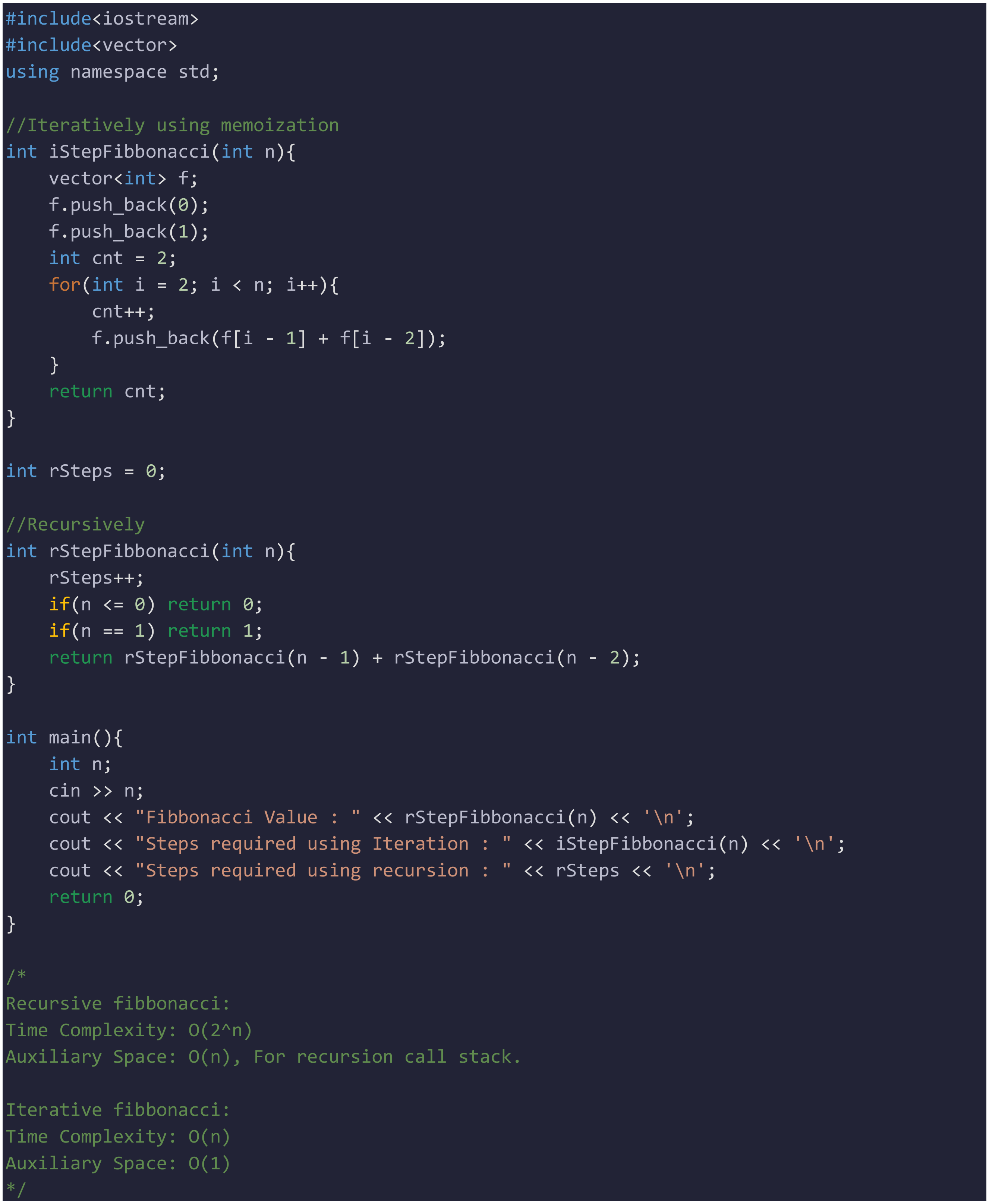
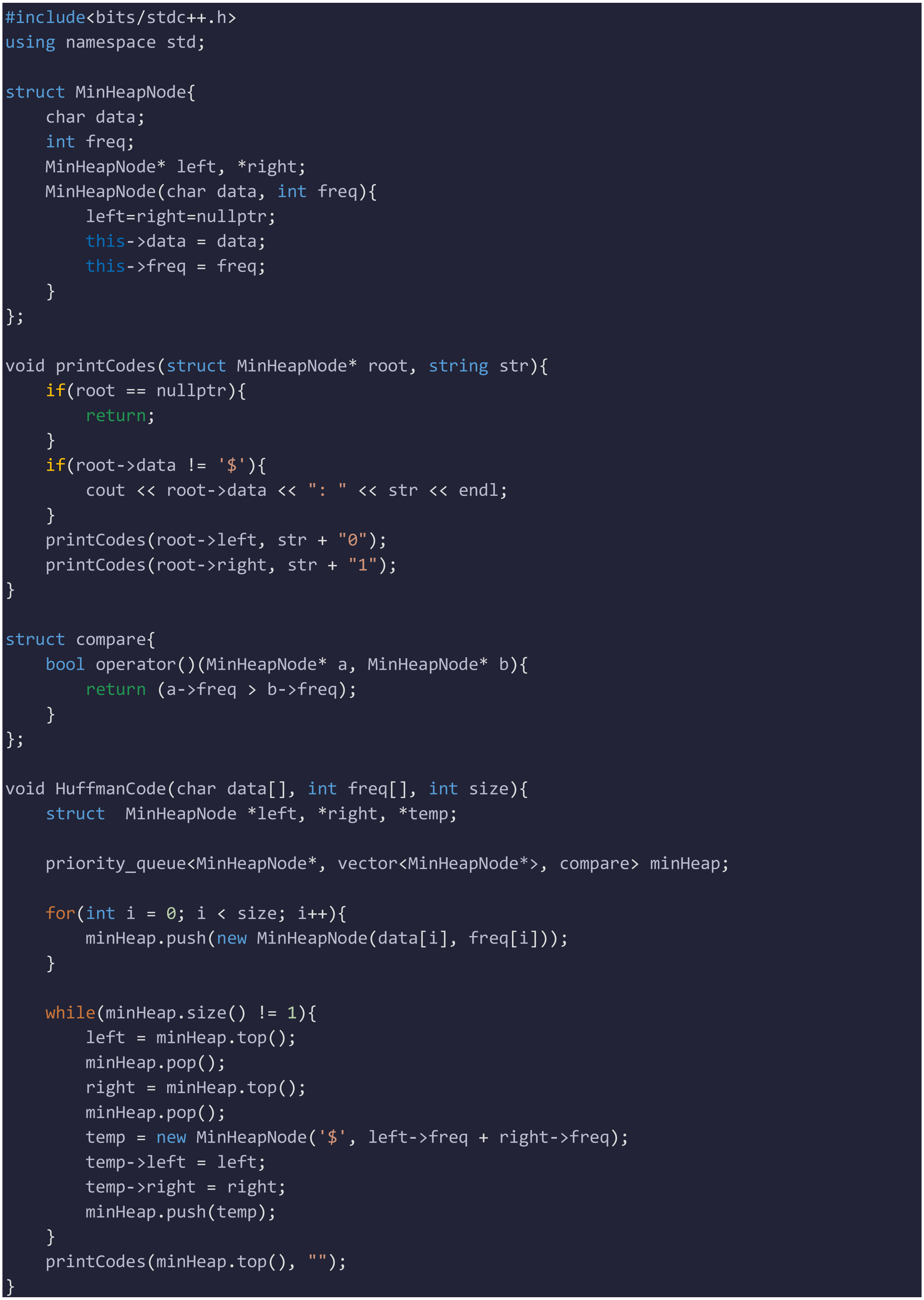
Fibonacci:



```
#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 cnt;
}

int rSteps = 0;

//Recursively
int rStepFibbonacci(int n){
    rSteps++;
    if(n <= 0) return 0;
    if(n == 1) 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;
}

/*
Recursive fibbonacci:
Time Complexity: O(2^n)
Auxiliary Space: O(n), For recursion call stack.

Iterative fibbonacci:
Time Complexity: O(n)
Auxiliary Space: O(1)
*/
```

