Missing number 1 to n

03 December 2024 00:50

```
N=4
 we need to enter 3 input only: 1 2 3
missing number is 4

total sum of natural numbers upto 4 is 10 (1,2,3,4)
actual array sum is 6 so 10-6=4
```

```
int n = sc.nextInt()
        int[] nums = new int[n - 1] // Since one number is missing

        for (int i = 0; i < n - 1; i++)
            nums[i] = sc.nextInt()


        int totalSum = n * (n + 1) / 2
        //bracket (n+1) evaluated then multiplied with n then /2

        int arraySum = 0;
        for (int num : nums)
            arraySum += num;

        int missing = totalSum - arraySum
```

```
Using Bitwise Xor:
----------------------

       int xortotal = 0;
        for (int i = 1; i <= n; i++)
            xortotal ^= i


        int xorarray = 0;
        for (int num : nums)
            xorarray ^= num

      // The missing number is the result of xortotal ^ xorarray
```

```
int missingNumber = xortotal ^ xorarray
```

if 0 to be included in the array then just change

```
 int n = sc.nextInt();
int[] nums = new int[n] //enter the values upto n

for (int i = 0; i < n; i++)
    nums[i] = sc.nextInt()
```

Pair with given Sum

02 December 2024 23:43

```
input: 1 2 3 4 5 6 7
sum=7

number of pairs=3 (3,4),(1,6),(5,2)
```

```
int n = scanner.nextInt();
int target = scanner.nextInt()
int[] arr = new int[n]
for (int i = 0; i < n; i++)
    arr[i] = scanner.nextInt()

HashMap<Integer, Integer> freq = new HashMap<>()
int cnt = 0

for (int i = 0; i < arr.length; i++)
    int complement=target-arr[i]
    if (freq.containsKey(complement))
        cnt += freq.get(complement)
    then
    freq.put(arr[i], freq.getOrDefault(arr[i], 0) + 1)


System.out.println(cnt)
```

Anagaram Strings

02 December 2024 23:45

listen and silent are anagarams

```
HashMap<Character, Integer> hmap = new HashMap<>()

        for (char ch : s1.toCharArray())
            hmap.put(ch, hmap.getOrDefault(ch, 0) + 1)

        for (char ch : s2.toCharArray())
            hmap.put(ch, hmap.getOrDefault(ch, 0) - 1)

         boolean flag = true

        for (var pair : hmap.entrySet())
            if (pair.getValue() != 0)
                flag = false
                break


        if (flag==true)
            print("strings are anagarams")
         else
            print("strings are not anagrams")
```

Sum with queries

03 December 2024 16:12

You are given an array of integers of size N. You are also given Q queries consisting of three integers i, j, and x.
For each query, increment each element of the array from index i to j by a value of x. At the end, print the sum of all the elements of the array.

```
Input:
2 (2 Test cases)
5
1 5 -3 2 8
2  (queries)
1 3 1   (1 and 3 are indicies 1 is the value to be added)
0 1 2
6
4 10 -1 2 8 -3
1
3 5 6

Output
20
38
```

```
int T=sc.nextInt()
 while(T-->0)
{
     int n=sc.nextInt()
     int[] arr=new int[n]
     long sum=0
     for(int i=0;i<n;i++)
         arr[i]=sc.nextInt()
         sum+=arr[i]



     int q=sc.nextInt()
     while(q-->0)
         long i=sc.nextInt()
         long j=sc.nextInt()
         long x=sc.nextInt()
         sum+=(j-i+1)*x

     print(sum)


 }
```

A query modifies elements in the range [i, j] by adding x to each.

- The number of elements in this range is given by:
- Count of Elements=(j−i+1)

Each element in this range contributes x to the total increment in the sum.

Therefore, the total contribution of this query to the array sum is:
(j-i+1)*x

Instead of modifying the array and recomputing the sum each time (which is time-consuming for large arrays), we directly add this contribution to the current sum.

in j-i+1 add 1 bcz Add 1 because both i and j are included in the range. Adding 1 is necessary because the subtraction j−i only gives the difference between the indices, not the count of elements.

