↳ Weekly test (tomorrow - 14 May)

↳ 10 am to 1 pm

↳ MCT → 17th may

⇒ Binary Search

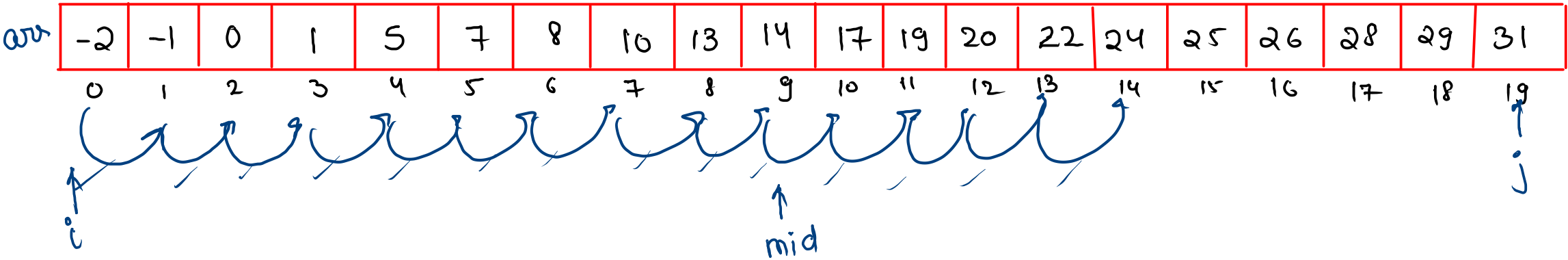
- searching algorithm (used to find an element)
- It works on divide and conqure method
- It works on a sorted array only
- advantage :- it take very less time as compared to linear search

T.C = $O(\log(n))$, where n is length of array

Algorithm

target = 23

array is Sorted



step 1 mid = (0 + 19)/2 = 9

step 2 mid = (10 + 19)/2 = 29/2 = 14

step 3 mid = (10 + 13)/2 = 11

step 4 mid = (12 + 13)/2 = 12

step 5 mid = (13 + 13)/2 = 13

in linear search = 14 steps

if array size is larger then complexity diff. will be huge

4 Binary Search

Code

```
i = 0, j = arr.length - 1;  (indexes)
while( i <= j ) {
    mid = (i + j)/2;
    if( arr[mid] == target) {
        return mid;
    } else if( arr[mid] > target){
        j = mid - 1;
    } else {
        i = mid + 1;
    }
}
Sysо(-1);
```

Imp if assume, whole array is of size 'x' only

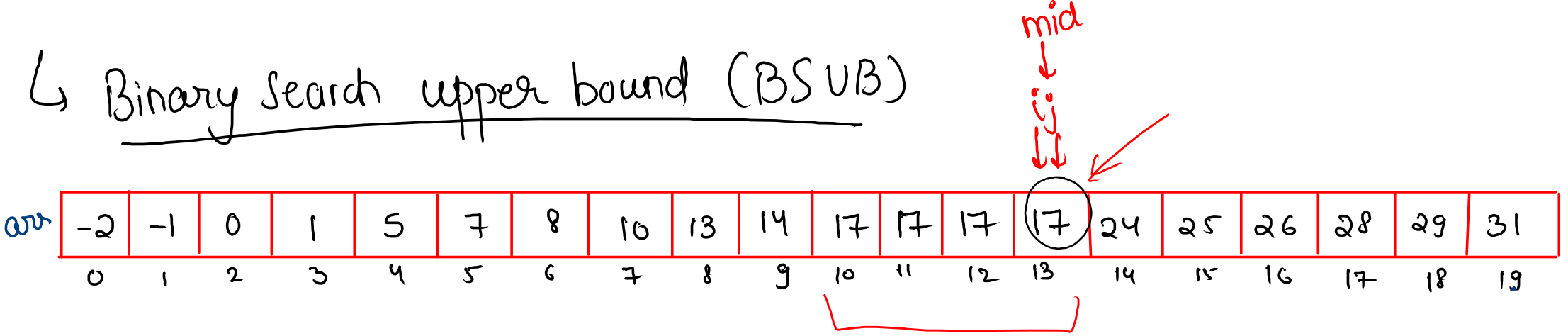
$$\Rightarrow x + \frac{x}{2} + \frac{x}{4} + \frac{x}{8} + \frac{x}{16} + \frac{x}{32} + \ldots\ldots + 2 + 1$$

$$\Rightarrow x\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \text{- - - - - - - -}\right)$$

