Code:

```
#include<iostream>
#include<vector>
using namespace std;
//Iteratively using memoization
int iStepFibbonacci(int n){
   vector<int> f;
   f.push_back(0);
   f.push_back(1);
   int cnt = 2;
   for(int i = 2; i < n; i++){
      cnt++;
      f.push_back(f[i - 1] + f[i - 2]);
   }
   return n;
}
int rSteps = 0;
int rStepFibbonacci(int n){
   rSteps++;
   if(n < 0) return 0;
   if(n == 1 || n == 0) return 1;
   return rStepFibbonacci(n - 1) + rStepFibbonacci(n - 2);
}
int main(){
   int n;
   cin >> n;
   cout << "Fibbonacci Value : " << rStepFibbonacci(n) << '\n';
   cout << "Steps required using Iteration : " << iStepFibbonacci(n) << '\n';
   cout << "Steps required using recursion : " << rSteps << '\n';
   return 0;
}
```

Output:

Ouput:

Fibbonacci Value : 8

Steps required using Iteration : 5

Steps required using recursion : 15

Code :

```
#include<bits/stdc++.h>
using namespace std;
struct MinHeapNode{
char data;
int freq;
MinHeapNode* left, *right;
MinHeapNode(char data, int freq){
left=right=nullptr;
this->data = data;
this->freq = freq;
}
};
void printCodes(struct MinHeapNode* root, string str){
if(root == nullptr){
return;
}
if(root->data != '$'){
cout << root->data << ": " << str << endl;
}
printCodes(root->left, str + "0");
printCodes(root->right, str + "1");
}
struct compare{
bool operator()(MinHeapNode* a, MinHeapNode* b){
return (a->freq > b->freq);
}
};void HuffmanCode(char data[], int freq[], int size){
struct MinHeapNode *left, *right, *temp;
priority_queue<MinHeapNode*, vector<MinHeapNode*>,compare> minHeap;
for(int i = 0; i < size; i++){
minHeap.push(new MinHeapNode(data[i], freq[i]));
```

```
}

while(minHeap.size() != 1){

left = minHeap.top();

minHeap.pop();

right = minHeap.top();

minHeap.pop();

temp = new MinHeapNode('$', left->freq + right->freq);

temp->left = left;

temp->right = right;

minHeap.push(temp);

}

printCodes(minHeap.top(), "");

}int main(){

char data[] = {'A', 'B', 'C', 'D'};

int freq[] = {23,12,34,10};

HuffmanCode(data, freq, 4);

return 0;

}

/*

Huffman Coding :

Time complexity: O(nlogn) where n is the number ofunique characters.

If there are n nodes, extractMin() is called 2*(n – 1) times.extractMin() takes O(logn) time as it calls minHeapify().So, overall complexity is O(nlogn).

*/
```

Output :

Ouput:
D: 100
B: 101
A: 11

Code :

```
#include<bits/stdc++.h>
using namespace std;
bool compare(pair<int,int> p1,pair<int,int> p2){
   double v1 = (double) p1.first/p1.second;
   double v2 = (double) p2.first/p2.second;
   return v1>v2;
}
int main(){
   int n;
   cin >> n;
   vector<pair<int,int>> a(n);
   for(int i=0;i<n;i++){
      cin >> a[i].first >> a[i].second;
   }
   int w;
   cin >> w;
   sort(a.begin(),a.end(),compare);
   double ans = 0;
   for(int i=0;i<n;i++){
      if(w>=a[i].second){
         ans+=a[i].first;
         w-=a[i].second;
         continue;
      }
      double vw = (double) a[i].first/a[i].second;
      ans += vw * w;
      w=0;
      break;
   }
   cout << ans << endl;
}
```

Output :

```
$ 5
10 60
20 100
30 120
5 30
15 50
50
```

Code :