Huffman:



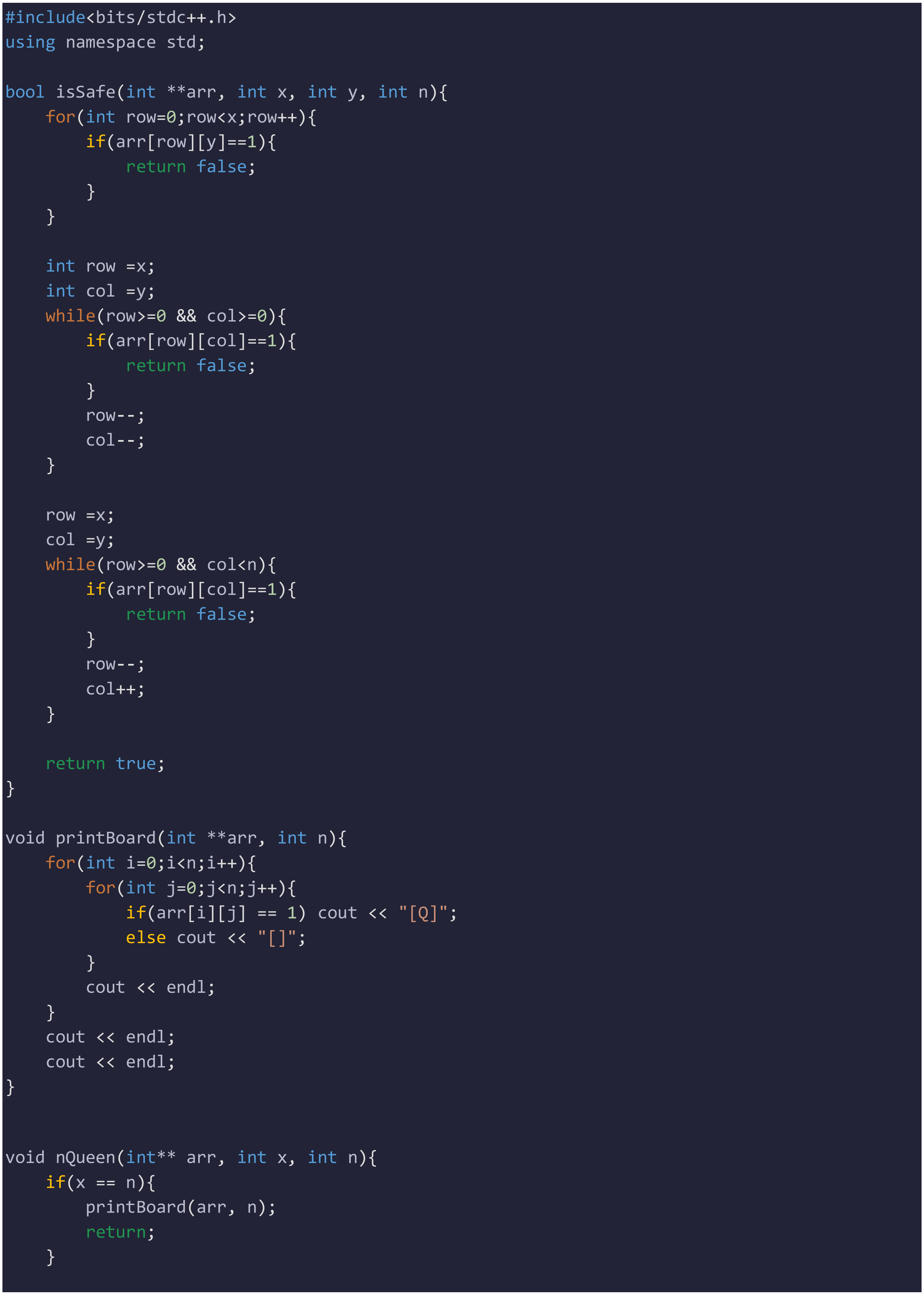
```cpp
#include<bits/stdc++.h>
using namespace std;

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(){
    int size;
    cout << "Enter the number of characters: ";
    cin >> size;
    char data[size];
    int freq[size];

    cout << "Enter characters and their frequencies:\n";
    for (int i = 0; i < size; i++) {
        cin >> data[i] >> freq[i];
    }
    HuffmanCode(data, freq, size);
}

/*
Huffman Coding :
Time complexity: O(nlogn) where n is the number of unique 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).
*/
```

Knapsack:

