Day 8 - Binary Search

Tuesday, 26 October 2021 1:18 PM

Unsorted Array

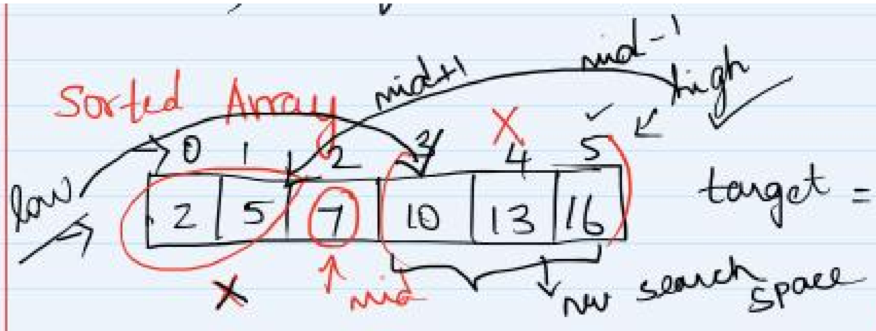
0	1	2	3	4
8	1	7	9	10

target = 10

```
for(int i=0; i< num.length; i++){
    if(num[i]==target){
        return i;
    }
}
return -1; (Not found)
```

→ O(n) → Size of the array

Sorted Array



target = 5   7 > 5

① Search space → (0, ... len-1) → (0 to 5) → all possible solutions

size = 6

low   high

② Mid element ⇒ $\frac{low + high}{2}$ ⇒ $(2^{31} - 1)$ → 2,147,483,647 (max) → exceed ✓ X will not occur

(− ") (min)   low + high =

int mid ⇒ $\boxed{low + (high - low)/2;}$

low = 2 high = 5 → $\frac{2+5}{2}$ = 7/2 = 3 ✓ 3

⇒ 2 + (5−2)/2 ⇒ 2 + 3/2 ⇒ 2 + 1 → 3

③ compare mid with target

(i) mid == target
Element found → stop

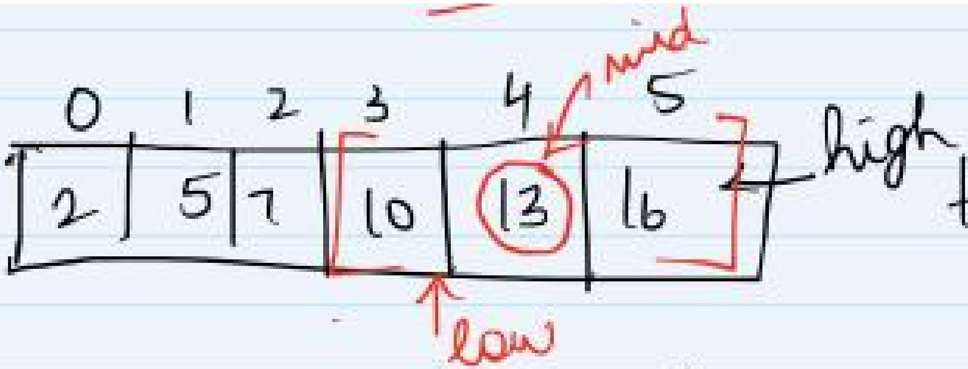
(ii) mid > target
high = mid − 1;

(iii) mid < target
low = mid+1

Repeat until I have atleast one element in search space

6 → n
3 → n/2
1 → $\frac{n/2}{2}$
reduced by half

TIME COMPLEXITY → O(log n)



target = 13

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

$\Rightarrow 0 + (5-0)/2 = 2$

→ 7 == 13 ? false

→ 7 > 13 → false

→ 7 < 13 → true

low = 3

high = 5

mid = 3 + (5 - 3/2)

$\Rightarrow 3 + (2/2) \Rightarrow 3+1 = 4$

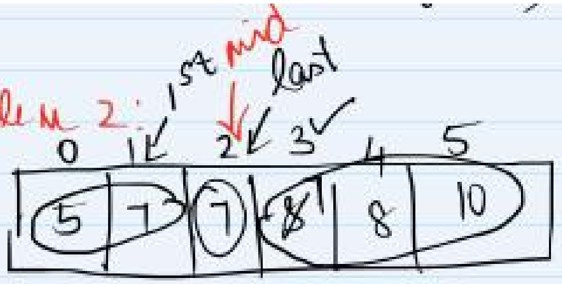
13 == 13 ✓

return mid;

Code:

