Codility_

CodeCheck Report: trainingBZE9PH-BPD

Test Name:

Check out Codility training tasks

Summary Timeline

Tasks summary

Task	Time spent	Score
CountDistinctSlices Java 8	1 min	100%

Total score

Tasks Details

1. **CountDistinctSlices**
Easy
Count the number of distinct slices (containing only unique numbers).

Task Score	Correctness	Performance
100%	100%	100%

Task description

An integer M and a non-empty array A consisting of N non-negative integers are given. All integers in array A are less than or equal to M.

A pair of integers (P, Q), such that 0 ≤ P ≤ Q < N, is called a *slice* of array A. The slice consists of the elements A[P], A[P + 1], ..., A[Q]. A *distinct slice* is a slice consisting of only unique numbers. That is, no individual number occurs more than once in the slice.

For example, consider integer M = 6 and array A such that:

```
A[0] = 3
A[1] = 4
A[2] = 5
A[3] = 5
A[4] = 2
```

There are exactly nine distinct slices: (0, 0), (0, 1), (0, 2), (1, 1), (1, 2), (2, 2), (3, 3), (3, 4) and (4, 4).

The goal is to calculate the number of distinct slices.

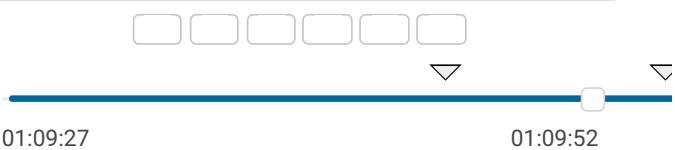
Write a function:

```
class Solution { public int solution(int M,
int[] A); }
```

Solution

Programming language used:	Java 8
Total time used:	1 minutes
Effective time used:	1 minutes
Notes:	*not defined yet*

Task timeline



01:09:27 01:09:52

Code: 01:09:52 UTC, java, final, score: **100**

show code in pop-up

```
1  class Solution {
2          public static final int MAX = 1000000(
```

that, given an integer M and a non-empty array A consisting of N integers, returns the number of distinct slices.

If the number of distinct slices is greater than 1,000,000,000, the function should return 1,000,000,000.

For example, given integer M = 6 and array A such that:

```
A[0] = 3
A[1] = 4
A[2] = 5
A[3] = 5
A[4] = 2
```

the function should return 9, as explained above.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [1..100,000];
- M is an integer within the range [0..100,000];
- each element of array A is an integer within the range [0..M].


```
3
4           public int solution(int M, int[] A) {
5
6                   int N = A.length;
7                   int[] indices = new int[M + 1]
8
9                   long count = 0L;
10
11                  int left = 0;
12
13                  for (int right = 0; right < N;
14
15                          int value = A[right];
16
17                          int previous = indices
18                          if (previous != -1) {
19                                  // add all pai
20                                  // reduce all
21                                  count += count
22                                  if (count >= M
23                                          return
24                                  }
25
26                                  // remove item
27                                  for (; left <=
28                                          indice
29                                  }
30                          }
31
32                          indices[value] = right
33                  }
34
35                  count += countPair(N - left);
36
37                  return count >= MAX ? MAX : (
38          }
39
40          private static long countPair(long n)
41                  return (n + 1) * n / 2;
42          }
43  }
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

collapse all	Example tests	
▼ example example test		✔ OK
1. 0.004 s OK		

collapse all	Correctness tests	
▼ single single element		✔ OK
1. 0.004 s OK		
2. 0.004 s OK		
3. 0.008 s OK		
▼ double double elements		✔ OK

1. 0.008 s **OK**
2. 0.008 s **OK**
3. 0.008 s **OK**

▼ simple1 ✔ OK
first simple test

1. 0.008 s **OK**

▼ simple2 ✔ OK
second simple test

1. 0.004 s **OK**

▼ small_random ✔ OK
small random test, length = 100

1. 0.008 s **OK**

collapse all

Performance tests

▼ medium_random ✔ OK
medium random test, length = 500

1. 0.008 s **OK**

▼ large ✔ OK
large tests, length = ~100,000

1. 0.192 s **OK**
2. 0.184 s **OK**

▼ large_range ✔ OK
large range tests, length = ~100,000

1. 0.292 s **OK**
2. 0.116 s **OK**

▼ large_random ✔ OK
large random tests, length = ~100,000

1. 0.288 s **OK**
2. 0.240 s **OK**

▼ extreme_the_same ✔ OK
all the same elements, length = ~100,000

1. 0.188 s **OK**
2. 0.332 s **OK**