Todays Content

a) Find Unique element

b) Search in rotated sorted arr[]

c) Find Sqrt()

When to apply BS

a) Target b) SearchSpace

c) Discard SearchSpace

1Q) Every element occurs twice except for 1, find unique element.

**Note:** Duplicates are adjacent to Each Other.

**Ex:**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| arr[] = { | 6 | 6 | 2 | 2 | 7 | 9 | 9 | 4 | 4 | 10 | 10 } | ans = 7 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| arr[] = { | 3 | 3 | 1 | 1 | 8 | 8 | 10 | 10 | 19 | 6 | 6 | 2 | 2 | 4 | 4 } | ans = 19 |

**Idea1:** 1. Iterate on arr[], check adjacent elements & find unique

2. Iterate on arr[], Calculate XOR of all elements

TC: O(N) SC: O(1)

**Idea2:**

Target: Unique element   SearchSpace: Entire arr[]

Discard ?

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| arr[] = { | 12 | 12 | 6 | 6 | 21 | 21 | 7 | 9 | 9 | 4 | 4 | 10 | 10 } |

left unique   right unique

| | 0 | | 2 | | 4 | | 6 | | 8 | 9 | | 11 | | 13 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| arr[] = { | 3 | 3 | 7 | 7 | 4 | 4 | 9 | 9 | 24 | 12 | 12 | 14 | 14 | 6 | 6 } |

left: 1st occurrence even   ↑ mid   right: 1st occurrence odd

If mid land on left: : goto right

If mid land on right : goto left

Note: If 1st occurrence is even mid is on left

Note: If 1st occurrence is odd mid is on right

|  |  |  |  |  |  |  |  | 1st<br>* | 2nd<br>* |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| ar[ ] = | 3 | 3 | 1 | 1 | 8 | 8 | 10 | 10 | 19 | 6 | 6 | 2 | 2 | 4 | 4 |
|  |  |  |  |  |  |  | m | m+1 | l h | m |  | m |  |  |  |

| l | h | m | if m is 2nd occ: ar[m] == ar[m-1] | left or right m%2 == 0 |
|---|---|---|---|---|
| 0 | 14 | 7 | m = m-1 = 6 | 6%2==0 left: goto right l=m+2 |
| 8 | 14 | 11 | ⟶ 11 | 11%2 == 1 right: goto left h = m-1 |
| 8 | 10 | 9 | ⟶ 9 | 9%2 == 1 right: goto left h = m-1 |
| 8 | 8 | 8 : unique element : return ar[m]; |  |  |

TC: O(log N) SC: O(1)

10:10pm

```
int unique(int ar[]){
    int n = ar.length;
    if(N==1) {return ar[0]}
    if( ar[0] != ar[1]) { return ar[0]}
    if( ar[N-1] != ar[N-2]) { return ar[N-1]}
    int l = 1, h = n-2;
    while( l<=h) {
        int m = (l+h)/2
        if(ar[m] != ar[m-1] && ar[m] != ar[m+1]){
            return ar[m]
        }
        if (ar[m] == ar[m-1]){ // m is on 2nd occurence bring to 1st occurence.
            m = m-1;
        }
        // m is 1st occurence:
        if(m%2 ==0){ // 1st ocurence even => left : goto right
            l=m+2;
        }
        else{ // 1st occurence odd => right: goto left
            h=m-1;
        }
    }
}
```

if m=0: ar[0] != ar[-1] = Error?

if m= N-1 : ar[N-1] != ar[N] Error?