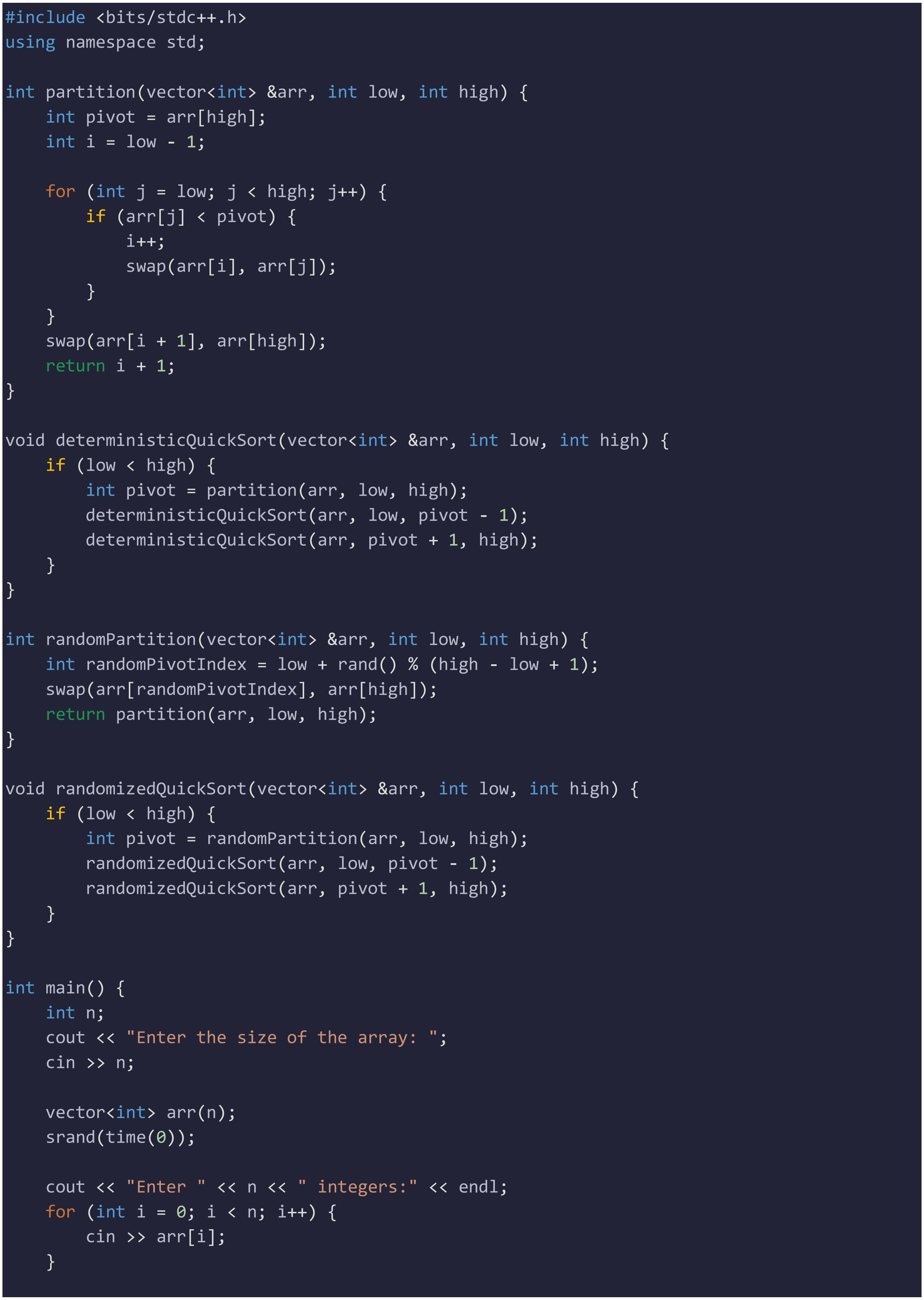
```
#include<iostream>
using namespace std;

int main(){
    int capacity;
    int items;

    cout << "Enter the capacity of the Knapsack: ";
    cin >> capacity;

    cout << "Enter the number of items: ";
    cin >> items;

    int price[items + 1];
    int wt[items + 1];

    cout << "Enter the prices of items (including item 0): ";
    for (int i = 0; i <= items; i++) {
        cin >> price[i];
    }
    cout << "Enter the weights of items (including item 0): ";
    for (int i = 0; i <= items; i++) {
        cin >> wt[i];
    }

    int dp[items + 1][capacity + 1];

    for(int i = 0; i <= items; i++){
        for(int j = 0; j <= capacity; j++){
            if(i == 0 || j == 0){
                // There's nothing to add to Knapsack
                dp[i][j] = 0;
            }
            else if(wt[i] <= j){
                // Choose previously maximum or value of the current item + value of
remaining weight
                dp[i][j] = max(dp[i - 1][j], price[i] + dp[i - 1][j - wt[i]]);
            }
            else{
                // Add previously added item to knapsack
                dp[i][j] = dp[i - 1][j];
            }
        }
    }

    cout << "Maximum Profit Earned: " << dp[items][capacity] << "\n";
    return 0;
}

/*
0/1 Knapsack :
Time Complexity: O(N*W).
where ‘N’ is the number of weight element and ‘W’ is capacity. As for every weight element
we traverse through all weight capacities 1<=w<=W.
Auxiliary Space: O(N*W).
The use of 2-D array of size ‘N*W’.
*/
```

N Queens:



```
#include<bits/stdc++.h>
using namespace std;

bool isSafe(int **arr, int x, int y, int n){
    for(int row=0;row<x;row++){
        if(arr[row][y]==1){
            return false;
        }
    }

    int row =x;
    int col =y;
    while(row>=0 && col>=0){
        if(arr[row][col]==1){
            return false;
        }
        row--;
        col--;
    }

    row =x;
    col =y;
    while(row>=0 && col<n){
        if(arr[row][col]==1){
            return false;
        }
        row--;
        col++;
    }

    return true;
}

void printBoard(int **arr, int n){
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            if(arr[i][j] == 1) cout << "[Q]";
            else cout << "[]";
        }
        cout << endl;
    }
    cout << endl;
    cout << endl;
}


void nQueen(int** arr, int x, int n){
    if(x == n){
        printBoard(arr, n);
        return;
    }
    for(int col=0;col<n;col++){
        if(isSafe(arr,x,col,n)){
            arr[x][col]=1;
            nQueen(arr,x+1,n);
            arr[x][col]=0;
        }
    }
}


int main(){
    int n;
    cin >> n;

    int **arr = new int*[n];
    for(int i=0;i<n;i++){
        arr[i] = new int[n];
        for(int j=0;j<n;j++){
            arr[i][j]=0;
        }
    }

    nQueen(arr, 0, n);

    cout << "--------All possible solutions--------";

    return 0;
}

/*
Time Complexity: O(N!)
Auxiliary Space: O(N^2)
*/
```

Quick Sort:

