Practice questions

HASKELL LAB EVAL-1

**1. Generate Prime Numbers Sequence**

Write a program to generate a sequence of prime numbers less than or equal to a given number n.

isPrime :: Int -> Bool

isPrime n

| n < 2 = False

| otherwise = null [x | x <- [2..n-1], n `mod` x == 0] -- No divisors other than 1 and n

primeSequence :: Int -> [Int]

primeSequence n

| n < 2 = [] -- No primes below 2

| otherwise = filter isPrime [2..n] -- Generate all primes up to n

**2. Generate an Alternating Sequence**

Write a program that generates a sequence starting from a number n:

* If the current term is divisible by 3, subtract 1.
* Otherwise, add 2. The sequence stops when the term exceeds a limit (e.g., 100).

alternatingSequence :: Int -> [Int]

alternatingSequence n

| n > 100 = [] -- Base case: Stop when the term exceeds 100

| n `mod` 3 == 0 = n : alternatingSequence (n - 1) -- If divisible by 3, subtract 1

| otherwise = n : alternatingSequence (n + 2) -- Otherwise, add 2

**3. Generate Fibonacci Sequence Until a Limit**

fibonacciSequence :: Int -> [Int]

fibonacciSequence limit = takeWhile (<= limit) (fib 0 1)

where

fib a b = a : fib b (a + b) -- Generate Fibonacci sequence

4.**Generate an Exponentiation Sequence with Custom Rules**

Extend the original question:

* If the current term is divisible by 5, double the term.
* If the current term is even but not divisible by 5, square it.
* If the current term is odd, cube it. The sequence stops when the term exceeds a limit (e.g., 2000).

customExponentiationSequence :: Int -> [Int]

customExponentiationSequence n

| n > 2000 = [] -- Stop if the term exceeds 2000

| n `mod` 5 == 0 = n : customExponentiationSequence (2 \* n) -- Double if divisible by 5

| even n = n : customExponentiationSequence (n \* n) -- Square if even

| otherwise = n : customExponentiationSequence (n \* n \* n) -- Cube if odd

**5. Generate a Collatz-Like Sequence**

 If the current term is divisible by 4, divide it by 4.

 If the current term is even but not divisible by 4, add 3.

 If the current term is odd, multiply it by 2 and subtract 1. The sequence stops when the term reaches 1.

collatzLikeSequence :: Int -> [Int]

collatzLikeSequence 1 = [1] -- Stop when the term reaches 1

collatzLikeSequence n

| n `mod` 4 == 0 = n : collatzLikeSequence (n `div` 4) -- If divisible by 4, divide by 4

| even n = n : collatzLikeSequence (n + 3) -- If even, add 3

| otherwise = n : collatzLikeSequence (2 \* n - 1) -- If odd, 2n – 1

6.**Generate a Sequence of Perfect Numbers**

Generate a sequence of "perfect numbers" (numbers equal to the sum of their proper divisors) below a given limit.

isPerfect :: Int -> Bool

isPerfect n = n == sum [x | x <- [1..n-1], n `mod` x == 0] -- Check if sum of divisors equals n(If n is equal to the sum of its proper divisors, it is a **perfect number**)

perfectNumbers :: Int -> [Int]

perfectNumbers limit = [x | x <- [1..limit], isPerfect x] -- Generate perfect numbers up to limit

**7. Generate a Modified Pascal’s Triangle Row**

pascalRow :: Int -> Int -> [Int]

pascalRow n limit = takeWhile (<= limit) [binomial n k | k <- [0..n]]

where

binomial n k = factorial n `div` (factorial k \* factorial (n - k))

factorial 0 = 1

factorial m = m \* factorial (m - 1)

8.**Alternate Arithmetic Sequence**

Write a program to generate a sequence where:

* Start with a number n.
* If the current term is divisible by 3, divide it by 3.
* If the term is not divisible by 3, alternate between adding and subtracting a value d. The sequence stops when a term becomes negative or exceeds a given limit.

