Report of The Project:→

Seminar

Sardar Vallabhbhai National Institute Of Technology, Surat

Department of Computer Science and Engineering

Subject:

Design and Analysis of Algorithm

Problem :

“Maximum profit in share market”

Submitted In Group Of 4 Members:

Joshi Prashant Manubhai(U19CS005)

Amin Saurabhkumar Rajeshbhai(U19CS007)

Vaghela Jaykumar harjibhai(U19CS013)

Sandeep Rathod(U19CS022)

EXPLANATION OF PROBLEM:→

We are given stock prices of certain days and we have to maximize the profit by doing at most ‘k’ transaction

Condition: Transaction cannot have in parallel one transaction has to end before other starts

Example:

Array of prize is given below of 6 days and at most ‘k’ transaction is given and output will be maximum profit of k transaction.

Input:

Price = {10,22,5,75,65,80}

k = 2

Output: 87

Explanation: We buy at price 10 and sell at 22 so profit would be 12 .(1st transaction)

again we buy at price 5 and sell at 80 so profit would be 75.(2nd transaction)

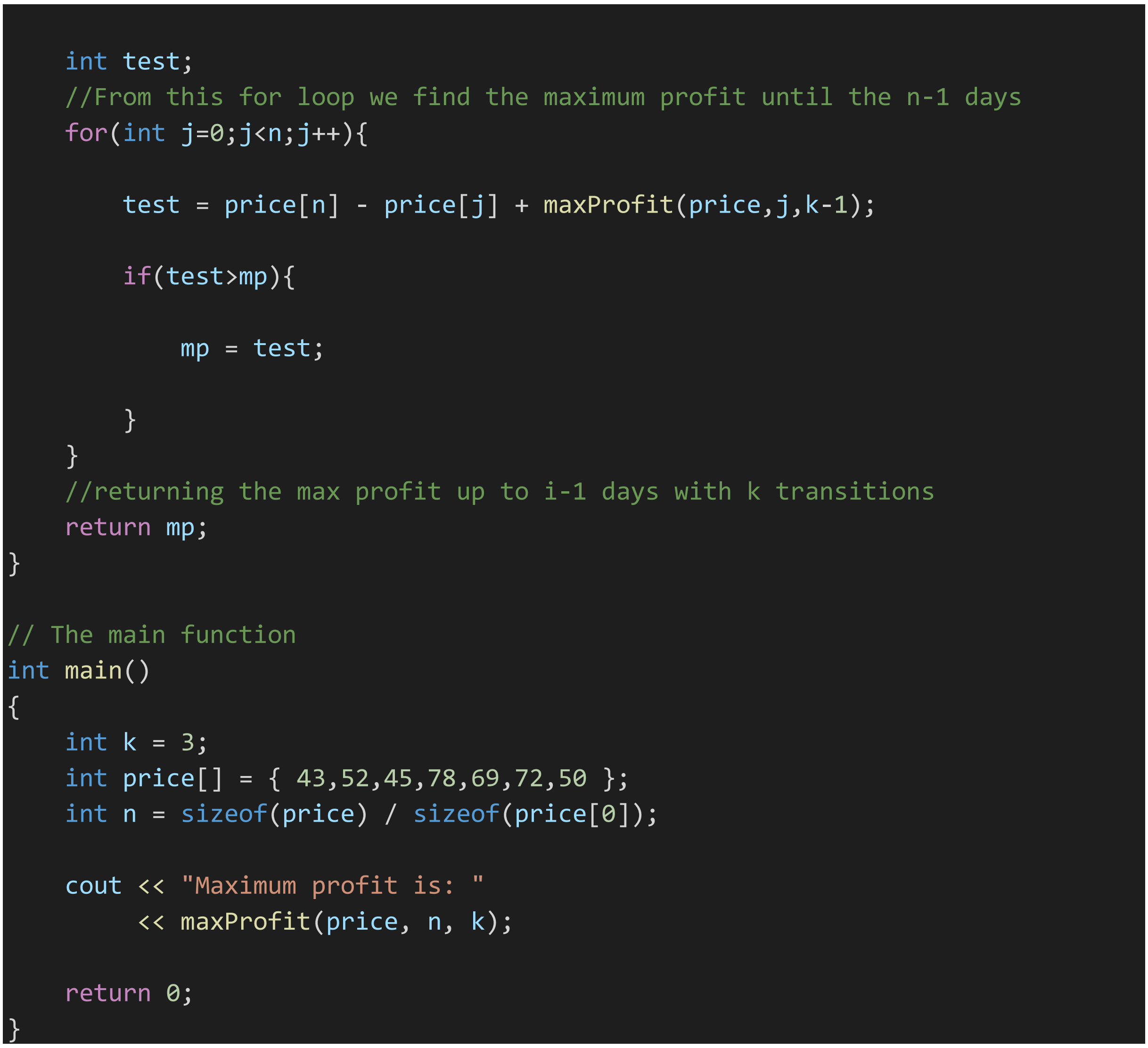
total profit = 75 + 12 = 87

NAIEVE RECURSIVE SOLUTION:→

(PSUEDOCODE AND FORMULA ARE EXPLAINED IN PPT)

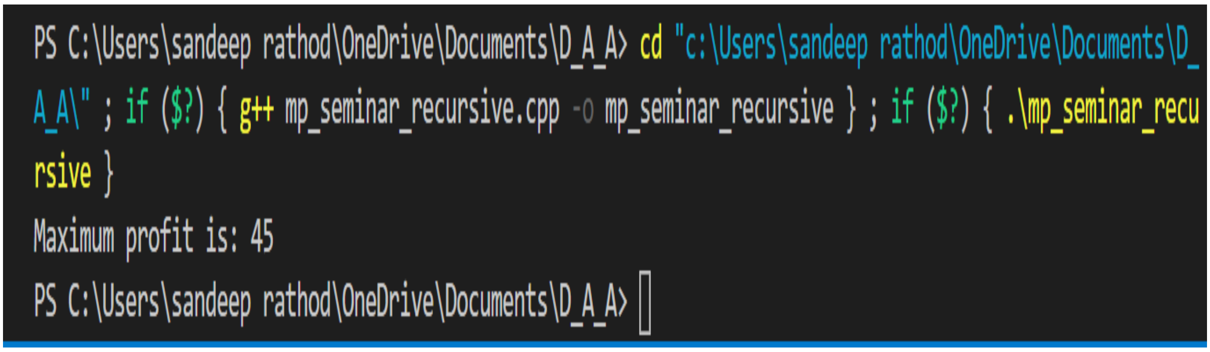
CODE:

```
// C++ program to find out maximum profit by
// buying and selling a share atmost k times
// given stock price of n days