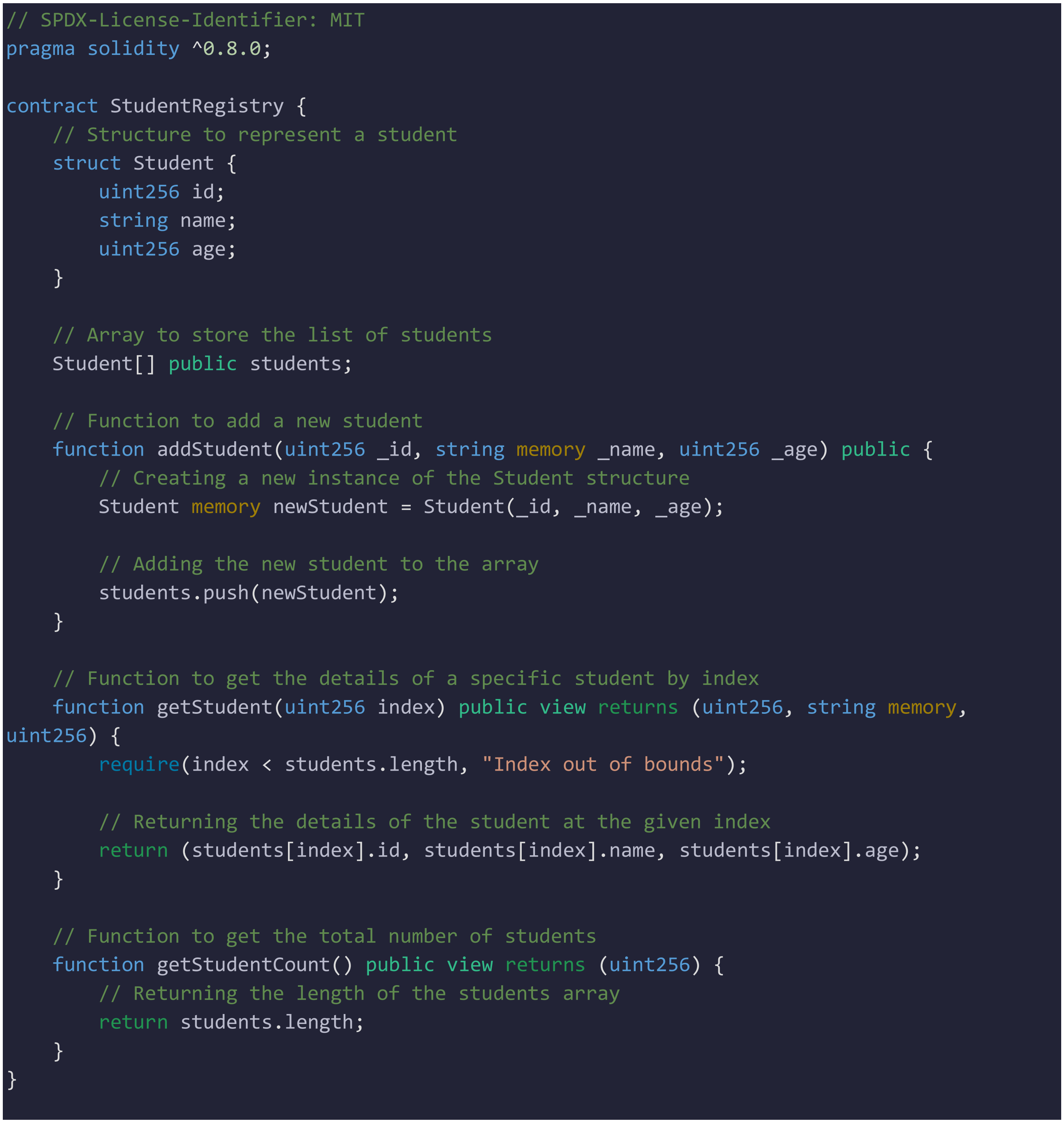


```
#include <bits/stdc++.h>
using namespace std;

int partition(vector<int> &arr, int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void deterministicQuickSort(vector<int> &arr, int low, int high) {
    if (low < high) {
        int pivot = partition(arr, low, high);
        deterministicQuickSort(arr, low, pivot - 1);
        deterministicQuickSort(arr, pivot + 1, high);
    }
}

int randomPartition(vector<int> &arr, int low, int high) {
    int randomPivotIndex = low + rand() % (high - low + 1);
    swap(arr[randomPivotIndex], arr[high]);
    return partition(arr, low, high);
}

void randomizedQuickSort(vector<int> &arr, int low, int high) {
    if (low < high) {
        int pivot = randomPartition(arr, low, high);
        randomizedQuickSort(arr, low, pivot - 1);
        randomizedQuickSort(arr, pivot + 1, high);
    }
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

    vector<int> arr(n);
    srand(time(0));

    cout << "Enter " << n << " integers:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    vector<int> deterministicArr = arr;
    vector<int> randomizedArr = arr;

    deterministicQuickSort(deterministicArr, 0, n - 1);
    randomizedQuickSort(randomizedArr, 0, n - 1);

    cout << "Deterministic Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << deterministicArr[i] << " ";
    }
    cout << endl;

    cout << "Randomized Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << randomizedArr[i] << " ";
    }
    cout << endl;

    return 0;
}

/*
Quick Sort:
Time Complexity: O(n log n)
Auxiliary Space: O(log n)
*/
```

Bank:

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

contract bank
{
    mapping(address => uint) public user_Account;
    mapping(address => bool) public user_Exists;

    function createAcc() public payable returns(string memory)
    {
        require( user_Exists[msg.sender] == false , "Account already created!");
        user_Exists[msg.sender] = true;
        user_Account[msg.sender] = msg.value;
        return "Account is created";
    }

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

    function withdraw(uint amount) public payable returns(string memory)
    {
        require( user_Exists[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 deposited");
        user_Account[msg.sender] -= amount;
        return "amount withdrawn";
    }

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

    function AccExists() public view returns(bool)
    {
        return user_Exists[msg.sender];
    }
}
```

Student:



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

contract StudentRegistry {
    // Structure to represent a student
    struct Student {
        uint256 id;
        string name;
        uint256 age;
    }

    // Array to store the list of students
    Student[] public students;

    // Function to add a new student
    function addStudent(uint256 _id, string memory _name, uint256 _age) public {
        // Creating a new instance of the Student structure
        Student memory newStudent = Student(_id, _name, _age);

        // Adding the new student to the array
        students.push(newStudent);
    }

    // Function to get the details of a specific student by index
    function getStudent(uint256 index) public view returns (uint256, string memory,
uint256) {
        require(index < students.length, "Index out of bounds");

        // Returning the details of the student at the given index
        return (students[index].id, students[index].name, students[index].age);
    }

    // Function to get the total number of students
    function getStudentCount() public view returns (uint256) {
        // Returning the length of the students array
        return students.length;
    }
}
```

In [1]:

```
#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")
```

In [2]:

```
data = pd.read_csv("uber.csv")
#Create a data copy
df = data.copy()
df.head()
```

Out[2]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_lat
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.73
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.72
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.74
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.79
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.74