Time Complexity
For Q queries and an array of size N:
naïve approach will take worst case O(QxN)

optimized approach will take: O(Q) for queries + O(N) for the final sum computation = O(N+Q)

Naïve Approach:

```
for (int q = 0; q < Q; q++)
    int i = queryStartIndex;
    int j = queryEndIndex;
    int x = incrementValue;

    // Update array for each query in the range [i, j]
    for (int k = i; k <= j; k++) {
        arr[k] += x;
```

```
long sum = 0;
for (int value : arr)
    sum += value;
```

bitonic Sequence

05 November 2024 01:21

```
/*
Input:
arr = 1 3 5 4 2
output=True

input=3 2 1 0
output=false

input=1 2 3 4 5
output=false

A valid bitonic sequence must have at least one element before and one element
after the peak.
its increasing upto certain element and then decreases

*/

N = input
arr = array of size N
for i from 0 to N - 1
    read array values


if N < 3
    output false
    return

i = 0
while i < N - 1 and arr[i] < arr[i + 1]
    i = i + 1
end while

if i == 0 or i == N - 1
    output false
    return
end if

while i < N - 1 and arr[i] > arr[i + 1]
    i = i + 1
end while

if(i==N-1)
print true
else
print false
```

Find Ciel of Sorted array

03 December 2024 00:06

```
Given a sorted array in ascending order and a value key,
the ceiling of key is the smallest element in array greater than or equal to key.


input=6
1 2 3 5 6 7
4
output=
The ceiling of 4 is: 5

int arr[]={1,2,3,5,6,7}
int n=arr.length
int x=0
int high=n-1
int x=given key


if (x <= arr[low])
            return arr[low]

        for (int i = low; i < high; i++)
            if (arr[i] == x)
                return arr[i]

            if (arr[i] < x && arr[i + 1] >= x)
                return arr[i+1]


        return -1
```

Find floor of the sorted array

03 December 2024 00:11

```
the floor of key is the greater element in the array less than or equal to key.

input=6
1 2 3 5 6 7
4
output=
The floor of 4 is: 3


if (key < arr[0])
            return -1


        for (int i = n - 1; i >= 0; i--)
            if (arr[i] <= key)
                return arr[i]


        return -1
```

Rotate 2D matrix in Place clockwise 90 degrees

02 December 2024 23:57

```
input matrix:

1 2 3
4 5 6
7 8 9

Rotated Matrix:

7 4 1
8 5 2
9 6 3
```

```
Step 1: Transpose the Matrix:
The transpose of a matrix is obtained by swapping the rows with the columns.
This turns the element at position (i,j) into position (j,i)

1 4 7
2 5 8
3 6 9
```

```
 int n = matrix.length;

for (int i = 0; i < n; i++)
  for (int j = i; j < n; j++)
      // Swap matrix[i][j] with matrix[j][i] to transpose
      int temp = matrix[i][j]
      matrix[i][j] = matrix[j][i]
      matrix[j][i] = temp
```

```
After transposing, to complete the 90-degree rotation, reverse each row.
to get rotated matrix.
7 4 1
8 5 2
9 6 3

Reversing each row of matrix using 2 pointer approach
-----------------------------------------------------
for (int i = 0; i < n; i++)
            int left = 0
            int right = n - 1
            while (left < right)
                int temp = matrix[i][left]
                matrix[i][left] = matrix[i][right]
                matrix[i][right] = temp
```

```
                left++
               right--

           end while
 end for
```

Another Approach:
----

the below method is just for printing the rotated value not changing matrix.
this will work for exam point of view.

```
      n=matrix.length
        for(int i=0;i<n;i++)
            for(int j=n-1;j>=0;j--)
                System.out.print(arr[j][i]+" ")

            System.out.println()
```

