##Design and Analysis of Algorithms

*Code-1(Fibo Recursive/java)

```
class Fibonacci
{
   public static int fibo(int n)
   {
      if(n<=1)
      {
         return n;
      }
      return fibo(n-1) + fibo(n-2);
   }
   public static void main(String[] args)
   {
       int n = 6;
      for (int i = 0; i <= n; i++)
      {
         System.out.print(fibo(i) + " ");
      }
   }
}
```

*Code-2(Huffman Coding using Greedy Strategy/python)

```
import heapq
```

```
class Node:
    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right
        self.huff = ''

    def __lt__(self, nxt):
        return self.freq < nxt.freq

def printNodes(node, val=''):
    newVal = val + str(node.huff)
    if node.left:
        printNodes(node.left, newVal)
    if node.right:
        printNodes(node.right, newVal)
    if not node.left and not node.right:
        print(f"{node.symbol} -> {newVal}")

n = int(input("Enter number of symbols: "))
chars = []
freq = []
for i in range(n):
    char = input(f"Enter symbol : ")
    chars.append(char)
    f = int(input(f"Enter frequency for {char}: "))
    freq.append(f)
```

```
nodes = []
for i in range(n):
    heapq.heappush(nodes, Node(freq[i], chars[i]))

start_time = time.time()

while len(nodes) > 1:
    left = heapq.heappop(nodes)
    right = heapq.heappop(nodes)

    left.huff = 0
    right.huff = 1

    newNode = Node(left.freq + right.freq, left.symbol + right.symbol, left, right)

    heapq.heappush(nodes, newNode)

huffman_tree_root = nodes[0]

print("Huffman Codes:")
printNodes(huffman_tree_root)
```

*Code-3(Greedy Fractional Knapsack/python)

```
class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight
```

```python
    def value_per_weight(self):
        return self.value / self.weight

def fractional_knapsack(items, capacity):
    items = sorted(items, key=lambda x: x.value_per_weight(), reverse=True)

    total_value = 0
    for item in items:
        if capacity == 0:
            break

        if item.weight <= capacity:
            total_value += item.value
            capacity -= item.weight
        else:
            fraction = capacity / item.weight
            total_value += item.value * fraction
            capacity = 0

    return total_value

if __name__ == "__main__":
    n = int(input("Enter the number of items: "))

    items = []

    for i in range(n):
        value = float(input(f"Enter value for item {i+1}: "))
```

```python
        weight = float(input(f"Enter weight for item {i+1}: "))
        items.append(Item(value, weight))

    capacity = float(input("Enter the capacity of the knapsack: "))

    max_value = fractional_knapsack(items, capacity)

    print(f"Maximum value in the knapsack = {max_value:.2f}")
```

*Code-4(0-1 Knapsack Using DP/python)

```python
def knapSack(W, wt, val, n):
    dp = [[0 for _ in range(W + 1)] for _ in range(n + 1)]
    for i in range(1, n + 1):
        for w in range(1, W + 1):
            if wt[i - 1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - wt[i - 1]] + val[i - 1])
            else:
                dp[i][w] = dp[i - 1][w]
    return dp[n][W]

if __name__ == '__main__':
    n = int(input("Enter the number of items: "))
    profit = []
    weight = []
    for i in range(n):
        value = int(input(f"Enter value for item {i + 1}: "))
        weight_value = int(input(f"Enter weight for item {i + 1}: "))
```

```
        profit.append(value)
        weight.append(weight_value)
    W = int(input("Enter the capacity of the knapsack: "))
    max_value = knapSack(W, weight, profit, n)
    print(f"Maximum value in the knapsack = {max_value}")
```

*Code-5(N-queens/python)

```
def is_safe(board, row, col, n):
    # Check this column on upper side
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check upper diagonal on right side
    for i, j in zip(range(row, -1, -1), range(col, n)):
        if board[i][j] == 1:
            return False

    return True

def print_board(board):
    for row in board:
```

```
        print(" ".join("[Q]" if x == 1 else "[]" for x in row))
    print()

def n_queen(board, row, n):
    if row == n:
        print_board(board)
        return

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1
            n_queen(board, row + 1, n)
            board[row][col] = 0 # backtrack

def main():
    n = int(input("Enter the number of queens: "))

    # Initialize the board with zeros
    board = [[0 for _ in range(n)] for _ in range(n)]

    n_queen(board, 0, n)

    print("--------All possible solutions--------")

if __name__ == "__main__":
    main()
```

