Binary Search.

- Real life example
- Coding Problem Example
- Iterative Code
- Recursive Code
- Time Complexity
- Overflow.

Binary Search:-

→ If you find from the first till last page → is known as LINEAR SEARCH.

→ Binary Search is applicable → if we have a sorted area. "raj".

→ Binary Search applied → where the Search Space is Sorted.

```
        0  1  2  3  4  5   6   7
Ex:→  [3, 4, 6, 7, 9, 12, 16, 17]   n=8.
```

array is Sorted.

target = 6.

→ then it is a 8 step.

→ But suppose we want 17. → then it is a 8 step process.

where TIME COMPLEXITY = O(N). because you have iterated the entire array.

→ Now if we take Binary Search.

```
[3, 4, 6, 7, 9, 12, 16, 17]      target = 6.
 ↑                       ↑
low                     high
```

$$mid = \frac{0+7}{2} = 3.5 = 3$$

at mid = 3 ⇒ a[mid] = a[3] = 7.

[3 4 5 6 7 . . .]

↑ low ↑ mid

a[mid] > a[target] → so now Binary Search will ly on LEFT HAND SIDE

[3 4 5 6 7 . .]

↑ low ↑ High ↑ mid

now this much is only the Search Space.

$$mid = \frac{low + high}{2} = \frac{0+2}{2} = 1 =$$

a[mid] = a[~~mid~~ 1] = 4.

[3 4 $\boxed{6}$ - - .]

↑ mid ↑ high ↑ low

now move low at ~~the~~ one place right of mid. and low and high pointing to the same index.

$$mid = \frac{low + high}{2} = 2,$$

a[mid] = a[2] = 6.

Iterative Soln. C++.

```
int Search (vector<int> & nums, int target) {
    int n = nums.size();
    int low=0, high = n-1;
    while (low <= high) {
        int mid = (low+high)/2;
        if (nums[mid] == target) return mid;
        else if (target > nums[mid])
                low = mid+1;
        else    high = mid-1;
    }
    return -1;
}
```

Recursive C++.

Recursion used when we are doing repeatitive steps.

$$f(arr, \overset{0}{low}, \overset{7}{high})$$

$$\downarrow$$

$$f(arr, 4, 7)$$

$$\downarrow$$

$$f(arr, 6, 7)$$

RECURSIVE

[3, 4, 6, 7, 9, 12, 16, 17]

Target = 13.

```
f( arr, low, high, target)
{
    if ( low >= high)
        return -1;

    mid = (low + high) / 2

    if(a[mid] == target)
        return mid

    else if ( target > a[mid])
        return f (arr, mid+1, high, target)
    else
        return f (arr, low, mid-1, target)
}
```

ultimately this will return -1.

[3, 4, 6, 7, 9, 12, 16, 17]
 0 1 2 3 4 5   6   7

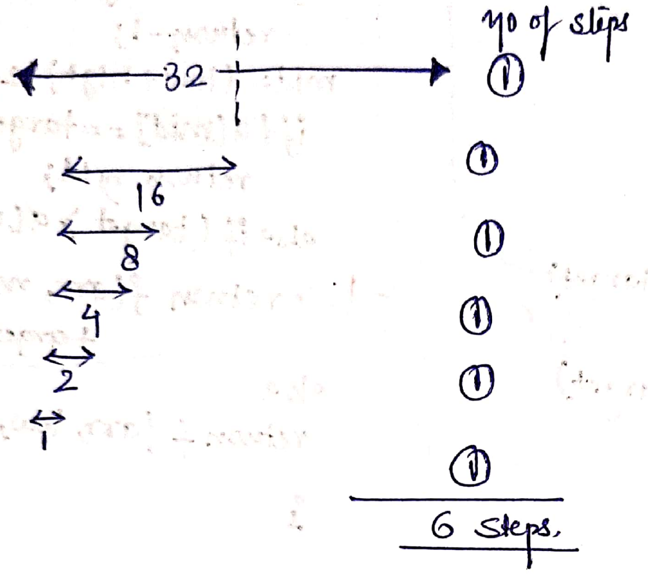
```
         0    7
f(arr, low, high, target)
{
  if (low > high)
     return -1;
  mid = (low+high)/2;   7
  if (a[mid]==target)
     return mid;
  else if (target > a[mid])
     return f(arr, mid+1, high, target)   -1
  else
     return f(arr, low, mid-1, target)
}
```

```
         4    7
f(arr, low, high, target)
{
  if (low>high)
     return -1;
  mid= (low+high)/2;   12
  if (a[mid]==target)
     return mid;
  else if (target > a[mid])
  -1 return f(arr, mid+1, high,
            target),
  else
     return f(arr, low, mid-1, target)
}
```

```
Index    6    7
f(arr, low, high, target)
{
  if (low>high)
     return -1;
  mid=(low+high)/2;   16
  if (a[mid]==target)
     return mid;
  else if (target>a[mid])
     return f(arr, mid+1,
              high, target)
  else
     return f(arr, low, mid-1,
              target)   -1
```

```
       6    5
f(arr, low, high, target)
{
  if (low>high)      ] Base Case
     return -1;      ] executed.
  mid=(low+high)/2;
  if (a[mid]==target)
     return mid;
  else if (target>a[mid])
     return f(arr, mid+1, high, target)
  else
     return f(arr, low, mid-1), target)
```

TIME COMPLEXITY OF BINARY SEARCH.



for $32 = 2^5$ | $64 = 2^6$.

∴ So Time Complexity is somewhat near about $O(\log_2 n)$.

* Overflow case of Binary Search.

0 ⟵⟶ INT_MAX

↑ low ↑ high

$$\boxed{mid = \frac{low + high}{2}}$$

if we keep on reducing, so low and high
both are low and high $= \frac{INT\_MAX + INT\_MAX}{2}$

So it becomes 2 * INT_MAX, thus it can't be
stored in the integer.

Solution of this problem is → either take long long (data type)

∴ Other option will be, if you don't want to use long long or long.

$$\boxed{mid = low + \frac{(high - low)}{2}}$$ it is same as → $$\boxed{mid = low + \frac{high}{2}}$$

So here if low and high
both become equal to INT_MAX
than $\frac{(INT\_MAX - INT\_MAX)}{2} = 0.$
So it will prevent overflow.