Kth smallest element naive

02 December 2024 23:10

```
input=
4
3
 7   4   8
 3   5   89
12  56   10
14  45   100
4
output=
The 4th smallest element is: 7
```

```
             int rows = matrix.length
            int cols = matrix[0].length

            int[] flatarray= new int[rows * cols]

            int index = 0
            for (int i = 0; i < rows; i++)
                for (int j = 0; j < cols; j++)
                    flatArray[index++] = matrix[i][j]


            Arrays.sort(flatArray)
            print flatArray[k - 1]
```

```
Better approach is to use Min heap to store the smallest elements
---------------
        int rows = matrix.length
        int cols = matrix[0].length


        PriorityQueue<Integer> minHeap = new PriorityQueue<>()

        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                minHeap.offer(matrix[i][j])
```

```
        int kthSmallest = -1
        for (int i = 0; i < k; i++)
          kthSmallest = minHeap.poll()

        print kthSmallest
```

In a min-heap, the smallest element is always at the root (top) of the heap, and every parent node has a value less than or equal to its children.

For a min-heap each poll operation will remove the smallest element

[7, 10, 3, 1, 5, 8, 4]

step by step creation of min heap

```
    7
   /
  10


   3
  / \
 10  7

   1
  / \
 3   7
/
10
   1
  / \
 3   7
/ \ /
10 5 8

   1
  / \
 3   4
/ \ / \
10 5 8  7
```

Insert 4 as the right child of 3.
We swap 4 with 7 to maintain the min-heap property, as 4 is smaller than 7.

final tree as min heap

```
   1
  / \
 3   4
/ \ / \
```

```
10 5 8  7

suppose K is 4:
4th smallest element is 5.


Suppose k = 4. 4th smallest elemnt is 5

after first poll 1 is removed.
    3
   / \
  5   4
 / \  /
7   10 8
after second poll 3 is removed.
    4
   / \
  5   8
 / \
7   10
after 3rd poll 4 is removed.
    5
   / \
  7   8
 /
10
after 4th poll 5 is removed.
    7
   / \
  10  8


*/
```

Quadruples of XOR

03 December 2024 00:18

```
You are given 4 arrays of integers: A, B, C, and D.
You have to find the number of quadruples (i, j, k, l)
such that A[i] ^ B[j] ^ C[k] ^ D[l] = 0,
where ^ is the bitwise XOR operator.

Input format:
-------------------
4  (sizes of array)
//next enter 4 arrays of size 4
31 8 28 10
18 7 22 5
16 25 20 14
39 9 34 19

Output:
2
```

```
Quadruples(int[] A, int[] B, int[] C, int[] D, int n)
        Map<Integer, Integer> map = new HashMap<>()
        int count = 0

        // Step 1: Store all possible XOR results of A[i] ^ B[j]
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                int xorAB = A[i] ^ B[j]
                map.put(xorAB, map.getOrDefault(xorAB, 0) + 1)



        // Step 2: Check for each pair C[k] ^ D[l] and look for its opposite in the map
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                int xorCD = C[i] ^ D[j]
                if (map.containsKey(xorCD))
                    count += map.get(xorCD)


        return count
```

Input:

- A=[1,2]
- B=[3,4]
- C=[5,6]
- D=[7,0]

Compute A[i]⊕B[j]:
calculate the XOR values of all pairs from A and B, storing their frequencies in the hashmap.

1. For A[0]=1
   - 1⊕3=2
   - 1⊕4=5

   For A[1]=2
   - 2⊕3=1

- ◦ 2⊕4=6

Now, the map contains:

map={2:1
 5:1
 1:1
 6:1}

Compute C[k]⊕D[l]:

We calculate the XOR values of all pairs from CCC and DDD, checking each against the hashmap.

For C[0]=5

5⊕7=2 → Found in map with frequency 1 → Increment count by 1.
5⊕0=5 → Found in map with frequency 1 → Increment count by 1.

For C[1]=6

C[1] = 6
6⊕7=1 → Found in map with frequency 1 → Increment count by 1.
6⊕0=6 → Found in map with frequency 1 → Increment count by 1.