In [3]:

```
df.info()
```

```
<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  object 
 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: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

In [4]:
```
#pickup_datetime is not in required data format
df["pickup_datetime"] = pd.to_datetime(df["pickup_datetime"])
df.info()
```

```
<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
```

In [5]:
```
df.describe()
```

Out[5]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dro
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	1
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658	
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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
```

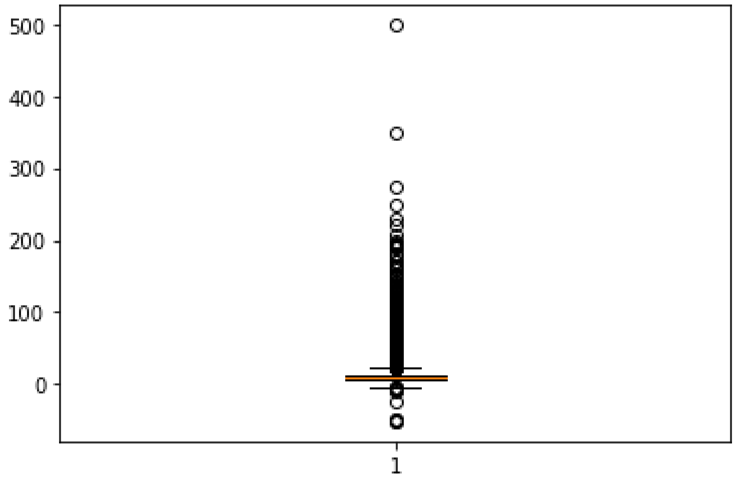
In [7]:
```
df.corr()
```

Out[7]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitu
Unnamed: 0	1.000000	0.000589	0.000230	-0.000341	0.0002
fare_amount	0.000589	1.000000	0.010457	-0.008481	0.0089
pickup_longitude	0.000230	0.010457	1.000000	-0.816461	0.8330
pickup_latitude	-0.000341	-0.008481	-0.816461	1.000000	-0.7747
dropoff_longitude	0.000270	0.008986	0.833026	-0.774787	1.0000
dropoff_latitude	0.000271	-0.011014	-0.846324	0.702367	-0.9170
passenger_count	0.002257	0.010150	-0.000414	-0.001560	0.0000

In [8]:
```
df.dropna(inplace=True)
plt.boxplot(df['fare_amount'])
```

Out[8]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1f440508fd0>,
  <matplotlib.lines.Line2D at 0x1f44051f370>],
 'caps': [<matplotlib.lines.Line2D at 0x1f44051f6d0>,
  <matplotlib.lines.Line2D at 0x1f44051fa30>],
 'boxes': [<matplotlib.lines.Line2D at 0x1f440508c70>],
 'medians': [<matplotlib.lines.Line2D at 0x1f44051fd90>],
 'fliers': [<matplotlib.lines.Line2D at 0x1f44052d130>],
 'means': []}
```



In [9]:

```
#Remove Outliers
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()
```

Out[9]:

```
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
```

In [10]:

```
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']

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,
                                                    random_state = 1)
```

In [11]:

```
from sklearn.linear_model import LinearRegression
lrmodel = LinearRegression()
lrmodel.fit(x_train, y_train)
```

Out[11]:

```
LinearRegression()
```

In [17]:

```
#Prediction
predict = lrmodel.predict(x_test)

#Check Error
from sklearn.metrics import mean_squared_error, r2_score
lrmodelrmse = np.sqrt(mean_squared_error(predict, y_test))
lrmodel_r2 = r2_score(y_test, predict)
print("RMSE error for the model is ", lrmodelrmse)
print("R-squared (R2) Error:", lrmodel_r2)
```

```
RMSE error for the model is  8.063863046328835
R-squared (R2) Error: -2.6395537326528995e-05
```

In [13]:

```
from sklearn.ensemble import RandomForestRegressor
rfrmodel = RandomForestRegressor(n_estimators = 100, random_state = 101)
rfrmodel.fit(x_train, y_train)
```

Out[13]:

```
RandomForestRegressor(random_state=101)
```

In [18]:

```
rfrmodel_pred = rfrmodel.predict(x_test)

rfrmodel_rmse = np.sqrt(mean_squared_error(rfrmodel_pred, y_test))
rfrmodel_r2 = r2_score(y_test, rfrmodel_pred)
print("RMSE value for Random Forest is:",rfrmodel_rmse)
print("R-squared (R2) Error:", rfrmodel_r2)
```

```
RMSE value for Random Forest is: 9.757713738069647
R-squared (R2) Error: -0.4642705335969681
```

In [ ]:

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:

```
df = pd.read_csv("emails.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastru
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	

5 rows × 3002 columns

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
Email No.     0
the           0
to            0
ect           0
and           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3002, dtype: int64
```

In [5]:
```
X = df.iloc[:,1:3001]
X
```

Out[5]:

	the	to	ect	and	for	of	a	you	hou	in	...	enhancements	connevey	jay	valu
0	0	0	1	0	0	0	2	0	0	0	...	0	0	0	
1	8	13	24	6	6	2	102	1	27	18	...	0	0	0	
2	0	0	1	0	0	0	8	0	0	4	...	0	0	0	
3	0	5	22	0	5	1	51	2	10	1	...	0	0	0	
4	7	6	17	1	5	2	57	0	9	3	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
5167	2	2	2	3	0	0	32	0	0	5	...	0	0	0	
5168	35	27	11	2	6	5	151	4	3	23	...	0	0	0	
5169	0	0	1	1	0	0	11	0	0	1	...	0	0	0	
5170	2	7	1	0	2	1	28	2	0	8	...	0	0	0	
5171	22	24	5	1	6	5	148	8	2	23	...	0	0	0	

5172 rows × 3000 columns

In [6]:
```
Y = df.iloc[:,-1].values
Y
```

Out[6]:
```
array([0, 0, 0, ..., 1, 1, 0], dtype=int64)
```

In [7]:
```
train_x,test_x,train_y,test_y = train_test_split(X,Y,test_size = 0.25)
```

In [8]:
```
svc = SVC(C=1.0,kernel='rbf',gamma='auto')
# C here is the regularization parameter. Here, L2 penalty is used(default)
#It is the inverse of the strength of regularization.
# As C increases, model overfits.
# Kernel here is the radial basis function kernel.
# gamma (only used for rbf kernel) : As gamma increases, model overfits.
svc.fit(train_x,train_y)
y_pred2 = svc.predict(test_x)
print("Accuracy Score for SVC : ", accuracy_score(y_pred2,test_y))
```

