**PART A:**

#include <stdbool.h>

bool isPrime(int number) {

if (number <= 1) return false;

if (number <= 3) return true;

if (number % 2 == 0 || number % 3 == 0) return false;

for (int i = 5; i \* i <= number; i += 6) {

if (number % i == 0 || number % (i + 2) == 0) return false;

}

return true;

}

**PART B:**

#include <stdio.h>

int main() {

for (int i = 1; i <= 10000; i++) {

if (isPrime(i)) {

printf("%d ", i);

}

}

return 0;

}

**PART B Question:** How many of these 10,000 numbers do you really have to test before being sure that you have found all the primes?

**PART B Question Answer:** In the most optimized approach without skipping known non-prime ranges, you would test approximately 5,000 numbers (every odd number from 1 to 10,000) to find all the primes within this range

**PART C:**

#include <stdio.h>

#include <time.h>

#include <stdbool.h>

// Function to check prime (efficient version using square root method)

bool isPrime(int number) {

if (number <= 1) return false;

if (number <= 3) return true;

if (number % 2 == 0 || number % 3 == 0) return false;

for (int i = 5; i \* i <= number; i += 6) {

if (number % i == 0 || number % (i + 2) == 0) return false;

}

return true;

}

// Function to check prime (less efficient version using n/2 method)

bool isPrime\_n2(int number) {

if (number <= 1) return false;

for (int i = 2; i <= number / 2; i++) {

if (number % i == 0) return false;

}

return true;

}

// Main function to compare the performance of the two methods

int main() {

clock\_t start, end;

double cpu\_time\_used;

// Version 1: n/2 method

start = clock();

for (int i = 1; i <= 10000; i++) {

isPrime\_n2(i);

}

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken (n/2 method): %f seconds\n", cpu\_time\_used);

// Version 2: square root method

start = clock();

for (int i = 1; i <= 10000; i++) {

isPrime(i);

}

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken (square root method): %f seconds\n", cpu\_time\_used);

return 0;

}

**OUTPUT:**

Time taken (n/2 method): 0.005656 seconds

Time taken (square root method): 0.000119 seconds

* The square root method is vastly more efficient than the n/2 method, as shown by the much shorter time it took to complete the same task.
* The performance improvement is substantial. To put it in perspective, the square root method is roughly 47.6 times faster than the n/2 method in this specific instance (0.005656 / 0.000119 ≈ 47.6).