#include <climits>
#include <iostream>
using namespace std;

//fuction to find maximum of two numbers
int max(int a,int b){
    if(a>b){
        return a;
    }else return b;
}

int mp_until(int price[],int n,int k);


// Function to find out maximum profit by buying
// & selling a share atmost k times given stock
// price of n days
int maxProfit(int price[], int n, int k)
{
    int mp;

    if(n==0||k==0){

        return 0;//terminate condition for the recursive calls

    }else{

       mp = mp_until(price,n-1,k);//calling mp_until function

       return max(maxProfit(price,n-1,k),mp);
    }
}

//function to find the maximum profit until the n-1 days
//from the k transitions
int mp_until(int price[],int n,int k){

    int mp=-10000;
```



```
    int test;
    //From this for loop we find the maximum profit until the n-1 days
    for(int j=0;j<n;j++){

        test = price[n] - price[j] + maxProfit(price,j,k-1);

        if(test>mp){

            mp = test;

        }
    }
    //returning the max profit up to i-1 days with k transitions
    return mp;
}

// The main function
int main()
{
    int k = 3;
    int price[] = { 43,52,45,78,69,72,50 };
    int n = sizeof(price) / sizeof(price[0]);

    cout << "Maximum profit is: "
         << maxProfit(price, n, k);

    return 0;
}
```

