**Lab Assignment 7**

**Q1.**

/\*

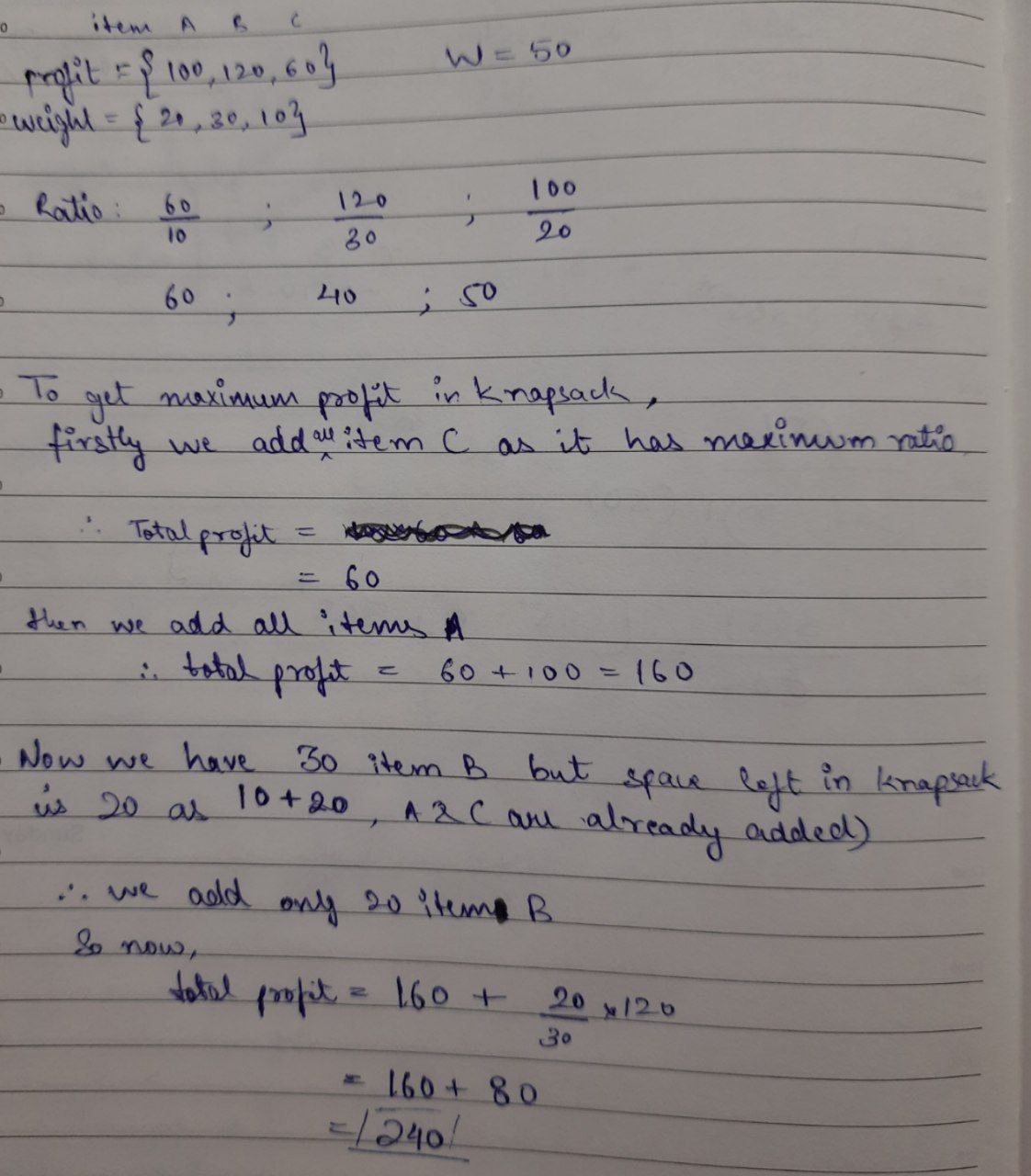
Roll Number: 102215255

Name: Diya Goyal

Description: Lab 7, question 1: (Fractional Knapsack) You are given arrays of weight and profit of N items. Select and put these items in a knapsack of capacity W to get the maximum total profit in the knapsack. Here the items can be broken down into smaller pieces. Example: Input: profit: {100,120,60} weight: {20,30,10} W = 50 Output: 240

We can select the third and first items with weights of 10 and 20 respectively. We can also select 2/3 of the second item. So total profit will be 60+100+(2/3) (120) = 240.

