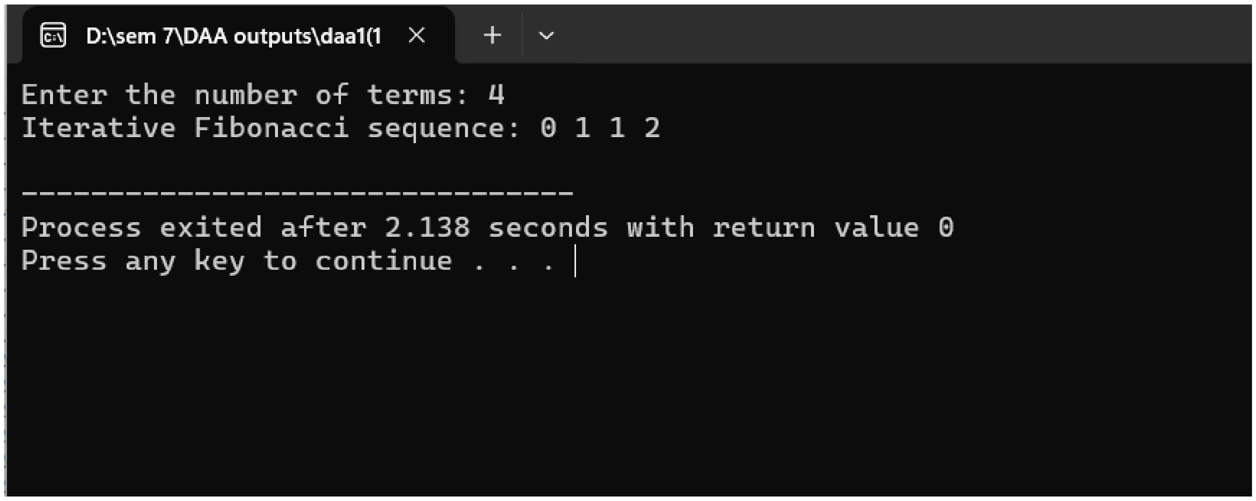
Fibonacci Sequence

(1)Non-recursive

Code:

```
#include <iostream>
using namespace std;
void fibonacciIterative(int n) {
    if (n < 1)
        return;
    int prev1 = 0, prev2 = 1;
    cout << prev1 << " ";
    if (n > 1) {
        cout << prev2 << " ";
    }
    for (int i = 2; i < n; i++) {
        int result = prev1 + prev2;
        cout << result << " ";
        prev1 = prev2;
        prev2 = result;
    }
    cout << endl;
}
int main() {
    int n;
    cout << "Enter the number of terms: ";
    cin >> n;
    cout << "Iterative Fibonacci sequence: ";
    fibonacciIterative(n);
    return 0;
}
```

Output:



```
Enter the number of terms: 4
Iterative Fibonacci sequence: 0 1 1 2

--------------------------------
Process exited after 2.138 seconds with return value 0
Press any key to continue . . .
```