Hence final answer is 4

Matches found are: if u cross verify all xor will be zero.
(A[0],B[0],C[0],D[0])
(A[0],B[1],C[0],D[1])
(A[1],B[0],C[1],D[0])
(A[1],B[0],C[1],D[0])
(A[1],B[1],C[1],D[1])

Longest Consecutive 1's

05 November 2024 00:53

```
declare maxlength as 0  // to store the longest sequence of 1s
declare a as 0  // to count current sequence of contiguous 1s

for i from 0 to n - 1
    if arr[i] equals 1
        increment count
        if coubt greater than maxlength
            set maxlength to count

    if arr[i] equals 0
        set count to 0

print maxlength
```

Bubble sort efficient

27 November 2024 01:15

```
int temp;
boolean flag;
for (int i = 0; i < n - 1; i++) {
    flag = false;
    for (int j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {

            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
            flag = true;
        }
    }
    if (!flag) {
        break;
    }

    System.out.print("Pass " + (i + 1) + ": ");
    for (int k = 0; k < n; k++) {
        System.out.print(arr[k] + " ");
    }
    System.out.println();
}

System.out.println("Sorted array is:");
for (int i = 0; i < n; i++) {
    System.out.print(arr[i] + " ");
}
```

Lucky Numbers

27 November 2024 01:16

```
            {3, 7, 8}
            {9, 11, 13}
            {15, 16, 17}

here lucky number is 15 which is minimum in its row and maximum in its column
```

```
Initialize matrix with values
Create an empty list called luckyNumbers

Get number of rows m and columns n in the matrix

For i from 0 to m - 1
    Initialize minRowIndex to 0
    Initialize minRowValue to matrix[i][0]

    For j from 1 to n - 1
        If matrix[i][j] is < minRowValue
            Update minRowValue to matrix[i][j] and minRowIndex to j

    Initialize flag as true
    For k from 0 to m - 1
        If matrix[k][minRowIndex] is > minRowValue
            Set flag to false
            Break


   if(flag)
            luckyNumber = minRowValue;
            break; // Since only one lucky number exists,
```

Longest Prefix Suffix

02 December 2024 23:47

```
input: grietcollegegriet
output:5
```

```
String s=sc.next();
 int n=s.length();
 int longest=0;
 for(int i=1;i<n;i++){
     if(s.substring(0,i).equals(s.substring(n-i)))
        longest=i


 print(longest)
```

```
n = len(s)
    longest = 0
    for i in range(1, n):
        if s[:i] == s[-i:]:
            longest = i

print longest
```

String GCD

28 November 2024 14:27

```
Input
ABABAB
ABAB

Output
AB
```

```
        String P = sc.nextLine()
        String Q = sc.nextLine()
        if (!(P + Q).equals(Q + P)) {
            System.out.println(-1);
        else
            int gcdLength = gcd(P.length(), 
Q.length())
            System.out.println(P.substring(0, 
gcdLength))


     int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }
}
```

