1. Binary Search Introduction

03 June 2023 12:00 PM

Binary Search :

Real Life Example:

Suppose In a Dictonary Alphabet are arranged

a b c d e f .... x y z

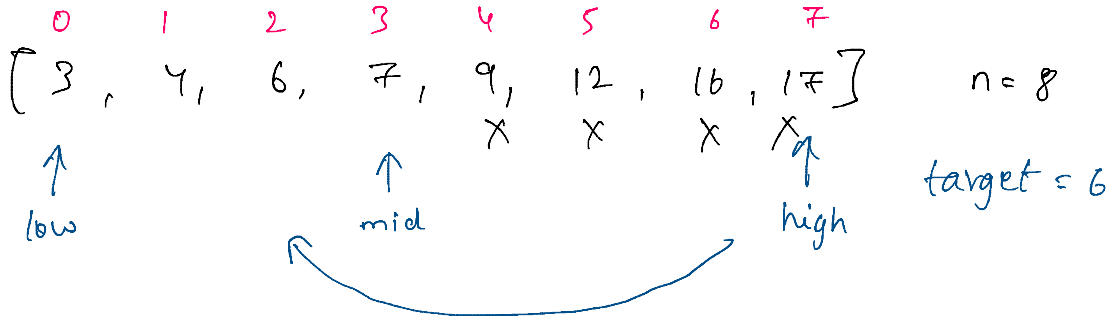
Now to search "rizon" you traverse linear and go to 'r' and then search it so it's a linear search.

Now it's a sorted array so for Binary Search it should be sorted array.

So now to implement Binary search

again left half.

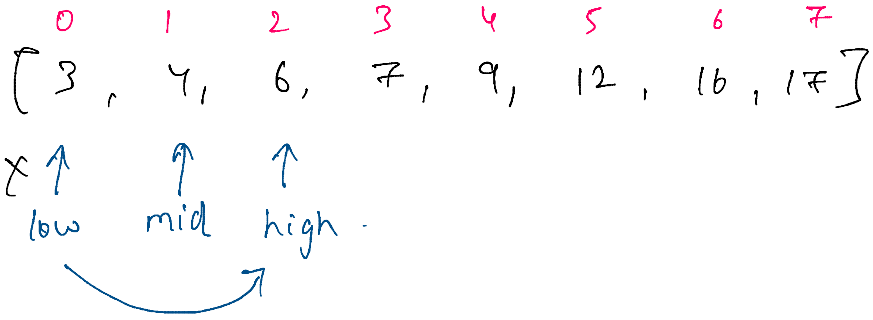
Coding Problem Example



→ everything b/w low & high is considered as search space.

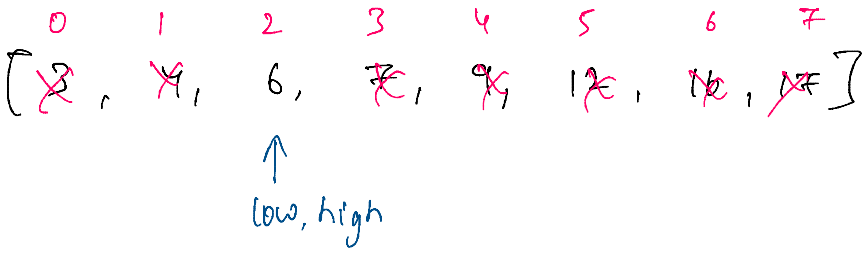
$mid = \frac{0+7}{2} = 3.5 \simeq 3$

mid value = 7 > 6 (target) it will be in left half.