```
Accuracy Score for SVC :  0.897138437741686
```

In [9]:
```
#Check Error
from sklearn.metrics import mean_squared_error
svcmodelrmse = np.sqrt(mean_squared_error(y_pred2, test_y))
print("RMSE error for the model is ", svcmodelrmse)
```

```
RMSE error for the model is  0.32072038017300053
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,
                                                     random_state=42)
```

In [11]:

```
knn = KNeighborsClassifier(n_neighbors=7)
```

In [12]:

```
knn.fit(X_train, y_train)
```

Out[12]:

```
KNeighborsClassifier(n_neighbors=7)
```

In [13]:

```
print(knn.predict(X_test))
```

```
[0 0 1 ... 0 1 0]
```

In [14]:

```
y_pred3 = knn.predict(X_test)
print("Accuracy Score for KNN : ", accuracy_score(y_pred3,y_test))
```

```
Accuracy Score for KNN :  0.8676328502415459
```

In [15]:

```
#Check Error
from sklearn.metrics import mean_squared_error
knnmodelrmse = np.sqrt(mean_squared_error(y_pred3, y_test))
print("RMSE error for the model is ", knnmodelrmse)
```

```
RMSE error for the model is  0.36382296485853405
```

In [ ]:

```

```

In [1]:

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

In [2]:

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

Out[2]:

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

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [4]:

```
df.describe()
```

Out[4]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	H
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	

In [5]:
```
df.dtypes
```

Out[5]:
```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [8]:
```
from sklearn.preprocessing import LabelEncoder
# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply LabelEncoder to 'Gender' and 'Geography'
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Geography'] = label_encoder.fit_transform(df['Geography'])
```

In [9]:
```
# Create Features and Target vars
Features = df[[ 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary']]
Target = df['Exited']

X = np.asarray(Features)
Y = np.asarray(Target)
```

In [10]:
```
# Train test split
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
```

In [11]:

```
# Scaling of Data
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
scaler = StandardScaler()
mlpc = MLPClassifier(hidden_layer_sizes=(64,32), activation='relu', solver='adam',
                     random_state=42, max_iter=1000)

X_Train_scaled = scaler.fit_transform(X_train)
X_Test_Scaled = scaler.transform(X_test)
```

In [12]:

```
# Model building
mlpc.fit(X_Train_scaled, Y_train)
```

Out[12]:

```
MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=1000, random_state=42)
```

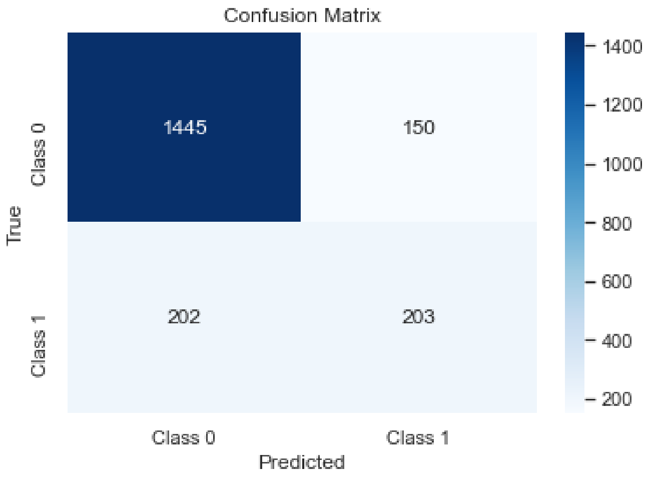
In [13]:

```
# Predict
Y_Pred = mlpc.predict(X_Test_Scaled)

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(Y_test, Y_Pred)*100, '%of data was classified correctly')
# help(accuracy_score)
```

```
82.39999999999999 %of data was classified correctly
```

In [14]:

```
cm = confusion_matrix(Y_test, Y_Pred)
#print(cm)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'],
            yticklabels=['Class 0', 'Class 1'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



In [ ]:

In [9]:

```
import numpy as np
import pandas as pd
import sympy as sym
import matplotlib as pyplot
from matplotlib import pyplot
```

In [2]:

```
def objective(x):
    return (x+3)**2
```

In [3]:

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

In [4]:

```
def gradient(alpha,start,max_iter):
    x_list=list()
    x=start
    x_list.append(x)
    for i in range(max_iter):
        gradi=derivative(x)
        x=x-(alpha*gradi)
        x_list.append(x)
    return x_list
x=sym.symbols('x')
expr=(x+3)**2.0
grad=sym.Derivative(expr,x)
print("{}".format(grad.doit()))
grad.doit().subs(x,2)
```

```
2.0*(x + 3)**1.0
```

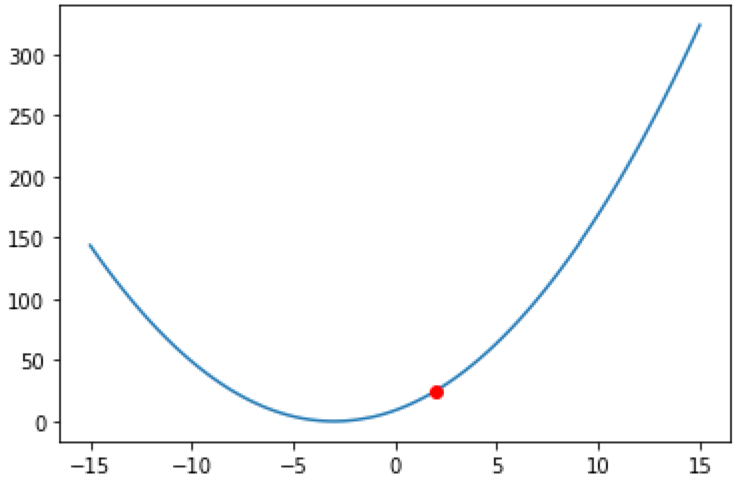
Out[4]: 10.0

In [5]:

```
alpha=0.1
start=2
max_iter=30
x=sym.symbols('x')
expr=(x+3)**2
```

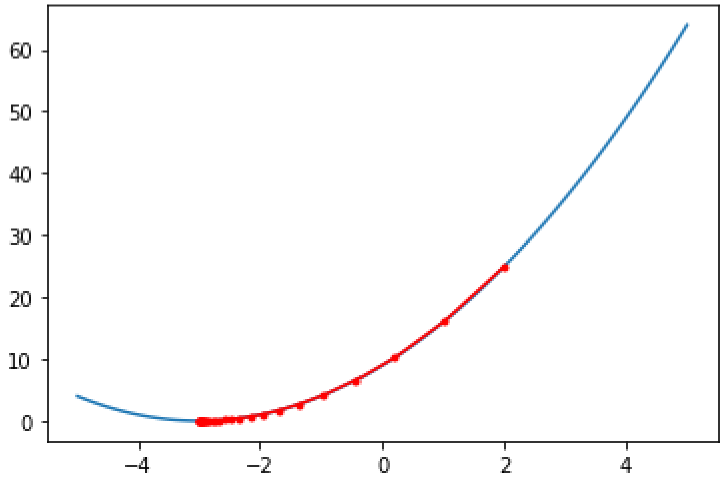
In [6]:

