Pat -2

Rod Cutting:

My Fibanocci Term
MyFibonacci series is one which is defined as follows:

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

int fiba(int n, int dp[]){
    if(n==1)
        return 0;
    if(n==2)
        return 1;
    if(n==3)
        return 2;
    if(dp[n]!=-1){
        return dp[n];
    }
    dp[n]=fiba(n-1,dp) + fiba(n-2,dp) + fiba(n-3,dp);
    return dp[n];
}

int main(){
    int n;
    cin>>n;
    int dp[n+1];
    for(int i = 0; i <= n; i++) {
        dp[i] = -1;
    }
    int fib = fiba(n, dp);
    cout<<fib;
}
```

Print Rod Lengths
Modify the bottom up dynamic programming code written to solve rod cutting problem to print the size of the pieces that we should cut the rod to get maximum revenue

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

void rod_cut(int n, int prices[]) {
    int dp[n + 1] = {0};
    int S[n + 1] = {0};
    int i, j;

    for (i = 1; i <= n; i++) {
        for (j = 1; j <= i; j++) {
            if (dp[i] < prices[j - 1] + dp[i - j]) {
                dp[i] = prices[j - 1] + dp[i - j];
                S[i] = j;
            }
        }
    }

    int x = n;
    while (n > 0) {
        cout << S[n] << " ";
        n -= S[n];
    }
    cout << dp[x];
}
int main() {
    int n;
    cin >> n;
    int prices[n];
    for (int i = 0; i < n; i++) {
        cin >> prices[i];
    }

    rod_cut(n, prices);
}
```

Cut Rods of Preferred Lengths

A rod cutting company sterling corporation will cut rods of specific lengths only. Given a list of lengths by which this company will make rods, a rod of length ‘n’ and list of prices of each piece of rod that shall be cut by the company, write an algorithm and implement it using top down dynamic programming to find maximum revenue that shall be generated. In that case, give a penalty of Rs 1 for each meter of the residue.

```
#include <iostream>
```

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

int rod_cut(int n, int prices[], int len[], int x, int dp[]) {
    if (n == 0) return 0;
    if (dp[n] != -1) return dp[n];

    int maxrev = INT_MIN;
    for (int i = 0; i < x; i++) {
        int l = len[i];
        if (n >= l) {
            int rev = prices[l-1] + rod_cut(n-l, prices, len, x, dp);
            maxrev = max(maxrev, rev);
        }
    }

    if (n > 0) {
        maxrev = max(maxrev, -n);
    }

    dp[n] = maxrev;
    return dp[n];
}

int main() {
    int n;
    cin >> n;
    int prices[n];

    for (int i = 0; i < n; i++) {
        cin >> prices[i];
    }

    int x;
    cin >> x;
    int len[x];

    for (int i = 0; i < x; i++) {
        cin >> len[i];
    }

    int dp[n + 1];
    for (int i = 0; i <= n; i++) {
        dp[i] = -1;
```

```
    }

    int rc = rod_cut(n, prices, len, x, dp);
    cout << rc;

}
```

Cut into lengths 3 or 5
Cut into lengths 3 or 5

A variation of rod cutting problem is one in which length of the rod will be of length greater than or equal to 8 and you can make cuts of rods of length 3 or 5 only. For each unit of wastage a penalty of Rs.1 should be given. Write an recursive algorithm and implement it to find the maximum revenue that may be generated.

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

int rod_cut(int n, int p3, int p5, int dp[]) {
    if (n == 0)
        return 0;

    if (n < 0)
        return -n;

    if (dp[n] != -1)
        return dp[n];

    int maxrev = INT_MIN;

    if (n >= 3) {
        int rev3 = p3 + rod_cut(n-3, p3, p5, dp);
        maxrev = max(maxrev, rev3);
    }

    if (n >= 5) {
        int rev5 = p5 + rod_cut(n-5, p3, p5, dp);
        maxrev = max(maxrev, rev5);
    }

    if (n < 3) {
```

```
        maxrev = max(maxrev, -n);
    }

    dp[n] = maxrev;

    return dp[n];
}

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

    int p3, p5;
    cin >> p3 >> p5;

    int dp[n + 1];
    for (int i = 0; i <= n; i++) {
        dp[i] = -1;
    }

    int rc = rod_cut(n, p3, p5, dp);
    cout << rc;}
```

MCM:

Catalan

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

int catalan(int n) {
    if (n <= 1)
        return 1;
    int ans = 0;
    for (int i = 0; i < n; i++)
        ans += catalan(i) * catalan(n - i - 1);
    return ans;
}

int main() {
    int elements = 4;
   int v = catalan(elements - 1);
    cout << v << endl;
```

}

Matrix chain multiplication problem aims at finding the optimal way to parenthesise the matrix chain so that the number of multiplications (cost) will be minimum. The matrices are compatible for matrix multiplication so the number of columns in the ith matrix will be equal to number of rows in the(i+1)th matrix. Given dimension of ‘n’ matrices in the chain, write a recursive algorithm and code o find the minimum cost required to multiply the matrices in the chain. For example, if there are three matrices

```
#include <iostream>
#include <climits>
using namespace std;
int minop(int arr[],int i,int j){
    if(i==j)
    return 0;
    int s=INT_MAX;
    for(int k=i;k<j;k++){
        int c=minop(arr,i,k)+minop(arr,k+1,j)+(arr[i-1]*arr[k]*arr[j]);
        if(c<s)
        s=c;
    }
    return s;
}
int main(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    int v=minop(arr,1,n-1);
    cout<<v;
```

```
}
```

```
#include<iostream>
using namespace std;
#include<vector>
#include<limits.h>
void matrix_chain_order(vector<int> p, vector<vector<int> >& m,
vector<vector<int> >& s)
{
    int n,l,i,j,k,q;
    n = p.size()-1;
    // try length of chain from 2 to n
    for(l=2;l<=n;l++)
    {
        // From where does the chain starts is in 'i'
        //cout<<"l is "<<l<<endl;
        for(i=1;i<=n-l+1;i++)
        {
            // till what position does the chain goes is in 'j'
            j = i+l-1;
            //cout<<"i is "<<i<<" j is "<<j;
            m[i-1][j-1] = INT_MAX;
            for(k=i;k<=j-1;k++)
            {
                //cout<< " k is "<<k<<endl;
                q = m[i-1][k-1] + m[k][j-1] + p[i-1]*p[k]*p[j];
                if(q<m[i-1][j-1])
                {
                    m[i-1][j-1] = q;
                    s[i-1][j-1] = k;
```

```
            }
        }
    }
}
}
void print_Optimal_Parens(vector<vector<int> >& s, int i, int j)
{
    if(i==j)
        cout<<"A"<<i;
    else
    {
        cout<<"(";
        print_Optimal_Parens(s,i,s[i-1][j-1]);
        print_Optimal_Parens(s,s[i-1][j-1]+1,j);
        cout<<")";
    }
}
int main()
{
    int n,i,j;
    cin>>n;
    vector<int> p(n);
    for(i=0;i<n;i++)
        cin>>p[i];
    vector<vector<int> > m(n-1,vector<int>(n-1,0));
    vector<vector<int> > s(n-1,vector<int>(n-1,0));
    matrix_chain_order(p,m,s);
    cout<<endl;
    for(i=0;i<n-2;i++)
    {
        for(j=1;j<n-1;j++)
            cout<<m[i][j]<<" ";
        cout<<endl;
    }
    for(i=0;i<n-2;i++)
    {
        for(j=0;j<n-1;j++)
            cout<<s[i][j]<<" ";
        cout<<endl;
    }

    cout<<m[0][n-2]<<endl;
    print_Optimal_Parens(s,1,n-1);
}
```

```
#include<iostream>
#include<iomanip>
#include<climits>
using namespace std;
void printTable(int** table, int n) {
    for (int i = 1; i < n; i++) {
        for (int j = i; j < n; j++) {
            if (table[i][j] != 0) {
                cout<<table[i][j]<< " ";
            }
        }
        if(i!=n-1)
        cout << endl;
    }
}
void matrixChainOrder(int* arr, int n) {
    int** M = new int*[n];
    int** S = new int*[n];
    for (int i = 0; i < n; i++) {
        M[i] = new int[n];
        S[i] = new int[n];
    }
    for (int i = 1; i < n; i++) {
        M[i][i] = 0;
    }
    for (int l = 2; l < n; l++) {
        for (int i = 1; i < n - l + 1; i++) {
            int j = i + l - 1;
            M[i][j] = INT_MAX;
            for (int k = i; k < j; k++) {
                int q = M[i][k] + M[k + 1][j] + arr[i - 1] * arr[k] * arr[j];
                if (q < M[i][j]) {
                    M[i][j] = q;
                    S[i][j] = k;
                }
            }
        }
    }
    printTable(M, n);
    printTable(S, n);
    cout << M[1][n - 1];

    for (int i = 0; i < n; i++) {
        delete[] M[i];
```

```
        delete[] S[i];
    }
    delete[] M;
    delete[] S;
}
int main() {
    int n;
    cin >> n;

    int* arr = new int[n];
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    matrixChainOrder(arr, n);
    delete[] arr;
    return 0;
}

