Revision -

- Check Anagram
  Success Rate: 90.38% Max Score: 10 Difficulty: Medium
- Isogramic String
  Success Rate: 95.65% Max Score: 10 Difficulty: Medium
- K Frequent Characters
  Success Rate: 100.00% Max Score: 10 Difficulty: Medium
- First Non-Repeating Character
  Success Rate: 95.45% Max Score: 10 Difficulty: Medium

s → "aman"
r → "naam"

a-2	a-2
m-1	m-1
n-1	n-1

arr

Approach.
↳ 2 freq arr
↳ individual cells.

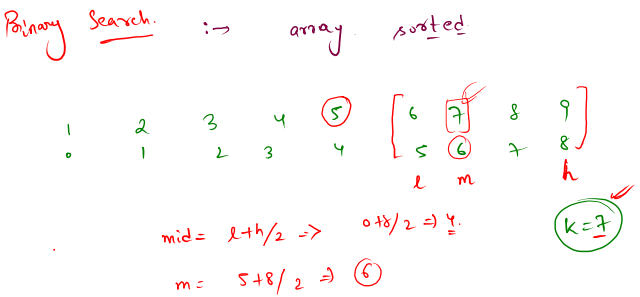
s → "aman" → not. a-2
↳ freq arr.
↳ freq(i) ≤ 1

aman
freq arr.
a
m

k=2.

k times
↳ max
↳ char

aabbdkpptt
↳ k
↳ freq arr.
0 → 25
freq[i] == 1

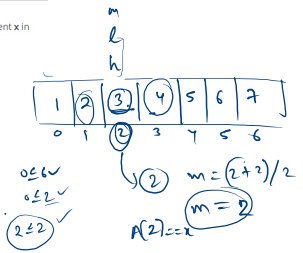


Binary Search in an Array

Problem | Submissions | Leaderboard | Discussions

Given a array of n elements sorted in the **increasing** order, write a function to search a given element **x** in array and print the **index** of x in the array.

Note: Consider the array is 0 based index and also that x definitely lies in range [0,arr.length].



Sample Input 0

```
7
1 2 3 4 5 6 7
3
```

Sample Output 0

```
2
```

```java
public static int binarySearch(int [] A, int x){
    int low = 0;
    int high = A.length-1;

    while(low <= high){
        int mid = (low + high) / 2;
        if(A[mid] == x){
            return mid;
        }
        else if (A[mid] > x){
            high = mid - 1;
        }
        else{
            low = mid + 1;
        }
    }

    return -1;

}
```

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
    public static int binarySearch(int [] A, int x){
        int low = 0;
        int high = A.length-1;

        while(low <= high){
            int mid = (low + high) / 2;
            if(A[mid] == x){
                return mid;
            }
            else if (A[mid] > x){
                high = mid - 1;
            }
            else{
                low = mid + 1;
            }
        }

        return -1;

    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
         int x = scn.nextInt();
        int ans = binarySearch(A,x);
        System.out.println(ans);
    }
}
```

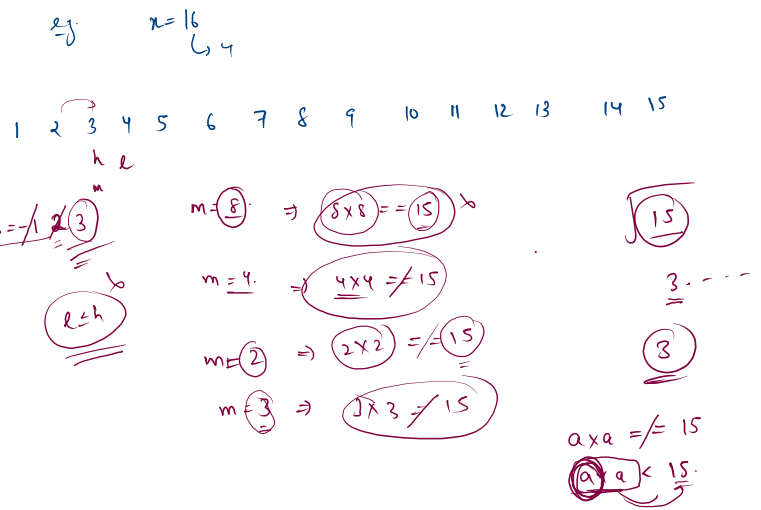
Find Square Root

Problem | Submissions | Leaderboard | Discussions

Given an integer number n, find its square root using binary search.

If exact square root of n is not possible then print the just **nearest** and **smaller** perfect square to n.

For example: if n=79, then nearest square root will be 8, not 9.




```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
    public static int func(int n){
        int low = 1;
        int high = n;
        int ans = -1;

        while(low <= high){
            int mid = (low + high)/2;

            if(mid * mid == n){
                return mid;
            }
            else if (mid * mid > n){
                high = mid - 1;
            }
            else{
                ans = mid;
                low = mid + 1;
            }
        }
        return ans;

    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int ans = func(n);
        System.out.println(ans);