Number of Monotonous Sub Strings

02 December 2024 23:51

```
Input: "abbcccaa"

For the first 'a': 1 substring ("a")
For the 'first b': 1 substring ("b")
For the first 'c': 1 substring ("c")


For the second 'c': 2 substrings ("c", "cc")
For the third 'c': 3 substrings ("c", "cc", "ccc")


For the first 'a': 1 substring ("a")
For the second 'a': 2 substrings ("a", "aa")

Total: 1 + 1 + 1 + 2 + 3 + 1 + 2 = 13

example 2
------------

String: "aaaa"
For the first 'a': 1 substring ("a")
For the second 'a': 2 substrings ("a", "aa")
For the third 'a': 3 substrings ("a", "aa", "aaa")
For the fourth 'a': 4 substrings ("a", "aa", "aaa", "aaaa")

Total: 1 + 2 + 3 + 4 = 10
```

```
int MOD = 1000000007
int n = sc.nextInt(); // Length of the string
String s = sc.next(); // The string itself
long result = 0;
int count = 1; //atleast one substring is der


for (int i = 1; i < n; i++)
    if (s.charAt(i) == s.charAt(i - 1))
        count++
    else
        // If the character changes, calculate the number of monotonous substrings
        result += (long) count * (count + 1) / 2;
        result %= MOD;
        count = 1;
    }
}
//for the last character
result += (long) count * (count + 1) / 2;
result %= MOD;


print(result)
```

```
if 3 consecutive: c, c, c
Length of consecutive c: 3
```

```
Number of monotonous substrings:
1 (Just "c") + 2 (Substrings of length 2: "c", "cc") + 3 (Substrings of length 3: "c,cc,ccc")
Total substrings are 6. same can be achieved using the following formula:

Total =3×(3+1)/2
 =6 substrings
```

Spiral Traversal Matrix

27 November 2024 01:26

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

int top = 0;
int bottom = matrix.length - 1;
int left = 0;
int right = matrix[0].length - 1;

while (top <= bottom && left <= right) {
    for (int i = left; i <= right; i++) {
        System.out.print(matrix[top][i] + " ");
    }
    top++;

     // Traverse from top to bottom along the right column
    for (int i = top; i <= bottom; i++) {
        System.out.print(matrix[i][right] + " ");
    }
    right--;


    if (top <= bottom) {
        // Traverse from right to left along the bottom row
        for (int i = right; i >= left; i--) {
            System.out.print(matrix[bottom][i] + " ");
        }
        bottom--;
    }

    if (left <= right) {
        // Traverse from bottom to top along the left column
        for (int i = bottom; i >= top; i--) {
            System.out.print(matrix[i][left] + " ");
        }
        left++;
    }
}
```

```
}
```

Zero Matrix

27 November 2024 01:39

```
Original Matrix:
 1 1 1
 1 0 1
 1 1 1
Modified Matrix:
 1 0 1
 0 0 0
 1 0 1
```

```
boolean[] row = new boolean[n];
boolean[] col = new boolean[n];
       for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        if (matrix[i][j] == 0)
            row[i] = true
            col[j] = true
```

```
// Step 2: Set the rows to zero
for (int i = 0; i < n; i++)
    if (row[i])
        for (int j = 0; j < n; j++)
            matrix[i][j] = 0
```

```
// Step 3: Set the columns to zero
for (int j = 0; j < n; j++)
    if (col[j])
        for (int i = 0; i < n; i++)
            matrix[i][j] = 0;
```

print the matrix.

XoR of sum of all Pairs

27 November 2024 01:50

```
if array is [1,2,3]
Pairs and their sums:
   (1 + 1) = 2
   (1 + 2) = 3
   (1 + 3) = 4
   (2 + 1) = 3
   (2 + 2) = 4
   (2 + 3) = 5
   (3 + 1) = 4
   (3 + 2) = 5
   (3 + 3) = 6

XOR of these sums=0
```

```
        int[] arr = {1, 2, 3}
        int n = arr.length
        int result = 0


        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                int sum = arr[i] + arr[j]
                result ^= sum;
```

```
       print result as xor of sum of pairs.
```

An Efficient approach is based upon the fact that xor of the same values is 0. All the pairs like (a[i], a[j]) and (a[j], a[i]) will have same sum. So, their xor values will be 0. Only the pairs like (a[i], a[i]) will give the different result. So, take the xor of all the elements of the given array and multiply it by 2.

