Searching & Sorting

Starting at 9:10

Lecture 1

1. Binary Search Basics
   - Transition Point
   - First Bad Version
   - Guess Higher or Lower
2. Floor & Ceil
   - First & last Occurence
   - Count Occurences (HW)
3. Upper & lower Bound
   - Search Insert Position
4. Closest Element
   - k-closest elements
5. Heaters

Binary Search

O(N) comparisons linear search

Time Complexity :→ $O(\log_2 n)$

Constraint :→

21354

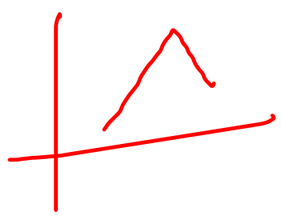
Nearly sorted array

Modified BS → Matrix

Infinite Array (unbounded)

Rotated Sorted Array

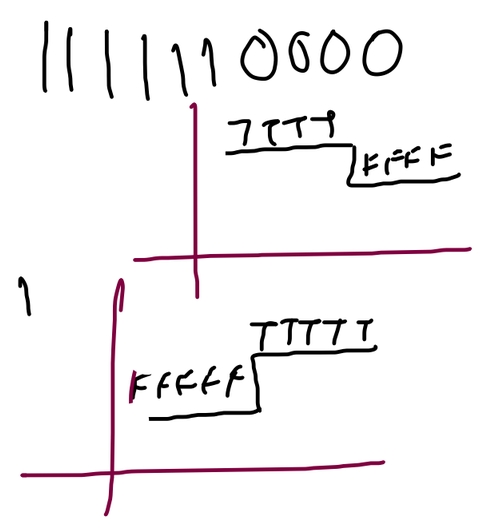
Bitonic Array

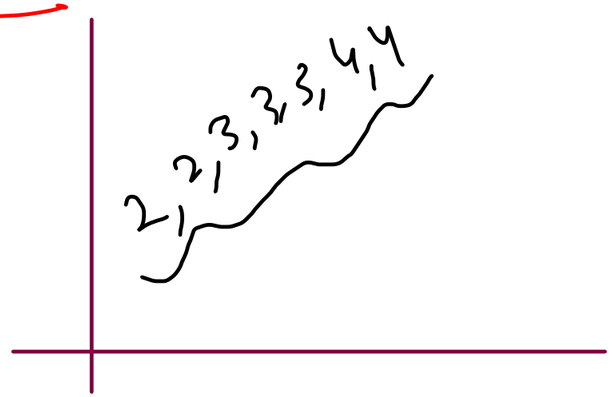


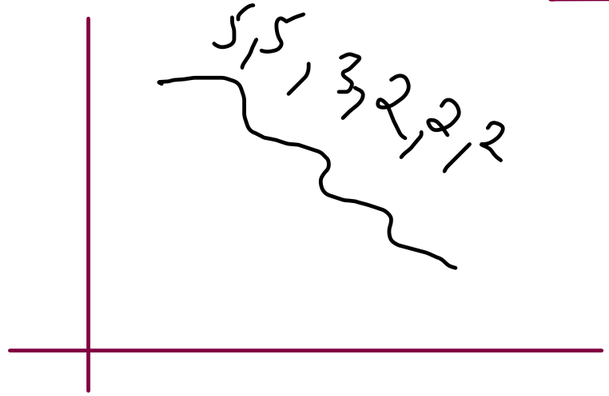
Monotonic

- Increasing
  - Strictly
- Decreasing
  - Strictly
- Binary
  - 0s 1s
  - 0s 1s → 00001111
  - 1s 0s → 11111 0000