alternateArithmeticSequence :: Int -> Int -> Int -> [Int]

alternateArithmeticSequence n d limit

| n < 0 || n > limit = [] -- Stop when term is out of bounds

| n `mod` 3 == 0 = n : alternateArithmeticSequence (n `div` 3) d limit -- Divide by 3

| otherwise = n : alternateArithmeticSequence (n + (-1)^n \* d) d limit -- Alternate add/subtract

**9. Generate Hailstone Sequence with Exponential Growth**

Extend the Hailstone sequence:

* If the current term is divisible by 2 and 3, cube it.
* If divisible only by 2, square it.
* If divisible only by 3, multiply it by 4.
* Otherwise, subtract 1. The sequence stops when the term exceeds a large limit.

hailstoneExponential :: Int -> [Int]

hailstoneExponential n

| n > 5000 = [] -- Stop when term exceeds 5000

| n `mod` 6 == 0 = n : hailstoneExponential (n ^ 3) -- Divisible by 2 and 3, cube it

| n `mod` 2 == 0 = n : hailstoneExponential (n ^ 2) -- Divisible by 2, square it

| n `mod` 3 == 0 = n : hailstoneExponential (n \* 4) -- Divisible by 3, multiply by 4

| otherwise = n : hailstoneExponential (n - 1) -- Otherwise, subtract 1

**1. Longest Ascending Sublist**

**Problem**: Write a recursive Haskell program longestAscending that takes a list of integers and returns the longest sublist where the elements are in strictly increasing order.

longestAscending :: [Int] -> [Int]

longestAscending [] = []

longestAscending (x:xs) = longestGroup (groupAscending [x] xs)

-- Helper function to build ascending sublists

groupAscending :: [Int] -> [Int] -> [[Int]]

groupAscending current [] = [current]

groupAscending current (y:ys)

| y > last current = groupAscending (current ++ [y]) ys

| otherwise = current : groupAscending [y] ys

-- Function to find the longest sublist from a list of sublists

longestGroup :: [[Int]] -> [Int]

longestGroup [] = []

longestGroup [x] = x

longestGroup (x:y:ys)

| length x >= length y = longestGroup (x:ys)

| otherwise = longestGroup (y:ys)

-- Main program

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

let numbers = map read (words input) :: [Int]

let result = longestAscending numbers

putStrLn $ "The longest ascending sublist is: " ++ show result

**2. Longest Decreasing Sublist**

**Problem**: Write a recursive Haskell program longestDecreasing that takes a list of integers and returns the longest sublist where the elements are in strictly decreasing order.

longdesc::[Int]->[Int]

longdesc []=[]

longdesc (x:xs)=longroup(groupcons [x] xs)

groupcons::[Int]->[Int]->[[Int]]

groupcons curr []=[curr]

groupcons curr (y:ys)

|y<last curr=groupcons(curr++[y]) ys

|otherwise=curr:groupcons [y] ys

longroup ::[[Int]]->[Int]

longroup []=[]

longroup[x]=x

longroup (x:y:ys)

|length x >length y=longroup (x:ys)

|otherwise=longroup (y:ys)

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

let numbers = map read (words input) :: [Int]

let result = longdesc numbers

putStrLn $ "The longest decreasing sublist is: " ++ show result

**3. Longest Alternating Sublist**

**Problem**: Write a recursive Haskell program longestAlternating that takes a list of integers and returns the longest sublist where the elements alternate between increasing and decreasing.

-- Main function

longestAlternating :: [Int] -> [Int]

longestAlternating [] = []

longestAlternating [x] = [x]

longestAlternating (x:y:xs) = longestGroup (groupAlternating [x, y] xs)

-- Group alternating sublists recursively

groupAlternating :: [Int] -> [Int] -> [[Int]]

groupAlternating curr [] = [curr]

groupAlternating curr (z:zs)

| isAlternating (lastButOne curr) (last curr) z = groupAlternating (curr ++ [z]) zs

| otherwise = curr : groupAlternating [last curr, z] zs

-- Check if three numbers form an alternating pattern

isAlternating :: Int -> Int -> Int -> Bool

isAlternating a b c = (a < b && b > c) || (a > b && b < c)

-- Utility to get second last element of a list

lastButOne :: [a] -> a

lastButOne xs = last (init xs)