##Machine Learning

*Code-1(Uber Fare Prediction/jupyter)

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import r2_score, mean_squared_error


file_path = r"D:\VStud\DSPractice\MLsem7\ML_datasets\uber.csv" #r refers to raw string(replace your own path)

df = pd.read_csv(file_path)

df.head()


#1.Data Preprocessing

df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])

df.dropna()

#Feature Engineering

df['pickup_hour'] = df['pickup_datetime'].dt.hour

df['pickup_day'] = df['pickup_datetime'].dt.day

df['pickup_day_of_week'] = df['pickup_datetime'].dt.dayofweek


#2.Identify Outliers

numerical_features = ['fare_amount', 'pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude', 'passenger_count']


for feature in numerical_features:
```

```
    sns.boxplot(x= df[feature])
    plt.title(f"Box plot for {feature}")
    plt.show

# 3. Check correlation (excluding non-numeric columns)
# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=[np.number])

# Calculate correlation matrix
correlation_matrix = numeric_df.corr()

# Plot the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

#4. Implement LR and Rf

X = df[['pickup_hour', 'pickup_day', 'pickup_day_of_week', 'passenger_count']]
y = df['fare_amount']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lin = lr.predict(X_test)

rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

```
y_pred_rf = rf.predict(X_test)

#5. Evaluate the Models

r2_lin = r2_score(y_test,y_pred_lin)
rmse_lin = np.sqrt(mean_squared_error(y_test, y_pred_lin))

print(f'Linear Reg r2:{r2_lin:3f}')
print(f'Linear Reg rmse:{rmse_lin:3f}')

r2_rf = r2_score(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))

print(f'Random Forrest r2:{r2_lin:3f}')
print(f'Random Forrest rmse:{rmse_lin:3f}')
```

*Code-2(Email Spam Detection/jupyter)

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import

file_path = r"D:\VStud\DSPractice\MLsem7\ML_datasets\emails.csv"  # Use raw string for Windows paths
df = pd.read_csv(file_path)
print(df.head())
```

```
# Step 2: Check the dataset structure
print(df.info())

# Step 3: Drop unnecessary columns
df.drop(columns=['Email No.'], inplace=True)

# Step 4: Check for missing values
print(df.isna().sum())

# Step 5: Split the data into features (X) and target (y)
X = df.iloc[:, :-1]  # Independent variables (all columns except the last)
y = df.iloc[:, -1]   # Dependent variable (last column)
print(X.shape, y.shape)

# Step 6: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=8)

# Step 7: Train and evaluate K-Nearest Neighbors
knn_model = KNeighborsClassifier(n_neighbors=2)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
accuracy_knn = metrics.accuracy_score(y_test, y_pred_knn)
print(f"Accuracy for K-Nearest Neighbors model: {accuracy_knn:.3f}")

# Step 8: Train and evaluate Linear SVM
svm_model = LinearSVC(random_state=8, max_iter=900000)
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
```

```
accuracy_svm = metrics.accuracy_score(y_test, y_pred_svm)
print(f"Accuracy for Linear SVM model: {accuracy_svm:.3f}")
```

*Code-3(Gradient Descent Algorithm/jupyter)

```
# Import necessary libraries (optional for visualization)
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Define the function and its derivative
def function(x):
    return (x + 3)**2

def derivative(x):
    return 2 * (x + 3)

# Step 2: Set initial parameters
x_current = 2  # Starting point
learning_rate = 0.1  # Step size
tolerance = 1e-6  # Convergence criterion
max_iterations = 1000  # Maximum number of iterations

# Step 3: Perform gradient descent
x_history = [x_current]  # To store x values for visualization

for i in range(max_iterations):
    gradient = derivative(x_current)
    x_next = x_current - learning_rate * gradient
```

```
    # Append to history for visualization
    x_history.append(x_next)

    # Check if the change is smaller than the tolerance
    if abs(x_next - x_current) < tolerance:
        break

    x_current = x_next  # Update the current point

# Step 4: Print the result
print(f"Local minima found at x = {x_current:.6f}")
print(f"Function value at local minima y = {function(x_current):.6f}")