```
x_cor=np.linspace(-15,15,100)
pyplot.plot(x_cor,objective(x_cor))
pyplot.plot(2,objective(2),'ro')
pyplot.show()
```



In [7]:

```
x=gradient(alpha,start,max_iter)
x_cor=np.linspace(-5,5,100)
pyplot.plot(x_cor,objective(x_cor))

x_arr=np.array(x)
pyplot.plot(x_arr,objective(x_arr),'.-',color='red')
pyplot.show()
```



In [ ]:

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.read_csv('sales_data_sample.csv', encoding='unicode_escape')
```

In [3]:
```
df.head()
```

Out[3]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDEI
0	10107	30	95.70	2	2871.00	2/2
1	10121	34	81.35	5	2765.90	5/7/20(
2	10134	41	94.74	2	3884.34	7/1/20(
3	10145	45	83.26	6	3746.70	8/2
4	10159	49	100.00	14	5205.27	10/1

5 rows × 25 columns

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ORDERNUMBER       2823 non-null   int64  
 1   QUANTITYORDERED   2823 non-null   int64  
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64  
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object 
 6   STATUS            2823 non-null   object 
 7   QTR_ID            2823 non-null   int64  
 8   MONTH_ID          2823 non-null   int64  
 9   YEAR_ID           2823 non-null   int64  
 10  PRODUCTLINE       2823 non-null   object 
 11  MSRP              2823 non-null   int64  
 12  PRODUCTCODE       2823 non-null   object 
 13  CUSTOMERNAME      2823 non-null   object 
 14  PHONE             2823 non-null   object 
 15  ADDRESSLINE1      2823 non-null   object 
 16  ADDRESSLINE2      302 non-null    object 
 17  CITY              2823 non-null   object 
 18  STATE             1337 non-null   object 
 19  POSTALCODE        2747 non-null   object 
 20  COUNTRY           2823 non-null   object 
 21  TERRITORY         1749 non-null   object 
 22  CONTACTLASTNAME   2823 non-null   object 
 23  CONTACTFIRSTNAME  2823 non-null   object 
 24  DEALSIZE          2823 non-null   object 
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

In [5]:

```
to_drop = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATE', 'POSTALCODE', 'PHONE']
df = df.drop(to_drop, axis=1)
```

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
ORDERNUMBER            0
QUANTITYORDERED        0
PRICEEACH              0
ORDERLINENUMBER        0
SALES                  0
ORDERDATE              0
STATUS                 0
QTR_ID                 0
MONTH_ID               0
YEAR_ID                0
PRODUCTLINE            0
MSRP                   0
PRODUCTCODE            0
CUSTOMERNAME           0
CITY                   0
COUNTRY                0
TERRITORY           1074
CONTACTLASTNAME        0
CONTACTFIRSTNAME       0
DEALSIZE               0
dtype: int64
```

In [7]:

```
df.dtypes
```

Out[7]:

```
ORDERNUMBER           int64
QUANTITYORDERED       int64
PRICEEACH           float64
ORDERLINENUMBER       int64
SALES               float64
ORDERDATE            object
STATUS               object
QTR_ID                int64
MONTH_ID              int64
YEAR_ID               int64
PRODUCTLINE          object
MSRP                  int64
PRODUCTCODE          object
CUSTOMERNAME         object
CITY                 object
COUNTRY              object
TERRITORY            object
CONTACTLASTNAME      object
CONTACTFIRSTNAME     object
DEALSIZE             object
dtype: object
```

In [8]:

```
#ORDERDATE Should be in date time
df['ORDERDATE'] = pd.to_datetime(df['ORDERDATE'])
```

In [9]:

```
#We need to create some features in order to create cluseters
#Recency: Number of days between customer's latest order and today's date
#Frequency : Number of purchases by the customers
#MonetaryValue : Revenue generated by the customers
import datetime as dt
snapshot_date = df['ORDERDATE'].max() + dt.timedelta(days = 1)
df_RFM = df.groupby(['CUSTOMERNAME']).agg({
    'ORDERDATE' : lambda x : (snapshot_date - x.max()).days,
    'ORDERNUMBER' : 'count',
    'SALES' : 'sum'
})

#Rename the columns
df_RFM.rename(columns = {
    'ORDERDATE' : 'Recency',
    'ORDERNUMBER' : 'Frequency',
    'SALES' : 'MonetaryValue'
}, inplace=True)
```

In [10]:

```
df_RFM.head()
```

Out[10]:

	Recency	Frequency	MonetaryValue
CUSTOMERNAME			
AV Stores, Co.	196	51	157807.81
Alpha Cognac	65	20	70488.44
Amica Models & Co.	265	26	94117.26
Anna's Decorations, Ltd	84	46	153996.13
Atelier graphique	188	7	24179.96

In [11]:

```
# Divide into segments
# We create 4 quartile ranges
df_RFM['M'] = pd.qcut(df_RFM['MonetaryValue'], q = 4, labels = range(1,5))
df_RFM['R'] = pd.qcut(df_RFM['Recency'], q = 4, labels = list(range(4,0,-1)
df_RFM['F'] = pd.qcut(df_RFM['Frequency'], q = 4, labels = range(1,5))

df_RFM.head()
```

Out[11]:

	Recency	Frequency	MonetaryValue	M	R	F
CUSTOMERNAME						
AV Stores, Co.	196	51	157807.81	4	2	4
Alpha Cognac	65	20	70488.44	2	4	2
Amica Models & Co.	265	26	94117.26	3	1	2
Anna's Decorations, Ltd	84	46	153996.13	4	3	4
Atelier graphique	188	7	24179.96	1	2	1

In [12]:

```
#Create another column for RFM score
df_RFM['RFM_Score'] = df_RFM[['R', 'M', 'F']].sum(axis=1)
df_RFM.head()
```

Out[12]:

	Recency	Frequency	MonetaryValue	M	R	F	RFM_Score
CUSTOMERNAME							
AV Stores, Co.	196	51	157807.81	4	2	4	10
Alpha Cognac	65	20	70488.44	2	4	2	8
Amica Models & Co.	265	26	94117.26	3	1	2	6
Anna's Decorations, Ltd	84	46	153996.13	4	3	4	11
Atelier graphique	188	7	24179.96	1	2	1	4

In [13]:

```
def rfm_level(df):
    if bool(df['RFM_Score'] >= 10):
        return 'High Value Customer'

    elif bool(df['RFM_Score'] < 10) and bool(df['RFM_Score'] >= 6):
        return 'Mid Value Customer'
    else:
        return 'Low Value Customer'
df_RFM['RFM_Level'] = df_RFM.apply(rfm_level, axis = 1)
df_RFM.head()
```

Out[13]:

	Recency	Frequency	MonetaryValue	M	R	F	RFM_Score	RFM_Level
CUSTOMERNAME								
AV Stores, Co.	196	51	157807.81	4	2	4	10	High Value Customer
Alpha Cognac	65	20	70488.44	2	4	2	8	Mid Value Customer
Amica Models & Co.	265	26	94117.26	3	1	2	6	Mid Value Customer
Anna's Decorations, Ltd	84	46	153996.13	4	3	4	11	High Value Customer
Atelier graphique	188	7	24179.96	1	2	1	4	Low Value Customer

In [14]:
```
# Time to perform KMeans
data = df_RFM[['Recency', 'Frequency', 'MonetaryValue']]
data.head()
```

Out[14]:

	Recency	Frequency	MonetaryValue
CUSTOMERNAME			
AV Stores, Co.	196	51	157807.81
Alpha Cognac	65	20	70488.44
Amica Models & Co.	265	26	94117.26
Anna's Decorations, Ltd	84	46	153996.13
Atelier graphique	188	7	24179.96

In [15]:
```
# Our data is skewed we must remove it by performing log transformation
data_log = np.log(data)
data_log.head()
```

Out[15]:

	Recency	Frequency	MonetaryValue
CUSTOMERNAME			
AV Stores, Co.	5.278115	3.931826	11.969133
Alpha Cognac	4.174387	2.995732	11.163204
Amica Models & Co.	5.579730	3.258097	11.452297
Anna's Decorations, Ltd	4.430817	3.828641	11.944683
Atelier graphique	5.236442	1.945910	10.093279

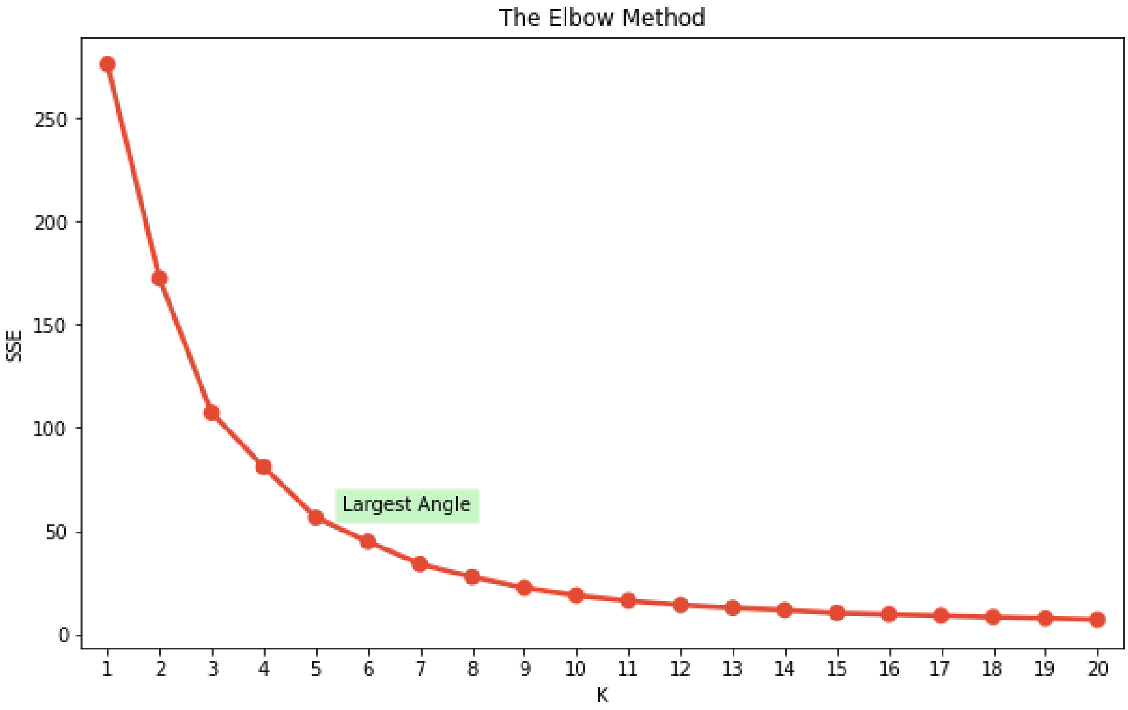
In [16]:
```
#Standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data_log)
data_normalized = scaler.transform(data_log)
data_normalized = pd.DataFrame(data_normalized, index = data_log.index,
                               columns=data_log.columns)
data_normalized.describe().round(2)
```

Out[16]:

	Recency	Frequency	MonetaryValue
count	92.00	92.00	92.00
mean	0.00	-0.00	0.00
std	1.01	1.01	1.01
min	-3.51	-3.67	-3.82
25%	-0.24	-0.41	-0.39
50%	0.37	0.06	-0.04
75%	0.53	0.45	0.52
max	1.12	4.03	3.92

In [17]:
```
#Fit KMeans and use elbow method to choose the number of clusters
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

# sum of squared errors
sse = {}

for k in range(1, 21):
    kmeans = KMeans(n_clusters = k, random_state = 1)
    kmeans.fit(data_normalized)
    sse[k] = kmeans.inertia_
```

In [18]:
```
plt.figure(figsize=(10,6))
plt.title('The Elbow Method')

plt.xlabel('K')
plt.ylabel('SSE')
plt.style.use('ggplot')

sns.pointplot(x=list(sse.keys()), y = list(sse.values()))
plt.text(4.5, 60, "Largest Angle", bbox = dict(facecolor = 'lightgreen',
                                               alpha = 0.5))
plt.show()
```



In [19]:

```
# 5 number of clusters seems good
kmeans = KMeans(n_clusters=5, random_state=1)
kmeans.fit(data_normalized)
cluster_labels = kmeans.labels_

data_rfm = data.assign(Cluster = cluster_labels)
data_rfm.head()
```

Out[19]:

	Recency	Frequency	MonetaryValue	Cluster
CUSTOMERNAME				
AV Stores, Co.	196	51	157807.81	0
Alpha Cognac	65	20	70488.44	4
Amica Models & Co.	265	26	94117.26	4
Anna's Decorations, Ltd	84	46	153996.13	0
Atelier graphique	188	7	24179.96	1

In [ ]:

```
```