Acknowledgement: did with Mili \*/



**Q2.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Lab 7, question 2: Using Dynamic programming technique write a program:

a) To find the factorial of a number using dynamic programming.

b) Implement combination formula n C r using dynamic programming

c) Implement the Fibonacci series using dynamic programming.

Acknowledgement: did with Mili \*/

**A.**

#include <iostream>

#include <vector>

using namespace std;

unsigned long long fact(int n) {

vector<unsigned long long> dp(n + 1, 0);

dp[0] = 1;

for (int i = 1; i <= n; ++i) {

dp[i] = dp[i - 1] \* i;

}

return dp[n];

}

int main() {

int num;

cout << "Enter a number to find its factorial: ";

cin >> num;

if (num < 0) {

cout << "Factorial is not defined for negative numbers." << endl;

} else {

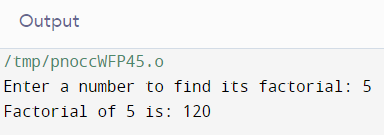
unsigned long long result = fact(num);

cout << "Factorial of " << num << " is: " << result << endl;

}

return 0;

}



**B.**

#include <iostream>

#include <vector>

using namespace std;

unsigned long long combination(int n, int r) {

vector<vector<unsigned long long>> dp(n + 1, vector<unsigned long long>(r + 1, 0));

for (int i = 0; i <= n; ++i) {

dp[i][0] = 1;

}

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= min(i, r); ++j) {

if (i == j || j == 0) {

dp[i][j] = 1;

} else {

dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j]; // Recurrence relation: C(n, r) = C(n-1, r-1) + C(n-1, r)

}

}

}

return dp[n][r];

}

int main() {

int n, r;

cout << "Enter the value of n: ";

cin >> n;

cout << "Enter the value of r: ";

cin >> r;

if (r < 0 || r > n) {

cout << "Invalid input: r should be between 0 and n." << endl;

}

else {

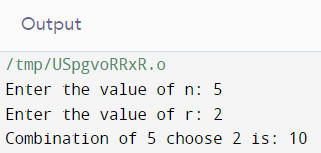
unsigned long long result = combination(n, r);

cout << "Combination of " << n << " choose " << r << " is: " << result << endl;

}

return 0;

}



**C.**

#include <iostream>

#include <vector>

using namespace std;

vector<unsigned long long> fibonacci(int n) {

vector<unsigned long long> fib(n + 1, 0);

fib[0] = 0;

if (n > 0) {

fib[1] = 1;

}

for (int i = 2; i <= n; ++i) {

fib[i] = fib[i - 1] + fib[i - 2];

}

return fib;

}

int main() {

int n;

cout << "Enter the number of terms in Fibonacci series: ";

cin >> n;

if (n < 0) {

cout << "Invalid input: Number of terms should be non-negative." << endl;

}

else {

vector<unsigned long long> fibSeries = fibonacci(n);

cout << "Fibonacci series up to " << n << " terms: ";

for (int i = 0; i <= n; ++i) {

cout << fibSeries[i] << " ";

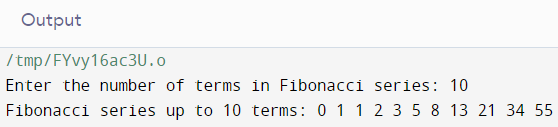
}

cout << endl;

}

return 0;

}



**Q3.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Lab 7, question 3: (0/1 Knapsack) You are given arrays of weight and profit of N items. Select and put these items in a knapsack of capacity W to get the maximum total profit in the knapsack. Here the items cannot be broken down into smaller pieces, i.e. either you can select an item in whole or will not select that item. Write a dynamic programming implementation for this problem.

