DAA

1. **Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.**

```
def fibonacci_iter(n):
    if n < 0:
        return -1, 1
    if n == 0 or n == 1:
        return n, 1
    steps = 0
    a = 0
    b = 1
    for i in range(2, n+1):
        c = a + b
        a = b
        b = c
        steps += 1
    return c, steps+1

def fibonacci_recur(n):
    if n < 0:
        return -1, 1
    if n == 0 or n == 1:
        return n, 1
    fib1, steps1 = fibonacci_recur(n-1)
    fib2, steps2 = fibonacci_recur(n-2)
    return fib1 + fib2, steps1 + steps2 + 1

if __name__ == '__main__':
    n = int(input("Enter a number: "))
    print("Iterative:", fibonacci_iter(n)[0])
    print("Steps:", fibonacci_iter(n)[1])
    print("Recursive:", fibonacci_recur(n)[0])
    print("Steps:", fibonacci_recur(n)[1])
```

2. **Write a program to implement Huffman Encoding using a greedy strategy.**

```
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, other):
        return self.freq < other.freq

def printNodes(node, val=""):
    newval = val + node.huff
    if node.left:
        printNodes(node.left, newval)
    if node.right:
        printNodes(node.right, newval)
    else:
        print(f"{node.symbol} -> {newval}")

chars = ["a", "b", "c", "d", "e", "f"]
freqs = [5, 9, 12, 13, 16, 45]
nodes = []

for i in range(len(chars)):
    heapq.heappush(nodes, node(freqs[i], chars[i]))

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)

printNodes(nodes[0])
```

3. **Write a program to solve a fractional Knapsack problem using a greedy method.**

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

def fractionalKnapsack(w, arr):
    arr.sort(key=lambda x: x.profit/x.weight, reverse=True)
    finalValue = 0.0
    for item in arr:
        if w >= item.weight:
            finalValue += item.profit
            w -= item.weight
        else:
            finalValue += item.profit * (w/item.weight)
            break
    return finalValue

if __name__ == "__main__":
    n = int(input("Enter number of items-\n"))
    arr = []
    for i in range(n):
        profit = int(input("Enter profit of item " + str(i + 1) + "-\n"))
        weight = int(input("Enter weight of item " + str(i + 1) + "-\n"))
        arr.append(Item(profit, weight))
    w = int(input("Enter capacity of knapsack-\n"))
    print("Maximum value in knapsack: ", fractionalKnapsack(w, arr))
```

4. **Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.**

```
def knapsack_01(n, values, weights, W):
    dp = [[0] * (W+1) for _ in range(n+1)]

    for i in range(n+1):
        for w in range(W+1):
            if i == 0 or w == 0:
                dp[i][w] = 0
            elif weights[i-1] <= w:
                dp[i][w] = max(dp[i-1][w], dp[i-1][w-weights[i-1]] + values[i-1])
            else:
                dp[i][w] = dp[i-1][w]

    selected_items = []
    i, w = n, W
    while i > 0 and w > 0:
        if dp[i][w] != dp[i-1][w]:
            selected_items.append(i-1)
            w -= weights[i-1]
        i -= 1

    return dp[n][W], selected_items

if __name__ == "__main__":
    n = 3
    values = [60, 100, 120]
    weights = [10, 20, 30]
    W = 50

    max_value, selected_items = knapsack_01(n, values, weights, W)
    print("Maximum value:", max_value)
    print("Selected items:", selected_items)
```

5. **Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen’s matrix.**

```
def solveNQueens(n: int, first_queen_col: int):
    col = set()
    posDiag = set()
    negDiag = set()

    res = []
    board = [["."] * n for _ in range(n)]
    def backtrack(r):
        if r == n:
            res.append(["".join(row) for row in board])
            return

        for c in range(n):
            if c in col or (r + c) in posDiag or (r - c) in negDiag:
                continue

            col.add(c)
            posDiag.add(r + c)
            negDiag.add(r - c)
            board[r][c] = "Q"

            backtrack(r + 1)

            col.remove(c)
            posDiag.remove(r + c)
            negDiag.remove(r - c)
            board[r][c] = "."

    col.add(first_queen_col)
    posDiag.add(0 + first_queen_col)
    negDiag.add(0 - first_queen_col)
    board[0][first_queen_col] = "Q"

    backtrack(1)  # Start with the second row
    return res

if __name__ == "__main__":
    n = 8
    first_queen_col = 1
    board = solveNQueens(n, first_queen_col)[0]
    for row in board:
        print(" ".join(row))
```

6. Write a program for analysis of quick sort by using deterministic and randomized variant.

