CMSC 12 Sample Exercises

Jan Neal Isaac D. Villamin

October 05, 2022

Instructions

Solve the problems strictly using input-output, variables, operators, if-else, and loops.

Problems and Solutions

1) Input an integer n, and print all divisors of n.

Solution 1: The simplest solution is to iterate from 1 to n and check if the current number is a divisor of n.

```
n = int(input())
for i in range(1, n + 1):
    if n % i == 0:
        print(i, end=' ')
```

Solution 2: We can solve this in $O(\sqrt{n})$. Iterate from 1 to $\lfloor\sqrt{n}\rfloor$ twice. For the first iteration, go from 1 to $\lfloor\sqrt{n}\rfloor$, print the current number i if i divides n. For the second iteration, go from $\lfloor\sqrt{n}\rfloor$ to 1, print $\frac{n}{i}$ if i divides n.

```
n = int(input())
i = 1
while i * i <= n:
    if n % i == 0:
        print(i, end=' ')
    i += 1
while i * i >= 1:
    if n % i == 0:
        print(n // i, end=' ')
    i -= 1
```

2) Input an integer n, followed by a list of n numbers, and count the number of zeroes in the list.

Solution: The best solution can be done in $O(n)$ because we have to consider all of the integers in the list. Check if the current number is 0, if yes, increment a counter variable.

```
n = int(input())
count = 0
for i in range(n):
    num = int(input())
    if num == 0:
        count += 1
print(count)
```

3) Input two positive integers A and B, and find the greatest common factor of A and B, i.e., the biggest integer that divides both A and B exactly.

Solution 1: The brute force solution is to iterate from 1 to $\min(A, B)$ and keep updating the *gcd* if the current number is a divisor of both A and B.

```
A = int(input())
B = int(input())
minimum = 0
if A > B:
    minimum = B
else:
    minimum = A
gcd = 0
for i in range(1, minimum + 1):
    if A % i == 0 and B % i == 0:
        gcd = i
print(gcd)
```

Solution 2: We can solve this in $O(\log(\min(A, B))$ using the Euclidean Algorithm. It uses the fact that $\gcd(A, B) = \gcd(A \bmod B, B)$, where $A \geq B$.

```
A = int(input())
B = int(input())
temp1 = A
temp2 = B
if temp1 < temp2:
    temp1, temp2 = temp2, temp1
remainder = temp1 % temp2
while remainder > 0:
    temp1 = temp2
    temp2 = remainder
    remainder = temp1 % temp2
print(temp2)
```

4) Input two positive integers A and B, and find the least common multiple of A and B, i.e., the smallest integer that is both a multiple of A and B.

Solution 1: The brute force solution is to iterate from $\max(A, B)$ to $A \cdot B$. Once the number is divisible by both A and B, then break the iteration.

```
A = int(input())
B = int(input())
maximum = 0
if A > B:
    maximum = A
else:
    maximum = B
lcm = 0
for i in range(maximum, A * B + 1):
    if i % A == 0 and i % B == 0:
        lcm = i
        break
print(lcm)
```

Solution 2: This can also be solved using the Euclidean Algorithm used in problem 3. We can solve for $\gcd(A, B)$, then use the fact that $\text{lcm}(A, B) = \frac{A \cdot B}{\gcd(A,B)}$.

```
A = int(input())
B = int(input())
temp1 = A
temp2 = B
if temp1 < temp2:
    temp1, temp2 = temp2, temp1
remainder = temp1 % temp2
while remainder > 0:
    temp1 = temp2
    temp2 = remainder
    remainder = temp1 % temp2
print(A * B // temp2)
```

5) Input three positive integers A, B, and C, and find the least common multiple of A, B, and C.

Solution 1: A similar algorithm from the previous problem can be used. Iterate from $\max(A, B, C)$ to $A \cdot B \cdot C$. If the current number is divisible by A, B, and C, that is the LCM, and break.

```
A = int(input())
B = int(input())
C = int(input())
maximum = 0
if A > B and A > C:
    maximum = A
elif B > C:
    maximum = B
else:
    maximum = C
lcm = 0
for i in range(maximum, A * B * C + 1):
    if i % A == 0 and i % B == 0 and i % C == 0:
        lcm = i
        break
print(lcm)
```

Solution 2: We use the same faster algorithm from the previous problem. This time, we use the fact that $\text{lcm}(A, B, C) = \text{lcm}(\text{lcm}(A, B), C)$.

```
A = int(input())
B = int(input())
C = int(input())
temp1 = A
temp2 = B
if temp1 < temp2:
    temp1, temp2 = temp2, temp1
remainder = temp1 % temp2
while remainder > 0:
    temp1 = temp2
    temp2 = remainder
    remainder = temp1 % temp2
lcmAB = A * B // temp2
temp1 = lcmAB
```

```
temp2 = C
if temp1 < temp2:
    temp1, temp2 = temp2, temp1
remainder = temp1 % temp2
while remainder > 0:
    temp1 = temp2
    temp2 = remainder
    remainder = temp1 % temp2
print(lcmAB * C // temp2)
```

6) Input a positive integer n, and determine whether n is prime (no divisors except 1 and itself) or composite (has divisors other than 1 and itself).