OUTPUT:→

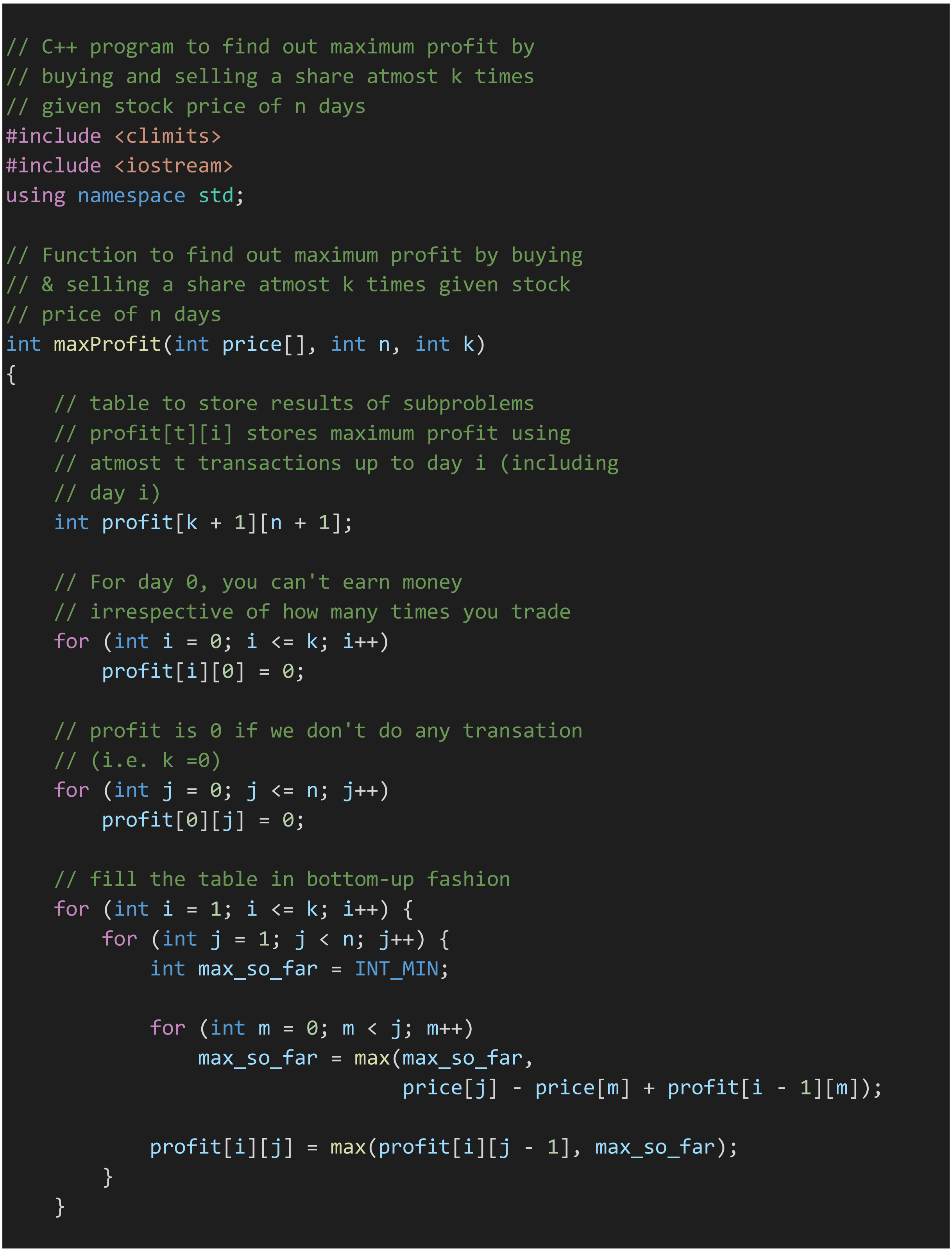


```
PS C:\Users\sandeep rathod\OneDrive\Documents\D_A_A> cd "c:\Users\sandeep rathod\OneDrive\Documents\D_
A_A\" ; if ($?) { g++ mp_seminar_recursive.cpp -o mp_seminar_recursive } ; if ($?) { .\mp_seminar_recu
rsive }
Maximum profit is: 45
PS C:\Users\sandeep rathod\OneDrive\Documents\D_A_A> 
```