```
import random
import timeit

def deterministic_partition(arr, low, high):
    pivot = arr[high]
    i = low - 1

    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def randomized_partition(arr, low, high):
    pivot_index = random.randint(low, high)
    arr[pivot_index], arr[high] = arr[high], arr[pivot_index]
    return deterministic_partition(arr, low, high)

def quick_sort(arr, low, high, pivot_selector):
    if low < high:
        pivot_index = pivot_selector(arr, low, high)
        quick_sort(arr, low, pivot_index - 1, pivot_selector)
        quick_sort(arr, pivot_index + 1, high, pivot_selector)

if __name__ == "__main__":
    arr_sizes = [100, 1000, 10000, 100000]
    for size in arr_sizes:
        arr = [random.randint(1, 1000) for _ in range(size)]
        arr.sort(reverse=True)

        deterministic_time = timeit.timeit("quick_sort(arr.copy(), 0, len(arr) - 1,
deterministic_partition)",
                                   globals=globals(),
                                   number=10)

        randomized_time = timeit.timeit("quick_sort(arr.copy(), 0, len(arr) - 1, randomized_partition)",
                                   globals=globals(),
                                   number=10)

        print(f"Array size: {size}")
        print(f"Deterministic Quick Sort time: {deterministic_time:.6f} seconds")
        print(f"Randomized Quick Sort time: {randomized_time:.6f} seconds")
        print("-" * 40)
```

ML

1. **Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks**

```
#Importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("uber.csv")

df.head()

df.info()

df.columns

df = df.drop(['Unnamed: 0', 'key'], axis= 1)

df.shape

df.dtypes

df.pickup_datetime = pd.to_datetime(df.pickup_datetime)

df.dtypes

df.isnull().sum()

df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True)

df.isnull().sum()

df= df.assign(hour = df.pickup_datetime.dt.hour,
day= df.pickup_datetime.dt.day,
month = df.pickup_datetime.dt.month,
year = df.pickup_datetime.dt.year,
dayofweek = df.pickup_datetime.dt.dayofweek)

df.head()

from math import *
# function to calculate the travel distance from the longitudes and latitudes
def distance_transform(longitude1, latitude1, longitude2, latitude2):
    travel_dist = []

    for pos in range(len(longitude1)):
        long1,lati1,long2,lati2 =
map(radians,[longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos]])
        dist_long = long2 - long1
        dist_lati = lati2 - lati1
        a = sin(dist_lati/2)**2 + cos(lati1) * cos(lati2) * sin(dist_long/2)**2
        c = 2 * asin(sqrt(a))*6371
        travel_dist.append(c)

    return travel_dist

df['dist_travel_km'] = distance_transform(df['pickup_longitude'].to_numpy(),
df['pickup_latitude'].to_numpy(),
df['dropoff_longitude'].to_numpy(),
df['dropoff_latitude'].to_numpy()
)

df.head()

df = df.drop('pickup_datetime',axis=1)

df.head()

df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))

def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

def treat_outliers_all(df1 , col_list):
    for c in col_list:
        df1 = remove_outlier(df , c)
    return df1

df = treat_outliers_all(df , df.iloc[: , 0::])

df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))

df= df.loc[(df.dist_travel_km >= 1) | (df.dist_travel_km <= 130)]
print("Remaining observastions in the dataset:", df.shape)

incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |(df.pickup_latitude < -90) |
(df.dropoff_latitude > 90) |(df.dropoff_latitude < -90) |
(df.pickup_longitude > 180) |(df.pickup_longitude < -180) |
(df.dropoff_longitude > 90) |(df.dropoff_longitude < -90)
]

df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
df.head()

df.isnull().sum()

sns.heatmap(df.isnull())

corr = df.corr()

corr

fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True)

x =
df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','passenger_count','
hour','day','month','year','dayofweek','dist_travel_km']]

y = df['fare_amount']

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.33)

from sklearn.linear_model import LinearRegression
regression = LinearRegression()

regression.fit(X_train,y_train)

regression.intercept_

regression.coef_

prediction = regression.predict(X_test)

print(prediction)

y_test

from sklearn.metrics import r2_score

r2_score(y_test,prediction)

from sklearn.metrics import mean_squared_error

MSE = mean_squared_error(y_test,prediction)