# Optional: Plot the function and gradient descent path
x_vals = np.linspace(-10, 5, 100)
y_vals = function(x_vals)

plt.plot(x_vals, y_vals, label='y = (x + 3)^2')
plt.scatter(x_history, [function(x) for x in x_history], color='red', label='Gradient Descent Path')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gradient Descent on y = (x + 3)^2')
plt.legend()
plt.show()
```

*Code-4(Implement KNN on Diabetes Dataset)

```
#Step-1
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

file_path = "D:\VStud\DSPractice\MLsem7\ML_datasets\diabetes.csv"
df = pd.read_csv(file_path)
df.head()

#Step-2
X = df.drop(columns=['Outcome'])
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)

#Step-3
```

```
# Compute evaluation metrics
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Display the results
print("Confusion Matrix:\n", conf_matrix)
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

*Code-5(K-Means Clustering on Sales.csv)

```
#Step-1
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

file_path = "D:\VStud\DSPractice\MLsem7\ML_datasets\sales_data_sample.csv"
df = pd.read_csv(file_path, encoding = 'latin1')
df.head()

#Step-2
```

```
#Select features
features = df[['SALES','QUANTITYORDERED']]
df = df.dropna()

#Normalize them
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Step 3: Determine the number of clusters using the elbow method
inertia = []
range_clusters = range(1, 11)

for i in range_clusters:
   kmeans = KMeans(n_clusters=i, random_state=42)
   kmeans.fit(scaled_features)
   inertia.append(kmeans.inertia_)

# Plot the elbow method graph
plt.figure(figsize=(10, 6))
plt.plot(range_clusters, inertia, marker='o', linestyle='--')
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal Number of Clusters")
plt.show()



# Step 4: Apply K-Means clustering with the optimal number of clusters(Optional Method)
optimal_clusters = 2
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
```

```
df['Cluster'] = kmeans.fit_predict(scaled_features)

# Visualize the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['SALES'], y=df['QUANTITYORDERED'], hue=df['Cluster'], palette='viridis')
plt.title('Clusters of Sales and Quantity Ordered')
plt.xlabel('Sales')
plt.ylabel('Quantity Ordered')
plt.show()
```

##Blockchain Technology

*Code-3(Bank Sol/remix ide)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Bank {
    address public accHolder;
    uint256 private balance = 0; // Make balance private for better encapsulation

    constructor() {
        accHolder = msg.sender;
    }

    function withdraw(uint256 amount) public payable {
        require(msg.sender == accHolder, "You are not the account owner.");
        require(amount > 0, "Withdraw amount must be greater than 0.");
        require(amount <= balance, "You don't have enough balance.");
```

```
        // Transfer the specified amount to the account holder
        payable(msg.sender).transfer(amount);

        // Deduct the withdrawn amount from the balance
        balance -= amount;
    }

    function deposit() public payable {
        require(msg.sender == accHolder, "You are not the account owner.");
        require(msg.value > 0, "Deposit amount should be greater than 0.");

        // Increase the balance by the deposited amount
        balance += msg.value;
    }

    function showBalance() public view returns (uint256) {
        require(msg.sender == accHolder, "You are not the account owner.");
        return balance;
    }
}
```

*Code-4(Student Sol/remix ide)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract StudentData {
    // Owner of the contract
    address public owner;
```

```
    // Structure to store student details
    struct Student {
        string name;
        uint age;
        string course;
        uint marks;
        bool isExist;
    }
    // Dynamic array of students
    Student[] public students;
    // Event to log the addition of a new student
    event StudentAdded(string name, uint age, string course, uint marks);
    // Modifier to allow only the contract owner to add students
    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can perform this action.");
        _;
    }
    // Constructor to initialize the owner of the contract
    constructor() {
        owner = msg.sender;
    }
    function addStudent(string memory _name, uint _age, string memory _course, uint _marks) public
onlyOwner {
        // Add the new student to the array
        students.push(Student(_name, _age, _course, _marks, true));
        emit StudentAdded(_name, _age, _course, _marks);
    }
    function getStudent(uint index) public view returns (string memory name, uint age, string memory
course, uint marks) {
```

```
        require(index < students.length, "Student does not exist.");
        Student storage student = students[index];
        return (student.name, student.age, student.course, student.marks);
    }
    receive() external payable {
        // Logic for receiving Ether can be added if needed
    }
    fallback() external payable {
        // Fallback logic can be added if needed
    }
    function getContractBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```