Vector#include<iostream>
#include<vector>
#include<climits>
using namespace std;
void printTable(const vector<vector<int>>& table, int n) {
    for (int i = 1; i < n; i++) {
        for (int j = i; j < n; j++) {
            if (table[i][j] != 0) {
                cout << table[i][j] << " ";
            }
        }
        if (i != n - 1)
            cout << endl;
    }
}
void matrixChainOrder(const vector<int>& arr, int n) {
    vector<vector<int>> M(n, vector<int>(n, 0));
    vector<vector<int>> S(n, vector<int>(n, 0));

    for (int i = 1; i < n; i++) {
        M[i][i] = 0;  // No cost for multiplying a single matrix
    }

    for (int l = 2; l < n; l++) {  // l is the chain length
        for (int i = 1; i < n - l + 1; i++) {
            int j = i + l - 1;
```

```
            M[i][j] = INT_MAX;
            for (int k = i; k < j; k++) {
                int q = M[i][k] + M[k + 1][j] + arr[i - 1] * arr[k] * arr[j];
                if (q < M[i][j]) {
                    M[i][j] = q;
                    S[i][j] = k;
                }
            }
        }
    }
    printTable(M, n);  // Print M table
    printTable(S, n);  // Print S table
    cout << M[1][n - 1] << endl;  // Print minimum cost
}
int main() {
    int n;
    cin >> n;

    vector<int> arr(n);
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    matrixChainOrder(arr, n);

    return 0;
}
```

parenthisisation

```
vector<int> diffWaysToCompute(string expression) {
        vector<int> g1;
        for(int i=0;i<expression.length();i++){
            char a=expression[i];
            if(a=='*'||a=='-'||a=='+'||a=='/'){

vector<int>l=diffWaysToCompute(expression.substr(0,i));
```

```
vector<int>r=diffWaysToCompute(expression.substr(i+1));
                for(int k:l){
                    for(int j:r){
                        if(a=='*'){
                            g1.push_back(k*j);
                        }
                        else if(a=='-'){
                            g1.push_back(k-j);
                        }
                        else if(a=='+'){
                            g1.push_back(k+j);
                        }
                        else {
                            g1.push_back(k/j);
                        }
                    }
                }
            }
        }
        if(g1.empty()){
            g1.push_back(stoi(expression));
        }
        return g1;
    }
```

Lcs:#include<iostream>
#include<climits>

```
using namespace std;
string s="";
int lcs(string s1,string s2,int i,int j){
   int m=0;
   if(i>=s1.length()||j>=s2.length())
   return 0;
   else if(s1[i]==s2[j]){
      s+=s1[i];
      return 1+lcs(s1,s2,i+1,j+1);

   }
   else{
      m=max(lcs(s1,s2,i+1,j),lcs(s1,s2,i,j+1));
```

```
        return m;
    }
}
int main(){
    string s1,s2;
    cin>>s1>>s2;
    int y=lcs(s1,s2,0,0);
    cout<<y;
    cout<<s;
}
```

```
Recursion#include<iostream>
#include<climits>
using namespace std;
string s="";
int lcs(string s1,string s2,int i,int j,string & s){
    int m=0;
    if(i>=s1.length()||j>=s2.length())
    return 0;
    else if(s1[i]==s2[j]){
        s+=s1[i];
        return 1+lcs(s1,s2,i+1,j+1,s);

    }
    else{
        string l=s,r=s;
        int ls=lcs(s1,s2,i+1,j,l);
        int rs=lcs(s1,s2,i,j+1,r);
        if(ls>rs){
            s=l;
            return ls;
        }
        else{
            s=r;
            return rs;