MSE

RMSE = np.sqrt(MSE)

RMSE
```

```
    from sklearn.ensemble import RandomForestRegressor

    rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of trees you
    want to build before making the prediction

    rf.fit(X_train,y_train)

    y_pred = rf.predict(X_test)

    y_pred

    R2_Random = r2_score(y_test,y_pred)

    R2_Random

    MSE_Random = mean_squared_error(y_test,y_pred)
    MSE_Random

    RMSE_Random = np.sqrt(MSE_Random)
    RMSE_Random
```

2.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics

df=pd.read_csv('emails.csv')

df.head()

df.columns

df.isnull().sum()

df.dropna(inplace = True)

df.drop(['Email No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']

from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("Prediction",y_pred)

print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))

print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))

# cost C = 1
model = SVC(C = 1)
# fit
model.fit(X_train, y_train)
# predict
y_pred = model.predict(X_test)

metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)

print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

3.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries

df = pd.read_csv("Churn_Modelling.csv")

df.head()

df.shape

df.describe()

df.isnull()

df.isnull().sum()

df.info()

df.dtypes

df.columns

df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary columns
```

```
df.head()

def visualization(x, y, xlabel):
 plt.figure(figsize=(10,5))
 plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
 plt.xlabel(xlabel,fontsize=20)
 plt.ylabel("No. of customers", fontsize=20)
 plt.legend()

df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']

visualization(df_churn_exited, df_churn_not_exited, "Tenure")

df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']

visualization(df_churn_exited2, df_churn_not_exited2, "Age")

X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary']]
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)

df = pd.concat([df,gender,states], axis = 1)

df.head()


X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Male','Germany','Spain']]

y = df['Exited']


from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()


X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

X_train

X_test
```

```
import keras #Keras is the wrapper on the top of tenserflow
#Can use Tenserflow as well but won't be able to understand the errors initially.

from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers

classifier = Sequential()

#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
#Uniform helps to distribute the weight uniformly
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_initializer = "uniform"))

classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform")) #Adding second
hidden layers

classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uniform"))


classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['accuracy'])


classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron in last

classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to training

y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result

from sklearn.metrics import confusion_matrix,accuracy_score,classification_report

cm = confusion_matrix(y_test,y_pred)

cm

accuracy = accuracy_score(y_test,y_pred)

accuracy

plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

print(classification_report(y_test,y_pred))
```

4.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def f(x):
    return (x+3)**2
def df(x):
    return 2*x + 6

def gradient_descent(initial_x, learning_rate, num_iterations):
    x = initial_x
    x_history = [x]
    for i in range(num_iterations):
        gradient = df(x)
        x = x - learning_rate * gradient
        x_history.append(x)
    return x, x_history

initial_x = 2
learning_rate = 0.1
num_iterations = 50
x, x_history = gradient_descent(initial_x, learning_rate, num_iterations)
print("Local minimum: {:.2f}".format(x))