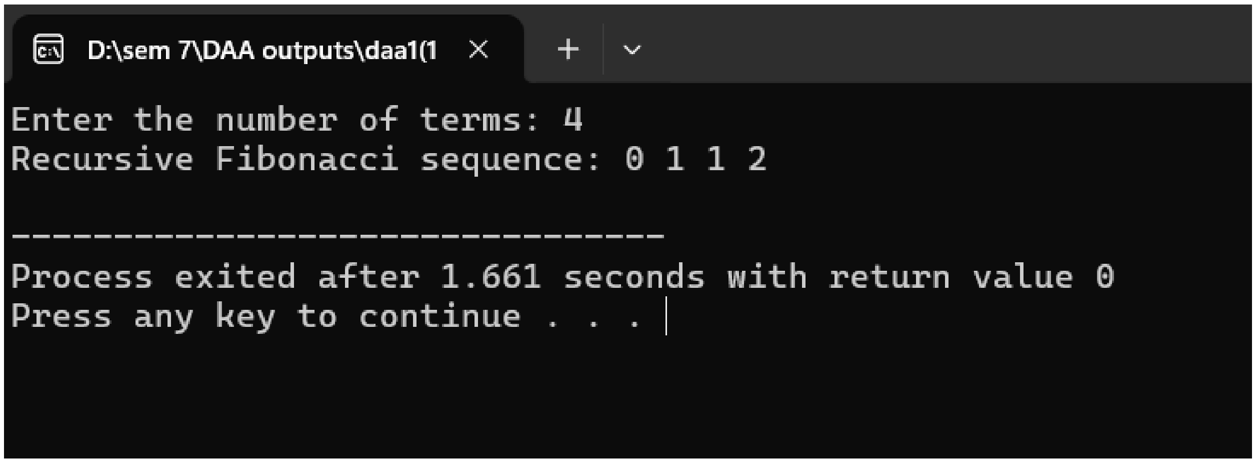
(2)Recursive

Code:

```
#include <iostream>

using namespace std;

void printFibonacciRecursive(int n, int a = 0, int b = 1) {

   if (n <= 0)

      return;

   cout << a << " ";

   printFibonacciRecursive(n - 1, b, a + b);

}

int main() {

   int n;

   cout << "Enter the number of terms: ";

   cin >> n;

   cout << "Recursive Fibonacci sequence: ";

   printFibonacciRecursive(n);

   cout << endl;
```

```
    return 0;
}
```

Output:



D:\sem 7\DAA outputs\daa1(1

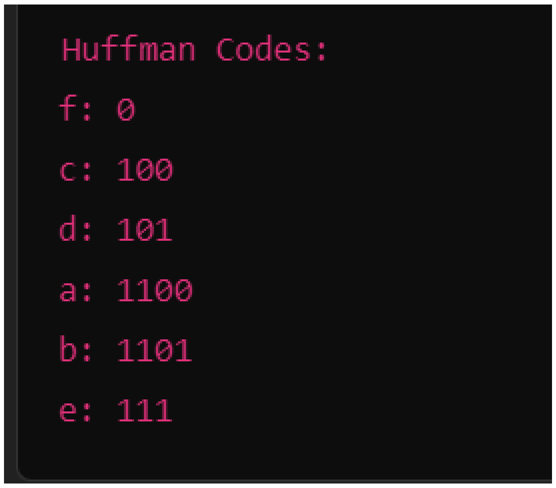
```
Enter the number of terms: 4
Recursive Fibonacci sequence: 0 1 1 2

--------------------------------
Process exited after 1.661 seconds with return value 0
Press any key to continue . . .
```

Huffman Tree

Code:

```
#include <iostream>
#include <queue>
#include <vector>
#include <unordered_map>
using namespace std;
struct HuffmanNode {
    char data;
    int freq;
    HuffmanNode *left, *right;
    HuffmanNode(char data, int freq) {
        left = right = nullptr;
        this->data = data;
        this->freq = freq;
    }
};
struct compare {
    bool operator()(HuffmanNode* left, HuffmanNode* right) {
        return left->freq > right->freq;
    }
};
void storeCodes(HuffmanNode* root, string code, unordered_map<char, string>& huffmanCodes) {
    if (root == nullptr)
        return;
    if (!root->left && !root->right) {
        huffmanCodes[root->data] = code;
    }
    storeCodes(root->left, code + "0", huffmanCodes);
    storeCodes(root->right, code + "1", huffmanCodes);
}
void huffmanEncoding(vector<char>& data, vector<int>& freq) {
    priority_queue<HuffmanNode*, vector<HuffmanNode*>, compare> minHeap;
    for (int i = 0; i < data.size(); i++) {
        minHeap.push(new HuffmanNode(data[i], freq[i]));
    }
    while (minHeap.size() != 1) {
        HuffmanNode* left = minHeap.top();
        minHeap.pop();
        HuffmanNode* right = minHeap.top();
        minHeap.pop();
        HuffmanNode* top = new HuffmanNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        minHeap.push(top);
    }
    unordered_map<char, string> huffmanCodes;
    storeCodes(minHeap.top(), "", huffmanCodes);
    cout << "Huffman Codes:\n";
    for (auto pair : huffmanCodes) {
        cout << pair.first << ": " << pair.second << endl;
    }
}
int main() {
    vector<char> data = { 'a', 'b', 'c', 'd', 'e', 'f' };
    vector<int> freq = { 5, 9, 12, 13, 16, 45 };
    huffmanEncoding(data, freq);
    return 0;
}
```