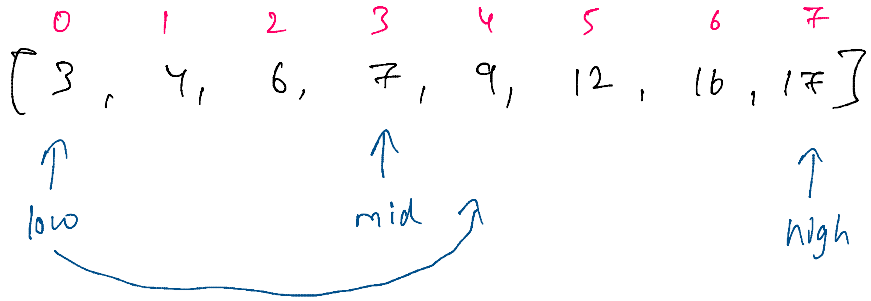
mid = 1

mid value = 4 < 6 (right side)



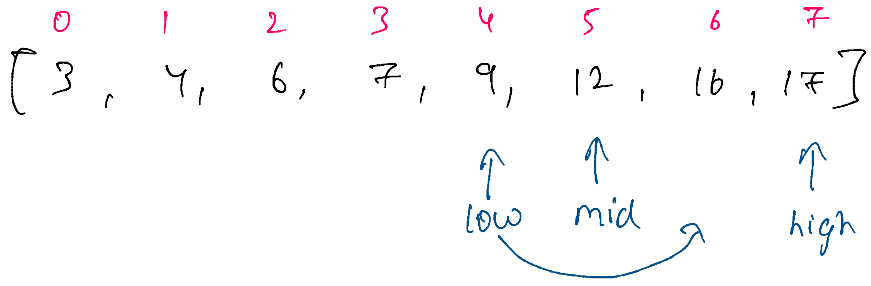
mid = 2 , mid value = 6 == target (6)

Now if target = 13

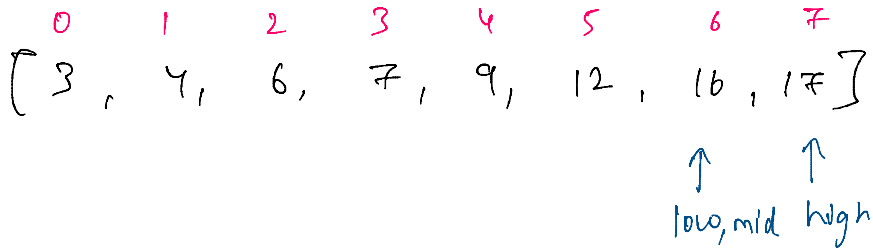


mid = 3, mid value = 7 < 13, so right half will be considered.

so low = mid + 1

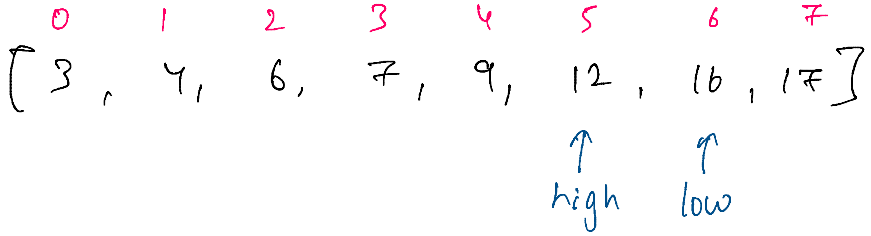


mid = 5, mid value = 12 < 13, again same



mid = 6, mid value = 16 > 13, now right side eliminated.

so high = mid - 1



— Now if you see high has crossed low so there is NO search space, search space always

b/w low to high.

- As we ran out of search space so no element is present.