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set()
dataset = pd.read_csv('/content/Churn_Modelling.csv', index_col = 'RowNumber')
dataset.head()
#Customer ID and Surname would not be relevant as features
X_columns = dataset.columns.tolist()[2:12]
Y_columns = dataset.columns.tolist()[-1:]
print(X_columns)
print(Y_columns)
X = dataset[X_columns].values
Y = dataset[Y_columns].values
#We need to encode categorical variables such as geography and gender
from sklearn.preprocessing import LabelEncoder
X_column_transformer = LabelEncoder()
X[:, 1] = X_column_transformer.fit_transform(X[:, 1])
#Lets Encode gender now
X[:, 2] = X_column_transformer.fit_transform(X[:, 2])
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

pipeline = Pipeline(
    [
        ('Categorizer', ColumnTransformer(
            [
                ("Gender Label Encoder", OneHotEncoder(categories = 'auto', drop
= 'first'), [2]),
                ("Geography Label Encoder", OneHotEncoder(categories = 'auto',
drop = 'first'), [1])
            ],
            remainder = 'passthrough', n_jobs = 1)),
        ('Normalizer', StandardScaler())
    ]
)
#Standardize the features
X = pipeline.fit_transform(X)
#Spilt the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,
random_state = 0)
#Let us create the Neural Network
from keras.models import Sequential
from keras.layers import Dense, Dropout
#Initialize ANN
classifier = Sequential()
#Add input layer and hidden layer
classifier.add(Dense(6, activation = 'relu', input_shape = (X_train.shape[1],
)))
classifier.add(Dropout(rate = 0.1))
#Add second layer
classifier.add(Dense(6, activation = 'relu'))
classifier.add(Dropout(rate = 0.1))
#Add output layer
classifier.add(Dense(1, activation = 'sigmoid'))
```

```
#Let us take a look at our network
classifier.summary()
#Optimize the weights
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])
#Fitting the Neural Network
history = classifier.fit(X_train, y_train, batch_size = 32, epochs = 200,
validation_split = 0.1, verbose = 2)
y_pred = classifier.predict(X_test)
print(y_pred[:5])
#Let us use confusion matrix with cutoff value as 0.5
y_pred = (y_pred > 0.5).astype(int)
print(y_pred[:5])
#Making the Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
#Accuracy of our NN
print(((cm[0][0] + cm[1][1])* 100) / len(y_test), '% of data was classified
correctly')
```

output :

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0
3	15619304	Onio	502	France	Female	42	8	159660.80	3	1
4	15701354	Boni	699	France	Female	39	1	0.00	2	0
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1

```
['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedS
alary']
['Exited']
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6)                 72

 dropout (Dropout)           (None, 6)                 0

 dense_1 (Dense)             (None, 6)                 42

 dropout_1 (Dropout)         (None, 6)                 0

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
 63/63 [==============================] - 0s 1ms/step
 [[0.21353428]
  [0.3550975 ]
  [0.1884149 ]
  [0.04963601]
  [0.2057534 ]]

[[0]
 [0]
 [0]
 [0]              [[1569   26]
 [0]]              [ 293  112]]
 84.05 % of data was classified correctly
```

Code :

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
#We do not want to see warnings
warnings.filterwarnings("ignore")
#import data
data = pd.read_csv("uber.csv")
#Create a data copy
df = data.copy()
#Print data
df.head
#pickup_datetime is not in required data format
df["pickup_datetime"] = pd.to_datetime(df["pickup_datetime"])
df.info()
df.describe()
df.isnull().sum()
#Correlation
df.corr()
#Drop the rows with missing values
df.dropna(inplace=True)
plt.boxplot(df['fare_amount'])
q_low = df["fare_amount"].quantile(0.01)
q_hi  = df["fare_amount"].quantile(0.99)
df = df[(df["fare_amount"] < q_hi) & (df["fare_amount"] > q_low)]
#Check the missing values now
df.isnull().sum()
#Time to apply learning models
from sklearn.model_selection import train_test_split
#Take x as predictor variable
x = df.drop("fare_amount", axis = 1)
#And y as target variable
y = df['fare_amount']
#Necessary to apply model
x['pickup_datetime'] = pd.to_numeric(pd.to_datetime(x['pickup_datetime']))
x = x.loc[:, x.columns.str.contains('^Unnamed')]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2
from sklearn.linear_model import LinearRegression
lrmodel = LinearRegression()
lrmodel.fit(x_train, y_train)
#Prediction
predict = lrmodel.predict(x_test)
#Check Error
from sklearn.metrics import mean_squared_error
lrmodelrmse = np.sqrt(mean_squared_error(predict, y_test))
print("RMSE error for the model is ", lrmodelrmse)
#Let's Apply Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
rfrmodel = RandomForestRegressor(n_estimators = 100, random_state = 101)
#Fit the Forest
rfrmodel.fit(x_train, y_train)
rfrmodel_pred = rfrmodel.predict(x_test)
#Errors for the forest
rfrmodel_rmse = np.sqrt(mean_squared_error(rfrmodel_pred, y_test))
print("RMSE value for Random Forest is:",rfrmodel_rmse)
```