#Create a range of x values to plot
x_vals = np.linspace(-1, 5, 100)
#Plot the function f(x)
plt.plot(x_vals, f(x_vals))
# Plot the values of x at each iteration
plt.plot(x_history, f(np.array(x_history)), 'rx')
#Label the axes and add a title
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Gradient Descent')
#Show the plot
plt.show()
```

5.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans, k_means
from sklearn.decomposition import PCA

df = pd.read_csv("sales_data_sample.csv")

df.head()

df.shape

df.describe()

df.info()
df.isnull().sum()

df.dtypes

df_drop = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATUS','POSTALCODE', 'CITY', 'TERRITORY',
'PHONE', 'STATE', 'CONTACTFIRSTNAME', 'CONTACTLASTNAME', 'CUSTOMERNAME', 'ORDERNUMBER']df = df.drop(df_drop, axis=1)

df.isnull().sum()

df.dtypes

df['COUNTRY'].unique()

df['PRODUCTLINE'].unique()

df['DEALSIZE'].unique()

productline = pd.get_dummies(df['PRODUCTLINE']) #Converting the categorical columns. Dealsize =
pd.get_dummies(df['DEALSIZE'])

df = pd.concat([df,productline,Dealsize], axis = 1)

df_drop = ['COUNTRY','PRODUCTLINE','DEALSIZE'] #Dropping Country too as there are alot of countries. df
= df.drop(df_drop, axis=1)

df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes

df.drop('ORDERDATE', axis=1, inplace=True)

df.dtypes

distortions = []

K = range(1,10)
for k in K: kmeanModel = KMeans(n_clusters=k)
kmeanModel.fit(df)
distortions.append(kmeanModel.inertia_)

plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()

X_train = df.values

X_train.shape

model = KMeans(n_clusters=3,random_state=2)
model = model.fit(X_train)
predictions = model.predict(X_train)

unique,counts = np.unique(predictions,return_counts=True)
```

```
counts = counts.reshape(1,3)

counts_df = pd.DataFrame(counts,columns=['Cluster1','Cluster2','Cluster3'])

counts_df.head()

pca = PCA(n_components=2)

reduced_X = pd.DataFrame(pca.fit_transform(X_train),columns=['PCA1','PCA2'])

reduced_X.head()

plt.figure(figsize=(14,10))
plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])

model.cluster_centers_

reduced_centers = pca.transform(model.cluster_centers_)

reduced_centers

plt.figure(figsize=(14,10))
plt.scatter(reduced_X['PCA1'],reduced_X['PCA2']) plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',marker='x',s=300)

reduced_X['Clusters'] = predictions

reduced_X.head()

plt.figure(figsize=(14,10))
plt.scatter(reduced_X[reduced_X['Clusters'] == 0].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] ==
0].loc[:,'PCA2'],color='slateblue')
plt.scatter(reduced_X[reduced_X['Clusters'] == 1].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] ==
1].loc[:,'PCA2'],color='springgreen')
plt.scatter(reduced_X[reduced_X['Clusters'] == 2].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] ==
2].loc[:,'PCA2'],color='indigo')

plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',marker='x',s=300)
```

BT

3.

```
// SPDX-License-Identifier: MIT
//https://betterprogramming.pub/developing-a-smart-contract-by-using-re mix-ide-81ff6f44ba2f
pragma solidity >=0.7.0 <0.9.0;
contract SimpleBank
{
struct client_account{
int client_id;
address client_address;
uint client_balance_in_ether;
}
client_account[] clients;
int clientCounter;
address payable manager;
modifier onlyManager() {

require(msg.sender == manager, "Only manager can call this!");
_;
}
modifier onlyClients() {
bool isclient = false;
for(uint i=0;i<clients.length;i++){
if(clients[i].client_address == msg.sender){
isclient = true;
break;
}
}
require(isclient, "Only clients can call this!");
_;
}
constructor() {
clientCounter = 0;
}
receive() external payable { }
function setManager(address managerAddress) public returns(string memory){
manager = payable(managerAddress);
return "";
}
function joinAsClient() public payable returns(string memory){

clients.push(client_account(clientCounter++, msg.sender, address(msg.sender).balance));

return "";
}
function deposit() public payable onlyClients{
payable(address(this)).transfer(msg.value);
}
function withdraw(uint amount) public payable onlyClients{
payable(msg.sender).transfer(amount * 1 ether);
}
function sendInterest() public payable onlyManager{
for(uint i=0;i<clients.length;i++){
address initialAddress = clients[i].client_address;
payable(initialAddress).transfer(1 ether);
}
}
function getContractBalance() public view returns(uint){
return address(this).balance;
}
}
```

4.

```
// SPDX-License-Identifier: MIT
//https://betterprogramming.pub/developing-a-smart-contract-by-using-remix-ide-81ff6f44ba2f
pragma solidity ^0.5.0;
contract Crud {
struct User {
uint id;
string name;
}
User[] public users;
uint public nextId = 0;
function Create(string memory name) public {
users.push(User(nextId, name));
nextId++;
}
function Read(uint id) view public returns(uint, string memory) {
for(uint i=0; i<users.length; i++) {
if(users[i].id == id) {
return(users[i].id, users[i].name);
}
}
}
function Update(uint id, string memory name) public {
for(uint i=0; i<users.length; i++) {
if(users[i].id == id) {
users[i].name =name;
}
}
}
function Delete(uint id) public {
delete users[id];
}
function find(uint id) view internal returns(uint) {
for(uint i=0; i< users.length; i++) {
if(users[i].id == id) {
return i;
}
}
// if user does not exist then revert back
revert("User does not exist");
}
}
```