Q1

Coding

```
power2 :: Integer -> Integer
power2 n
   | n == 0 = 1
   | n `mod` 2 == 0 = (power2 (n `div` 2))^2
   | otherwise = (power2 ((n - 1) `div` 2))^2 * 2

main = do
  print(power2 5)
```

Explanation

power2 :: Integer -> Integer
This takes integer as input value and output integer value

```
power2 n
   | n == 0 = 1
```

If n is equal to 0 then it the result will be 1

```
  | n `mod` 2 == 0 = (power2 (n `div` 2))^2
```

If "n mod 2" is equal to 0 then it equals to **(power2 (n `div` 2))^2.** n mod 2 == 0 only occurs when n is even. Since it has function inside the function which the power2(n) will be done recursively. It divide n by 2 everytime when n mod 2 which the n value decrease until n == 0 = 1.

If n is odd then do **(power2 ((n - 1) `div` 2))^2 * 2**, we minus 1 in odd number which it becomes even and we multiple by 2 at the end for missing 2 from n - 1.

If I put n == 5 then it comes to this line **(power2 ((5 - 1) `div` 2))^2 * 2,** then power2(4 div 2) which it equals to power(2)^2*2 so we need to recursively find the value of power(2).

2 `mod` 2 == 0 = (power2 (2 `div` 2))^2 it becomes **(power2 (1)^2)** so recursively find power2(1) by using **(power2 ((1 - 1) `div` 2))^2 * 2** which it equal to **(power2 (0 `div` 2))^2 * 2**

(power2 (0))^2 * 2 becomes 2 since n == 0 = 1 so 1^2 * 2 = 2.

Then we put back the values that we got to the recursive function.
N = 0, when power(0) = 1
N = 1, when power(1) = (power2 ((1-1) `div` 2))^2 * 2 = (power2 (0))^2 * 2 = 1^2 * 2 = 2
N = 2, when power(2) = (power2 (2 `div` 2))^2 = (power2 (1))^2 = 2^2 = 4
N = 5, when power(5) = (power2 ((5 - 1) `div` 2))^2 * 2 = (power2 (2))^2 * 2 = 4^2 * 2 = 32

Q2

Coding from tutorial lab6

```
divisors :: Integer -> [Integer]
divisors n = [x | x <- [1..n], n `mod` x == 0]

isPrime :: Integer -> Bool
isPrime n
   | divisors n == [1,n]   = True
   | otherwise             = False

main = do
  let n = 8
  print(divisors n)
  print(isPrime n)
```

Explanation

divisors :: Integer -> [Integer]

This takes integer as input and output integer list

divisors n = [x | x <- [1..n], n `mod` x == 0]

This is list comprehension that x has the value from [1,2,3,4…….n] and if the n is divisible by x then we put that value inside of n. If I do n = 8 then we already contain 1 so starting from x = 2 , 8 mod 2 == 0 so we put 2 in n list. 8 mod 3 =/= 0, 8 mod 4 == 0, 8 mod 5 =/= 0, 8 mod 6 == 0, 8 mod 7 =/= 0 and 8 mod 8 == 0. If we gather the values that is divisible by x then we get values of [1,2,4,8].

isPrime :: Integer -> Bool

Integer as input and output true or false.

```
isPrime n
   | divisors n == [1,n]   = True
   | otherwise             = False
```

Once we get the divisor, we also can get whether the n is the prime number by checking whether the integer n has more than 2 numbers in list. Prime number won't get divided by any number so it will always contain n = [1 and n] and that doesn't exceed 2 numbers in list. If there are more than 2 numbers in list then we know that the number isn't prime.

Q3

Coding

```
instrictorder :: [Int] -> Bool
instrictorder [] = True
instrictorder [_] = True
instrictorder (x:y:z) = x > y && instrictorder (y:z)

main = do
   let list = [4,3,2,1]
   print (show (instrictorder list))
```

Explanation

instrictorder :: [Int] -> Bool

Takes integer list and output boolean value.

instrictorder [] = True
instrictorder [_] = True

If the instrictorder have less than 2 elements then we don't have elements to compare each values to know whether it's decreasing so I return value as true.

instrictorder (x:y:z) = x > y && instrictorder (y:z)

X will be the first element in the list and y will be the second element in the list and z will be rest in the list.Check if first element x is greater than the second element y and we run the function instrictorder recursively to check other values in the list. Due to the use of AND, we need to get true for both cases. If one of condition fails, we are returning false. We are recursively run instrictorder (y:z) which the y will be the first element to compare with next element in the list.

For example, list = [4,3,2,1] then first x = 4 and y = 3 z = 2,1 and we are comparing value x and y so x is bigger than y and we do recursive function instrictorder(y:z) which compares y = 3 and z = 2,1. instrictorder function, y = 3 will now become x = 3 due to recursive function and z = 2 will be y = 2 value and z = 1. it compares first element with second element, x = 3 will be compared to y = 2, and x is greater than y and we do instrictorder with y = 2 and z = 1. Y becomes x and z becomes y from recursive. We will be eventually come to the 1 element and 0 element which one element or 0 element must be true to return true at the end. Everything must be true to return true otherwise false.

Q4

Coding

```haskell
binom :: Integer -> Integer -> Integer
binom n k
  | k < 0 || k > n = error "n must be greater than k and k must be greater than 0"
  | n < 1 = error "n must be greater than 0"
  | k == 0 = 1
  | k == n = 1
  | otherwise = product [n-k+1..n] `div` product [1..k]

main = do
  let n = 10
  let k = 5

  print (binom n k)
```

Explanation

binom :: Integer -> Integer -> Integer

This is taking two integer value as input and output integer

The binomial coefficient n k is defined as follows for integers n >= 1. And 0 <= k <= n.

```haskell
    | k == 0 = 1
    | k == n = 1
```

If k is equal to 0, the equation of binomial coefficient will always equal to 1.
If k is equal to n then the equation will always equal to 1. Therefore we are returning value 1 as result for those two cases.

```haskell
    | otherwise = product [n-k+1..n] `div` product [1..k]
```

Otherwise we do multiply all the numbers with the equation of product [n-k+1..n] and then divide product [1..k].

Denominator has increasing multiplication from 1 to k. Numerator also has value increasing from n-k+1 to n. Let say if I do n = 10 and k = 5, product[10-5+1..10] = product[6….10] = Product[6,7,8,9,10], it will multiply all those numbers. Also product[1…k] = product [1….5] = product [1,2,3,4,5]. Result would be 30240/120 = 252

Q5

Coding

```
duplicate :: String -> Integer -> String
duplicate _ 0 = ""
duplicate str 1 = str
duplicate str n = str ++ duplicate str (n-1)

main = do
   let result = duplicate "CPSC449 " 3
   print(result)
```

Explanation

duplicate :: String -> Integer -> String

We input string and integer value and output string.

duplicate _ 0 = ""

If we put n = 0 then we are returning empty string.

duplicate str 1 = str

If we put n = 1 then we are printing string

duplicate str n = str ++ duplicate str (n-1)

If we put integer n >= 2 then we are printing n times the number of string.
Since if n >= 2 then we are printing string first and we recursively run duplicate str(n-1).
For example, n = 3, the n we are doing duplicate str 3 = str ++ duplicate str(3-1), we are printing one string at that line and we run duplicate str(2) which we get duplicate str 2 = str ++ duplicate str(2-1), this also prints one string. We run duplicate str 1 = str. This prints 3 times as total.