Searching and Sorting-3

19/feb/23

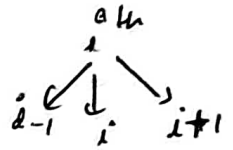
Q. ~~Binary~~ Search in a nearly sorted array.

	0	1	2	3	4	5	6
arr →	10	3	40	20	50	80	70

sorted array →

0	1	2	3	4	5	6
3	10	20	40	50	70	80

here nearly sorted means that the element (in a sorted array) is in i^{th} index, so it can be $(i-1)^{th}$ or i^{th} or $(i+1)^{th}$ index in nearly sorted array.

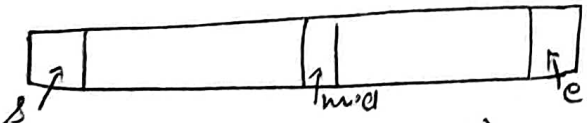


Approach 1 - Linear Search.

T.C = O(n)

Approach 2 - Binary Search

Sorted Array




```
if (arr[mid] == target)
    return mid

if (arr[mid] > target)
    e = mid - 1

if (arr[mid] < target)
    s = mid + 1
```

Nearly Sorted


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

} 3 condition because target can be in one of these three places.

```
if (target > arr[mid])
    s = mid + 2;
if (target < arr[mid])
    e = mid - 2;
```

→ 2 because, $(mid+1)^{th}$ element is already checked.

code →

```
int binarySearch ( vector<int>& arr , int target){
    int s = 0, e = arr.size()-1;
    int mid = s + (e-s)/2;
    while (s<=e){
        if (arr[mid] == target)
            return mid;                    → O(1)

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

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

T.C = O(log n)

because we already checked mid-1 or mid+1 and we did not find target there, so we will not check again.

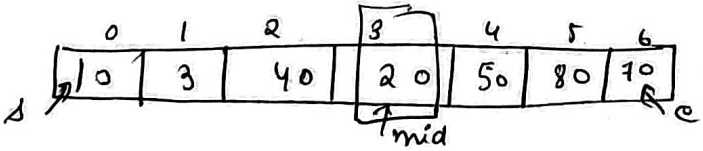
We have to make sure that mid+1 and mid-1 are valid indexes.

so we will check them also.

```
if ( mid-1 >= s && arr[mid-1] == target)
        return mid-1;

if ( mid+1 <= e && arr[mid+1] == target)
        return mid+1;
```

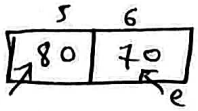
dry run →



, target = 70

1st cond" → 20 == 70 → F
2nd cond" → 40 == 70 → F
3rd cond" → 50 == 70 → F
70 > 20 → T ↳ search in right.

s = 3+2 = 5
e = 6



mid = 5

1st cond" → 80 == 70 → F
IIIrd cond" → 80 == 70 → T

⊙

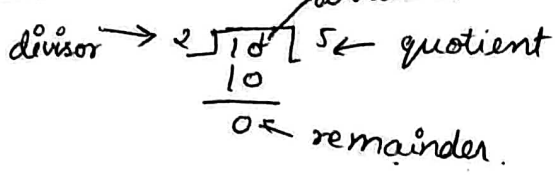
6	3	2	5	1	8	4

⇒ This isn't a nearly sorted array.

Q- Divide 2 numbers using binary search →

i/p { dividend → 10
divisor → 2

o/p quotient → 5

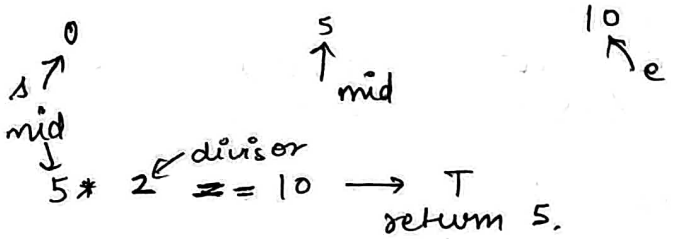


quotient can lie between
0 to dividend.

The formula we know → (0 or a positive no.)

quotient * divisor + remainder = dividend

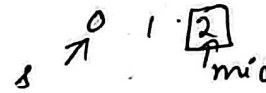
quotient * divisor <= dividend



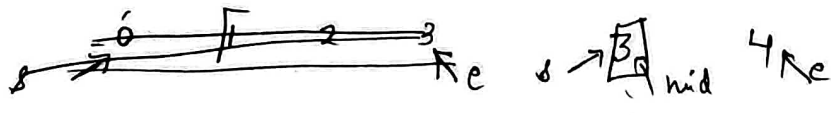
⊙ 22 ← dividend
7 ← divisor

11 X 7 = 77 <= 22 → F
77 > 22 → T (left search)

5 X 7 = 35
35 <= 22 → F
35 > 22 → T (left search)



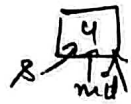
2 X 7 = 14
14 <= 22 → ~~F~~ T + search in right
~~14 > 22 → F~~



$3 \times 7 = 21$

$21 <= 22 \rightarrow T$ ↳ store 21 and search in right

→



$4 \times 7 = 28$

$28 <= 22 \rightarrow F$

$28 > 22 \rightarrow T$ (search in left)

$s > e \rightarrow$ stop

code -

