Perfect Number Checker

📄 Problem Set — Perfect Numbers

Difficulty: Beginner → Intermediate **Topic:** Loops, Number Theory

Problem 1 — Perfect Number Checker

A **Perfect Number** is a positive integer equal to the sum of its **proper divisors** (excluding itself).

For example:

- 6 → divisors: 1, 2, 3 → sum = 6 → **Perfect**
- 28 → divisors: 1, 2, 4, 7, 14 → sum = 28 → **Perfect**
- 12 → divisors: 1, 2, 3, 4, 6 → sum = 16 ≠ 12 → **Not Perfect**

Your task is to:

- Read an integer `n`.
- Print:
  - `1` if `n` is a perfect number.
  - `0` if it is not.

Input Format

- One integer `n` ($1 \le n \le 10^9$).

Output Format

- Print `1` if `n` is perfect, otherwise `0`.

Sample Input 1

```
6
```

Sample Output 1

```
1
```

Sample Input 2

```
12
```

Sample Output 2

```
0
```

Explanation

- For `n = 6`: divisors are {1, 2, 3}, sum = 6 → perfect → output 1.
- For `n = 12`: divisors are {1, 2, 3, 4, 6}, sum = 16 → not perfect → output 0.

Constraints & Notes

- Perfect numbers are rare. The first few are:

```
6, 28, 496, 8128, 33550336
```

Problem 2 — List All Perfect Numbers Less Than n

Now let's take it further: instead of checking one number, print **all perfect numbers strictly less than a given** `n`.

Input Format

- One integer `n` ($1 \le n \le 10^9$).

Output Format

- Print all perfect numbers less than `n` in ascending order, separated by spaces.
- If no perfect number exists below `n`, print nothing.

Sample Input 1

```
30
```

Sample Output 1

```
6 28
```

Sample Input 2

```
6
```

Sample Output 2

(no output)

Explanation

- For `n = 30`, perfect numbers below it are 6 and 28.
- For `n = 6`, there are no perfect numbers strictly less than it.

Constraints & Notes

- You may use a precomputed list of known perfect numbers (e.g., `6, 28, 496, 8128, ...`) or write a program to check each number.
- Efficiency matters if you use checking for larger `n`.

SOLUTIONS

Complete Solution Code

python

```python
def is_perfect_number(n):
    """
    Check if a number is a perfect number.
    A perfect number equals the sum of its proper divisors (excluding itself).
    """
    if n <= 1:
        return False

    divisor_sum = 1  # 1 is always a proper divisor for n > 1

    # Only check up to sqrt(n) for efficiency
    i = 2
    while i * i <= n:
        if n % i == 0:
            divisor_sum += i
            # Add the corresponding divisor (n/i) if it's different from i
            if i * i != n:
                divisor_sum += n // i
        i += 1

    return divisor_sum == n


def solve_problem_1():
    """
    Problem 1: Perfect Number Checker
    Read an integer n and print 1 if perfect, 0 otherwise.
    """
    n = int(input().strip())
    result = 1 if is_perfect_number(n) else 0
    print(result)


def solve_problem_2():
    """
    Problem 2: List All Perfect Numbers Less Than n
    Print all perfect numbers strictly less than n.
    """
    n = int(input().strip())

    # Known perfect numbers up to reasonable limits
    # This is more efficient than checking each number for large n
    known_perfect = [6, 28, 496, 8128, 33550336, 8589869056, 137438691328]
```

```
    perfect_below_n = [num for num in known_perfect if num < n]

    if perfect_below_n:
        print(' '.join(map(str, perfect_below_n)))
    # If no perfect numbers below n, print nothing (empty line)


def solve_problem_2_brute_force():
    """
    Problem 2: Alternative solution using brute force checking
    This approach checks each number but is slower for large n.
    """
    n = int(input().strip())

    perfect_numbers = []

    # For very large n, we can optimize by only checking up to a reasonable limit
    # since perfect numbers are extremely rare
    check_limit = min(n, 10000000)  # Adjust based on time constraints

    for i in range(2, check_limit):
        if is_perfect_number(i):
            perfect_numbers.append(i)

    if perfect_numbers:
        print(' '.join(map(str, perfect_numbers)))


# Example usage and testing
if __name__ == "__main__":
    print("Testing Perfect Number functions:")

    # Test cases for Problem 1
    test_cases_1 = [6, 12, 28, 496, 100]
    print("\nProblem 1 test cases:")
    for num in test_cases_1:
        result = 1 if is_perfect_number(num) else 0
        print(f"n = {num}: {result}")

    # Test cases for Problem 2
    test_cases_2 = [30, 6, 500, 10000]
    print("\nProblem 2 test cases:")
    for num in test_cases_2:
```

```
        known_perfect = [6, 28, 496, 8128, 33550336, 8589869056, 137438691328]
        perfect_below = [n for n in known_perfect if n < num]
        if perfect_below:
            print(f"n = {num}: {' '.join(map(str, perfect_below))}")
        else:
            print(f"n = {num}: (no output)")


# Uncomment the line below to run the solution for Problem 1
# solve_problem_1()

# Uncomment the line below to run the solution for Problem 2
# solve_problem_2()
```

COMPETITIVE PROGRAMMING TEMPLATE - COPY THESE FOR SUBMISSION

Problem 1 Solution - Perfect Number Checker

Copy this code for Problem 1 submission:

python

```
# Problem 1 - Compact version for submission
def is_perfect(n):
    if n <= 1:
        return False
    divisor_sum = 1
    i = 2
    while i * i <= n:
        if n % i == 0:
            divisor_sum += i
            if i * i != n:
                divisor_sum += n // i
        i += 1
    return divisor_sum == n

# Main code for Problem 1
n = int(input())
print(1 if is_perfect(n) else 0)
```

Problem 2 Solution - List All Perfect Numbers Less Than n

Copy this code for Problem 2 submission:

```python
# Problem 2 - Optimized version using known perfect numbers
n = int(input())
known_perfect = [6, 28, 496, 8128, 33550336, 8589869056, 137438691328]
perfect_below_n = [num for num in known_perfect if num < n]
if perfect_below_n:
    print(' '.join(map(str, perfect_below_n)))
```

Alternative Problem 2 Solution - Brute Force

Copy this code if you need to check each number:

```python
# Problem 2 - Brute force version
def is_perfect_bf(n):
    if n <= 1:
        return False
    divisor_sum = 1
    i = 2
    while i * i <= n:
        if n % i == 0:
            divisor_sum += i
            if i * i != n:
                divisor_sum += n // i
        i += 1
    return divisor_sum == n

n = int(input())
perfect_numbers = []
check_limit = min(n, 10000000)  # Reasonable limit for time constraints
for i in range(2, check_limit):
    if is_perfect_bf(i):
        perfect_numbers.append(i)
if perfect_numbers:
    print(' '.join(map(str, perfect_numbers)))
```

Algorithm Notes

Time Complexity

- **is_perfect_number()**: O(√n) - only checks divisors up to square root
- **Problem 1**: O(√n) per test case
- **Problem 2 (optimized)**: O(1) - uses precomputed list
- **Problem 2 (brute force)**: O(n√n) - checks each number up to n

Space Complexity

- **Problem 1**: O(1)
- **Problem 2**: O(k) where k is the number of perfect numbers found

Key Optimizations

1. **Square root optimization**: Only check divisors up to √n
2. **Precomputed list**: Use known perfect numbers for faster lookup
3. **Early termination**: Stop checking when divisor sum exceeds n

The optimized versions will easily handle the constraint $n \leq 10^9$ within typical competitive programming time limits.