-1)
4.    IF LA[J] is equal ITEM Then Goto Step 6
5.    Set J = J +1
6.  If J=N then Print "Number Not Found" Else Print J, ITEM
7. Stop
```

Challenges:

- Best Case : Found on 1st Position (complexity = 1)
- Worst Case : Found on Last Position (Complexity = n)
- Average case : Complexity = n/2
- **Given :**

- The Tycho-2 star catalog contains information about the brightest 25,39,913 (Approx 2.5 lacs) stars in our galaxy. Suppose that you want to search the catalog for a particular star, based on the star's name. When the program examined every star in the star catalog in order starting with the first, an algorithm called **linear search and** the computer might have to examine all 2,539,913 stars to find the star you were looking for, in the worst case.
- **Note : The given Dataset is sorted**

- **Motivation for Binary search**

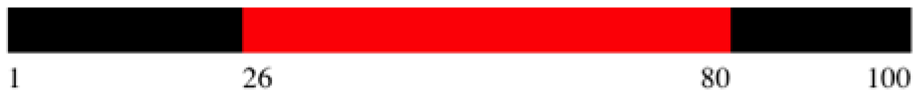
Number Game:

100 Numbers are stored in sorted manner. Player has to find the number guessed by the computer asking some question to computer

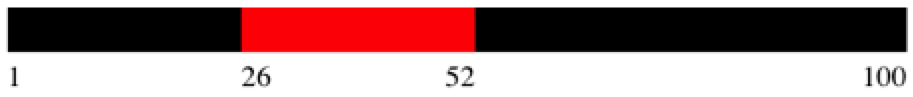
How to play (28)

- If you guessed 25 : Computer replied it is lower
- If you guessed 81 : Computer replied it is higher
- So your number is lying between 26 and 80

Here, the red section of the number line contains the reasonable guesses, and the black section shows the guesses that we've ruled out:



- In each turn, you choose a guess that divides the set of reasonable guesses into two ranges of roughly the same size. If your guess is not correct, then computer will you whether it's too high or too low, and you can eliminate about half of the reasonable guesses.
    - For example, if the current range of reasonable guesses is 26 to 80, you would guess the halfway point, (26 + 80) / 2 = 53.
    - If I then tell you that 53 is too high, you can eliminate all numbers from 53 to 80, leaving 26 to 52 as the new range of reasonable guesses, halving the size of the range.

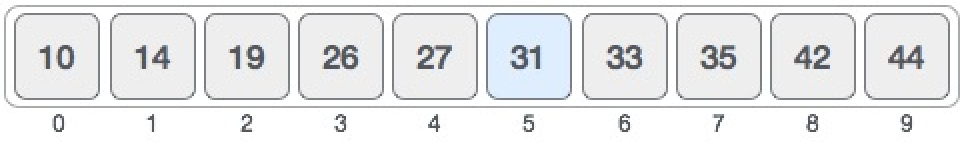


- Initially variable max, min are the boundary of the list.
- **step-by-step description of using binary search to play the guessing game:**

1. Let min = 1and max = n
2. Guess the average of max and *min and* rounded down so that it is an integer.
3. If you guessed the number, stop. You found it!
4. If the guess was too low, set min to be one larger than the guess.
5. If the guess was too high, set mx to be one smaller than the guess.
6. Go back to step two.

How Binary Search Works?

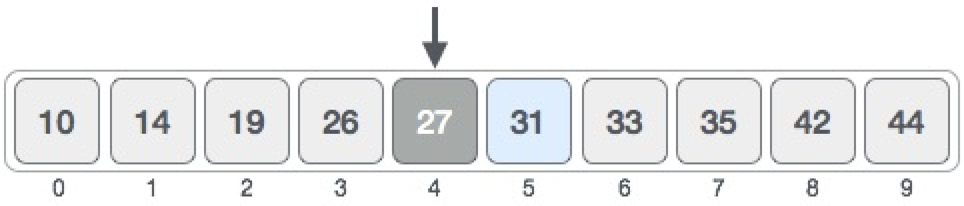
For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to **search the location of value 31** using binary search.



