3

19/02/2023

Q1 Binary Search in a nearly sorted array.

i/p → 10 3 40 20 50 80 70

First of all we need to know what is a nearly sorted array.

If the above array was sorted, then elements would be

3 10 20 40 50 70 80

In the nearly sorted array, the element present at the ith index in the sorted array can be present in 3 places i.e (i-1) th index or i th index or (i+1) th index.

i=0 in sorted array can be present at -1 or 0 or 1st index in nearly sorted array & is found at 1st index.

i=1 in sorted array can be present at 0, 1 or 2nd index in the nearly sorted array & hence is found at 0th index.
Similarly we can verify for all the elements as the condition will always be true.

Approach can be that we can apply linear search however it has time complexity = O(n) but can we solve in the logn approach / time complexity.
Other approach can be like we can sort the array & then apply binary search but the complexity of this solution will be O(n logn).

Algorithm

Sorted	Nearly sorted
1) Find mid = $\frac{s+e}{2}$ & compare arr[mid] and target.	Find mid & compare target with value at mid, mid-1 or mid+1 index and return index in each case.

2) target > arr [mid],
search in right side
i.e s = mid + 1

target > arr [mid]
search in right side i.e s = mid + 2
We added 2 as we have already checked mid + 1 index.

3) target < arr [mid],
Search in left side
i.e e = mid - 1

Here we will search in left side i.e e = mid - 2 as we have already checked mid - 1 index.

Code

```
int binarySearch (vector <int> arr, int target){

    int s = 0;
    int e = arr.size() - 1;
    int mid = s + (e - s) / 2;
    while (s <= e) {
        if (arr [mid] == target) {
                return mid;
        }       -> valid index check
        if (mid-1 >= 0 && arr [mid-1] == target){
                return mid - 1;
        }
        if (mid+1 < arr.size() && arr[mid+1] == target){
                return mid+1;
        }
```

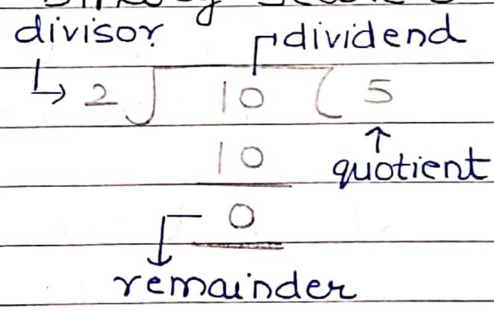
```
    if(target > arr[mid]){
            s = mid + 2;
    }
    else {
            e = mid - 2;
    }
    mid = s + (e-s)/2;
}
return -1;
}
```

Note → We can optimize the above code by modifying the condition of valid index check to other condition.

mid - 1 >= 0 → mid - 1 >= s
mid + 1 < arr.size() → mid + 1 <= e

Time complexity : O(logn) same as that of binary search.

Q2 Divide two numbers using binary search



i/p → dividend = 10
divisor = 2
quotient = ?

o/p → 5

Now the question is how can we use binary search algorithm to find the quotient. The similar kind of approach of finding the square root of a number using binary search.

Here we will take the search space from 0 to dividend.

Dividend = Quotient * Divisor + Remainder
Also we can modify the above formulae to

quotient * remainder <= dividend

Algorithm for 10/2

1) start = 0
end = 10
mid = $\frac{0+10}{2}$ = 5

mid * divisor = 5 * 2 = = 10 return mid;

Algorithm for 22/7

1) Start = 0
end = 22
mid = $\frac{0+22}{2}$ = 11

mid * divisor = 11 * 7 = 77 > 22
move to the left side by e = mid - 1.

2) Start = 0
end = 10
mid = $\frac{0+10}{2}$ = 5

mid * divisor = 5 * 7 = 35 > 22
Again search in the left side by e = mid - 1

3) start = 0
end = 4
mid = $\frac{0+4}{2} = 2$

mid * divisor = 2 * 7 = 14 <= 22
(i) Store ans as ans = mid as it might be the answer.

