Mean Median Mode

04 November 2024 23:02

Mean=(double) sum of array elements/n

Median=

```
double median=0
     if(n%2==0)
        median=(arr[(n/2)-1]+arr[(n/2)])/2.0

     else
         median=arr[n/2]
```

Mode= to calculate Mode use HashMap

```
Map<Integer,Integer> count=new HashMap<>()
```

```
mode = arr[0]
maxcount = 0
for i from 0 to n - 1
    curcount = count.get(arr[i])
    if curcount > maxcount
        maxcount = curcount
        mode = arr[i]
```

output mean, median, mode formatted to 2 decimal places

Duplicate Number Using HashMap

04 November 2024 23:07

```
int[] arr=new int[n]

        HashMap<Integer,Integer> mapobj=new HashMap<>()

        for(int i=0;i<n;i++)
            arr[i]=sc.nextInt()
            mapobj.put(arr[i],mapobj.getOrDefault(arr[i],0)+1)
```

OR

```
        for (int value : arr) {
        mapobj.put(value, mapobj.getOrDefault(value, 0) + 1)

        for(Map.Entry<Integer,Integer> e:mapobj.entrySet())
            if(e.getValue()>1)

                System.out.println(e.getKey()+" "+e.getValue())
```

Pair Gauntlets

04 November 2024 23:07

```
/* 1 ≤ N ≤ 10^2
1 ≤ A[i] ≤ 10^3

Input:
6
4 1 7 4 1 4
Output:
2
*/

  int[] arr=new int[n]


 Map<Integer, Integer> color = new HashMap<>()
        for (int i = 0; i < n; i++)
            colorCount.put(A[i], color.getOrDefault(A[i], 0) + 1)


        int pairs = 0
        for (int count : color.values())
            pairs += count / 2


        System.out.println(pairs)
```

Missing Number

04 November 2024 23:07

```
int[] arr=new int[99]

sum = 0
for i from 0 to 99
    input arr[i]
    sum = sum + arr[i]

total = 100 * (100 + 1) / 2
output (total - sum)
```

Max Altitude

04 November 2024 23:07

Print the highest altitude the pilot can reach.

Constraints
---
1 <= N <= 1000
-1000 <= A[i] <= 1000

Input

5
-5 1 5 0 -7
Output
1

```
declare max as 0
declare sum as 0

for i from 0 to n - 1
    sum =sum +a[i]
    if sum is greater than max
        set max to sum

print max
```

Print Array A from B

05 November 2024 00:53

```
/* Given two arrays, A and B, for each index 'i' in array B, print the
corresponding element from array A if B[i] is within the range of A, otherwise
print -1.

Constraints
----------
1 <= N <= 100
1 <= Arr1[i] <= 1000
1 <= M <= 100
0 <= Arr2[i] <= 1000
*/

declare arr1 as new integer array of size n

for i from 0 to n - 1
    read arr1 value

declare m as input integer

declare arr2 as new integer array of size m

for i from 0 to m - 1
     read arr2 value

for i from 0 to m - 1
    if arr2[i] >=0 and arr2[i]<n
        print arr1[arr2[i]]
    else
        print -1
```

Merge Sorted Arrays

05 November 2024 00:53

```
declare n as input integer
declare arr as new integer array of size n

for i from 0 to n - 1
    read arr value

declare m as input integer
declare arr1 as new integer array of size m

for i from 0 to m - 1
    read arr1 value

declare arr2 as new integer array of size n + m

for i from 0 to n - 1
    set arr2[i] to arr[i]

for i from 0 to m - 1
    set arr2[n + i] to arr1[i]

sort arr2

for i from 0 to n + m - 1
    print arr2[i]
```

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 a
        if a greater than maxlength
            set maxlength to a

    if arr[i] equals 0
        set a to 0

print maxlength
```

Alternate Seating

05 November 2024 00:59

```
/*
 You are given an integer N, denoting the number of people who need to be seated,
and a list of M seats, where 0 represents a vacant seat and 1 represents an already occupied
seat.
Find whether all N people can find a seat, provided that no two people can sit next to each
other.
Constraints
---------
1 ≤ N ≤ 10^5 (number of ppl)
1 ≤ M ≤ 10^5 (number of seats)
Ai ∈ {0, 1}

Input:
2
7
0 0 1 0 0 0 1
Output:
YES
*/


        read integer N  // number of people
        read integer M  // number of seats

        create array seats of size M

        loop from i = 0 to M - 1
            read seats[i]

        initialize count to 0

        loop from i = 0 to M - 1
            if seats[i] is 0
                if i is 0 or seats[i - 1] is 0  (if it's the first seat or previous seat is
vacant)
                    increment count  (a person can sit here)
                    increment i  (skip the next seat to ensure no adjacent seating)

        if count is greater than or equal to N
            print "YES"
        else
            print "NO"
```

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
```

Sum of all Subarrays

05 November 2024 01:28

The total number of subarrays in an array of size N is N * (N + 1) / 2.
elements should be contigous.

For an array [1, 2, 3], subarrays are: [1], [2], [3], [1, 2], [1, 2, 3], [2,3]

1 appears in subarrays: [1], [1, 2], [1, 2, 3]
Contribution of 1 = 1 * 3 = 3

2 appears in subarrays: [2], [1, 2], [2, 3], [1, 2, 3]
Contribution of 2 = 2 * 4 = 8

3 appears in subarrays: [3], [2, 3], [1, 2, 3]
Contribution of 3 = 3 * 3 = 9

Hence sum of all subarrays can be said as: 3 + 8 + 9 = 20.

```
n = arr.length
long totalsum = 0

for i from 0 to n - 1
    long contribution = arr[i] * (i + 1) * (n - i)
    totalsum = totalsum + contribution
end for

print totalsum
```

Sum of all Subsequences

05 November 2024 01:28

The total number of subsequences possible is equal to 2^n. (including empty subsequence)
elements need not to be contigous.

For an array [1, 2, 3], subsequences are:
[]
[1]
[2]
[3]
[1, 2]
[1, 3]
[2, 3]
[1, 2, 3]

In an array subsequence, each element appears in 2^n-1 subsequences.

first calculate array sum: which is 1+2+3=6
count of each element: 2^3-1=2^2=4

print(sum*countofeachelement) 6*4=24

```
sum = 0
for num in arr
    sum = sum + num
```

```
long res = (long) Math.pow(2, n - 1);
or
long res= 1L << (n-1))
```

```
print (sum * res)
```

Maximum Subarray Sum (Kadane's Algorithm)

05 November 2024 01:25

```
/*
Input: arr[] = {2, 3, -8, 7, -1, 2, 3}
Output: 11
Explanation: The subarray {7, -1, 2, 3} has the largest sum 11. */



n = scanner.nextInt()

for i from 0 to n - 1
    arr[i] = scanner.nextInt()
end for

res = arr[0]
maxEnding = arr[0]

for i from 1 to n - 1
    maxEnding = max(arr[i], maxEnding + arr[i])
    res = max(res, maxEnding)


output res
```