-- Find the longest sublist among groups

longestGroup :: [[Int]] -> [Int]

longestGroup [] = []

longestGroup [x] = x

longestGroup (x:y:ys)

| length x >= length y = longestGroup (x:ys)

| otherwise = longestGroup (y:ys)

-- Main program to run

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

let numbers = map read (words input) :: [Int]

let result = longestAlternating numbers

putStrLn $ "The longest alternating sublist is: " ++ show result

**4. Longest Sublist with Same Difference**

**Problem**: Write a recursive Haskell program longestArithmetic that takes a list of integers and returns the longest sublist where the difference between consecutive elements is the same.

longestArithmetic([2, 4, 6, 8, 3, 6, 9, 12]) → Output: [3, 6, 9, 12]

longestArithmetic([1, 3, 5, 10, 20, 30]) → Output: [1, 3, 5]

longestArithmetic :: [Int] -> [Int]

longestArithmetic [] = []

longestArithmetic [\_] = []

longestArithmetic xs = findLongest (groupArithmetic xs)

-- Helper: Group elements with the same difference

groupArithmetic :: [Int] -> [[Int]]

groupArithmetic [] = []

groupArithmetic [\_] = []

groupArithmetic (x:y:xs) = buildGroup [x, y] xs (y - x)

where

buildGroup current [] \_ = [current]

buildGroup current (z:zs) diff

| z - last current == diff = buildGroup (current ++ [z]) zs diff

| otherwise = current : groupArithmetic (z:zs)

-- Helper: Find the longest sublist

findLongest :: [[Int]] -> [Int]

findLongest [] = []

findLongest [x] = x

findLongest (x:xs)

| length x > length (findLongest xs) = x

| otherwise = findLongest xs

**5. Longest Sublist of Even Numbers**

**Problem**: Write a recursive Haskell program longestEvenSublist that takes a list of integers and returns the longest sublist where all elements are even.

-- Main function

longestEvenSublist :: [Int] -> [Int]

longestEvenSublist xs = findLongest (groupEven xs)

-- Group consecutive even numbers

groupEven :: [Int] -> [[Int]]

groupEven [] = []

groupEven (x:xs)

| even x = extendGroup [x] xs

| otherwise = groupEven xs

-- Helper to extend an even group

extendGroup :: [Int] -> [Int] -> [[Int]]

extendGroup current [] = [current]

extendGroup current (y:ys)

| even y = extendGroup (current ++ [y]) ys

| otherwise = current : groupEven ys

-- Find the longest sublist

findLongest :: [[Int]] -> [Int]

findLongest [] = []

findLongest [x] = x

findLongest (x:y:ys)

| length x >= length y = findLongest (x:ys)

| otherwise = findLongest (y:ys)

-- Main IO function

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

let numbers = map read (words input) :: [Int]

let result = longestEvenSublist numbers

putStrLn $ "The longest sublist of even numbers is: " ++ show result

**7. Longest Sublist of Prime Numbers**

**Problem**: Write a recursive Haskell program longestPrimeSublist that takes a list of integers and returns the longest sublist where all elements are prime numbers.

-- Check if a number is prime

isPrime :: Int -> Bool

isPrime n

| n < 2 = False

| otherwise = null [x | x <- [2..(floor . sqrt $ fromIntegral n)], n `mod` x == 0]

-- Main function

longestPrimeSublist :: [Int] -> [Int]

longestPrimeSublist xs = findLongest (groupPrimes xs)

-- Group consecutive prime numbers

groupPrimes :: [Int] -> [[Int]]

groupPrimes [] = []

groupPrimes (x:xs)

| isPrime x = extendGroup [x] xs

| otherwise = groupPrimes xs

-- Extend a current prime group

extendGroup :: [Int] -> [Int] -> [[Int]]

extendGroup current [] = [current]

extendGroup current (y:ys)

| isPrime y = extendGroup (current ++ [y]) ys

| otherwise = current : groupPrimes ys

-- Find the longest sublist from list of sublists

findLongest :: [[Int]] -> [Int]

findLongest [] = []

findLongest [x] = x

findLongest (x:y:ys)

| length x >= length y = findLongest (x:ys)

| otherwise = findLongest (y:ys)