(ii) Search in right part
s = mid + 1;

4) start = 3
end = 4
mid = $\frac{3+4}{2} = 3$

mid * divisor = 3 * 7 = 21
(i) store ans as ans = mid
ans = 3
(ii) Search in right part
s = mid + 1

5) start = 4
end = 4
mid = $\frac{4+4}{2} = 4$

mid * divisor = 4 * 7 = 28 > 22
e = mid - 1 ; } Search in the left part

Now start > end, exit the loop.

Code

int solve (int dividend, int divisor) {

```
int s = 0;
int e = abs(dividend);
int mid = s+(e-s)/2;
int ans = 1;

while (s <= e) {

if (abs (mid*divisor) == abs (dividend)){
        ans = mid; // Got the answer
        break;
}

if (abs (mid * divisor) > abs (dividend)){
        e = mid - 1; // Search in left
}
else {
        ans = mid; // Store answer
        s = mid + 1; // search in right
}
mid = s + (e-s)/2;
}
// To handle the -ve case
if ((divisor < 0 && dividend < 0) || (divisor > 0 &&
                                       dividend >0){
        return ans;
}
else {
        return -ans;
}
}
```

Note → abs(-3); → 3
abs(3); → 3

abs () always return the +ve value.

Q3 Find the odd occuring element in the array.

i/p → 1 1 2 2 3 3 4 4 3 600 600 4 4

In this question all the elements occur even number of times except one. Also all the repeating occurence of element appear in the pairs and the pairs are not adjacent. Also note that there can not be more than 2 consecutive occurence of any element. We have to find the element that appears odd number of times.

Algorithm - 1

Brute force approach can be doing XOR of all the elements of the array. Time complexity of this approach is O(n) but can we

Algorithm - 2

1 1 2 2 3 3 4 4 $\boxed{3}$ 600 600 4 4

ans

Left of ans → pair

First value on the even index

Second value on the odd index

Right of ans → pair

First value on odd index

Last/second value on even index.

Also from observation, ans always lies on the even index.

Note → 0^{th} index will be considered as even in this question.

```
S = 0
e = n-1
mid = (s+e)/2   } Change in code to s+(e-s)/2;

while (s<=e) {
 if (s==e)
        return s;   } Only one element remaining case
```

Case-1

```
if (mid % 2 == 0) { → Even index
                      ↳ left side of answer
    if (arr[mid] == arr[mid+1]) {
            //Search in the right part
            s = mid + 2;
                      ↳ As mid+1 already checked.
    }
    else {
        e = mid; // if we do e=mid-1, then
                    we might loose the answer
                    as mid may be an answer.
    }
}
```

Case-2

```
else { → odd index
                ↳ left part as mid is odd
    if (arr[mid-1] = arr[mid]) {
        //Search in right part
        s = mid + 1;
                ↳ Here 1 & not 2 because
    }             mid+1 is not explored
```

```
else {

        e = mid-1;  // Here mid can not be
                       answer as the mid index
}                      is odd but answer is at
                       even index.
```

Code

```
int oddOccurence (vector <int> v) {
    int s = 0;
    int e = v.size() -1;
    int mid = s + (e-s)/2;
    while (s <= e) {
        if (s == e)
                return s;

        if (mid % 2 == 0) {

                if (v[mid] == v[mid+1]) {
                        s = mid + 2;
                }
                else {
                        e = mid;
                }
        }
        else {
                if (v[mid] == v[mid-1]) {
                        s = mid+1;
                }
                else {
                        e = mid-1;
                }
    }
```

```
    mid = s + (e-s)/2;
}
return -1;
}
```

Types of questions in binary search

1) Classic Questions like lower bound, upper bound etc.

2) Search space predicate function question such as aggressive cows, book allocation etc.

3) Observing index value like the question of finding odd occuring element.