```
int solve( int dividend , int divisor){
    int s = 0, e = dividend; ans = -1;
    int mid = s + (e-s)/2;
    while ( s <= e)
        //Perfect case (⇒rem = 0)
        if ( mid * divisor == dividend)
            return mid;
        // not perfect case (rem > 0)
        if (mid * divisor > dividend)
            e = mid - 1;
        else {
            ans = mid;
            s = mid + 1;
        }
        mid = s + (e-s)/2;
    }
    return ans;
}
```

But this will not work in case of negative numbers.
How to solve for negative numbers?

We will ignore negative sign and at least when we get ans we will update -ve sign.

⇒ int solve(- - -){

int s = 0;

```
int e = abs(dividend);
int ans = -1;
while(---){
    if (abs(divisor * mid) == dividend)){
        return mid;
        break;
    }
    if (abs(mid * divisor) :- - -
}
if (divident==0 && divisior<0 || divident>0 && divisor>0)
    return ans;
else return ans;
}
}
int main(){
    int dividend, divisor;
    cin >> dividend >> divisor;
    cout << solve(dividend, divisor);
}
```

dry run -

$\frac{64}{4}$

0 ——— 32 ——— 65
↑s ↑mid ↑e

$4 \times 32 = 128 > 65$ ↳ left

0 ——— 15 ——— 31
↑s ↑mid ↑e

$4 \times 15 = 60 < 65$ ↳ store and search in right.
ans = 15

16 ——— 23 ——— 31
↑s ↑mid ↑e

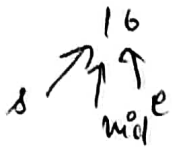
$4 \times 23 = 92 > 65$ ↳ left

16 — 19 — 22
↑s ↑mid ↑e

$19 \times 4 = 72 > 65$ ↳ left

16 — 17 — 18
↑s ↑mid ↑e

$17 \times 4 = 68 > 65$ ↳ left



$16 \times 4 = 64 < 65$

↳ store and search in right.

ans = 16

$s = 17, e = 16$

$s > e \rightarrow$ stop.

↳ return ans = (16)

H·W → Calculate fractional part the same way we did in square root question.

Q → Find Odd occuring element in an array.

index based questions

i/p →

1	1	2	2	3	3	4	4	3	600	600	4	4
0	1	2	3	4	5	6	7	8	9	10	11	12

→ all element occur even no. of times except one.

→ all repeating occurence of element apper in pairs

→ Pairs are not adjacent.

→ There can't be more than two consecutive occurence of any element.

Aproach 1 –

XOR operation.

$a \wedge a = 0$
$a \wedge b = 1$

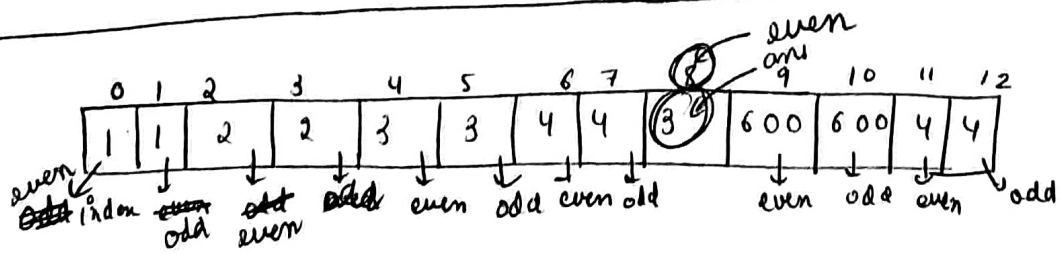
$1 \wedge 1 \wedge 2 \wedge 2 \wedge 3 \wedge 3 \wedge 4 \wedge 4 \wedge 3 \wedge 600 \wedge 600 \wedge 4 \wedge 4$

$= (3)$

T.C = O(n)

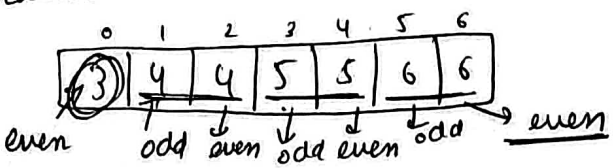
Binar Search Questions (Types) →

1. Classic Question (lower, upper bound, first, last, total occurence, peak, pivot element)
2. Search Space (Predicate func questions)
3. Observation in Index



pairs one element is in even index and one element is in odd index always.

And our ans (single element) will be always in a even index.



One more observation →

→ ans ka left me → pair ka 1st el → even index
pair ka 2nd el → odd index

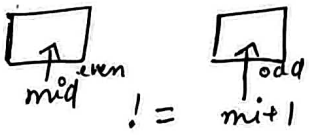
→ ans ka right me → pair ka 1st el → odd index
pair ka 2nd el → even index

Cases –

① mid is even

That means we are in left and the ans is in right part.

s = mid + 2



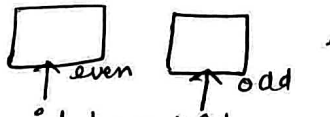
Two possibilities –

① ↳ mid can be ans

② ↳ ans can be in left part.

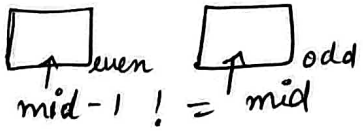
e = mid

② Mid is odd –



→ That means we are in left and ans will be in right.

s = mid + 1



Two possibilities

① ↳ mid is answer left

② ↳ ans is in ~~right~~ part.

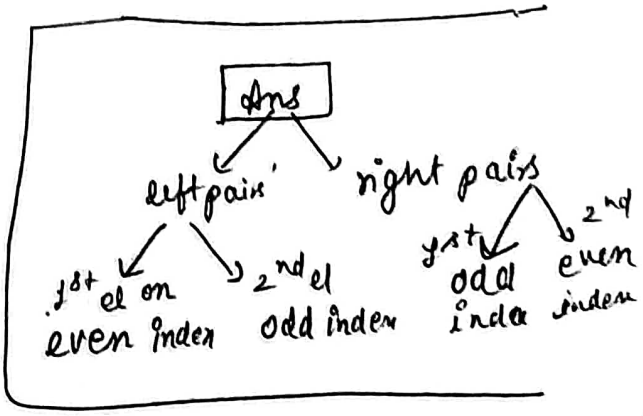
But we can remove one possibility that ans cannot be in a odd index.

so e = mid - 1

Code -