Solution 1: The brute force solution is to iterate from 2 to $n-1$. If the current number is divides n, then n is not a prime. If there exists no number that divides n, then n is prime.

```
n = int(input())
is_prime = True
for i in range(2, n):
    if n % i == 0:
        is_prime = False
        break
if is_prime:
    print(n, "is prime.")
else:
    print(n, "is composite.")
```

Solution 2: It turns out that we only need to check divisors from 2 to $\lfloor\sqrt{n}\rfloor$. Additionally, we can omit all even number except 2 by checking first if 2 divides n, where $n \neq 2$. This means that instead of incrementing by 1, we can start from 3 and increment by 2.

```
n = int(input())
is_prime = True
if n != 2 and n % 2 == 0:
    is_prime = False
i = 3
while i * i <= n:
    if n % i == 0:
        is_prime = False
        break
    i += 2
if is_prime:
    print(n, "is prime.")
else:
    print(n, "is composite.")
```

7) Input an integer n and print the number of primes between n and $2n$.

Solution 1: We use the same slower algorithm to check if a number is prime. The difference is that we iterate from n to $2n$ and check the current number. If the current number is a prime, then increment a counter variable.

```
n = int(input())
count = 0
for i in range(n, 2 * n + 1):
```

```
    is_prime = True
    for j in range(2, i):
        if i % j == 0:
            is_prime = False
            break
    if is_prime:
        count += 1
print(count)
```

Solution 2: We use the same faster algorithm to check if a number is prime. The difference is the same as before.

```
n = int(input())
count = 0
for i in range(n, 2 * n + 1):
    is_prime = True
    if i != 2 and i % 2 == 0:
        is_prime = False
    j = 3
    while j * j <= n:
        if i % j == 0:
            is_prime = False
            break
        j += 2
    if is_prime:
        count += 1
print(count)
```

8) Twin primes are prime numbers that differ by two, (e.g. 3 and 5 is the smallest pair of twin primes). Find and print the first thirty (30) pairs of twin primes.

Solution 1: Notice that both primes in a twin prime are odd numbers. We can therefore start from 3 and obtain the current number i and $i + 2$. If both numbers are primes, then increment a counter variable, and print the numbers. If the counter variable becomes 30, we stop the iteration. We can use the slower algorithm to check if a number is a prime number.

```
count = 0
i = 3
while count < 30:
    are_primes = True
    for j in range(2, i):
        if i % j == 0:
            are_primes = False
            break
    for j in range(2, i + 2):
        if (i + 2) % j == 0:
            are_primes = False
            break
    if are_primes:
        print(i, i + 2)
        count += 1
    i += 2
```

Solution 2: We use the same observation as the slower algorithm. But this time, we use the faster method of checking if a number is prime.

```python
count = 0
i = 3
while count < 30:
    are_primes = True
    j = 3
    while j * j <= i:
        if i % j == 0:
            are_primes = False
            break
        j += 2
    j = 3
    while j * j <= i + 2:
        if (i + 2) % j == 0:
            are_primes = False
            break
        j += 2
    if are_primes:
        print(i, i + 2)
        count += 1
    i += 2
```

9) Input a positive integer n and neatly print an n x n multiplication table.

Solution: Iterate through an n by n grid using nested loops. Then, multiply the coordinates and print the product separated by tabs.

```python
n = int(input())
for i in range(1, n + 1):
    for j in range(1, n + 1):
        print(i * j, end='\t')
    print()
```

10) Input a positive integer n and draw an isosceles triangle-shaped figure of chars, of height n.

Solution: We can semantically solve this problem. Notice that an isosceles triangle expands 1 character on both sides from the middle for each row. Iterate through an n by $2(n-1)+1 = 2n-1$ grid. If the current coordinate is covered by the expansion, then print a character. Otherwise, print space.

```python
n = int(input())
for i in range(n):
    for j in range(2 * n - 1):
        if n - i - 1 <= j and j < n + i:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

11) Input positive integers L and W and draw a box-shaped figure L chars long and W chars thick.

Solution: Iterate through an L by W grid. If the current coordinate touches the edge, then print a character. Otherwise, print space.

```python
L = int(input())
W = int(input())
```

```
for i in range(L):
    for j in range(W):
        if i == 0 or i == L - 1 or j == 0 or j == W - 1:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

12) Input a positive integer n and draw a Christmas tree-shaped figure of chars, consisting of n stacked "centered" trapezoids (see diagram below).

Solution: We can again semantically solve this problem. We can think of iterating for each trapezoid of the n trapezoids. Notice that the ith trapezoid has height $i+2$, upper base of $2i-1$, and lower base of $4i+1$. The longest base is $4n+1$ characters long. Hence, we can iterate for each trapezoid, then for each line of the trapezoid, then for each character in that line which has $4n+1$ characters. We can use the same expansion semantics in the previous problem, i.e., if the expansion from the middle covers the current distance, then print a character. Otherwise, print space.

For the expansion, we only need to know the upper base, and expand from the middle on both sides, for each line of the trapezoid. For the kth trapozoid, it has an upper base of $2k-1$ expanding for each $k+2$ lines.

```
n = int(input())
for k in range(n):
    for i in range(k + 3):
        for j in range(4 * n + 1):
            if 2 * n - (k + i) <= j and j <= 2 * n + (k + i):
                print('#', end='')
            else:
                print(' ', end='')
        print()
print(" Merry Christmas!")
```