Q: sorted arr[] : { 3 9 14 16 20 28 35 40 49 }

Given k, Can k be present in arr[] or not ?

| k | {min | max} | chance |
|---|---|---|---|
| 16 ✓ : | 3 <= k <= 49 | | Yes |
| 25 ✓ : | 3 <= 25 <= 49 | | Yes |
| 60 ✗ : | 3 <= 60 <= 49 | | No |
| 1 ✗ : | 3 <= 1 <= 49 | | No |

k : { min <= k && k <= max }, k can be in given range

## Rotate arr[] Revision

Given ar[N], rotate arr[] Right to left

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| ar[9] = | 3 | 6 | 9 | 10 | 11 | 14 | 20 | 23 | 30 |

Rotate 3 times:

Rotate 3 : { 20 23 30 3 6 9 10 11 14 }

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| ar[10] = | 2 | 4 | 6 | 8 | 12 | 15 | 19 | 21 | 26 | 30 |

Rotate 4 times

{ 19 21 26 30 2 4 6 8 12 15 }

Note: If we rotate sorted arr[] k times, we get 2 sorted arrays

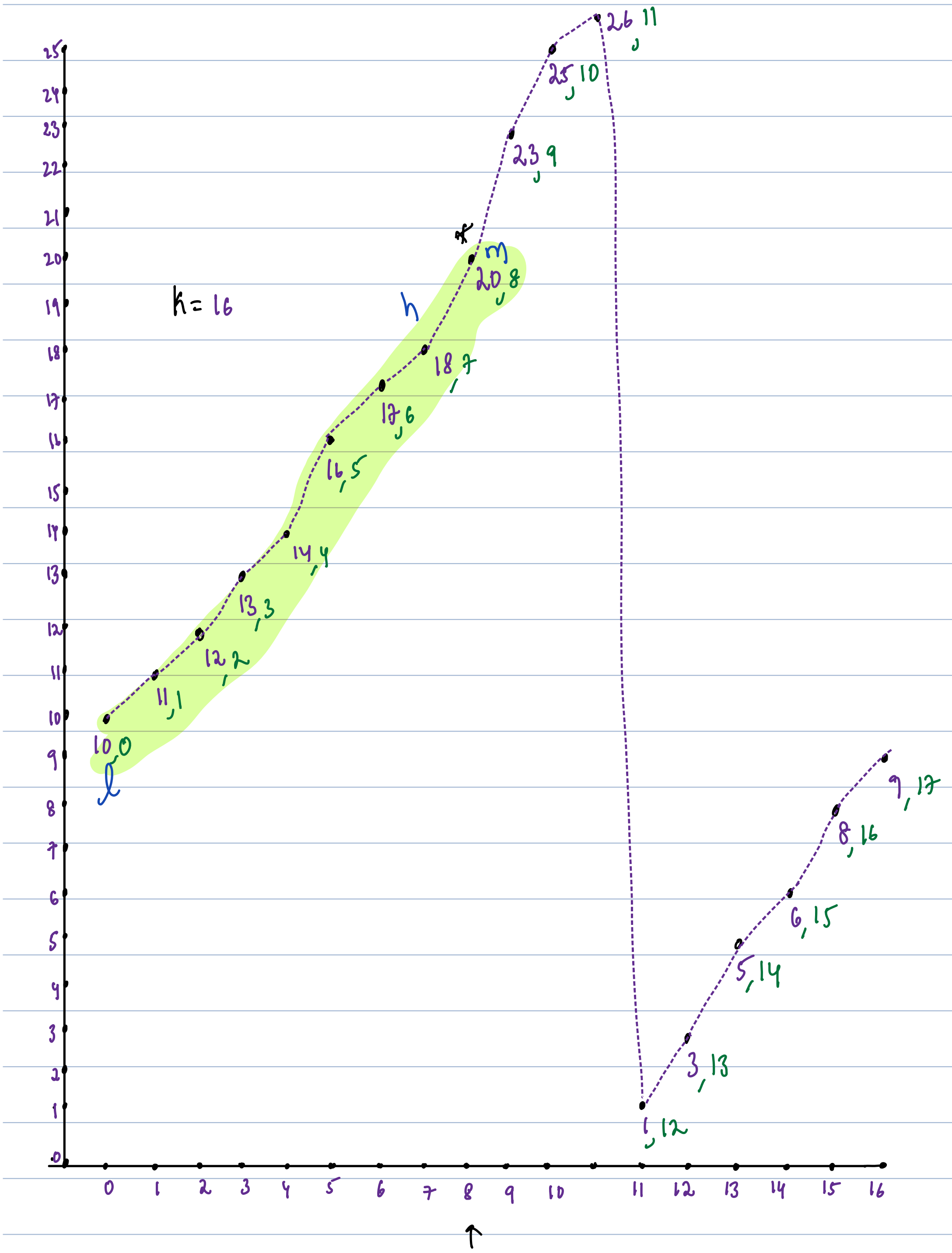
1st sorted arr[] : k elements   2nd sorted arr[] : N-k