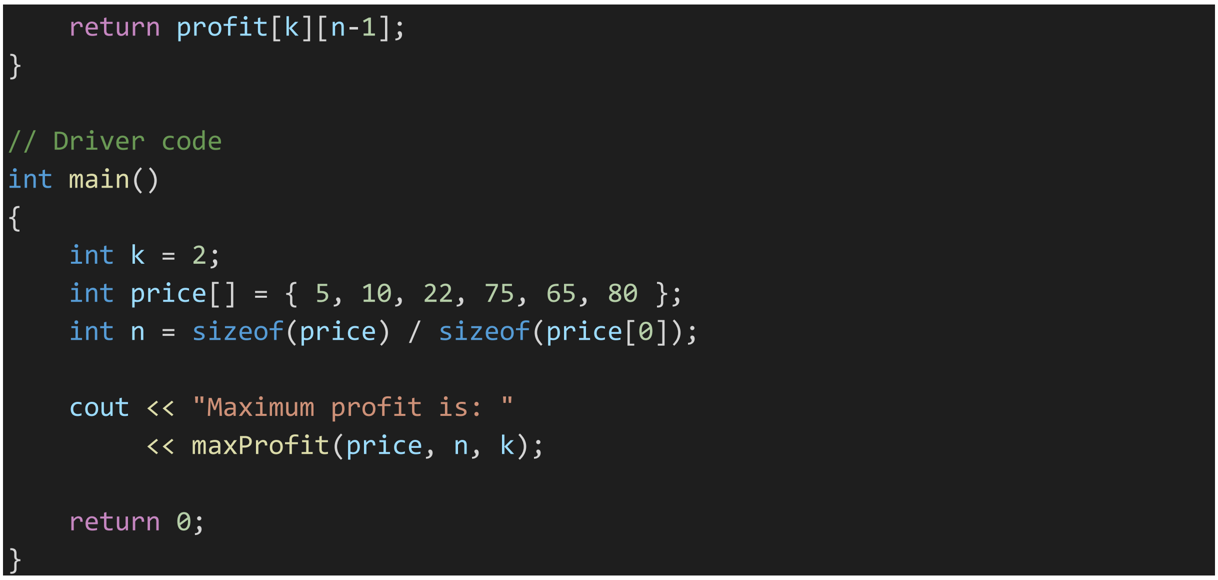
BOTTOM UP SOLUTION:→

(PSUEDOCODE AND FORMULA ARE EXPLAINED IN PPT)

CODE:

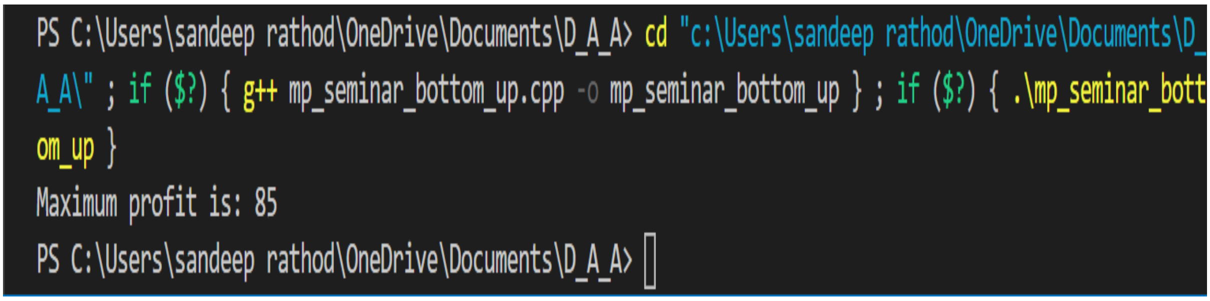


```
// C++ program to find out maximum profit by
// buying and selling a share atmost k times
// given stock price of n days
#include <climits>
#include <iostream>
using namespace std;

// Function to find out maximum profit by buying
// & selling a share atmost k times given stock
// price of n days
int maxProfit(int price[], int n, int k)
{
    // table to store results of subproblems
    // profit[t][i] stores maximum profit using
    // atmost t transactions up to day i (including
    // day i)
    int profit[k + 1][n + 1];

    // For day 0, you can't earn money
    // irrespective of how many times you trade
    for (int i = 0; i <= k; i++)
        profit[i][0] = 0;

    // profit is 0 if we don't do any transation
    // (i.e. k =0)
    for (int j = 0; j <= n; j++)
        profit[0][j] = 0;

    // fill the table in bottom-up fashion
    for (int i = 1; i <= k; i++) {
        for (int j = 1; j < n; j++) {
            int max_so_far = INT_MIN;

            for (int m = 0; m < j; m++)
                max_so_far = max(max_so_far,
                                 price[j] - price[m] + profit[i - 1][m]);

            profit[i][j] = max(profit[i][j - 1], max_so_far);
        }
    }
```



```
    return profit[k][n-1];
}

// Driver code
int main()
{
    int k = 2;
    int price[] = { 5, 10, 22, 75, 65, 80 };
    int n = sizeof(price) / sizeof(price[0]);

    cout << "Maximum profit is: "
         << maxProfit(price, n, k);

    return 0;
}
```