$\log(n)$

binary search is $O(\log(n))$ because each time, we have to analyse only half of array

↳ Binary Search upper bound (BSUB)



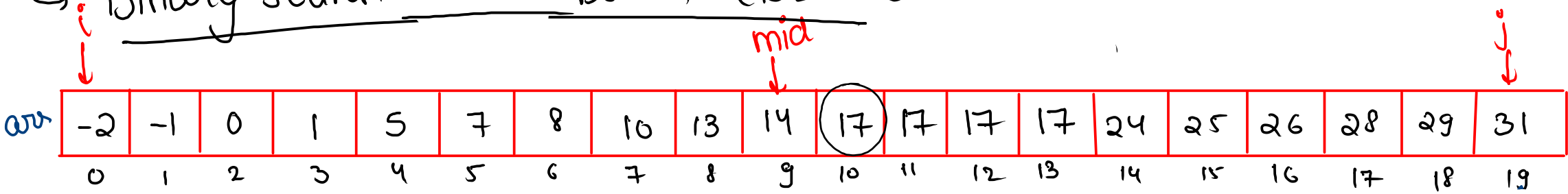
target = 17

```
int i=0, j=n-1;
while( i<=j) {
    mid = (i+j)/2;
    if( arr[mid] == tar){
        if( arr[mid] == arr[mid+1] )  i=mid+1;
        else  return mid;
    } else if (arr[mid] < tar){
        i= mid+1;
    } else {
        j= mid-1;
    }
}
return -1;
```

code

ans = 13

↳ Binary Search lower bound (BSLB)



target = 17

$$mid = \frac{10+10}{2} = 10$$

code

```
int i=0, j=n-1;
while( i<=j ){
    mid = (i+j)/2;
    if( arr[mid] == tar){
        if( arr[mid] == arr[mid-1] )  j = mid-1;
        else  return mid;
    } else if (arr[mid] < tar){
        i = mid+1;
    } else {
        j = mid-1;
    }
}
return -1;
```

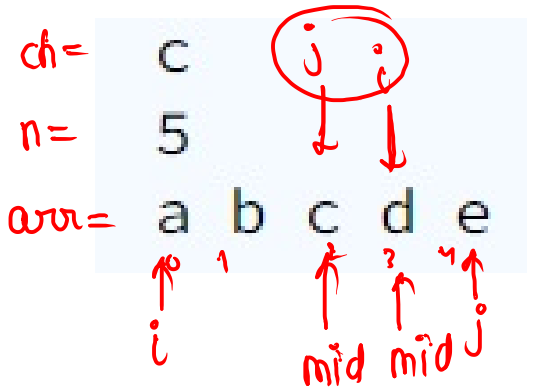
ans = 10

Search Character

code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    char ch = scn.next().charAt(0);
    int n = scn.nextInt();
    char[] arr = new char[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.next().charAt(0);
    }

    find(arr, ch, n);
}

public static void find(char[] arr, char ch, int n) {
    int i = 0;
    int j = n - 1;
    char ans = ' ';
    while (i <= j) {
        int mid = (i + j) / 2;
        char c = arr[mid];

        if ( c <= ch ) {
            i = mid + 1;
        } else {
            ans = arr[mid];
            j = mid - 1;
        }
    }

    if (i == arr.length) {
        System.out.println(-1);
    } else {
        System.out.println(ans);
    }
}
```

ans = 'd'



ans = d

char c = 'c'

c <= ch

'd' <= 'c'

Find Last Occurrence

(BSUB)

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++)
        arr[i] = scn.nextInt();
    int target = scn.nextInt();

    int i = 0;
    int j = arr.length - 1;
    while (i <= j) {
        int mid = (i + j) / 2;
        if (arr[mid] == target) {

            if ( mid + 1 < arr.length && arr[mid] == arr[mid + 1] ) {
                i = mid + 1;
            } else {
                System.out.println(mid);
                return;
            }

        } else if ( arr[mid] < target ) {
            i = mid + 1;
        } else {
            j = mid - 1;
        }
    }

    System.out.println(-1);
}
```