BS on Answer



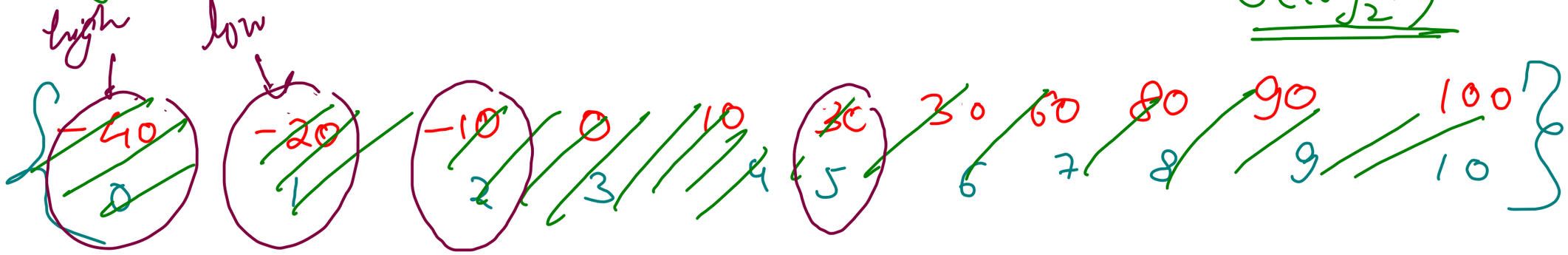


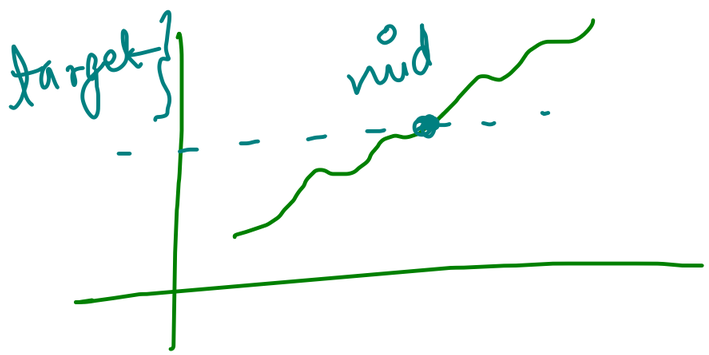


binary Search

$T(n) = T(n/2) + k$

$\Rightarrow O(\log_2 n)$






```java
public int search(int[] nums, int target) {
    int left = 0, right = nums.length - 1;

    while(left <= right){
        int mid = (left + right) / 2;

        if(nums[mid] == target){
            return mid; // search succesful
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1; // search unsuccesfull
}
```

target => -30

Transition Point

First-Occurence of (1)

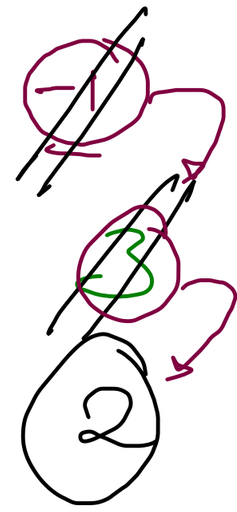
{ 0, 0, 1, 1, 1, 1, 1, 1 }

0 1 2 3 4 5 6 7

high

low

ans =




```
int transitionPoint(int arr[], int n) {
    int left = 0, right = n - 1;
    int ans = -1;

    while(left <= right){
        int mid = left + (right - left) / 2;
        if(arr[mid] == 0){
            left = mid + 1;
        } else {
            ans = mid;
            right = mid - 1;
        }
    }

    return ans;
}
```

First Bad Version

bool isBadVersion(v)