OUTPUT:→



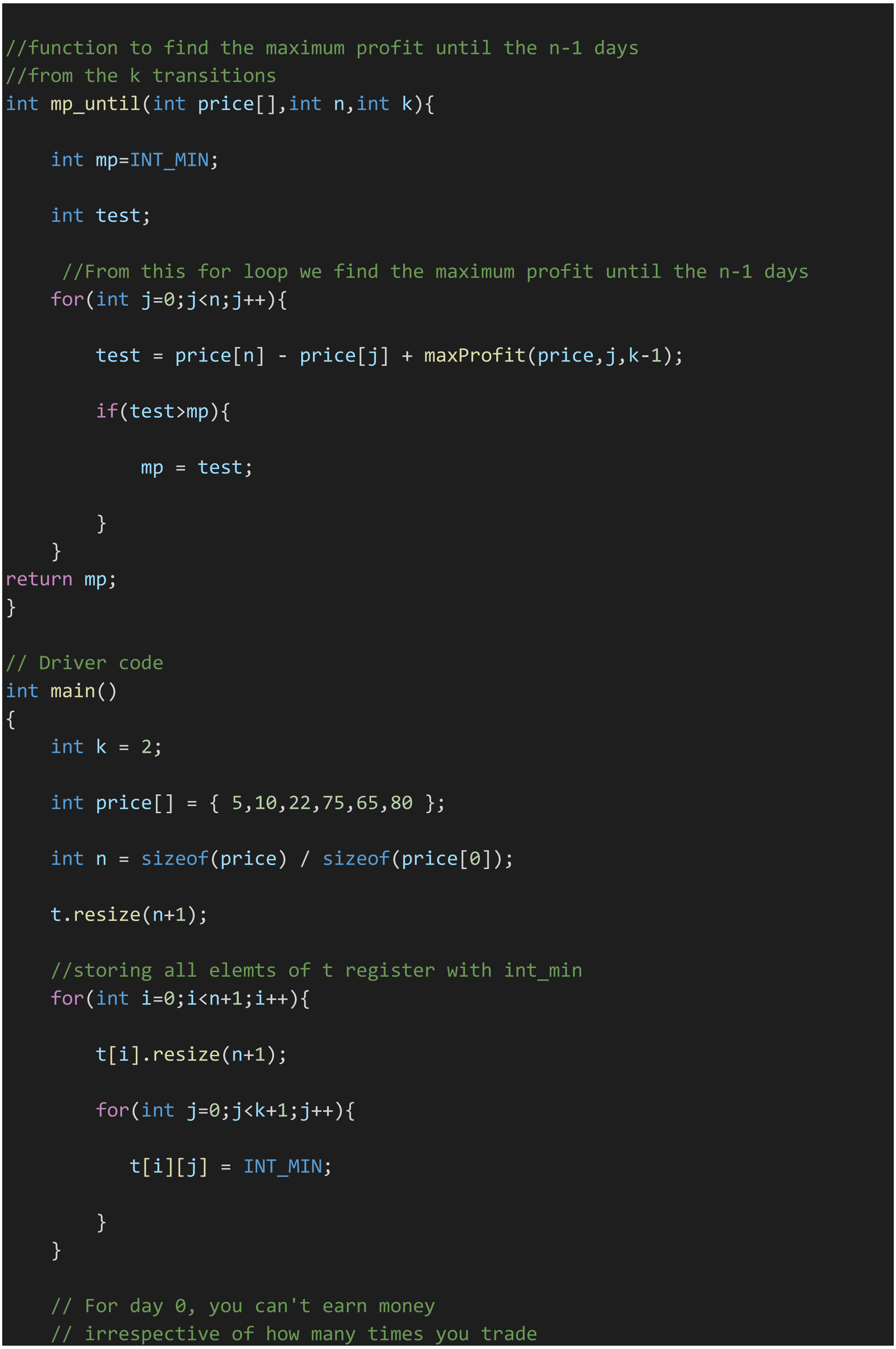
```
PS C:\Users\sandeep rathod\OneDrive\Documents\D_A_A> cd "c:\Users\sandeep rathod\OneDrive\Documents\D_
A_A\" ; if ($?) { g++ mp_seminar_bottom_up.cpp -o mp_seminar_bottom_up } ; if ($?) { .\mp_seminar_bott
om_up }
Maximum profit is: 85
PS C:\Users\sandeep rathod\OneDrive\Documents\D_A_A> 
```

