Binary Search

Condition: All element should be in monotonic function

→ Means Shorted

Algorithm :-

- find mid element
- compare mid element with key
- > go to right and repeat the same
- < go to left and repeat same

Example:-

0	1	2	3	4	5	6	7
3	7	11	13	19	27	6	15

n = 8

key = 27 ← means we want to find 27

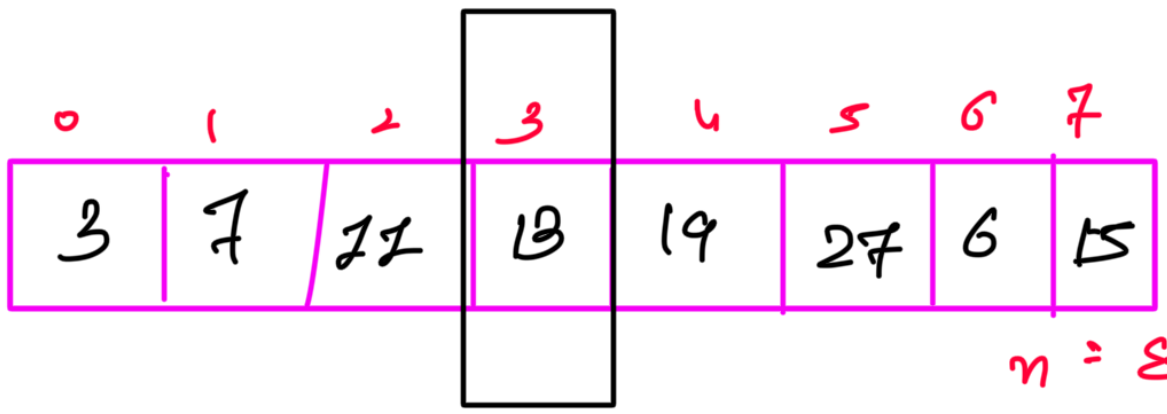
now we go to the mid element of

array.

mid element in int : $\left(\frac{\text{start} + \text{end index}}{2}\right)$

Here mid $= \frac{7}{2} = 3.5$

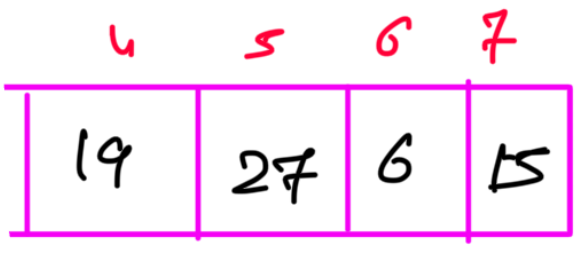
$= 3^{rd}$ index



now compare it with key.

Key = 27 $\quad$ 27 > 13

Because 27 is greater than 13 we will move to right.



Now we find mid **again.**

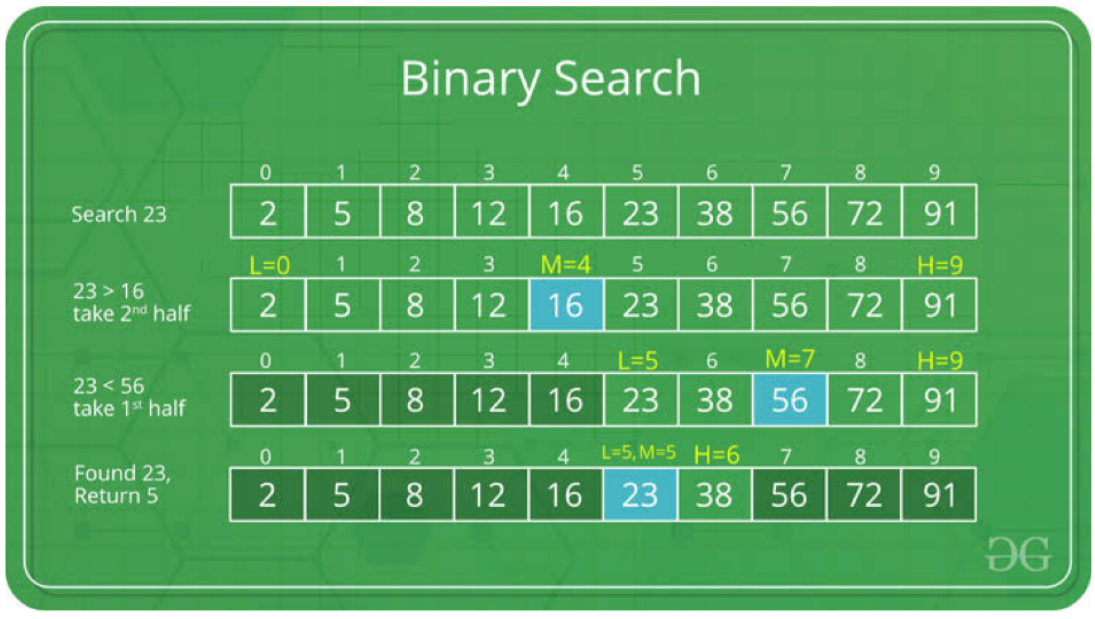
$$mid = \left(\frac{4 + 7}{2} \right)$$

$$= \left(\frac{11}{2} \right) = \underline{\underline{5.5}}$$

$$= 5$$

At index : 5

27 = 27



Binary Search

```cpp
#include <iostream>
using namespace std;

int binarySearch(int arr[], int n, int target, int hi, int lo)
{
  while(lo <= hi)
  {
    int mid = lo + (hi-lo)/2;
    if(arr[mid] == target)              return mid;
    else if(arr[mid] < target)
      lo = mid + 1;
    else if(arr[mid] > target)
      hi = mid - 1;
  }
     return 0;
}

int main()
{
  int arr[10] = {1,12,14,16,18,20,24,26,28,30};
  int n = 10;
  int target = 26;

  int lo = 0;
  int hi = n-1;
  cout<<"The index of the target is - "<<binarySearch(arr, n, target, hi,
lo);
  return 0;
}
```

Here

low = start = first index

high: end = final index

Here is diffrent formula.

$$mid = start + \left(\frac{end - start}{2}\right)$$

Because range of int is $2^{31} - 1$

what if my array is big enough that it's some is out of int range.

Then we need this **formula**.

Finding time complexity

Here there is some sequance is followed in binary search.

while first iteration = $\frac{N}{2}$

second iteration - $\frac{N}{4} = \frac{N}{2^2}$

third iteration = $\frac{N}{8} = \frac{N}{2^3}$

Here N = size of array.

final equation : $\frac{N}{2^k}$

Now according to decreasing infinite geometric series.

(G.P.) = $\frac{r}{1-r}$

Here $r = 1/2$

$1/2^k = \frac{1}{1 - 1/2}$

$= 1$

means $\frac{1}{2^k} = 1$

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

$$N = 2^k$$

$$k = \log n$$

upper bound $k = \log n$

Time complexity $= O(\log N)$