2Q)

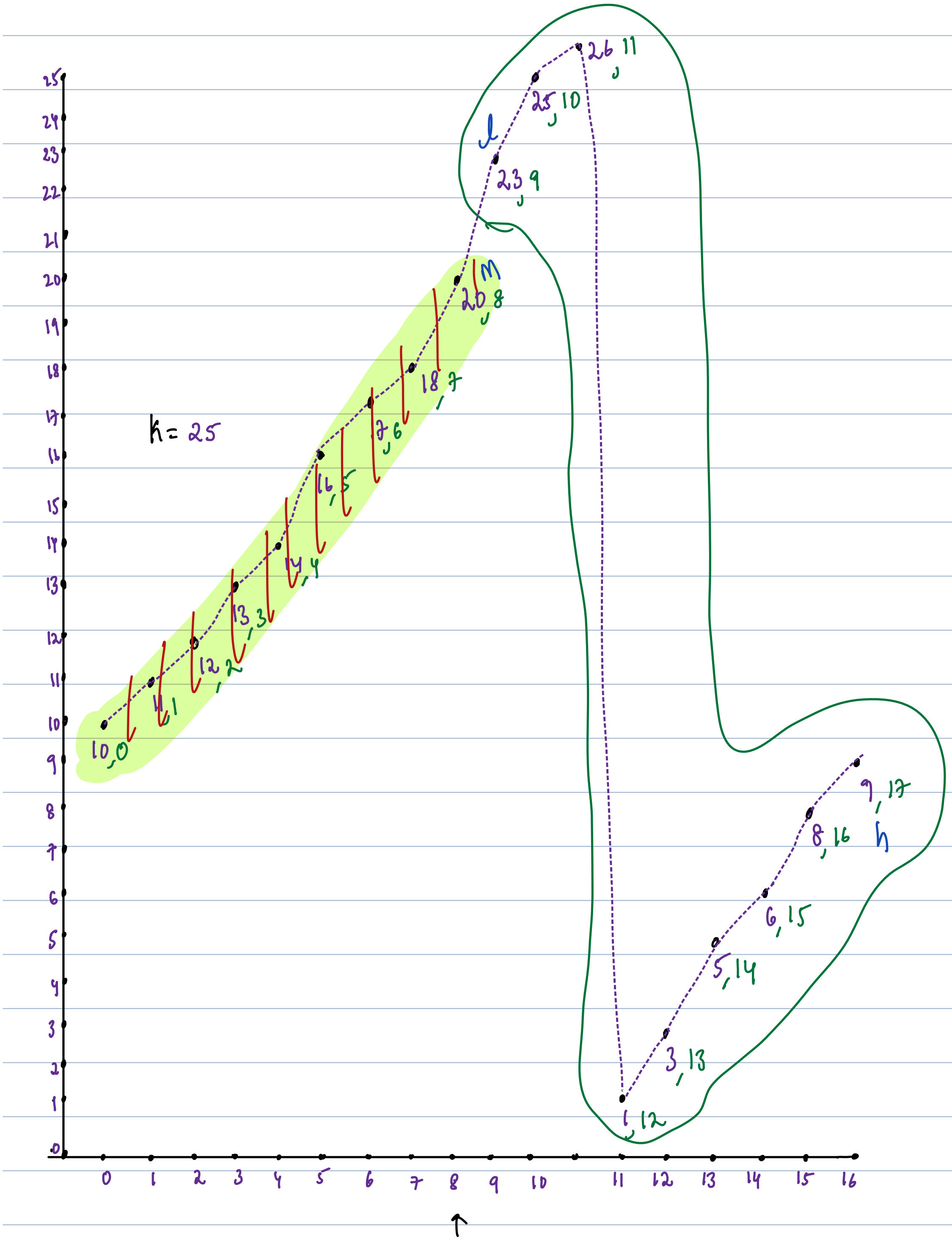
Given an input arr[], formed by rotatating a distinct sorted array right to left by some no: of times.

