20/09/2023
Wednesday

* Divide using Binary Search →

I/P = dividend = 29
divisor = -7

O/P = quotient is : -4

Logic → Search space = 0 ⟶ dividend

As quotient * divisor + remainder = dividend

= if (mid * divisor == dividend) {
return mid;
}

This is the case when number is perfectly divisible and remainder is 0. Simply, return mid.
Here, mid is the quotient.

= else if (mid * divisor < dividend) {
ans = mid;
s = mid+1;
}

Here, remainder is some value other than 0.
So, we can re-write the equality,
This → quotient * divisor + remainder = divident

as this → quotient * divisor < dividend
in the form of inequality by neglecting remainder.
So, in this case → store ans and move to right.

= else {
e = mid-1;
}

move to the left when mid * divisor > dividend

Program →

```
#include <bits/stdc++.h>
using namespace std;

int getQuotient (int dividend , int divisor) {
        int s = 0;
        int e = dividend;
        int mid = s + (e-s)/2;
        int ans = -1;
        while (s<=e) {
            if (mid * divisor == dividend) {
                return mid;
            }
            else if (mid * divisor < dividend) {
                ans = mid;
                s = mid + 1;
            }
            else {
                e = mid - 1;
            }
            mid = s + (e-s)/2;
        }
    return ans;
}
int main() {
        int dividend = 29;
        int divisor = -7;
        int ans = getQuotient(abs(dividend), abs(divisor));
// when either dividend or divisor is negative
    if (dividend>0 && divisor<0 || dividend<0 && divisor>0) {
            ans = 0 - ans;
        }
    cout << "quotient is : " << ans << endl;
}
```

__/__/____

* **Binary Search in nearly sorted array →**

I/P = 45 15 30 75 60 90 105
Target = 90

O/P = Target found at index : 5

logic →

Nearly sorted array - The element at ith index in sorted array may be present at (i-1)th, ith or (i+1)th indices in the nearly sorted array.
Let a sorted array - 15 30 45 60 90 105
In nearly sorted array - 30 15 60 45 105 90

```
if (mid-1>=0 && v[mid-1]==target){
        return mid-1;
    }
```

when (mid-1)th index is the targeted position.

```
if (v[mid]==target){
        return mid;
    }
```

when (mid)th index is the targeted position.

```
if (mid+1<n && v[mid+1]==target){
        return mid+1;
    }
```

when (mid+1)th index is the targeted position.

if (v[mid] < target) ⟶ s = mid+2

s = mid+2 to prevent the double checking of any element in the array

__/__/__

if (v[mid] > target) → e = mid - 2

e = mid - 2 to prevent double checking of any element in array.

If it is done → s = mid + 1 and e = mid - 1, it may be possible that some of the elements are checked twice.

Code →

```
#include <bits|stdc++.h>
using namespace std;
int nearlySorted (vector<int> v, int target){
    int n= v.size();
    int s=0 ; int e=n-1;
    int mid= s+(e-s)/2;
    while(s<=e){
if (mid-1>=0 && v[mid-1]==target){
        return mid-1;
    }
else if (v[mid]==target){
        return mid;
    }
else if ( mid+1<n && v[mid+1]==target){
        return mid+1;
    }
else if (v[mid]<target){
        s=mid+2;
    }
else{
    e=mid-2;
  }
mid= s+(e-s)/2;
}
return -1;
}
```

__/__/__

```
int main(){
 vector<int> v{45,15,30,75,60,90,105};
 int target = 90;

 int ans= nearlySorted (v, target);

 if (ans==-1){
        cout<<" target not found"<<endl;
      }
 else{
    cout<<"target found at index: "<<ans<<endl;
  }
}
```

