Searching & Sorting - Level-3

Search space Example.

Q. NO 1

~~Ip~~

Input Number ⇒

Divide using Binary Search

Formula ⇒

Quotient * Divisor + Remainder = Dividend

$\frac{a}{b}$ ⇒ Quotient

Quotient * Divisor <= Dividend

Cases:

1. Q x divisor == dividend
   ↳ Q → Final Ans

2. Q x divisor < dividend
   ↳ ans store
   ↳ right

3. Q x divisor > dividend
   ↳ Left

Code:

```
int getQuotient( int divisor, int dividend)
{
 int s=0;
 int e= dividend;
 int ans = -1;
 int mid =s+(e-s)/2;

 while (s<=e) {
    cout<< "s: "<<s<< "e: "<<endl;
    if (mid * divisor == dividend) {
    return mid;
}
 if (mid* divisor < dividend) {
     //ans store
     ans=mid;
   //right me jao
     s=mid+1;
 }
else {
    //left
   e=mid-1;
}
 mid = s+(e-s)/2;
}
return ans;
}
```

H.W ⇒ 2 digit Precision

Note:-

Just remember if we want to find quotient of -ve outcome then we have to find the quotient by considering -ve number then after finding we can change by seeing the numbers.

for e.g ⇒

divisor → 2 4 ← dividend

$$\begin{array}{r} 2 \\ 2\overline{)\,4} \\ -\,4 \\ \hline 0 \end{array} \qquad \begin{array}{r} -2 \\ -2\overline{)\,4} \\ -\,4 \\ \hline 0 \end{array}$$

Time complexity ⇒ $O(\log_2 N)$

where N is dividend.

Binary Search on Nearly Sorted Array

Sorted Array

10	20	30	40	50	60	70
0	1	2	3	4	5	6

Nearly Sorted Array

20	10	30	50	40	70	60
0	1	2	3	4	5	6

If in sorted Array, any number is in ith index then that same number will be in i-1, i, or i+1 index.

Comparison.

Normal sorted Array	Nearly sorted Array

if(arr[mid] == target)
return mid;
	if(arr[mid-1] == target)
	return mid-1;
	if(arr[mid] == target)
	return mid;
	if(arr[mid+1] == target)
	return mid+1;

if(target > arr[mid])
↳ Right;
	if(target > arr[mid])
	↳ Right
	else
	↳ Left
	} There is a catch

else ↳ Left	

Code:

```
int searchNearlySorted(int arr[], int n,
    int target){
    int s=0;
    int e=n-1;
    int mid= s+(e-s)/2;
    while(s<=e){
        if(mid-1>=0 && arr[mid-1]==target){
            return mid-1;
        }
        if(arr[mid]==target){
            return mid;
        }
```

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

Time complexity $\Rightarrow O(\log_2 N)$

where,

N is size of array.

Find the odd occurring elements.

Approach - 1
- O(N)
- XOR

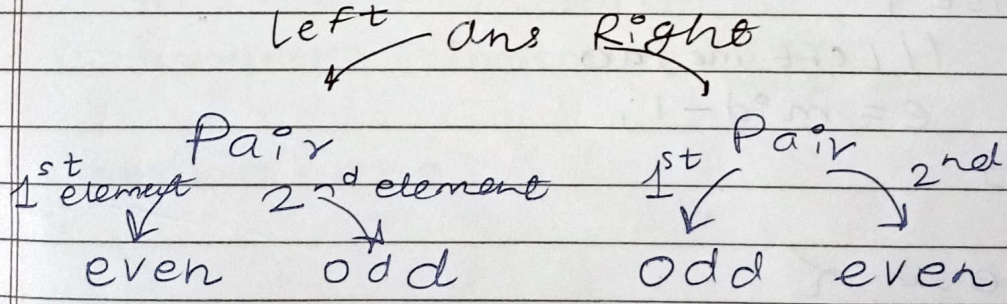
Binary search Problem types.

1. classical
2. Searchspace
3. Predicate Function
4. Index Logic.

Approach 2
counting with map datastructure.

Approach 3
sorting

Observation ⇒



If single element then that number is the answer.

Code:

```
int FindoddOccuringelement ( int arr[],
  int n) {
   int s=0;
   int e =n-1;
   int mid = s+(e-s)/2;

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

    //mid - check → even or odd
    if(mid & 1){
    // mid & 1 → true ⇒ odd number
 if(mid-1>=0 && arr[mid-1]==arr[mid]){
     //right me jao
     s=mid+1;
   }
 else{
     //Left me jao
     e=mid-1;
 }
 }
  else{
      //even
  if(mid+1<n && arr[mid]==arr[mid+1]){
        //right me jao
        s=mid+2;
 }
  else{
```

// ya toh main right part pr khada hu
// ya toh main answer k upar khada hu
// thats why e=mid kra hu.
// kyoki e=mid-1 jse ans lost hosakta hi
e=mid;
}
}
mid=s+(e-s)/2;
}
return -1;
}

Home work

float increment=0.1;
for(int i=0;i<precision;i++){
while(ans*ans<=number){
ans+=increment;
}
//loop terminates when ans*ans>number
ans=ans-increment;
increment=increment/10;
}
return ans;
}