TOP DOWN SOLUTION:→

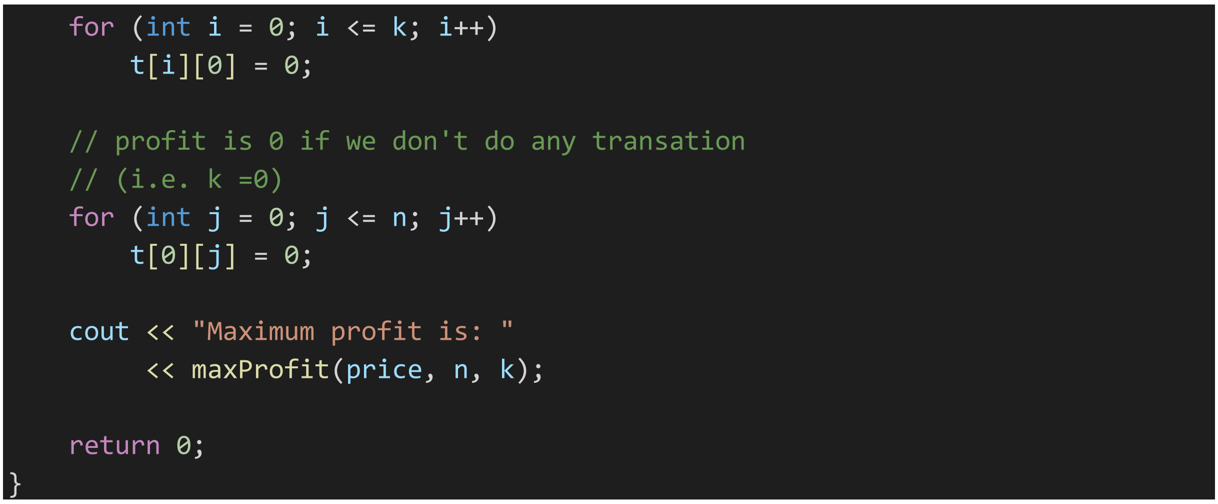
(PSUEDOCODE AND FORMULA ARE EXPLAINED IN PPT)

CODE:

```
// C++ program to find out maximum profit by
// buying and selling a share atmost k times
// given stock price of n days
#include <climits>
#include <iostream>
#include <vector>
using namespace std;
//Array to for the memoization
vector<vector<int>> t;

//fuction to find maximum of two numbers
int max(int a,int b){
    if(a>b){
        return a;
    }else return b;
}

int mp_until(int price[],int n,int k);

// Function to find out maximum profit by buying
// & selling a share atmost k times given stock
// price of n days
int maxProfit(int price[], int n, int k)
{   int mp;
    if(n==0||k==0){
        return 0;//termination condition for the recursive calls
    }

    //if the t[n][k] array has updated the value the return it directly
    if(t[n][k]!=INT_MIN){
        return t[n][k];
    }

     mp = mp_until(price,n-1,k);
     //memoization of the result in t array
     t[n][k] = max(maxProfit(price,n-1,k),mp);


return t[n][k];

}
```