__/__/__

* **Find odd occuring element →**

I/P = 4 4 5 5 6 6 7 7 6

O/P = odd occuring element : 6

Conditions for the input →

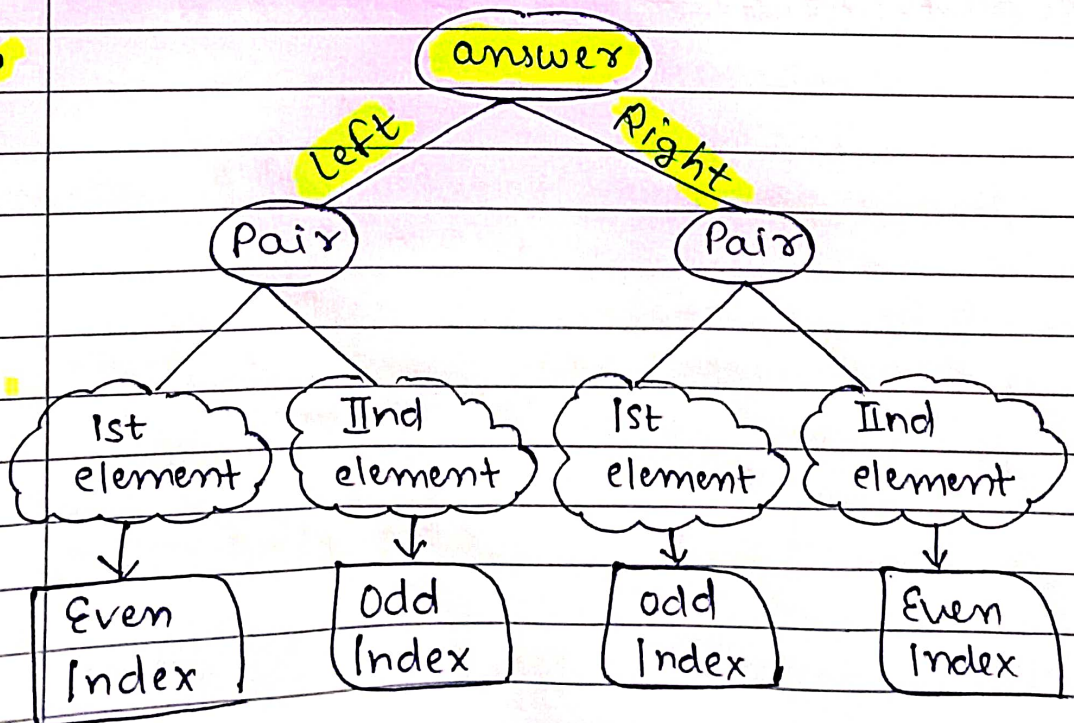
= All elements occur even number of times except one that occurs odd number of times.

= More than two times, a digit can't be placed adjacently, i.e. a same digit can't be placed in sequence 3 or more times.

= Every number is repeating in pair and placed adjacent. Also, no two same pairs are placed adjacently.

Observations →

•



__/__/____

- Answer i.e. odd occuring element always occurs on even index for every case keeping in mind the conditions for input.
- When array contains single element i.e. when s==e, that single element will be answer

Logic →

```
// single element
if (s==e) {
    return s;
}

                    → // odd index
if (mid %2==1) {
    if (v[mid]==v[mid-1]) {
        s=mid+1; → // move to right
    }
    else {
        e= mid-1; → // move to left
    }

                    → // even index
else if (mid %2==0) {
    if (v[mid]==v[mid+1]) {
        s=mid+2; → // move to right
    }
    else{ → either in right part or answer
        e=mid; That's why e=mid kiya. If
    }   e=mid-1 krte to ans lost ho skta hai
```

As we are checking $(mid+1)^{th}$ index element in condition, so, to prevent double checking of an element, we have done s=mid+2.

Code →

```
#include <bits/stdc++.h>
using namespace std;

int findOddOccuringElement (vector<int>&v){
    int n=v.size();
    int s = 0;
    int e = n-1;
    int mid= s+(e-s)/2;
    while(s<=e){
        if (s==e){
            return s;
        }
        else if (mid%2==1){
    if (mid-1>=0 && v[mid-1]==v[mid]){
            s=mid+1;
        }
        else{
            e=mid-1;
        }}
        else{
    if (mid+1<n && v[mid]==v[mid+1]){
            s=mid+2;
        }
        else {
            e=mid;
        }
    }
    mid= s+(e-s)/2;
    }
    return -1;
}
```

```
int main(){

    vector<int> v{4,4,5,5,6,6,7,7,6};
    int ans= findOddOccuringElement (v);

    cout<<"odd occuring element: "<<v[ans]<<endl;

}
```

* **Types of Binary search questions →**

- Classical
- Search-space based
- Predicate function based
- Index logic based