Output:



```
Huffman Codes:
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

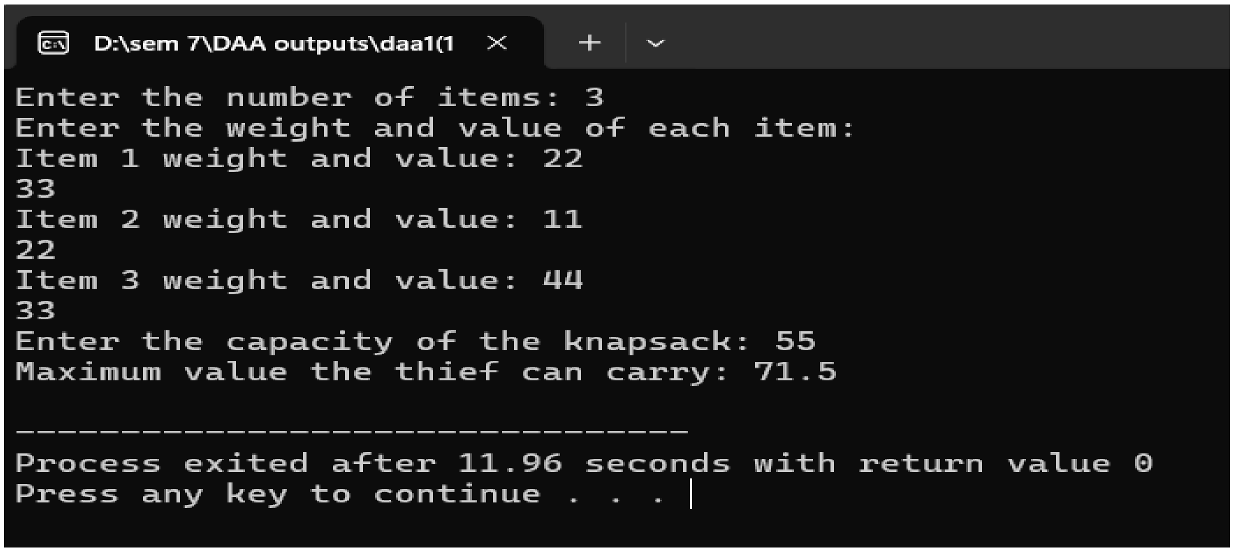
Fractional Knapsack Problem

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Item {
  int weight;
  int value;
};
bool compare(Item a, Item b) {
  double r1 = (double)a.value / a.weight;
  double r2 = (double)b.value / b.weight;
  return r1 > r2;
}
double fractionalKnapsack(int W, vector<Item>& items) {
  sort(items.begin(), items.end(), compare);
  int currentWeight = 0;   // Current weight in knapsack
  double totalValue = 0.0; // Total value in knapsack
  for (int i = 0; i < items.size(); i++) {
    if (currentWeight + items[i].weight <= W) {
      currentWeight += items[i].weight;
      totalValue += items[i].value;
    }
    else {
      int remainingCapacity = W - currentWeight;
      totalValue += items[i].value * ((double)remainingCapacity / items[i].weight);
      break; // No more items can be taken after this
    }
  }
```

```
    return totalValue;
}
int main() {
    int n, W;
    cout << "Enter the number of items: ";
    cin >> n;
    vector<Item> items(n);
    cout << "Enter the weight and value of each item:\n";
    for (int i = 0; i < n; i++) {
        cout << "Item " << i + 1 << " weight and value: ";
        cin >> items[i].weight >> items[i].value;
    }
    cout << "Enter the capacity of the knapsack: ";
    cin >> W;
    double maxValue = fractionalKnapsack(W, items);
    cout << "Maximum value the thief can carry: " << maxValue << endl;
    return 0;
}
```

Output:



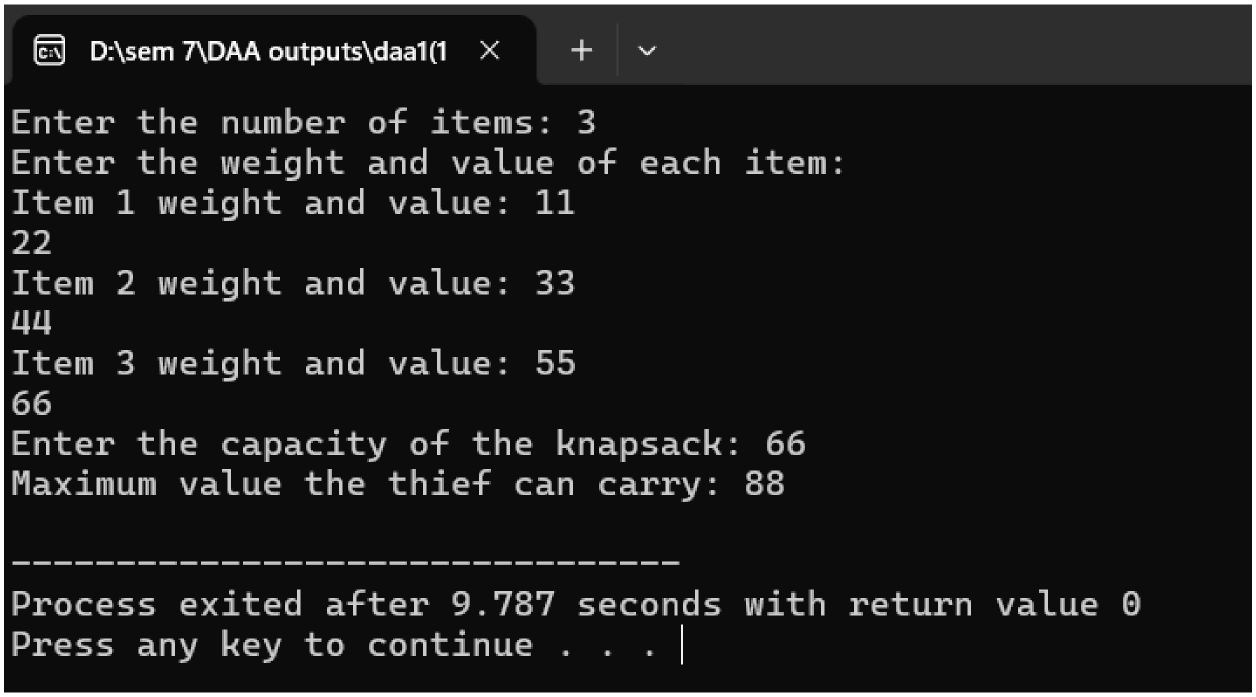
```
Enter the number of items: 3
Enter the weight and value of each item:
Item 1 weight and value: 22
33
Item 2 weight and value: 11
22
Item 3 weight and value: 44
33
Enter the capacity of the knapsack: 55
Maximum value the thief can carry: 71.5

--------------------------------
Process exited after 11.96 seconds with return value 0
Press any key to continue . . .
```