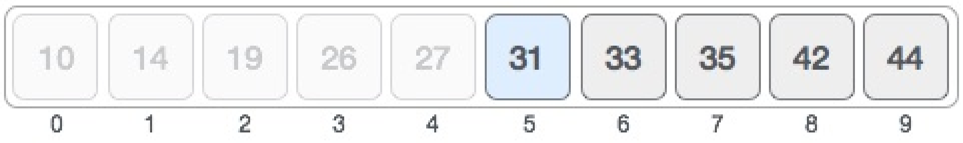
First, we shall determine half of the array by using this formula −

mid = (high - low) / 2 ………[mid = low + (high - low) / 2]

Here it is, (9 - 0) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.



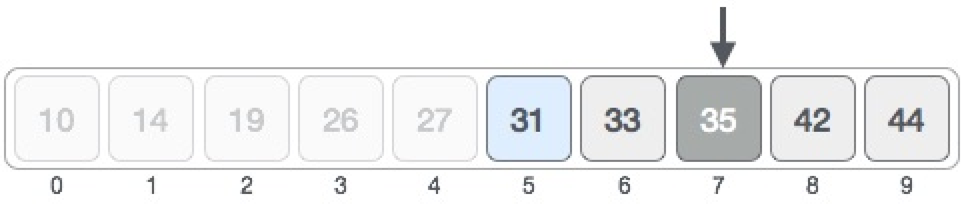
Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.



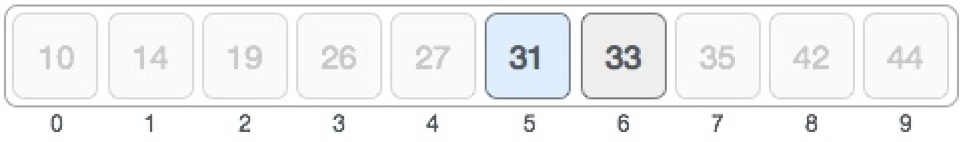
We change our low to mid + 1 and find the new mid value again.

low = mid + 1
mid = low + (high - low) / 2

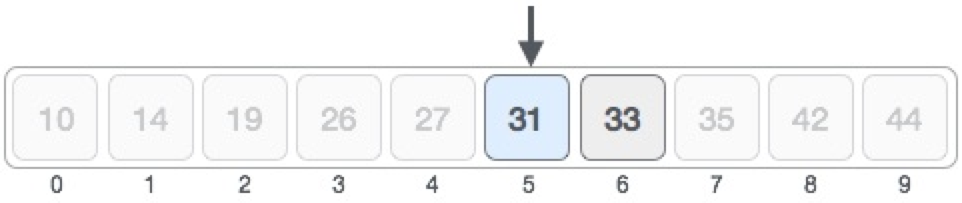
Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



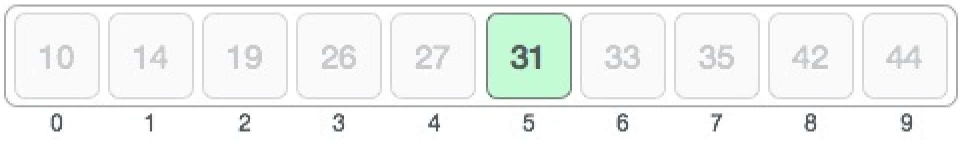
The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.



Hence, we calculate the mid again. This time it is 5.



We compare the value stored at location 5 with our target value. We find that it is a match.

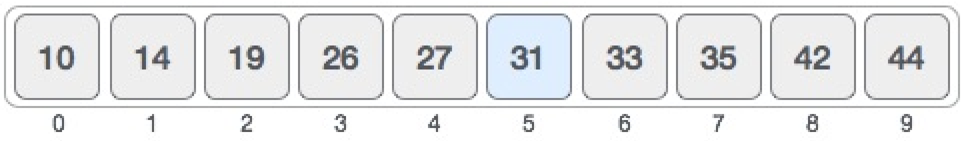


We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Practice

Search 28



Min = 0   Maz = 9

1. Mid = 4 … 27   Compare Fail

2. Min =5 Max = 9
Mid = 7 … 35   Compare Fail

3. Max = 6 Min = 5
Mid = 5   Compare Fail

4. Max = 4, Min = 5
Mid = 5   (Compare Fail)

5. Max = 4 so stop (Max < Min)

Result …. Not found

Algorithm

The pseudocode of binary search algorithms should look like this −

```
binary_search (A,n,x)
  A ← sorted array
  n ← size of array
  x ← value to be searched

1.  Set lowerBound = 1
2.  Set upperBound = n
3.  while x not found ……. [UB>=LB]
      4.  if upperBound < lowerBound
          a. EXIT: x does not exists.