Acknowledgement: did with Mili \*/

#include <iostream>

#include <vector>

using namespace std;

int knapSack(int W, vector<int> wt, vector<int> val, int n) {

vector<vector<int>> dp(n + 1, vector<int>(W + 1));

for (int i = 0; i <= n; i++) {

for (int w = 0; w <= W; w++) {

if (i == 0 || w == 0) {

dp[i][w] = 0;

}

else if (wt[i - 1] <= w) {

dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);

}

else {

dp[i][w] = dp[i - 1][w];

}

}

}

return dp[n][W];

}

int main() {

int W = 50;

vector<int> wt = {10, 20, 30};

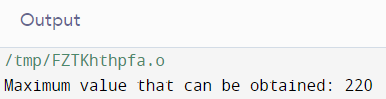
vector<int> val = {60, 100, 120};

int n = wt.size();

cout << "Maximum value that can be obtained: " << knapSack(W, wt, val, n) << endl;

return 0;

}



**Q4.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Lab 7, question 4: Given an integer array of coins of size N representing different types of denominations and an integer S. Write a program to find the minimum number of coins required to make a given value S. You can assume that you have an infinite supply of each type of coin. If it is not possible to make a change, print invalid.

Examples: Input: coins = {2,3,5}, S=10 Output: 2 {you can choose 5 two times}

Acknowledgement: did with Mili \*/

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int minCoins(vector<int> coins, int S) {

int n = coins.size();

vector<int> dp(S + 1, INT\_MAX);

dp[0] = 0;

for (int i = 1; i <= S; i++) {

for (int j = 0; j < n; j++) {

if (coins[j] <= i) {

int sub\_res = dp[i - coins[j]];

if (sub\_res != INT\_MAX && sub\_res + 1 < dp[i]) {

dp[i] = sub\_res + 1;

}

}

}

}

if (dp[S] == INT\_MAX) {

return -1;

} else {

return dp[S];

}

}

int main() {

vector<int> coins = {2, 3, 5};

int S = 10;

int result = minCoins(coins, S);

if (result == -1) {

cout << "Invalid" << endl;

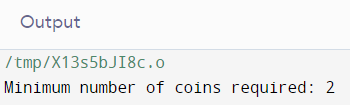
} else {

cout << "Minimum number of coins required: " << result << endl;

}

return 0;

}



**Q5.**

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Lab 7, question 5: (Subset Sum problem) Given a set of non-negative integers and a value sum, the task is to check if there is a subset of the given set whose sum is equal to the given sum. Note that more than one subset may exist, you just need to check if there is any such subset. For example: if a set is given S = {2,8,6,10,5,7} and sum=16, then you must return true because the required subset is {6,10}. Here, another set possible is {2,8,6}. When the value of sum=100, then there is no such subset and you will return false

Acknowledgement: did with Mili \*/

#include <iostream>

#include <vector>

using namespace std;

bool isSubsetSum(vector<int> set, int sum, int n) {

vector<vector<bool>> dp(n + 1, vector<bool>(sum + 1));

for (int i = 0; i <= n; i++) {

dp[i][0] = true;

}

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= sum; j++) {

if (set[i - 1] <= j) {

dp[i][j] = dp[i - 1][j - set[i - 1]] || dp[i - 1][j];

} else {

dp[i][j] = dp[i - 1][j];

}

}

}

return dp[n][sum];

}

int main() {

vector<int> set = {2, 8, 6, 10, 5, 7};

int sum1 = 16;

int sum2 = 35;

int n = set.size();

int m = set.size();

bool result1 = isSubsetSum(set, sum1, n);

bool result2 = isSubsetSum(set, sum2, m);

if (result1) {

cout << "True" << endl;

} else {

cout << "False" << endl;

}

if (result2) {

cout << "True" << endl;

} else {

cout << "False" << endl;

}

return 0;

}