Search ele & return index in input arr[] if ele is not present return -1

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ar[ ] = { | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 20 | 23 | 25 | 26 | 1 | 3 | 5 | 6 | 8 | 9 } |
|  | ↑<br>l |  |  |  |  |  |  |  | ↑<br>m |  |  |  |  |  |  |  |  | ↑<br>h |



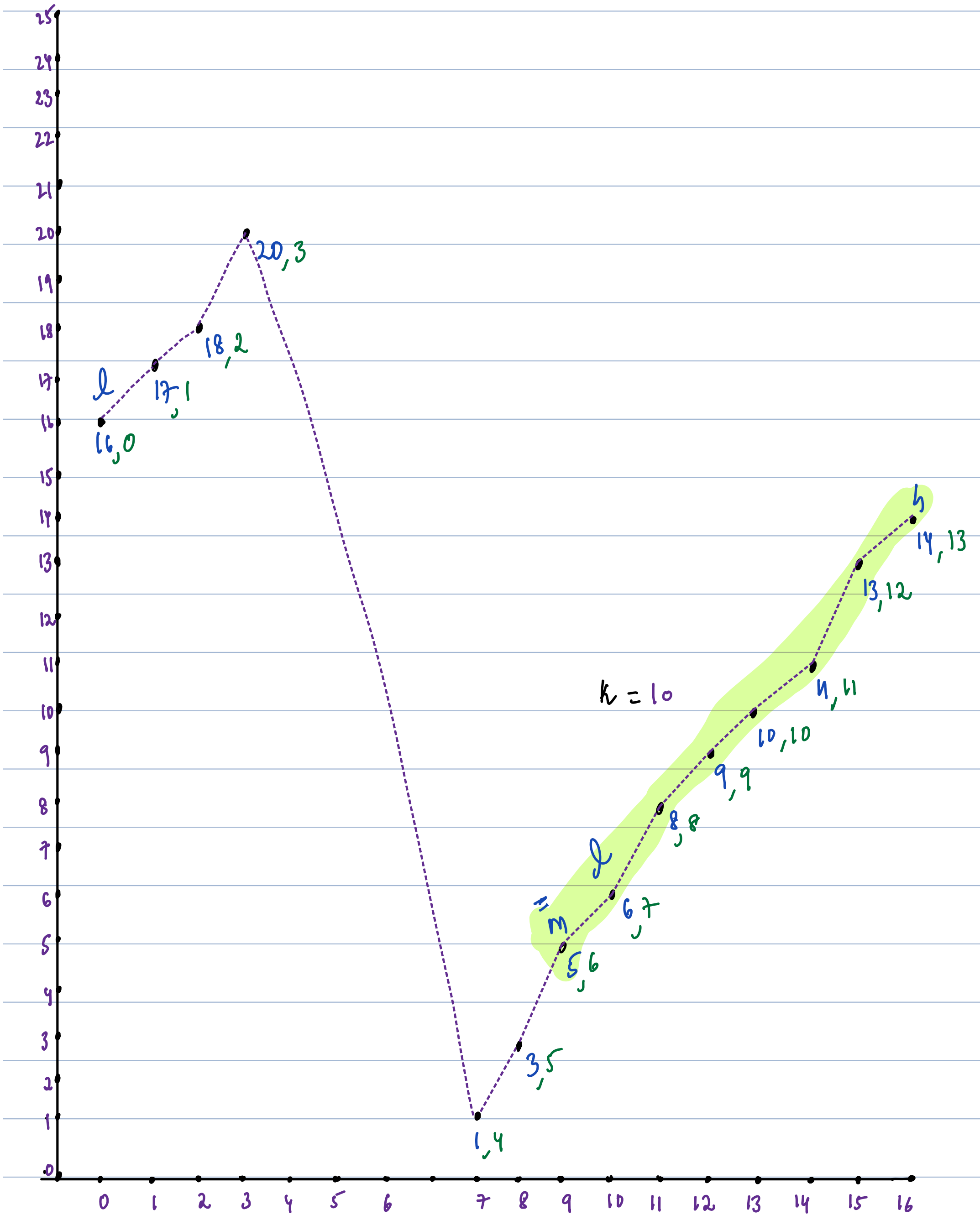
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ar[ ] ={ | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 20 | 23 | 25 | 26 | 1 | 3 | 5 | 6 | 8 | 9} |
| k= 25 | l | | | | | | | | m | | | | | | | | | h |