-- Main program

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

let numbers = map read (words input) :: [Int]

let result = longestPrimeSublist numbers

putStrLn $ "The longest sublist of prime numbers is: " ++ show result

**8. Longest Palindromic Sublist**

**Problem**: Write a recursive Haskell program longestPalindromeSublist that takes a list of integers and returns the longest sublist that forms a palindrome (i.e., it reads the same forwards and backwards).

longestPalindromeSublist([1, 2, 3, 2, 1, 4, 5, 6]) → Output: [1, 2, 3, 2, 1]

longestPalindromeSublist([10, 20, 30, 20, 10, 15, 25]) → Output: [10, 20, 30, 20, 10]

-- Check if a list is a palindrome

isPalindrome :: [Int] -> Bool

isPalindrome xs = xs == reverse xs

-- Generate all contiguous sublists recursively

allSublists :: [Int] -> [[Int]]

allSublists [] = []

allSublists (x:xs) = subFrom (x:xs) ++ allSublists xs

-- Helper to build sublists from the current head

subFrom :: [Int] -> [[Int]]

subFrom [] = []

subFrom ys = ys : subFrom (init ys)

-- Find the longest palindrome from sublists

longestPalindromeSublist :: [Int] -> [Int]

longestPalindromeSublist xs = findLongest (filter isPalindrome (allSublists xs))

-- Find the longest list from a list of lists

findLongest :: [[Int]] -> [Int]

findLongest [] = []

findLongest [x] = x

findLongest (x:y:ys)

| length x >= length y = findLongest (x:ys)

| otherwise = findLongest (y:ys)

-- Main program

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

let numbers = map read (words input) :: [Int]

let result = longestPalindromeSublist numbers

putStrLn $ "The longest palindromic sublist is: " ++ show result

**9. Longest Sublist with Unique Elements**

**Problem**: Write a recursive Haskell program longestUniqueSublist that takes a list of integers and returns the longest sublist where all elements are distinct.

longestUniqueSublist([1, 2, 3, 1, 4, 5, 6]) → Output: [1, 2, 3]

longestUniqueSublist([10, 20, 30, 10, 40]) → Output: [10, 20, 30, 40]

-- Check if a list has all unique elements

allUnique :: [Int] -> Bool

allUnique [] = True

allUnique (x:xs) = x `notElem` xs && allUnique xs

-- Generate all contiguous sublists

allSublists :: [Int] -> [[Int]]

allSublists [] = []

allSublists (x:xs) = subFrom (x:xs) ++ allSublists xs

-- Helper to generate sublists from current position

subFrom :: [Int] -> [[Int]]

subFrom [] = []

subFrom ys = ys : subFrom (init ys)

-- Find the longest sublist with unique elements

longestUniqueSublist :: [Int] -> [Int]

longestUniqueSublist xs = findLongest (filter allUnique (allSublists xs))

-- Get the longest list from list of lists

findLongest :: [[Int]] -> [Int]

findLongest [] = []

findLongest [x] = x

findLongest (x:y:ys)

| length x >= length y = findLongest (x:ys)

| otherwise = findLongest (y:ys)

-- Main function

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

let numbers = map read (words input) :: [Int]

let result = longestUniqueSublist numbers

putStrLn $ "The longest unique sublist is: " ++ show result

**10. Longest Sublist Divisible by a Number**

**Problem**: Write a recursive Haskell program longestDivisibleSublist that takes a list of integers and a divisor d, and returns the longest sublist where all elements are divisible by d.

