Codility

Candidate Report: trainingRNXXND-NRG

Test Name:

Check out Codility training tasks

Summary | Timeline

Tasks summary

Task	Time spent	Score
CountSemiprimes Python	3 min	100%

Total score

Tasks Details

Medium

1. CountSemiprimes

Count the semiprime numbers in the given range [a..b]

Task Score	Correctness	Performance
100%	100%	100%

Task description

A *prime* is a positive integer X that has exactly two distinct divisors: 1 and X. The first few prime integers are 2, 3, 5, 7, 11 and 13.

A *semiprime* is a natural number that is the product of two (not necessarily distinct) prime numbers. The first few semiprimes are 4, 6, 9, 10, 14, 15, 21, 22, 25, 26.

You are given two non-empty arrays P and Q, each consisting of M integers. These arrays represent queries about the number of semiprimes within specified ranges.

Query K requires you to find the number of semiprimes within the range (P[K], Q[K]), where 1 ≤ P[K] ≤ Q[K] ≤ N.

For example, consider an integer N = 26 and arrays P, Q such that:

```
P[0] = 1    Q[0] = 26
P[1] = 4    Q[1] = 10
P[2] = 16   Q[2] = 20
```

The number of semiprimes within each of these ranges is as follows:

- (1, 26) is 10,
- (4, 10) is 4,
- (16, 20) is 0.

Write a function:

```
def solution(N, P, Q)
```

that, given an integer N and two non-empty arrays P and Q consisting of M integers, returns an array consisting of M elements specifying the consecutive answers to all the queries.

For example, given an integer N = 26 and arrays P, Q such that:

```
P[0] = 1    Q[0] = 26
P[1] = 4    Q[1] = 10
P[2] = 16   Q[2] = 20
```

the function should return the values [10, 4, 0], as explained above.

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

- N is an integer within the range [1..50,000];
- M is an integer within the range [1..30,000];
- each element of arrays P, Q is an integer within the range [1..N];
- P[i] ≤ Q[i].

Solution

Programming language used:	Python
Total time used:	3 minutes
Effective time used:	3 minutes
Notes:	*not defined yet*

Task timeline

23:36:54 23:39:31

Code: 23:39:31 UTC, py, final, score: 100

show code in pop-up

```
1  import math
2  # Utility function to check whether
3  # number is semiprime or not
4  def checkSemiprime(num):
5      cnt = 0
6      for i in range(2, int(math.sqrt(num)) + 1):
7          while num % i == 0:
8              num /= i
9              cnt += 1 # Increment count
10                      # of prime number
11         # If count is greater than 2,
12         # break loop
13         if cnt >= 2:
14             break
15     # If number is greater than 1, add it to
16     # the count variable as it indicates the
17     # number remain is prime number
18     if(num > 1):
19         cnt += 1
20
21     # Return '1' if count is equal to '2' else
22     # return '0'
23     return cnt == 2
24
25 def solution(N, P, Q):
26     # write your code in Python 3.6
27     if N == 1:
28         return [0]
29
30     cumulative = []
31     cumulative.append(0)
32     for i in range(N+1):
33         if checkSemiprime(i):
34             cumulative.append(cumulative[-1]+1)
35         else:
36             cumulative.append(cumulative[-1])
37
38     result = []
39     for i in range(len(Q)):
40         result.append(cumulative[Q[i]+1]-cumulative[P[
41
42     return result
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: O(N * log(log(N)) + M)

expand all	Example tests	
▶	example example test	✔ OK
expand all	**Correctness tests**	
▶	extreme_one small N = 1	✔ OK
▶	extreme_four small N = 4	✔ OK
▶	small_functional small functional	✔ OK
▶	small_random small random, length = ~40	✔ OK
expand all	**Performance tests**	
▶	medium_random small random, length = ~300	✔ OK
▶	large_small_slices large with very small slices, length = ~30,000	✔ OK
▶	large_random1 large random, length = ~30,000	✔ OK
▶	large_random2 large random, length = ~30,000	✔ OK
▶	extreme_large all max ranges	✔ OK