0 1 2 3 4 5 6 7 8 9 10 11 12 13

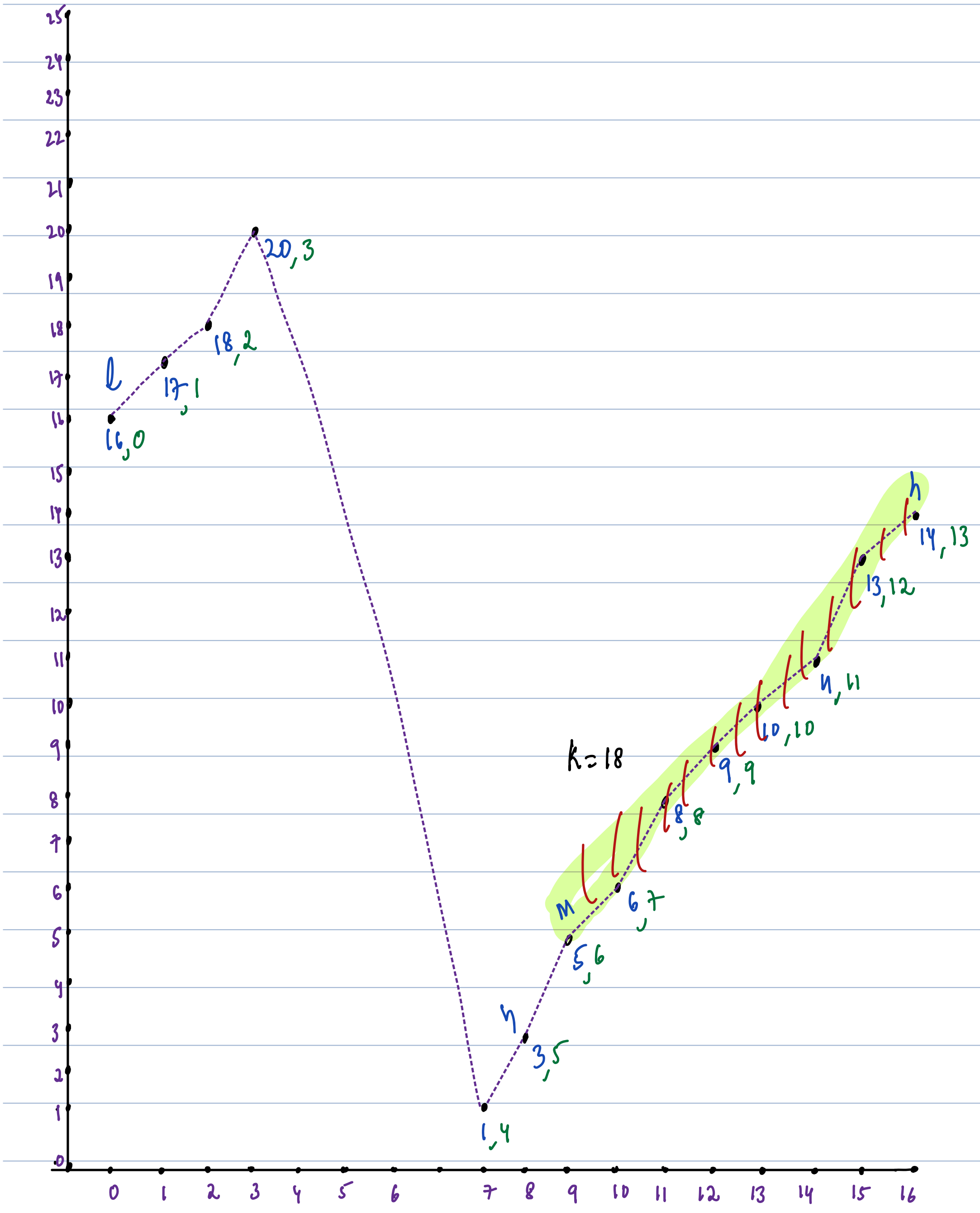
Ex: ar[] = {16 17 18 20 1 3 5 6 8 9 10 11 13 14}