        }

    }
}
int main(){
    string s1,s2;
    cin>>s1>>s2;
    int y=lcs(s1,s2,0,0,s);
    cout<<y<<endl;
    cout<<s;
}
```

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

void lcs_length(string x, string y, vector<vector<int>>& c, vector<vector<char>>& b) {
    int m = x.length();
    int n = y.length();

    // Filling the LCS length and direction table
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (x[i-1] == y[j-1]) {
                c[i][j] = c[i-1][j-1] + 1;
                b[i][j] = 'd';  // Diagonal, match found
            }
            else if (c[i-1][j] >= c[i][j-1]) {
                c[i][j] = c[i-1][j];
                b[i][j] = 'u';  // Coming from above
            }
            else {
                c[i][j] = c[i][j-1];
                b[i][j] = 'l';  // Coming from the left
            }
        }
    }
}

// Function to print the LCS by tracing the `b` table
void print_LCS(vector<vector<char>>& b, string x, int i, int j) {
    if (i == 0 || j == 0)
        return;
    if (b[i][j] == 'd') {
        print_LCS(b, x, i-1, j-1);  // Move diagonally
        cout << x[i-1];  // Output character
    }
    else if (b[i][j] == 'u') {
        print_LCS(b, x, i-1, j);  // Move up
    }
    else {
        print_LCS(b, x, i, j-1);  // Move left
    }
}

int main() {
    string x, y;
    cin >> x >> y;

    // Create tables
    vector<vector<int>> c(x.length() + 1, vector<int>(y.length() + 1, 0));
    vector<vector<char>> b(x.length() + 1, vector<char>(y.length() + 1, ' '));
```

```
    // Compute LCS lengths and directions
    lcs_length(x, y, c, b);

    // Output LCS length
    cout << "LCS Length: " << c[x.length()][y.length()] << endl;

    // Output LCS
    cout << "LCS: ";
    print_LCS(b, x, x.length(), y.length());
    cout << endl;

    return 0;
}
```

LCS LPS:

```
#include<iostream>
#include<climits>
#include <vector>
using namespace std;
int lcs(string s1,string s2){
    vector<int> pr(s2.length()+1,0);
    vector<int> cn(s2.length()+1,0);
    for(int i=1;i<=s1.length();i++){
        for(int j=1;j<=s2.length();j++){
            if(s1[i-1]==s2[j-1]){
                cn[j]=pr[j-1]+1;
            }
            else{
                cn[j]=max(pr[j],cn[j-1]);
            }
        }
        pr=cn;
    }
    return pr[s2.length()];
}
int main(){
    string s1,s2;
    cin>>s1>>s2;
    int p=lcs(s1,s2);
    cout<<p;
}
```

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;
void lcs(string s1, string s2, vector<vector<int>>& c, vector<vector<char>>& b) {
```

```
    int n = s1.length();
    int m = s2.length();
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s1[i - 1] == s2[j - 1]) {
                c[i][j] = c[i - 1][j - 1] + 1;
                b[i][j] = 'd';
            } else if (c[i - 1][j] > c[i][j - 1]) {
                c[i][j] = c[i - 1][j];
                b[i][j] = 'u';
            } else if (c[i - 1][j] < c[i][j - 1]) {
                c[i][j] = c[i][j - 1];
                b[i][j] = 'l';
            } else {
                c[i][j] = c[i - 1][j];
                b[i][j] = 'b';
            }
        }
    }
}
void alllcs(string s1, string s2, vector<vector<char>>& b, int i, int j, string currentLCS,
set<string>& al) {
    if (i == 0 || j == 0) {
        al.insert(currentLCS);
        return;
    }
    if (b[i][j] == 'd') {
        alllcs(s1, s2, b, i - 1, j - 1, s1[i - 1] + currentLCS, al);
    }
    if (b[i][j] == 'u' || b[i][j] == 'b') {
        alllcs(s1, s2, b, i - 1, j, currentLCS, al);
    }
    if (b[i][j] == 'l' || b[i][j] == 'b') {
        alllcs(s1, s2, b, i, j - 1, currentLCS, al);
    }
}
int main() {
    string s1, s2;
    cin >> s1 >> s2;
    vector<vector<int>> c(s1.length() + 1, vector<int>(s2.length() + 1, 0));
    vector<vector<char>> b(s1.length() + 1, vector<char>(s2.length() + 1, ' '));
    lcs(s1, s2, c, b);
    set<string> al;
    alllcs(s1, s2, b, s1.length(), s2.length(), "", al);
    for (const string& lcs : al) {
        cout << lcs << endl;
    }}
```

Activity Selection:

Recursion vector#include <iostream>