Output:

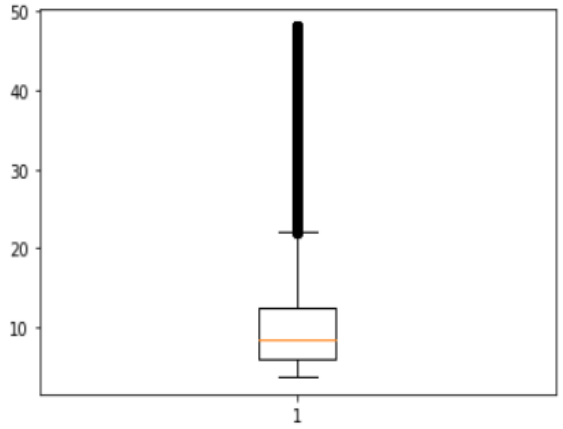
```
 <bound method NDFrame.head of         Unnamed: 0                            key  fare_amount  \
 0         24238194    2015-05-07 19:52:06.0000003          7.5
 1         27835199    2009-07-17 20:04:56.0000002          7.7
 2         44984355   2009-08-24 21:45:00.00000061         12.9
 3         25894730    2009-06-26 08:22:21.0000001          5.3
 4         17610152  2014-08-28 17:47:00.000000188         16.0
 ...            ...                           ...          ...
 199995    42598914   2012-10-28 10:49:00.00000053          3.0
 199996    16382965    2014-03-14 01:09:00.0000008          7.5
 199997    27804658   2009-06-29 00:42:00.00000078         30.9
 199998    20259894    2015-05-20 14:56:25.0000004         14.5
 199999    11951496   2010-05-15 04:08:00.00000076         14.1

                 pickup_datetime  pickup_longitude  pickup_latitude  \
 0       2015-05-07 19:52:06 UTC        -73.999817        40.738354
 1       2009-07-17 20:04:56 UTC        -73.994355        40.728225
 2       2009-08-24 21:45:00 UTC        -74.005043        40.740770
 3       2009-06-26 08:22:21 UTC        -73.976124        40.790844
 4       2014-08-28 17:47:00 UTC        -73.925023        40.744085
 ...                         ...               ...              ...
 199995  2012-10-28 10:49:00 UTC        -73.987042        40.739367
 199996  2014-03-14 01:09:00 UTC        -73.984722        40.736837
 199997  2009-06-29 00:42:00 UTC        -73.986017        40.756487
 199998  2015-05-20 14:56:25 UTC        -73.997124        40.725452
 199999  2010-05-15 04:08:00 UTC        -73.984395        40.720077

         dropoff_longitude  dropoff_latitude  passenger_count
 0               -73.999512         40.723217                1
 1               -73.994710         40.750325                1
 2               -73.962565         40.772647                1
 3               -73.965316         40.803349                3
 4               -73.973082         40.761247                5
 ...                    ...               ...              ...
 199995          -73.986525         40.740297                1
 199996          -74.006672         40.739620                1
 199997          -73.858957         40.692588                2
 199998          -73.983215         40.695415                1
 199999          -73.985508         40.768793                1

 [200000 rows x 9 columns]>
 <class 'pandas.core.frame.DataFrame'>
 RangeIndex: 200000 entries, 0 to 199999
 Data columns (total 9 columns):
  #   Column             Non-Null Count   Dtype
 ---  ------             --------------   -----
  0   Unnamed: 0         200000 non-null  int64
  1   key                200000 non-null  object
  2   fare_amount        200000 non-null  float64
  3   pickup_datetime    200000 non-null  datetime64[ns, UTC]
  4   pickup_longitude   200000 non-null  float64
  5   pickup_latitude    200000 non-null  float64
  6   dropoff_longitude  199999 non-null  float64
  7   dropoff_latitude   199999 non-null  float64
  8   passenger_count    200000 non-null  int64
 dtypes: datetime64[ns, UTC](1), float64(5), int64(2), object(1)
 memory usage: 13.7+ MB
```