```
int solve(vector<int> arr){
    int s = 0;
    int e = arr.size()-1;
    int mid = s + (e-s)/2;
    while(s <= e)
    {
        if(s==e){
            return s;
        }
        // Case 1
        if(mid % 2 == 0){ → even index
            if(arr[mid]==arr[mid+1]){
                s = mid+2;
            }
            else
                e = mid;
            }
        }
        //Case 2 → odd index
        else{
            if(arr[mid-1]==arr[mid]){
                s = mid+1;
            }
            else{
                e = mid-1;
            }
        } mid = s + (e-s)/2;
    } return -1;
}
```

if we left with only one el



because mid+1 is already checked.

we might loose ans if we do e = mid-1;

mid can't be our answer because answer is at even index (always)

dry run →

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

s, e

s = 0 , e = 12 mid = $\frac{0+12}{2}$ = 6

mid == mid+1

4 (even) == 4 (odd) → go to right.

left गई; s = mid + 2

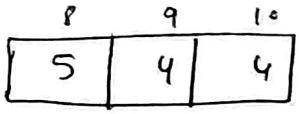
8	9	10	11	12
5	4	4	2	2

s, e

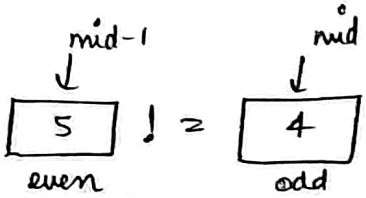
s = 8 , e = 12 , mid = 10

4 (even) != 2 (odd) → या तो यही ans है या left में है।

e = mid



mid = 9



left में search करना है,

e = mid - 1

s == e

↳ वही ans है,

H.W →

1. Find Pairs with difference 'K' in an array.
2. Find 'K' closest element to a given value in an array.
3. Exponential Search
4. Unbounded Binary search
5. Advanced Binary search
   - ↳ Book Allocation → ↳ Roti/ Paratha SPOJ
   - ↳ Painter's Partition → ↳ EKO SPOJ.
   - ↳ Aggressive Cows