k = 10 l m h



1

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ex: | ar[ ] = { | 16 | 17 | 18 | 20 | 1 | 3 | 5 | 6 | 8 | 9 | 10 | 11 | 13 | 14 | } |
| k=18 | | l | | | | | | m | | | | | | | h | |



```
int    searchrotated(int arr[], int k){ TC:O(logN) SC:O(1)
    int n = ar.length;
    int l=0, h=N-1;
    while(l<=h){
        int m = (l+h)/2
        if(ar[m]== k){return m }
        if(ar[m]>=ar[l]){ [l..m] sorted
            if(ar[l] <= k && k<=ar[m]){//go to left
                h=m-1;
            }
            else{ l=m+1}
        }
        else{ [m..h] sorted
            if( ar[m] <= k && k <= ar[h]){ //goto right
                l=m+1
            }
            else{
                h=m-1
            }
    }   }
    return -1;
}
```

3Q: Given +ve find SQRT(N)

Find greatest i such that i*i<=N

Sqrt(25) = 5*5 <=25

Sqrt(30) = 6*6 <=30   4*4<=30   5*5<=30   ans=5

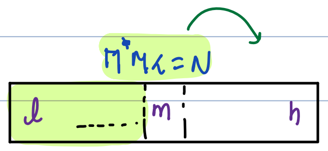
N=30:

| i | i*i<=N | ans |
|---|---|---|
| 1 | 1*1<=30 | ans=1 |
| 2 | 2*2<=30 | ans=2 |
| 3 | 3*3<=30 | ans=3 |
| 4 | 4*4<=30 | ans=4 |
| 5 | 5*5<=30 | ans=5 |
| 6 | 6*6<=30 ✗ | return ans=5 |
| 7<br>:<br>: | } No way | |

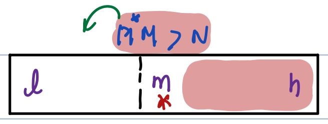
```
int sqrt(int N){ TC: O(√N)  SC: O(1)
    int i=1, ans=1;                          i=1
    while( i*i<=N){ ⇒ i^2<=N ⇒ while (i<=√N)
        ans=i; //update                      i++
        i=i+1;
    }
    return ans;
}
```

Find Sqrt(N) using BS

1. Target: greatest ele, such that ele*ele <=N
2. SearchSpace: sqrt(N) will be in range [1..N]
3. Discard



```
if(m*m <= N){
    ans=m; l=m+1
}
```



```
if(m*m > N){
    h=m-1;
}
```

N = 24

| l | h | m | m*m <= N | ans | |
|---|---|---|---|---|---|
| 1 | 24 | 12 | 12*12 <= 24 | 12 13 14 ....... | goto left h = m-1; |
| 1 | 11 | 6 | 6*6 <= 24 | 6 7 8 .. | goto left h = m-1; |
| 1 | 5 | 3 | 3*3 <= 24 | 1 2 3 ✓ | ans = 3 goto right l = m+1; |
| 4 | 5 | 4 | 4*4 <= 24 | 4 | ans = 4 goto right l = m+1; |
| 5 | 5 | 5 | 5*5 <= 24 | 5 ... | goto left h = m-1; |
| 5 | 4 | break, return ans = 4 | | | |

```
int sqrt(int N){   TC: O(logN)   SC: O(1)
    int l=1, h=N, ans=1;
    while( l<=h){
        int m= (l+h)/2
        if( m*m <= N){
            ans = m;
        } l=m+1; // l=m;
        else{ // m*m > N
            h=m-1;
        }
    }
    return ans;
}
```