```
 int xoR = 0;
        for (int i = 0; i < n; i++) {
            xoR = xoR ^ arr[i]
        }


        print xoR * 2
```

```
}
```

When optimizing the code, we recognize that:

1. Symmetric pairs cancel out (XOR becomes 0 for those pairs).
2. Only diagonal pairs (a[i], a[i]) contribute, and their sums are 2 * a[i].

Thus, instead of explicitly summing diagonal pairs, we:

1. Compute the XOR of all array elements (xoR = a[0] ^ a[1] ^ ... ^ a[n-1]).
2. Multiply the result by 2 because each diagonal contributes twice the value of the element.
3. Diagonal Pairs in the Array
4. For an array [1, 2, 3], the diagonal pairs are:
    1. (1 + 1) = 2
    2. (2 + 2) = 4
    3. (3 + 3) = 6
5. Each diagonal pair is the sum of an element with itself, i.e., a[i] + a[i] = 2 * a[i].

The final XOR computation is equivalent to XORing these diagonal values:

- result = (2) ^ (4) ^ (6)

Sum of XoR of all pairs

27 November 2024 01:51

```
Input : arr[] = {5, 9, 7, 6}
Output : 47
5 ^ 9 = 12
9 ^ 7 = 14
7 ^ 6 = 1
5 ^ 7 = 2
5 ^ 6 = 3
9 ^ 6 = 15
Sum = 12 + 14 + 1 + 2 + 3 + 15
    = 47
```

```
int[] arr = {5, 9, 7, 6}
int n = arr.length
int result = 0

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        int xorSum = arr[i] ^ arr[j]
        result += xorSum



System.out.println("XOR sum of all pairs: " + result)
```

Sum of XOR of all subarrays

29 November 2024 20:39

Given an array containing N positive integers, the task is to find the sum of XOR of all sub-arrays of the array.

```
Input : arr[] = {3, 8, 13}
Output : 46

XOR of {3} = 3
XOR of {3, 8} = 11
XOR of {3, 8, 13} = 6
XOR of {8} = 8
XOR of {8, 13} = 5
XOR of {13} = 13
Sum = 3 + 11 + 6 + 8 + 5 + 13 = 46
```

```
int sum = 0

       for (int i = 0; i < n; i++)
           int xorr = 0
           for (int j = i; j < n; j++)
               xorr = xorr ^ arr[j]
               sum += xorr


      print sum
```

```
  when i=0
  xorr = 0.
inner loop j is 0 so
 Subarray: {3}
  xorr = 0 ^ 3 = 3
 sum = 0 + 3 = 3
 j becomes 1
 Subarray: {3, 8}
 xorr = 3 ^ 8 = 11
  sum = 3 + 11 = 14
 j becomes 2
 Subarray: {3, 8, 13}
 xorr = 11 ^ 13 = 6
  sum = 14 + 6 = 20

 continue similary for for i=1 and subarrays are 8 and 8,13
 total sum would be 33
 when i=2, subarrays is only 13.
 so final sum will 33+13=43
```

Tracing the Execution
Outer Loop (i = 0):

- Starting from index i = 0.
- Initialize xorr = 0.

Inner Loop (j = 0):

- Subarray: {3}
- Compute XOR: xorr = 0 ^ 3 = 3
- Add to sum: sum = 0 + 3 = 3

Inner Loop (j = 1):

- Subarray: {3, 8}
- Compute XOR: xorr = 3 ^ 8 = 11
- Add to sum: sum = 3 + 11 = 14

Inner Loop (j = 2):

- Subarray: {3, 8, 13}
- Compute XOR: xorr = 11 ^ 13 = 6
- Add to sum: sum = 14 + 6 = 20

Outer Loop (i = 1):

- Starting from index i = 1.
- Initialize xorr = 0.

Inner Loop (j = 1):

- Subarray: {8}
- Compute XOR: xorr = 0 ^ 8 = 8
- Add to sum: sum = 20 + 8 = 28

Inner Loop (j = 2):

- Subarray: {8, 13}
- Compute XOR: xorr = 8 ^ 13 = 5
- Add to sum: sum = 28 + 5 = 33

Outer Loop (i = 2):

- Starting from index i = 2.
- Initialize xorr = 0.

Inner Loop (j = 2):

- Subarray: {13}
- Compute XOR: xorr = 0 ^ 13 = 13
- Add to sum: sum = 33 + 13 = 46