V_1 V_2 V_3 V_4 V_5 V_6 V_7

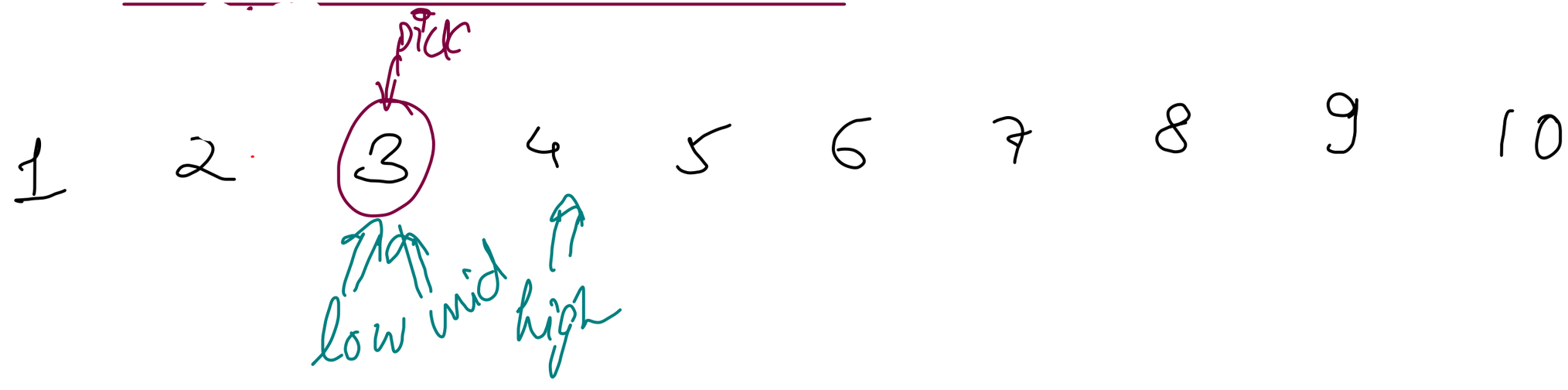
F F F F T T T

```java
public int firstBadVersion(int n) {
    int left = 1, right = n, ans = -1;

    while(left <= right){
        int mid = left + (right - left) / 2;

        if(isBadVersion(mid) == false){
            left = mid + 1;
        } else {
            ans = mid;
            right = mid - 1;
        }
    }
    return ans;
}
```

(0374) Guess Number Or lower




```java
public int guessNumber(int n) {
    int left = 1, right = n;
    while(left <= right){
        int mid = left + (right - left) / 2;

        int ans = guess(mid);
        if(ans == 0) return mid;
        else if(ans == -1) right = mid - 1;
        else left = mid + 1;
    }
    return -1;
}
```

guess(5) :- -1

guess(2) :- +1

guess(3) :- 0

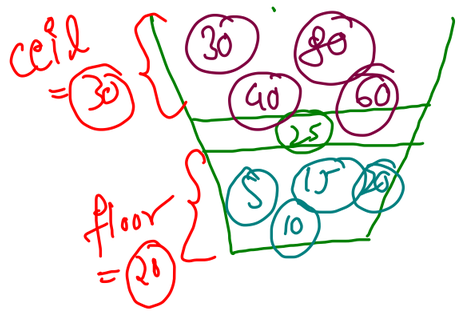
Floor & Ceil of Broken Economy

target = 60

0	1	2	3	4	5	6	7
5	10	15	22	33	40	42	55

target = 41

40, 42



floor, ceil
↳ (r, l)

```java
public static int floor(int[] arr, int target){
    int left = 0, right = arr.length - 1;
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(arr[mid] >= target)
            right = mid - 1;
        else left = mid + 1;
    }
    return arr[right];
}

public static int ceil(int[] arr, int target){
    int left = 0, right = arr.length - 1;
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(arr[mid] <= target)
            left = mid + 1;
        else right = mid - 1;
    }
    return arr[left];
}
```

right = -1 → floor does not exist

left == n → ceil does not exist

(134) First And Last Occurence

target = 40

Count occurences
if not found return 0
else return l-f+1

0	1	2	3	4	5	6	7	8	9	10
10	10	20	20	20	30	30	40	40	40	50

f

```
public int firstOcc(int[] nums, int target){
    int left = 0, right = nums.length - 1;
    int ans = -1;
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(nums[mid] == target){
            ans = mid;
            right = mid - 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            // nums[mid] > target
            right = mid - 1;
        }
    }
    return ans;
}
```