    }
}
```

Find Last Occurrence

Problem | Submissions | Leaderboard | Discussions

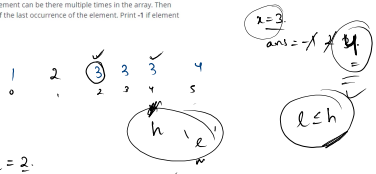
Given an array sorted in the increasing order and an element can be there multiple times in the array. Then take an integer input x. In the end the print the index of the last occurrence of the element. Print -1 if element is not present in array.

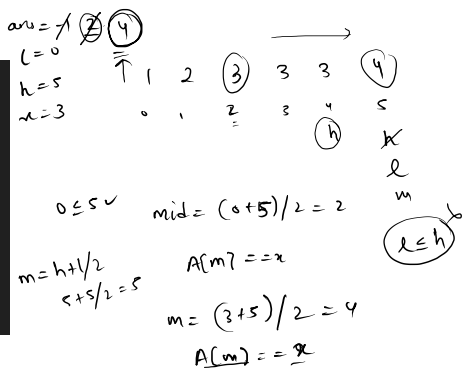
Sample Input 0

```
6
1 2 3 3 3 4
3
```

Sample Output 0

```
4
```






```java
public class Solution {
    public static int func(int [] A, int x){
        int low = 0;
        int high = A.length-1;
        int ans = -1;

        while(low <= high){
            int mid =  (low + high) / 2;

            if(A[mid] == x){
                ans = mid;
                low = mid + 1;
            }
            else if(A[mid] > x){
                high = mid - 1;
            }
            else{
                low = mid + 1;
            }

        }
        return ans;
    }
```

```java
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
    public static int func(int [] A, int x){
        int low = 0;
        int high = A.length-1;
        int ans = -1;

        while(low <= high){
            int mid =  (low + high) / 2;

            if(A[mid] == x){
                ans = mid;
                low = mid + 1;
            }
            else if(A[mid] > x){
                high = mid - 1;
            }
            else{
                low = mid + 1;
            }

        }
        return ans;

    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        int x = scn.nextInt();
        int ans = func(A,x);
        System.out.println(ans);
    }
}
```

Subtract the Product and Sum of Digits of an Integer 1

Problem | Submissions | Leaderboard | Discussions

Given an integer number n, return the difference between the product of its digits and the sum of its digits.

Sample Input 0

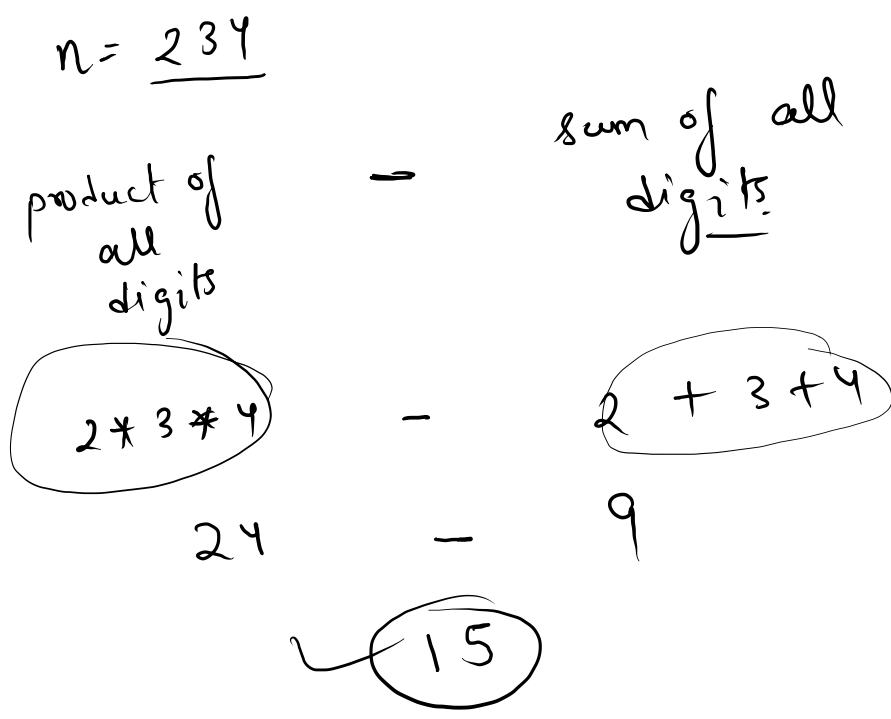
```
234
```

Sample Output 0

```
15
```

Explanation 0

1. Product of Digits->2*3*4=24
2. Sum Of Digit->2+3+4=9
3. Difference->15

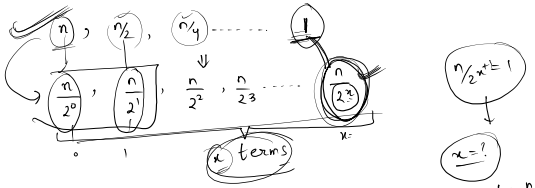


Time Complexity of Binary Search.


```
public static int binarySearch(int [] A, int x){
    int low = 0;
    int high = A.length-1;

    while(low <= high){
        int mid = (low + high) / 2;
        if(A[mid] == x){
            return mid;
        }
        else if (A[mid] > x){
            high = mid - 1;
        }
        else{
            low = mid + 1;
        }
    }

    return -1;
}
```



$\frac{n}{2^x} = 1$

$n = 2^x$

$\log_2 n = x \log_2 2$

$\log_2 n = x$

$\Rightarrow x = \log_2 n$

if n elements
do x iterations

$n/2^{x+1} = 1$

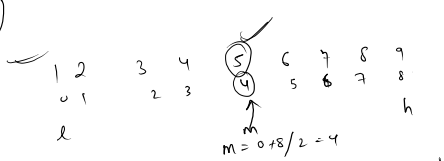
$x = ?$

$x+1 = \log_2 n$

$x = \log_2 n - 1$

$x = O(\log n)$

$x = 5$



$\omega(1)$

$O(\log n)$