0/1 Knapsack using Dynamic Programming

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int knapsackDP(int W, vector<int>& weights, vector<int>& values, int n) {
    vector< vector<int> > dp(n + 1, vector<int>(W + 1, 0));
    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            if (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];
            }
        }
    }
    return dp[n][W];
}
int main() {
    int n, W;
    cout << "Enter the number of items: ";
    cin >> n;
    vector<int> weights(n);
    vector<int> values(n);
    cout << "Enter the weight and value of each item:\n";
    for (int i = 0; i < n; i++) {
        cout << "Item " << i + 1 << " weight and value: ";
        cin >> weights[i] >> values[i];
    }
    cout << "Enter the capacity of the knapsack: ";
    cin >> W;
    int maxValue = knapsackDP(W, weights, values, n);
    cout << "Maximum value the thief can carry: " << maxValue << endl;
    return 0;
}
```

Output:



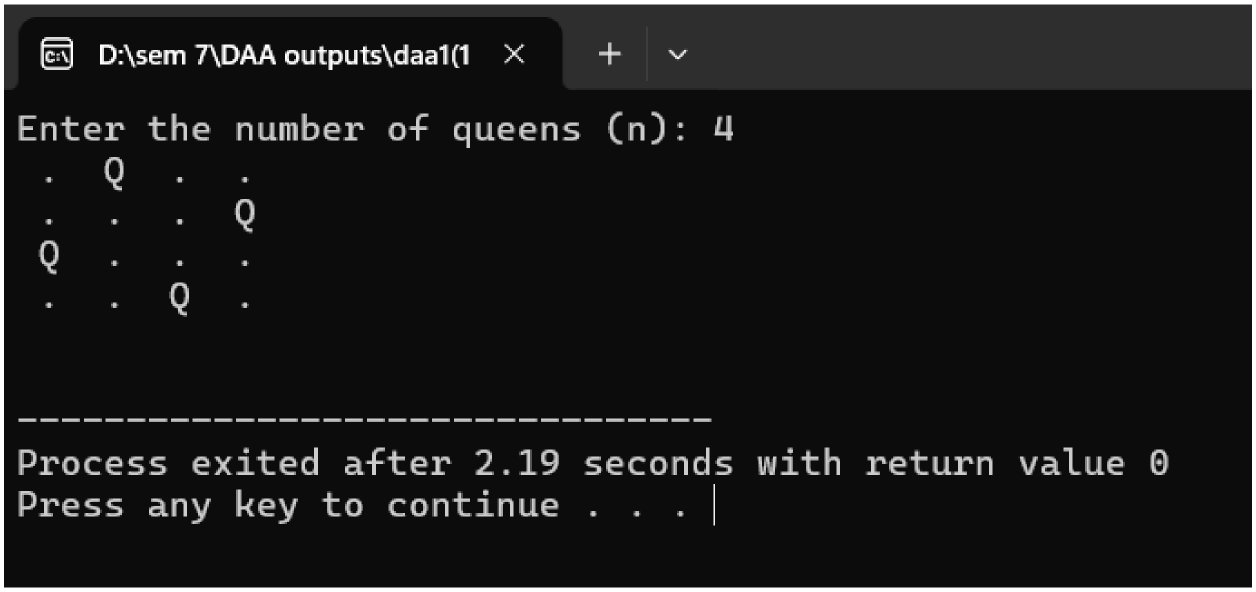
```
Enter the number of items: 3
Enter the weight and value of each item:
Item 1 weight and value: 11
22
Item 2 weight and value: 33
44
Item 3 weight and value: 55
66
Enter the capacity of the knapsack: 66
Maximum value the thief can carry: 88

--------------------------------
Process exited after 9.787 seconds with return value 0
Press any key to continue . . .
```

N-Queen Problem

Code:

```
#include <iostream>
#include <vector>
using namespace std;
void printBoard(const vector< vector<int> >& board) {
    for (int i = 0; i < board.size(); i++) {
        for (int j = 0; j < board[i].size(); j++) {
            cout << (board[i][j] ? " Q " : " . ");
        }
        cout << endl;
    }
    cout << endl;
}
bool isSafe(const vector< vector<int> >& board, int row, int col) {
    int n = board.size();
    for (int i = 0; i < row; i++) {
        if (board[i][col] == 1) return false;
    }
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j] == 1) return false;
    }
    for (int i = row, j = col; i >= 0 && j < n; i--, j++) {
        if (board[i][j] == 1) return false;
    }
    return true;
}
bool solveNQueens(vector< vector<int> >& board, int row) {
    int n = board.size();
    if (row == n) {
        return true;
    }
    for (int col = 0; col < n; col++) {
        if (isSafe(board, row, col)) {
            board[row][col] = 1;
            if (solveNQueens(board, row + 1)) {
                return true;
            }
            board[row][col] = 0;
        }
    }
    return false;
}
int main() {
    int n;
    cout << "Enter the number of queens (n): ";
    cin >> n;
    vector< vector<int> > board(n, vector<int>(n, 0));
    for (int firstCol = 0; firstCol < n; firstCol++) {
        board[0][firstCol] = 1;
        if (solveNQueens(board, 1)) {
            printBoard(board);
            break;
        }
        board[0][firstCol] = 0;
    }
    return 0;
}
```

Output:


D:\sem 7\DAA outputs\daa1(1

```
Enter the number of queens (n): 4
 .  Q  .  .
 .  .  .  Q
 Q  .  .  .
 .  .  Q  .



--------------------------------
Process exited after 2.19 seconds with return value 0
Press any key to continue . . .
```