

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 .

```
1 n = int(input())
2 for i in range(1, n + 1):
3     if n % i == 0:
4         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 .

```
1 n = int(input())
2 i = 1
3 while i * i <= n:
4     if n % i == 0:
5         print(i, end=' ')
6     i += 1
7 while i * i >= 1:
8     if n % i == 0:
9         print(n // i, end=' ')
10    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.

```
1 n = int(input())
2 count = 0
3 for i in range(n):
4     num = int(input())
5     if num == 0:
6         count += 1
7 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 .

```
1 A = int(input())
2 B = int(input())
3 minimum = 0
4 if A > B:
5     minimum = B
6 else:
7     minimum = A
8 gcd = 0
9 for i in range(1, minimum + 1):
10     if A % i == 0 and B % i == 0:
11         gcd = i
12 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 $A \geq B$.

```
1 A = int(input())
2 B = int(input())
3 temp1 = A
4 temp2 = B
5 if temp1 < temp2:
6     temp1, temp2 = temp2, temp1
7 remainder = temp1 % temp2
8 while remainder > 0:
9     temp1 = temp2
10    temp2 = remainder
11    remainder = temp1 % temp2
12 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.

```
1 A = int(input())
2 B = int(input())
3 maximum = 0
4 if A > B:
5     maximum = A
6 else:
7     maximum = B
8 lcm = 0
9 for i in range(maximum, A * B + 1):
10     if i % A == 0 and i % B == 0:
11         lcm = i
12         break
13 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 $\text{lcm}(A, B) = \frac{A \cdot B}{\gcd(A, B)}$.

```

1 A = int(input())
2 B = int(input())
3 temp1 = A
4 temp2 = B
5 if temp1 < temp2:
6     temp1, temp2 = temp2, temp1
7 remainder = temp1 % temp2
8 while remainder > 0:
9     temp1 = temp2
10    temp2 = remainder
11    remainder = temp1 % temp2
12 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.

```

1 A = int(input())
2 B = int(input())
3 C = int(input())
4 maximum = 0
5 if A > B and A > C:
6     maximum = A
7 elif B > C:
8     maximum = B
9 else:
10    maximum = C
11 lcm = 0
12 for i in range(maximum, A * B * C + 1):
13     if i % A == 0 and i % B == 0 and i % C == 0:
14         lcm = i
15         break
16 print(lcm)

```

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

```

1 A = int(input())
2 B = int(input())
3 C = int(input())
4 temp1 = A
5 temp2 = B
6 if temp1 < temp2:
7     temp1, temp2 = temp2, temp1
8 remainder = temp1 % temp2
9 while remainder > 0:
10    temp1 = temp2
11    temp2 = remainder
12    remainder = temp1 % temp2
13 lcmAB = A * B // temp2
14 temp1 = lcmAB

```

```

15 temp2 = C
16 if temp1 < temp2:
17     temp1, temp2 = temp2, temp1
18 remainder = temp1 % temp2
19 while remainder > 0:
20     temp1 = temp2
21     temp2 = remainder
22     remainder = temp1 % temp2
23 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 divides n , then n is not a prime. If there exists no number that divides n , then n is prime.

```

1 n = int(input())
2 is_prime = True
3 for i in range(2, n):
4     if n % i == 0:
5         is_prime = False
6         break
7 if is_prime:
8     print(n, "is prime.")
9 else:
10    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.

```

1 n = int(input())
2 is_prime = True
3 if n != 2 and n % 2 == 0:
4     is_prime = False
5 i = 3
6 while i * i <= n:
7     if n % i == 0:
8         is_prime = False
9         break
10    i += 2
11 if is_prime:
12     print(n, "is prime.")
13 else:
14     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.

```

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

```

```

4     is_prime = True
5     for j in range(2, i):
6         if i % j == 0:
7             is_prime = False
8             break
9     if is_prime:
10        count += 1
11 print(count)

```

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

```

1 n = int(input())
2 count = 0
3 for i in range(n, 2 * n + 1):
4     is_prime = True
5     if i != 2 and i % 2 == 0:
6         is_prime = False
7     j = 3
8     while j * j <= i:
9         if i % j == 0:
10            is_prime = False
11            break
12        j += 2
13    if is_prime:
14        count += 1
15 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.

```

1 count = 0
2 i = 3
3 while count < 30:
4     are_primes = True
5     for j in range(2, i):
6         if i % j == 0:
7             are_primes = False
8             break
9     for j in range(2, i + 2):
10        if (i + 2) % j == 0:
11            are_primes = False
12            break
13    if are_primes:
14        print(i, i + 2)
15        count += 1
16    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.

```

1 count = 0
2 i = 3
3 while count < 30:
4     are_primes = True
5     j = 3
6     while j * j <= i:
7         if i % j == 0:
8             are_primes = False
9             break
10        j += 2
11    j = 3
12    while j * j <= i + 2:
13        if (i + 2) % j == 0:
14            are_primes = False
15            break
16        j += 2
17    if are_primes:
18        print(i, i + 2)
19        count += 1
20    i += 2

```

9) Input a positive integer n and neatly print an $n \times n$ multiplication table.

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

```

1 n = int(input())
2 for i in range(1, n + 1):
3     for j in range(1, n + 1):
4         print(i * j, end='\t ')
5     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.

```

1 n = int(input())
2 for i in range(n):
3     for j in range(2 * n - 1):
4         if n - 1 - i <= j and j <= n - 1 + i:
5             print('#', end='')
6         else:
7             print('_', end='')
8     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.

```

1 L = int(input())
2 W = int(input())

```

```

3  for i in range(L):
4      for j in range(W):
5          if i == 0 or i == L - 1 or j == 0 or j == W - 1:
6              print( '#', end='')
7          else:
8              print( ' ', end='')
9      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 i th 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 k th trapezoid, it has an upper base of $2k - 1$ expanding for each $k + 2$ lines.

```

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

```