    5.  set midPoint = lowerBound + ( upperBound - lowerBound ) / 2
    6.  if A[midPoint] < x
          a.  set lowerBound = midPoint + 1
    7.  Else if A[midPoint] > x
          a. set upperBound = midPoint - 1
    8.  Else ……….[if A[midPoint] = x ]
        a.  Print x "found at location" midpoint, EXIT
9.  end while
10. end procedure
```

Running time of binary search Google ClassroomFacebookTwitter Email

- Linear search on an array of n elements might have to make as many as n guesses.
- Binary search halves the size of the reasonable portion upon every incorrect guess.

- For an array of length 8,
  - Incorrect guesses reduce the size of the reasonable portion to 4, then 2, and then 1.
  - Once the reasonable portion contains just one element, no further guesses occur;
    - the guess for the 1-element portion is either correct or incorrect, and we're done.
  - So with an array of length 8, binary search needs at most four guesses.

- For an array of length 16,
  - Incorrect guesses reduce the size of the reasonable portion to 8, then 4, then 2, and then 1.
  - So with 16 elements, we need at most five guesses.

    16 = 2 ^ 4
    $4 = \log_2(16)$

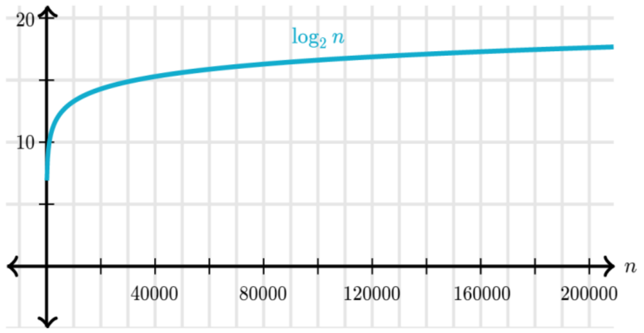
- Every time we double the size of the array, we need at most one more guess.
- the general case of an array of length n. We can express the number of guesses, in the worst case, as
  - "the number of times we can repeatedly halve, starting at n, until we get the value 1, plus one.
  - We will use "log base-2" to represent this values

base-2 logarithms of various values of n:

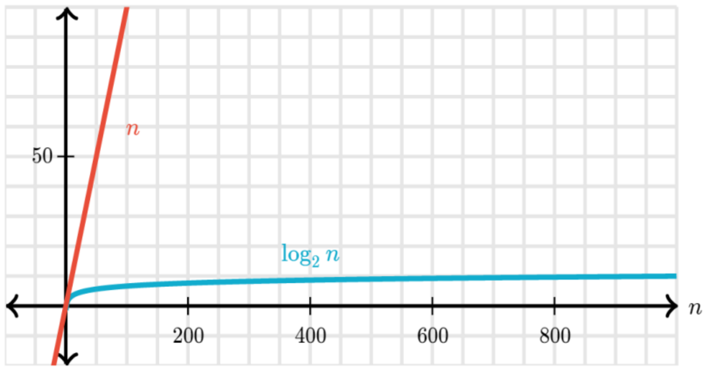
N*n*n	$\log_2(n)$
1	0
2	1
4	2
8	3
16	4
32	5
64	6

128	7
256	8
512	9
1024	10
1,048,576	20
2,097,152	21

We can view this same table as a graph: (X axix – number of comparisons, Y axis = Size of an array)



Compare N od Linear search with $\log_2(n)$ of Binary search



Complexity of the Binary Search Algorithm = O ($\log_2$N)
O(N)