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658	40.768001	2.000000
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
Unnamed: 0	1.000000	-0.000223	-0.000266	0.000061	-0.000310	0.000938	0.002311
fare_amount	-0.000223	1.000000	0.004654	-0.003154	0.003021	-0.004621	0.010705
pickup_longitude	-0.000266	0.004654	1.000000	-0.806902	0.830658	-0.844705	-0.000644
pickup_latitude	0.000061	-0.003154	-0.806902	1.000000	-0.770049	0.691893	-0.001441
dropoff_longitude	-0.000310	0.003021	0.830658	-0.770049	1.000000	-0.912750	0.000105
dropoff_latitude	0.000938	-0.004621	-0.844705	0.691893	-0.912750	1.000000	-0.000726
passenger_count	0.002311	0.010705	-0.000644	-0.001441	0.000105	-0.000726	1.000000

```
: {'whiskers': [<matplotlib.lines.Line2D at 0x241e10fadc0>,
   <matplotlib.lines.Line2D at 0x241e11130d0>],
  'caps': [<matplotlib.lines.Line2D at 0x241e1113460>,
   <matplotlib.lines.Line2D at 0x241e11137f0>],
  'boxes': [<matplotlib.lines.Line2D at 0x241e10fa970>],
  'medians': [<matplotlib.lines.Line2D at 0x241e1113b80>],
  'fliers': [<matplotlib.lines.Line2D at 0x241e1113f10>],
  'means': []}
```




```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

```
RMSE error for the model is  7.083585521002763

RMSE value for Random Forest is: 8.565996490346976
```

Code :

```
import numpy as np
import pandas as pd

data = pd.read_csv('./diabetes.csv')
data.head()

#Check for null or missing values

data.isnull().sum()

#Replace zero values with mean values
for column in data.columns[1:-3]:
    data[column].replace(0, np.NaN, inplace = True)
    data[column].fillna(round(data[column].mean(skipna=True)), inplace = True)
data.head(10)

X = data.iloc[:, :8] #Features
Y = data.iloc[:, 8:] #Predictor

#Perform Spliting
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=0)

#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn_fit = knn.fit(X_train, Y_train.values.ravel())
knn_pred = knn_fit.predict(X_test)

from sklearn.metrics import confusion_matrix, precision_score, recall_score,
f1_score, accuracy_score
print("Confusion Matrix")
print(confusion_matrix(Y_test, knn_pred))
print("Accuracy Score:", accuracy_score(Y_test, knn_pred))
print("Reacal Score:", recall_score(Y_test, knn_pred))
print("F1 Score:", f1_score(Y_test, knn_pred))
print("Precision Score:",precision_score(Y_test, knn_pred))
```

Output :

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Pregnancies       0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
Pedigree          0
Age               0
Outcome           0
dtype: int64
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
0	6	148.0	72.0	35.0	156.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	156.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	156.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
5	5	116.0	74.0	29.0	156.0	25.6	0.201	30	0
6	3	78.0	50.0	32.0	88.0	31.0	0.248	26	1
7	10	115.0	72.0	29.0	156.0	35.3	0.134	29	0
8	2	197.0	70.0	45.0	543.0	30.5	0.158	53	1
9	8	125.0	96.0	29.0	156.0	32.0	0.232	54	1

```
Confusion Matrix
[[88 19]
 [19 28]]
Accuracy Score: 0.7532467532467533
Reacal Score: 0.5957446808510638
F1 Score: 0.5957446808510638
Precision Score: 0.5957446808510638
```

Code :

```
pragma solidity >= 0.7.0;
// Write a smart contract on a test network, for Bank account of a
customer for
  // following operations: Deposit money | Withdraw Money | Show balance
contract Bank{
    mapping(address => uint) public user_account;
    mapping(address => bool) public user_exist;

    function create_account() public payable returns(string memory){
        require(user_exist[msg.sender] == false, "Account Already created!");
        user_account[msg.sender] = msg.value;
        user_exist[msg.sender] = true;
        return "Account created";
    }

    function deposit(uint amount) public payable returns(string memory){
        require(user_exist[msg.sender] == true, "Account not created!");
        require(amount > 0, "Amount should be greater than 0");
        user_account[msg.sender] += amount;
        return "Amount deposisted sucessfully";
    }

    function withdraw(uint amount) public payable returns(string memory){
        require(user_exist[msg.sender] == true, "Account not created!");
        require(amount > 0, "Amount should be greater than 0");
        require(user_account[msg.sender] >= amount, "Amount is greater than
money deposisted");
        user_account[msg.sender] -= amount;
        return "Amount withdrawn sucessfully";
    }

    function account_balance() public view returns(uint){
        return user_account[msg.sender];
    }

    function account_exists() public view returns(bool){
        return user_exist[msg.sender];
    }

}
```