-- Check if all elements in the list are divisible by d

allDivisible :: [Int] -> Int -> Bool

allDivisible [] \_ = True

allDivisible (x:xs) d = x `mod` d == 0 && allDivisible xs d

-- Generate all contiguous sublists

allSublists :: [Int] -> [[Int]]

allSublists [] = []

allSublists (x:xs) = subFrom (x:xs) ++ allSublists xs

-- Generate sublists from a given starting point

subFrom :: [Int] -> [[Int]]

subFrom [] = []

subFrom ys = ys : subFrom (init ys)

-- Find the longest sublist from list of lists

findLongest :: [[Int]] -> [Int]

findLongest [] = []

findLongest [x] = x

findLongest (x:y:ys)

| length x >= length y = findLongest (x:ys)

| otherwise = findLongest (y:ys)

-- Main function to find the longest sublist divisible by d

longestDivisibleSublist :: [Int] -> Int -> [Int]

longestDivisibleSublist xs d = findLongest (filter (`allDivisible` d) (allSublists xs))

-- Main I/O

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

putStrLn "Enter the divisor:"

divStr <- getLine

let numbers = map read (words input) :: [Int]

let divisor = read divStr :: Int

let result = longestDivisibleSublist numbers divisor

putStrLn $ "The longest sublist divisible by " ++ show divisor ++ " is: " ++ show result

**11. Longest Sublist of Multiples of a Number**

**Problem**: Write a recursive Haskell program longestMultiplesSublist that takes a list of integers and a number n, and returns the longest sublist where all elements are multiples of n.

longestMultiplesSublist([2, 4, 6, 7, 8, 10, 12], 2) → Output: [2, 4, 6]

longestMultiplesSublist([5, 10, 15, 20, 25, 30], 5) → Output: [5, 10, 15, 20, 25, 30]

-- Check if a number is a multiple of n

isMultiple :: Int -> Int -> Bool

isMultiple x n = x `mod` n == 0

-- Main function

longestMultiplesSublist :: [Int] -> Int -> [Int]

longestMultiplesSublist xs n = findLongest (groupMultiples xs n)

-- Group all sublists of consecutive multiples of n

groupMultiples :: [Int] -> Int -> [[Int]]

groupMultiples [] \_ = []

groupMultiples (x:xs) n

| isMultiple x n = buildGroup [x] xs n

| otherwise = groupMultiples xs n

-- Build one group and continue with remaining

buildGroup :: [Int] -> [Int] -> Int -> [[Int]]

buildGroup current [] \_ = [current]

buildGroup current (y:ys) n

| isMultiple y n = buildGroup (current ++ [y]) ys n

| otherwise = current : groupMultiples ys n

-- Find the longest list from a list of lists

findLongest :: [[Int]] -> [Int]

findLongest [] = []

findLongest [x] = x

findLongest (x:y:ys)

| length x >= length y = findLongest (x:ys)

| otherwise = findLongest (y:ys)

-- Main IO

main :: IO ()

main = do

putStrLn "Enter a list of integers separated by spaces:"

input <- getLine

putStrLn "Enter a number to check for multiples:"

nStr <- getLine

let numbers = map read (words input) :: [Int]

let n = read nStr :: Int

let result = longestMultiplesSublist numbers n

putStrLn $ "Longest sublist of multiples of " ++ show n ++ ": " ++ show result

**12. Longest Consecutive Sublist with a Fixed Step**

**Problem**: Write a recursive Haskell program longestFixedStepSublist that takes a list of integers and a step size k, and returns the longest sublist where the difference between consecutive elements is k.

longestFixedStepSublist([1, 3, 5, 7, 2, 4, 6, 8], 2) → Output: [1, 3, 5, 7]

longestFixedStepSublist([10, 20, 30, 40, 25, 35], 10) → Output: [10, 20, 30, 40]

-- Check if the difference between two numbers is k

hasStep :: Int -> Int -> Int -> Bool

hasStep x y k = abs (x - y) == k

-- Find the longest sublist with a fixed step k

longestFixedStepSublist :: [Int] -> Int -> [Int]

longestFixedStepSublist [] \_ = []

longestFixedStepSublist xs k = findLongest (groupSteps xs)

where

groupSteps [] = []

groupSteps (y:ys) = buildGroup [y] ys

buildGroup current [] = [current]

buildGroup current (z:zs)

| hasStep (last current) z k = buildGroup (current ++ [z]) zs

| otherwise = current : groupSteps (z:zs)

findLongest [] = []

findLongest [x] = x

findLongest (x:xs)

| length x > length (findLongest xs) = x

| otherwise = findLongest xs