```
#include <vector>
using namespace std;
void ssort(vector<vector<int>>&act,int n){
    for(int i=1;i<n;i++){
        int key=act[1][i];
        int j=i-1;
        while(j>=0&&act[1][j]>key){
            act[1][j+1]=act[1][j];
            j--;
        }
        act[1][j+1]=key;
    }
}
void actv(vector<vector<int>>&act,int i,int n){
    int m=i+1;
    while(m<n&&act[0][m]<act[1][i]){
        m++;
    }
    if(m<n){
        cout<<m<<" ";
        actv(act,m,n);
    }
}
int main(){
    int n;
    cin>>n;
    vector<vector<int>> act(2,vector<int>(n,0));
    for(int i=0;i<2;i++){
        for(int j=0;j<n;j++){
            cin>>act[i][j];
        }
    }
    ssort(act,n);
    cout<<0<<" ";
    actv(act,0,n);
}
```

Greedy#include <iostream>

```
#include <vector>
using namespace std;
void ssort(vector<vector<int>>&act,int n){
    for(int i=1;i<n;i++){
        int key=act[1][i];
        int j=i-1;
        while(j>=0&&act[1][j]>key){
            act[1][j+1]=act[1][j];
            j--;
        }
        act[1][j+1]=key;
```

```
    }
}
void actv(vector<vector<int>>&act,int n){
    int i=0;
    cout<<i<<" ";
    for(int k=1;k<n;k++){
        if(act[0][k]>=act[1][i]){
            cout<<k<<" ";
            i=k;
        }
    }
    cout<<endl;
}
int main(){
    int n;
    cin>>n;
    vector<vector<int>> act(2,vector<int>(n,0));
    for(int i=0;i<2;i++){
        for(int j=0;j<n;j++){
            cin>>act[i][j];
        }
    }
    ssort(act,n);
    actv(act,n);
```

}all combinations of max length

```
#include <iostream>
#include <vector>
using namespace std;
void ssort(vector<vector<int>>&act,int n){
    for(int i=1;i<n;i++){
        int key=act[1][i];
        int j=i-1;
        while(j>=0&&act[1][j]>key){
            act[1][j+1]=act[1][j];
            j--;
        }
        act[1][j+1]=key;
    }
}
void findMaxActivities(vector<vector<int>>& act, int n, int index, vector<int>& current,
vector<vector<int>>& results, int& maxLength) {
    current.push_back(index);
    for (int nextIndex = index + 1; nextIndex < n; nextIndex++) {
        if (act[0][nextIndex] >= act[1][index]) {
            findMaxActivities(act, n, nextIndex, current, results, maxLength);
        }
    }
    if (current.size() > maxLength) {
        maxLength = current.size();
        results.clear();
        results.push_back(current);
```

```
    } else if (current.size() == maxLength) {
        results.push_back(current);
    }
    current.pop_back();
}

void printMaxCombinations(vector<vector<int>>& act, int n) {
    vector<vector<int>> results;
    vector<int> current;
    int maxLength = 0;
    for (int i = 0; i < n; i++) {
        findMaxActivities(act, n, i, current, results, maxLength);
    }
    for (const auto& combination : results) {
        for (int activity : combination) {
            cout << activity << " ";
        }
        cout << endl;
    }
}

int main() {
    int n;
    cin >> n;
    vector<vector<int>> act(2, vector<int>(n, 0));
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < n; j++) {
            cin >> act[i][j];
        }
    }
    ssort(act, n);
    printMaxCombinations(act, n);
    return 0;
}
```

Maximum comparable activities

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Activity {
    int start;
    int finish;
    int value;
};

bool compare(Activity a, Activity b) {
    return a.finish < b.finish;
}
```

```
int findLastCompatibleActivity(vector<Activity>& activities, int index) {
    for (int j = index - 1; j >= 0; j--) {
        if (activities[j].finish <= activities[index].start||activities[index].finish <= activities[j].start) {
            return j;
        }
    }
    return -1; // No compatible activity found
}

int maxValueActivities(vector<Activity>& activities) {
    int n = activities.size();
    sort(activities.begin(), activities.end(), compare);

    vector<int> dp(n);
    dp[0] = activities[0].value;

    for (int i = 1; i < n; i++) {
        int inclValue = activities[i].value;
        int lastIndex = findLastCompatibleActivity(activities, i);
        if (lastIndex != -1) {
            inclValue += dp[lastIndex];
        }
        dp[i] = max(inclValue, dp[i - 1]);
    }

    return dp[n - 1];
}

int main() {
    int n;
    cout << "Enter number of activities: ";
    cin >> n;
    vector<Activity> activities(n);

    cout << "Enter start time, finish time, and value for each activity:\n";
    for (int i = 0; i < n; i++) {
        cin >> activities[i].start >> activities[i].finish >> activities[i].value;
    }

    int maxValue = maxValueActivities(activities);
    cout << "Maximum value of compatible activities: " << maxValue << endl;
    return 0;
}
```

