Wellerson comentários - Solução:

A complexidade do algoritmo otimizado da função **is_prime** é O(sqrt(n)/3), onde **n** é o número que está sendo verificado.

Isso ocorre porque o algoritmo verifica a divisibilidade do número **n** por todos os números primos ímpares até a raiz quadrada de n, incrementando de 6 em 6 (ou seja, verificando apenas os primos da forma **6k ± 1**, onde k é um número inteiro). Isso reduz pela metade o número de divisões necessárias em comparação com uma verificação simples.

Em termos de notação Big O, pode ser aproximado para O(sqrt(n)), pois a contribuição adicional do fator 1/3 se torna insignificante em comparação com a raiz quadrada.

—

A função **get_circular_primes** itera sobre cada número no intervalo fornecido. Para cada número, ele verifica se é um primo circular, ou seja, verifica se todas as rotações dos seus dígitos também são números primos.

Para fazer isso, o código converte o número em uma string e, em seguida, itera sobre cada dígito. Em cada iteração, ele verifica se o número atual é primo usando a função **is_prime**. Se o número não for primo, o valor da variável is_circular_prime é definido como False e o loop interno é interrompido.

Caso o número seja um primo circular, ou seja, o loop interno é concluído sem definir is_circular_prime como False, o número é adicionado à lista circular_primes.

—

Lista dos primos circulares do intervalo de 1 até 10^6:
[2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, 97, 113, 131, 197, 199, 311, 337, 373, 719, 733, 919, 971, 991, 1193, 1931, 3119, 3779, 7793, 7937, 9311, 9377, 11939, 19391, 19937, 37199, 39119, 71993, 91193, 93719, 93911, 99371, 193939, 199933, 319993, 331999, 391939, 393919, 919393, 933199, 939193, 939391, 993319, 999331].

*Uma solução "**safada**" é gerar a lista dos primos circulares antes e fixar essa lista no código, dessa forma basta percorrer a lista. Com isso, o código será muito mais rápido.*