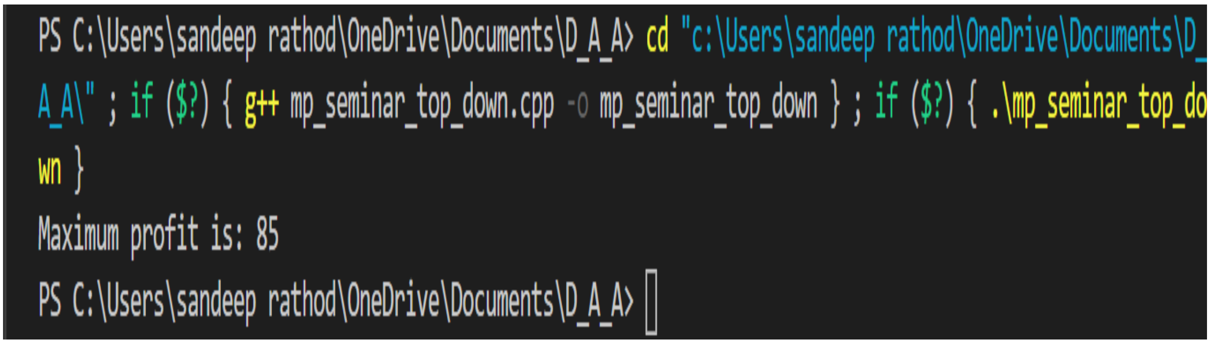


```
//function to find the maximum profit until the n-1 days
//from the k transitions
int mp_until(int price[],int n,int k){

    int mp=INT_MIN;

    int test;

     //From this for loop we find the maximum profit until the n-1 days
    for(int j=0;j<n;j++){

        test = price[n] - price[j] + maxProfit(price,j,k-1);

        if(test>mp){

            mp = test;

        }
    }
return mp;
}

// Driver code
int main()
{
    int k = 2;

    int price[] = { 5,10,22,75,65,80 };

    int n = sizeof(price) / sizeof(price[0]);

    t.resize(n+1);

    //storing all elemts of t register with int_min
    for(int i=0;i<n+1;i++){

        t[i].resize(n+1);

        for(int j=0;j<k+1;j++){

           t[i][j] = INT_MIN;

        }
    }

    // For day 0, you can't earn money
    // irrespective of how many times you trade
```



```
    for (int i = 0; i <= k; i++)
        t[i][0] = 0;

    // profit is 0 if we don't do any transation
    // (i.e. k =0)
    for (int j = 0; j <= n; j++)
        t[0][j] = 0;

    cout << "Maximum profit is: "
         << maxProfit(price, n, k);

    return 0;
}
```