Iterative Binary Search Code:

0	1	2	3	4	5	6	7	
[3,	4,	6,	7,	9,	12,	16,	17]	n = 8

target = 13

Pseudo Code:

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

3

Code:

LeetCode

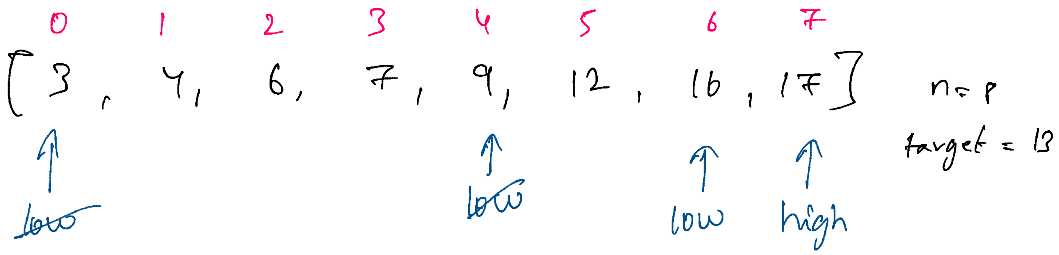
```cpp
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int n = nums.size();
        int low = 0, high = n - 1;
        while(low <= high){
            int mid = (low + high) / 2;
            if (nums[mid] == target) return mid;
            else if (target > nums[mid])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

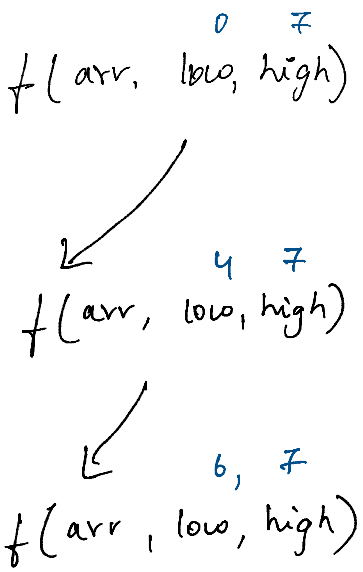
CodeStudio

```cpp
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;
}
```

Recursion Solution :

- Recursion occurs when we perform same task repeatedly.





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

C++ (g++ 5.4) Autocomplete

```cpp
int bs(vector<int> &nums, int low, int high, int target) {
  if (low > high)
    return -1; // base case
  int mid = (low + high) / 2;
  if (nums[mid] == target)
    return mid;
  else if (target > nums[mid])
    return bs(nums, mid + 1, high, target);
  return bs(nums, low, mid - 1, target);
}

int search(vector<int> &nums, int target) {
  return bs(nums, 0, nums.size() - 1, target);
}
```

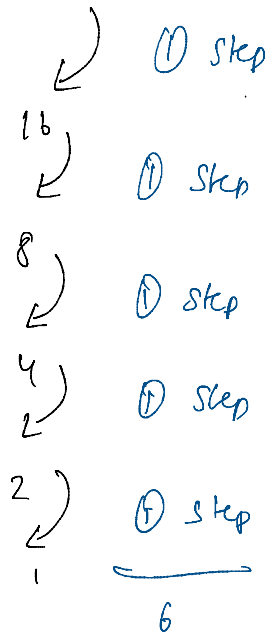
C++ • Auto

```cpp
class Solution {
private:
    int bs(vector<int> &nums, int low, int high, int target) {
    if (low > high)
        return -1; // base case
    int mid = (low + high) / 2;
    if (nums[mid] == target)
        return mid;
    else if (target > nums[mid])
        return bs(nums, mid + 1, high, target);
    return bs(nums, low, mid - 1, target);
    }

public:
    int search(vector<int>& nums, int target) {
        return bs(nums, 0, nums.size() - 1, target);
    }
};
```

T.C

Suppose 32 element then it get half



$T.C = O(\log_2 N)$

Overflow Case :

If high = INT_MAX

low = 0

$$mid = \frac{(low + high)}{2}$$

Now if low comes to last element

$$mid = \frac{(INT\_MAX + INT\_MAX)}{2}$$ = it will overflow as it can't be stored as Integer

So solution is use long long

OR

$$mid = low + \frac{(high - low)}{2}$$ if you use both

$$mid = low + \frac{(high - low)}{2}$$ if you sure both low and high = INT_MAX