Activity hall problem

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
using namespace std;
```

```
struct Activity {
    int start;
    int finish;
};

struct Hall {
    int end;
    int hallNumber;
};

bool compareActivities(Activity a, Activity b) {
    return a.finish < b.finish;
}

void extendedActivitySelection(vector<Activity>& activities) {
    int n = activities.size();
    sort(activities.begin(), activities.end(), compareActivities);

    priority_queue<Hall, vector<Hall>, greater<Hall>> halls; // Min-heap based on finish time
    vector<int> hallAssignments(n, -1); // To store which hall each activity is assigned to

    for (int i = 0; i < n; i++) {
        // Check if the earliest finishing hall is free
        if (!halls.empty() && halls.top().end <= activities[i].start) {
            Hall hall = halls.top();
            halls.pop(); // Remove the hall as it's now being used
            hallAssignments[i] = hall.hallNumber; // Assign this hall to the current activity
            hall.end = activities[i].finish; // Update finish time of this hall
            halls.push(hall); // Add it back with updated finish time
        } else {
            // Assign a new hall
            Hall newHall;
            newHall.end = activities[i].finish;
            newHall.hallNumber = halls.size(); // Use the size of the heap to assign hall numbers
            hallAssignments[i] = newHall.hallNumber; // Assign this new hall to the current
activity
            halls.push(newHall); // Add this new hall to the heap
        }
    }

    // Output the assignments
    for (int i = 0; i < n; i++) {
        cout << "Activity " << i << " assigned to Hall " << hallAssignments[i] << endl;
    }
}

int main() {
    int n;
    cout << "Enter number of activities: ";
    cin >> n;
    vector<Activity> activities(n);

    cout << "Enter start and finish times for each activity (s f):" << endl;
```

```
    for (int i = 0; i < n; i++) {
        cin >> activities[i].start >> activities[i].finish;
    }