l

```
public int lastOcc(int[] nums, int target){
    int left = 0, right = nums.length - 1;
    int ans = -1;
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(nums[mid] == target){
            ans = mid;
            left = mid + 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            // nums[mid] > target
            right = mid - 1;
        }
    }
    return ans;
}
```

first = -1 ⊗ (7)

last = -1 ⊗ (9)

```
public int[] searchRange(int[] nums, int target) {
    int[] ans = {-1, -1};
    if(nums.length == 0) return ans;

    ans[0] = firstOcc(nums, target);
    ans[1] = lastOcc(nums, target);
    return ans;
}
```

1, 2, 3, 4, 5

l r

(1) [l, r] ⇒ 5 - 1 + 1 ⇒ r - l + 1

(2) (l, r] [l, r) ⇒ 5 - 1 ⇒ r - l

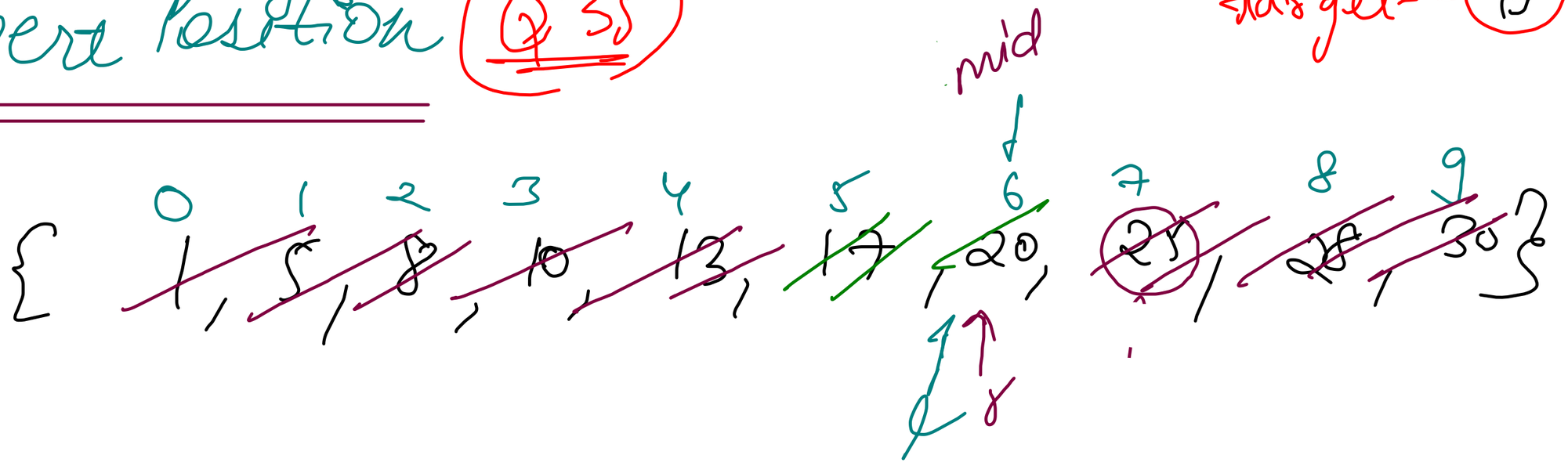
(3) (l, r) ⇒ 5 - 1 - 1 ⇒ r - l - 1

Search Insert Position (35)

target = 19

lower bound

on unique elements




```java
public int searchInsert(int[] nums, int target) {
    int left = 0, right = nums.length - 1;
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(nums[mid] == target){
            return mid;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return left;
}
```

lower & upper bound

- lower
  - if element found → return first occurance
  - if element not found, then return just greater value (ceil)
- upper bound
  - just greater value {ceil}
    - first occurence of ceil

{ 10, 10, 10, 20, 20, 40, 40, 40, 50, 50 }

0 1 2 3 4 5 6 7 8 9 (10)