Task 1.

Implement the function to find the combination of 4 digits in a row which gives the max multiplication. If object is not a string or there are no combinations found - return nil. If combination is found - return it's multiplication result.
E.g.
max_multiplication('abc12345def') => 120 # 2*3*4*5
max_multiplication('a1b2c3d4e') => nil

# Solution 1:

```
#I have written the code in such a manner that, it will prompt the user to give input.
#Based on the input it will execute.
# Actually it will basically search for max consecutive 4 digits and do the operation.

def max_multiplication(input_string):
    consecutive_digits = ""  # To store consecutive digits
    result = 1  # Initialize the result to 1
    stored_numbers = []  # To store the individual numbers contributing to the product

    for char in input_string:
        if char.isdigit():
            consecutive_digits += char
        else:
            if len(consecutive_digits) >3:
                # Sort the consecutive digits and keep the highest four
                sorted_digits = sorted(consecutive_digits, reverse=True)
                highest_four = sorted_digits[:4]

                product = 1
                for digit in highest_four:
                    int_digit = int(digit)
                    product *= int_digit
                    stored_numbers.append(int_digit)  # Store individual number
                result *= product
            consecutive_digits = ""  # Reset if a non-digit character is encountered
    if result == 1:
        return "nil", []  # Return "nil" and an empty list if no consecutive digits are found
    else:
        return result, stored_numbers

# Get input from the user
user_input = input("Enter a string: ")
result, stored_numbers = max_multiplication(user_input+" ")

if result != "nil":
    numbers_str = ' * '.join(str(num) for num in stored_numbers)
    print(f"Result is: {result}  # {numbers_str} = {result}")
else:
    print("nil.")
```

******************************END**

Task 2.

Implement the function to sort array of numbers by amount of '1' in its binary representation. Numbers with identical amount of '1's should be ordered by decimal representation.
E.g.
# 3 = 11, 7 = 111, 8 = 1000, 9 = 1001.
sort([3,7,8,9]) => [8,3,9,7] # 1000, 11, 1001, 111

#Solution 2:

```
# Function to convert a decimal number to binary
def decimal_to_binary(decimal_number):
    binary_representation = ""
    if decimal_number == 0:
        binary_representation = "0"
    else:
        while decimal_number > 0:
            remainder = decimal_number % 2
            binary_representation = str(remainder) + binary_representation
            decimal_number = decimal_number // 2
    return binary_representation

# Function to get the count of 1s in a binary number
def count_ones(binary_string):
    return binary_string.count('1')

# Input the size of the array
array_size = int(input("Enter the size of the array: "))

# Initialize an empty list to store the binary representations
binary_array = []

# Input the decimal numbers and convert them to binary
for i in range(array_size):
    decimal_number = int(input(f"Enter decimal number {i + 1}: "))
    binary_representation = decimal_to_binary(decimal_number)
    binary_array.append(binary_representation)

# Create a dictionary to store the counts
counts = {}

# Loop through the binary numbers and count '1's
for binary_number in binary_array:
    count = count_ones(binary_number)
    counts[binary_number] = count

# Sort the binary numbers based on the counts and then the decimal values
def custom_sort_key(item):
    binary_string, decimal_number = item
    return (counts[binary_string], decimal_number)
```

```
sorted_binary_numbers = sorted(zip(binary_array, [int(binary, 2) for binary in binary_array]),
key=custom_sort_key)

# Print the sorted binary numbers in decimal format
for _, decimal_number in sorted_binary_numbers:
    print(decimal_number)
```