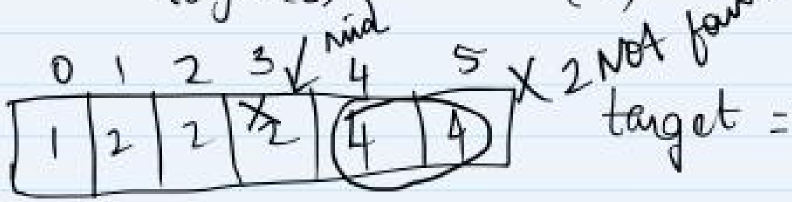
```
int[] num = new int[]{2, 5, 7, 10, 13, 16];    int target = 10

int low = 0                    } -> Search Space
int high = num.length -1;

while (low <= high) {  -> atleast one element in search space)

      int mid = low +(high-low)/2;  -> calculating mid
      if (num[mid] == target) {
            return mid;  ->                      } -> Shifting low/high to reduce the
      }else if(num[mid] > target) {                   Search Space
         high = mid -1;
      }else {
         low = mid +1;
      }
}
return -1 (not found);
```

Problem 2:



target = ~~7~~ ~~8~~ 10 | return 1st & last position

o/p: [3, 4]   o/p: [1, 2]   o/p: [-1, -1]   o/p: [5, 5]
target (8)   (7)   (1)   (10)



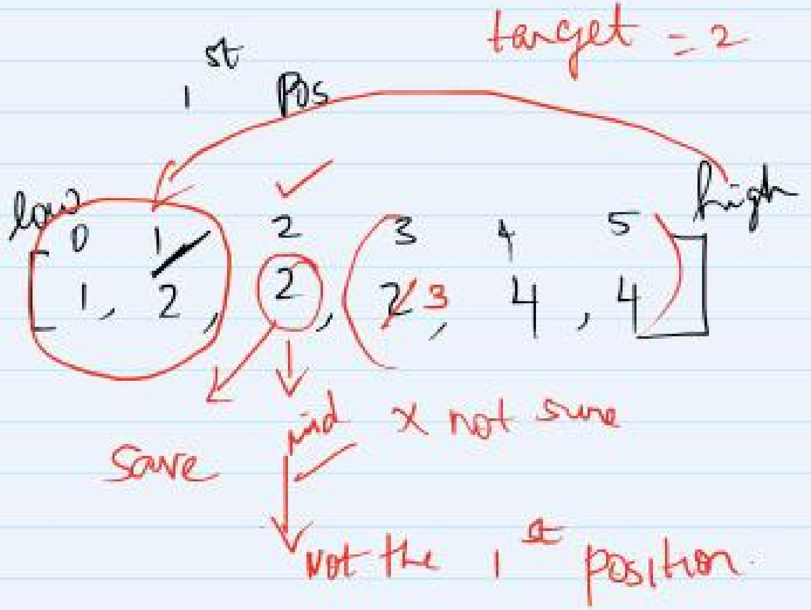
target = 2

o/p = [1, 3]

① first position → ?

② last position → ? ✓

```
int ans = -1;
if(nums[mid] == target){ ✓ element found
  → ans = mid;
    high = mid-1; → low = mid+1;
}
return ans;
```



Problem 3: KoKo Eating Bananas

i/p: 0 1 2 3
[3, 4, 7, 11] hour < 8 hour

3 11 → 11

→ Speed → K : value → K = } minimum spee

① low = 1 ✓ high = max element ✓

(2 ③ 4) K = 5 ✓ (right subarray) → I can eat all bananas

K ⇒ 5 > 5 → also possible ? X

↓ K => 5 > 5 → also possible ? X

① low = 1 $\underbrace{\quad}$ high = max element (11) → all possible values of K

```
int ans = -1;
while (low <= high) {
        int K ✓ = ?;  → Value  [6 → K ✓]  5 ✓ < 4 ✓ <
    ← if (isPossible(K)) { →                6 X
          ans = K;                          6 X cannot eat all bananas
          high = mid-1;                     ↓
                                            increasing the speed.
    } else
        low = mid+1
    }                TIME = O(nlogn) → :X
}
```

O(logn)

10 hour

6 hours < 8 return true

i/p : [3, 4, 7, 11]   hour = 8 ✓   K = 6 ✓

① ① 1 → 6, 1 → 1, 1 → 6, 1 → 5

```
boolean isPossible(int speed) {
    int count = 0
    for(int i=0; i< nums.length; i++) {
        count += nums[i]/speed;
        ~~ew~~ if(num[i] % speed != 0) {
            count++;
        }
    }
    ~~count~~ return count <= hour;
```

O(n)

✓ 3/6 => 0 → 3 % 6 = 3 ✓ ✓ +1

num[i] / speed (K) + 1

✓ 18/6 → 3

19/6 → 3 + 1 (if you have remainder)
↓
1