Perfect Numbers in a Range

Difficulty: Beginner → Intermediate
Topic: Loops, Number Theory

Problem Description

A **Perfect Number** is a positive integer that equals the sum of its **proper divisors** (all positive divisors except the number itself).

Examples:

- **6** → proper divisors: 1, 2, 3 → sum = 1 + 2 + 3 = 6 ✓ **Perfect**
- **28** → proper divisors: 1, 2, 4, 7, 14 → sum = 1 + 2 + 4 + 7 + 14 = 28 ✓ **Perfect**

Task: Find and print all perfect numbers between two given integers `n` and `m` (inclusive).

Input Format

- Two integers `n` and `m` separated by a space
- Constraints: $1 \leq n, m \leq 10^9$

Output Format

- Print all perfect numbers in the range `[min(n, m), max(n, m)]`, separated by spaces
- If no perfect numbers exist in the range, print nothing (empty output)

Sample Test Cases

Sample Input 1

```
1 30
```

Sample Output 1

```
6 28
```

Sample Input 2

```
30 1
```

Sample Output 2

```
6 28
```

Sample Input 3

```
100 200
```

Sample Output 3

(no output - no perfect numbers in this range)

Explanation

- The program works regardless of input order (`n < m` or `n > m`) by automatically checking in ascending order
- Range 1-30: Contains perfect numbers 6 and 28
- Range 100-200: Contains no perfect numbers

Important Notes

Known Perfect Numbers

The first few perfect numbers are extremely rare:

```
6, 28, 496, 8128, 33550336, 8589869056, 137438691328, ...
```

Performance Considerations

- Perfect numbers are very rare - only 51 are known to exist
- For large ranges, direct divisor-sum checking may be slow
- Consider optimizations for better performance with large inputs
- The 5th perfect number (33550336) is already quite large, so most practical ranges will contain at most the first few perfect numbers

Sample Solution

```c
```

```c
#include <stdio.h>

int main() {
    int n, m;
    scanf("%d %d", &n, &m);

    // Ensure n <= m for easier processing
    if (n > m) {
        int temp = n;
        n = m;
        m = temp;
    }

    // Check each number in the range
    for (int i = n; i <= m; i++) {
        if (i < 6) continue; // skip numbers too small to be perfect

        int sum = 1; // 1 is always a proper divisor

        // Find all divisors up to sqrt(i)
        for (int j = 2; j * j <= i; j++) {
            if (i % j == 0) {
                sum += j;                // add the divisor
                if (j != i / j) sum += i / j; // add the complement divisor
            }
        }

        // Check if sum of proper divisors equals the number
        if (sum == i) {
            printf("%d ", i);
        }
    }

    return 0;
}
```

Algorithm Explanation

1. **Input handling**: Read n and m, swap if necessary to ensure $n \leq m$
2. **Range optimization**: Skip numbers less than 6 (no perfect numbers below 6)
3. **Divisor finding**: For each number i, find all proper divisors efficiently:
   - Start with sum = 1 (since 1 is always a proper divisor)

- Check divisors from 2 to √i
- For each divisor j, also add its complement i/j (if different)

4. **Perfect number check**: If sum of proper divisors equals i, print it

Time Complexity

- **O((m-n) × √max_value)** where max_value is the largest number in range
- The √i optimization reduces divisor checking from O(i) to O(√i)