    extendedActivitySelection(activities);
    return 0;
}
```

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

struct Activity {
    int start;
    int finish;
};

bool compareActivities(Activity a, Activity b) {
    return a.finish < b.finish;
}

void extendedActivitySelection(vector<Activity>& activities) {
    sort(activities.begin(), activities.end(), compareActivities);

    vector<int> hallEndTimes; // To keep track of end times for each hall
    vector<int> hallAssignments(activities.size(), -1); // To store hall assignments

    for (int i = 0; i < activities.size(); i++) {
        int j;
        // Check for available hall
        for (j = 0; j < hallEndTimes.size(); j++) {
            if (activities[i].start >= hallEndTimes[j]) {
                hallAssignments[i] = j; // Assign this activity to hall j
                hallEndTimes[j] = activities[i].finish; // Update the end time of this hall
                break;
            }
        }
        // If no hall is available, add a new one
        if (j == hallEndTimes.size()) {
            hallAssignments[i] = hallEndTimes.size(); // Assign new hall
            hallEndTimes.push_back(activities[i].finish); // Add new hall end time
        }
    }

    // Output the assignments
    for (int i = 0; i < activities.size(); i++) {
        cout << "Activity " << i << " assigned to Hall " << hallAssignments[i] << endl;
    }
}

int main() {
    int n;
    cout << "Enter number of activities: ";
    cin >> n;
    vector<Activity> activities(n);

    cout << "Enter start and finish times for each activity (s f):" << endl;
    for (int i = 0; i < n; i++) {
        cin >> activities[i].start >> activities[i].finish;
    }

    extendedActivitySelection(activities);
    return 0;
}
```

```
dp#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Activity {
    int start;
    int finish;
};

bool compareActivities(Activity a, Activity b) {
```

```
    return a.finish < b.finish;
}

int latestNonConflict(Activity arr[], int i) {
    for (int j = i - 1; j >= 0; j--) {
        if (arr[j].finish <= arr[i].start) {
            return j;
        }
    }
    return -1;
}

int dpActivitySelection(Activity arr[], int n) {
    sort(arr, arr + n, compareActivities);

    vector<int> dp(n);
    dp[0] = 1; // Base case: only one activity can be selected

    for (int i = 1; i < n; i++) {
        int incl = 1; // Count this activity
        int l = latestNonConflict(arr, i); // Last non-conflicting activity
        if (l != -1) {
            incl += dp[l];
        }
        dp[i] = max(incl, dp[i - 1]);
    }

    return dp[n - 1];
}

int main() {
    int n;
    cout << "Enter number of activities: ";
    cin >> n;
    Activity arr[n];

    cout << "Enter start and finish times for each activity (s f):" << endl;
    for (int i = 0; i < n; i++) {
        cin >> arr[i].start >> arr[i].finish;
    }

    int maxActivities = dpActivitySelection(arr, n);
    cout << "Maximum number of compatible activities: " << maxActivities << endl;
    return 0;
}


Knapsack#include <iostream>
#include <vector>
using namespace std;

void insertionSort(vector<vector<int>>& items, int n) {
    for (int i = 1; i < n; i++) {
```

```
        int keyValue = items[0][i];
        int keyWeight = items[1][i];
        int j = i - 1;

        while (j >= 0 && (double)items[0][j] / items[1][j] < (double)keyValue / keyWeight) {
            items[0][j + 1] = items[0][j]; // value
            items[1][j + 1] = items[1][j]; // weight
            j--;
        }
        items[0][j + 1] = keyValue;
        items[1][j + 1] = keyWeight;
    }
}

int knapsack(vector<vector<int>>& items, int W, int n) {
    int totalValue = 0;
    for (int i = 0; i < n; i++) {
        if (items[1][i] <= W) {
            W -= items[1][i];
            totalValue += items[0][i];
        }
    }
    return totalValue;
}

int main() {
    int n, W;
    cout << "Enter the number of items: ";
    cin >> n;
    cout << "Enter the maximum weight of the knapsack: ";
    cin >> W;

    vector<vector<int>> items(2, vector<int>(n, 0));
    cout << "Enter value and weight for each item:\n";
    for (int i = 0; i < n; i++) {
        cin >> items[0][i] >> items[1][i]; // items[0] for values, items[1] for weights
    }

    insertionSort(items, n);

    int maxValue = knapsack(items, W, n);

    cout << "Maximum value in Knapsack = " << maxValue << endl;

    return 0;
}
```

Code audi book

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Event {
  string name;
  int start;
  int end;
  int participants;
};


bool compare(const Event& e1, const Event& e2) {
  return e1.end < e2.end;
}

int main() {
  int n;
  cin >> n;
  vector<Event> events(n);

  for (int i = 0; i < n; ++i) {
    cin >> events[i].name >> events[i].start >> events[i].end >> events[i].participants;
  }


  sort(events.begin(), events.end(), compare);

  vector<string> result;
  int last_end = 0;

  for (int i = 0; i < n; ++i) {
    int vacate_time = (events[i].participants + 9) / 10;
    if (last_end <= events[i].start) {
      result.push_back(events[i].name);
      last_end = events[i].end + vacate_time;
    }
  }

  for (const string &name : result) {
    cout << name << " ";
  }
```

```
    return 0;
}
```

Rod cutting

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int calculatePrice(int x) {
    int sum = 0;
    while (x > 0) {
        sum += x % 10;
        x /= 10;
    }
    return (sum % 2 == 0) ? sum : sum + 1;
}

int maxRevenue(int n, int k) {
    vector<vector<int>> dp(k + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= k; ++i) {
        for (int j = 1; j <= n; ++j) {
            for (int x = 1; x <= j; ++x) {
                dp[i][j] = max(dp[i][j], dp[i - 1][j - x] + calculatePrice(x));
            }
        }
    }
    return dp[k][n];
}

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

    if (k == 1) {
        cout << calculatePrice(n) << endl;
    } else {
        cout << maxRevenue(n, k) << endl;
    }

    return 0;
}
```