Searching 1
Tuesday, 9 March 2021 7:58 PM

Say we want to search DOG in dictionary

→ T words: DOG leftside? // on right you will not find DOG [D > T]

→ B words: DOG rightside? // on left side you will not find DOG [D > B]

The reason we are able to make this decision is because → DOG is ordered/sorted

Search space → ordered

↳ Can we make the decision

// 1B: How are we choosing random position to land: $\frac{(s+e)}{2}$

Note: Search space is simply defined by a start & end point

	s		e	m					e
ar[] =	3	6	8	12	14	19	20	23	25	27
	0	1	2	3	4	5	6	7	8	9

k = 12

6+12 / 6×12 / 14+12 / 14×12

s	e	m	

0	9	4	ar[m] > 12 : // we are search to left
			e = m-1;

0	3	1	ar[1] < 12 : // we are search to right
			s = m+1

2	3	2	ar[2] < 12 : // we are search to right
			s = m+1

3	3	3	ar[3] = 12 : [Return 3];

```
bool BS1 (int ar[], int n, int k) {
    int l = 0, h = n-1;
    while ( l ≤ h ) {
        int mid = (l+h)/2;
        if ( ar[mid] == k) return True;
        else if ( ar[mid] > k) {
            // goto left
            h = m-1;
        }
        else {
            l = m+1;
        }
    }
    return False;
}
```

size of array

Element we want to search

Why mid: Because it has Best worst case Scenario

: we neglect N/2 → worst N/2

but: 2N/3 → worst: N/3

worst: N/3

N/3 | 2N/3

1B: Given an array of size n, & q queries for every query find max(ar[i]) ≤ q

	0	1	2	3	4	5	6	7	8
ar[] :	6	2	3	-5	10	18	9	11	4

Q	Ans
5 :	4
4 :	4
10 :	10
20 :	18
-7 :	INT_MIN

DF: q = 12, ans = INT_MIN

	[ar[i] ≤ q] [Is an ans]	update ans
6	✓	6
2	✓	*
3	✓	*
-5	✓	*
10	✓	10
18	*	*
9	✓	*
11	✓	11
4	✓	*

→ Pseudo Code:

```
given query q
ans = INT_MIN
i = 0 to N-1,
    if(ar[i] <= q){
        ans = max(ans, ar[i])
    }
}
```

TC: QN
SC: 1

optimization

Sort the given array:

	0	1	2	3	4	5	6	7	8
ar[] :	-5	2	3	4	6	9	13	15	18

valued part | Invalid part

k = 12, ans = INT_MIN

Note:
If we land on valid part
→ update ans
→ look for a better ans
If we land on invalid part
→ go to valid side

lo	hi	mid
0	8	4
5	8	6
5	5	5

ar[4] ≤ k, ar[4] can be a ans, [ans = 6], we will look for a better ans we search on right side lo = 5

ar[6] > k // goto leftside
lo > hi
hi = mid-1 = hi = 5

ar[5] ≤ k // q ≤ 12
[ans = 9] → [find ans]
// look for a better ans right
→ l = mid+1

Pseudo Code:

[ar[] is sorted to make it faster than we are using it] → q is single query

```
int findq (int ar[], int n, int q) {
    int lo = 0, hi = n-1, ans = INT_MIN
    while ( l ≤ h){
        int mid = (lo+hi)/2
        if (ar[mid] <= q) {
            ans = ar[mid]
            lo = mid+1 // look for a better ans on right side
        }
        else { hi = mid-1 }
    }
    return ans;
}
```

[6 5]
[Ex step]

TC: QK + logN + NlogN
querying | BS | sorting

2B: Given an sorted array, & q queries for every query find its frequence: Note: Cannot use extra space

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
ar[] :	-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15

q: 8 : 1
q: 5 : 7
1:7 : 0

P1 → first occurence
P2 → last occurence
count: P2 − P1 + 1

Solutions:

1B: loop for each query
TC: QN

2B: Use BS & iterate on left & right
TC: Q[logN + N]
Iterate on left & right

3B: using BS & get first occurence(P1) & last occurence(P2)
TC: Q[logN + logN]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15

P1 P2

left Invalid | valid part | right Invalid

BS1:
if(ar[mid] < q){
    mid landing here we go to right side
    l = mid+1
}

if(ar[mid] == q){
    mid landing here update ans & look for a better ans
    we get better ans ↑
    P1 = mid h = mid-1
}

if(ar[mid] > q){
    mid landing here we go to leftside
    h = mid-1
}

// Initialize P1 = -1
while(l <= h)

BS2:
if(ar[mid] < q){
    we go right
    l = mid+1
}

if(ar[mid] == q){
    P2 = mid, goto right
    low = mid+1
}

if(ar[mid] > q){
    we go left
    h = mid-1
}

// Initialize P2 = -1
while (l <= h)

if (P1 == -1) return 0
return: P2 − P1 + 1

3B: Given N distinct array elements, find [any] peak element

Note: A element is said to be peak element if it is greater than its adjacent elements

	0	1	2	3	4	5	6
ar[] :	3	2	1	5	7	4	8

DF:
Iterate find & return were element: O(n) SC: O(1)

l	h	m
0	6	3

Not an answer

Above we apply BS and we land mid

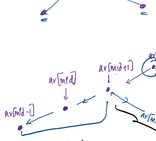
```
int peak (int ar[], int n){
    int lo = 0, hi = n-1;
    while( lo ≤ hi ){
        int mid = (lo+hi)/2
        if( ar[mid-1] < ar[mid] && ar[mid] > ar[mid+1]) {
            return ar[mid];
        }
        if( ar[mid] < ar[mid+1] ) { low = mid+1;
            peak on right }
        else { high = mid-1
            peak on left }
    }
}
```

TC: logN

```
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]
```

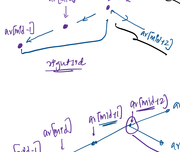
write the above base condition before while loop

Case 1:



Note: There is always a peak element on greater side.

Case 2:



Doubt:

Note: To apply BS, we need to condition to determine whether to go left or right: If we have condition then can we apply BS [YES]