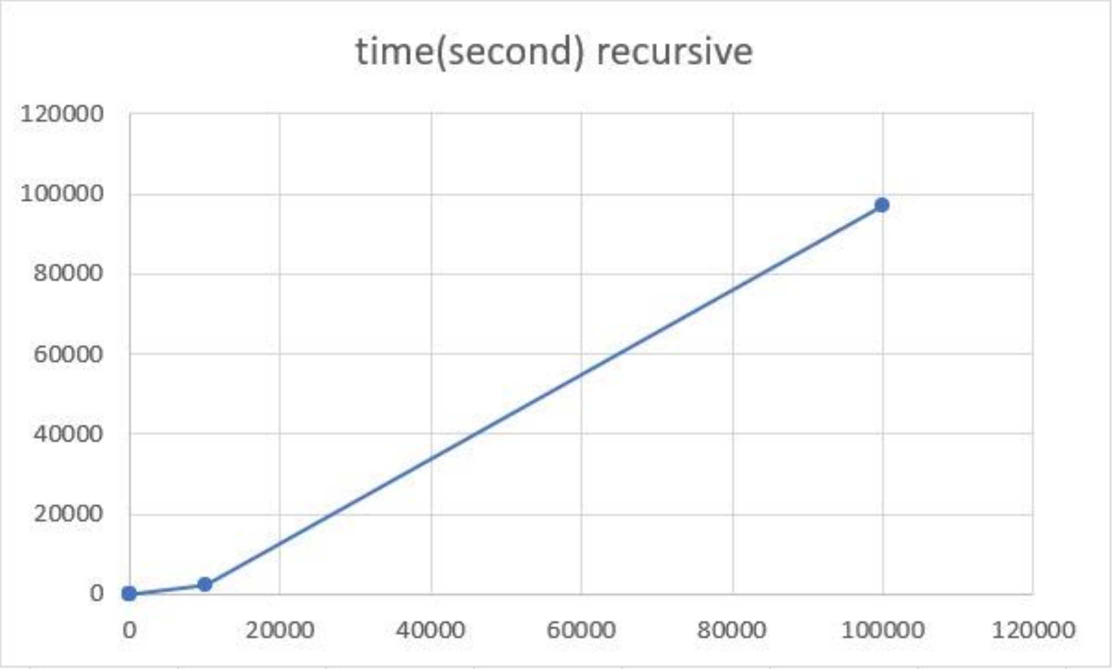
OUTPUT:→



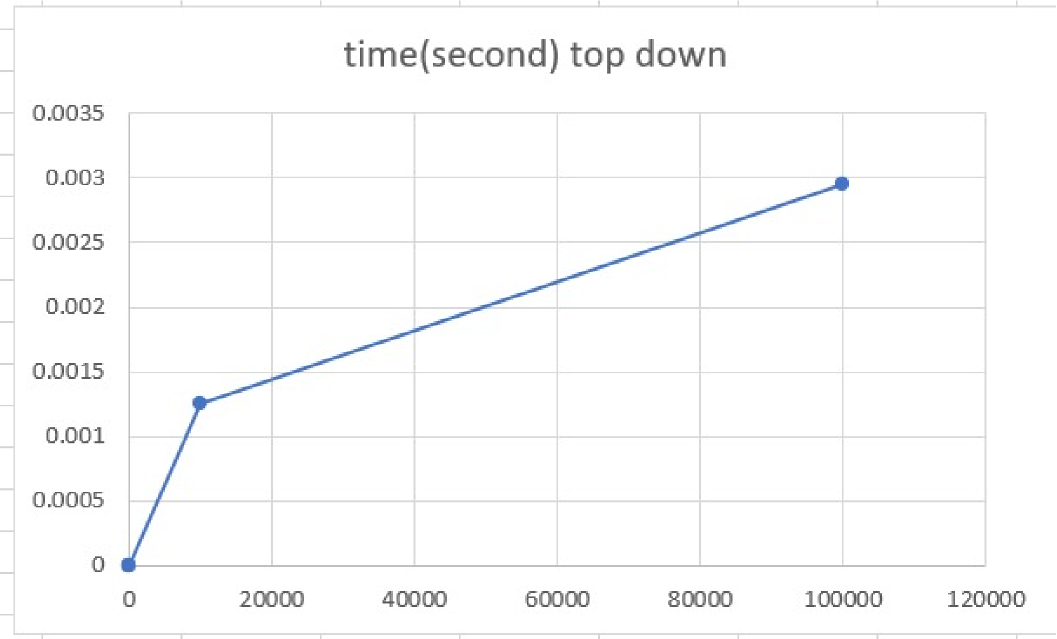
```
PS C:\Users\sandeep rathod\OneDrive\Documents\D_A_A> cd "c:\Users\sandeep rathod\OneDrive\Documents\D_
A_A\" ; if ($?) { g++ mp_seminar_top_down.cpp -o mp_seminar_top_down } ; if ($?) { .\mp_seminar_top_do
wn }
Maximum profit is: 85
PS C:\Users\sandeep rathod\OneDrive\Documents\D_A_A> 
```

The graph for time taken with data added for

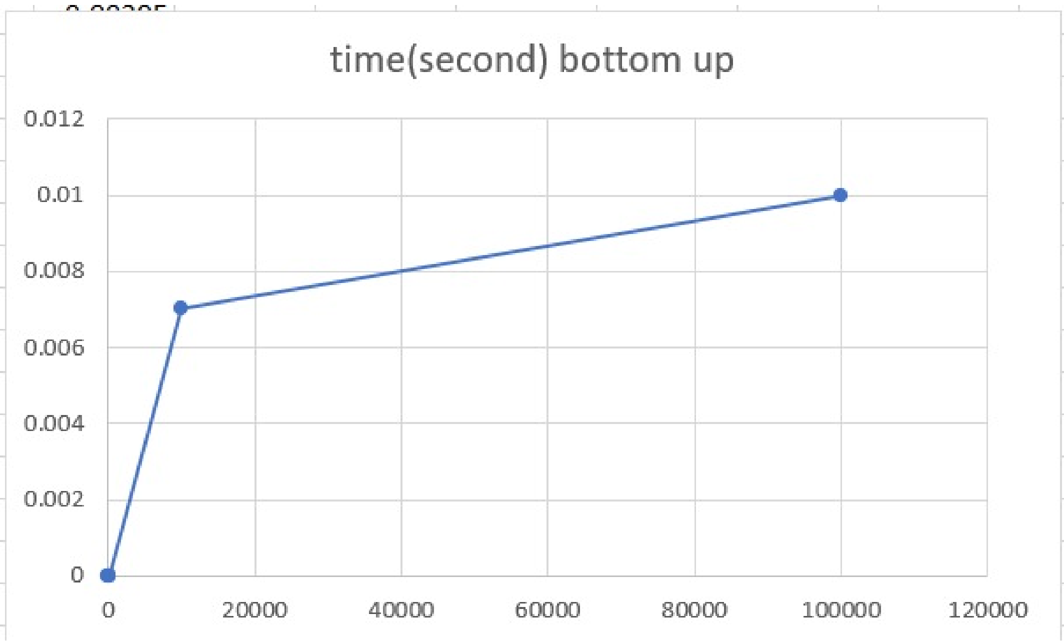
Recursive:



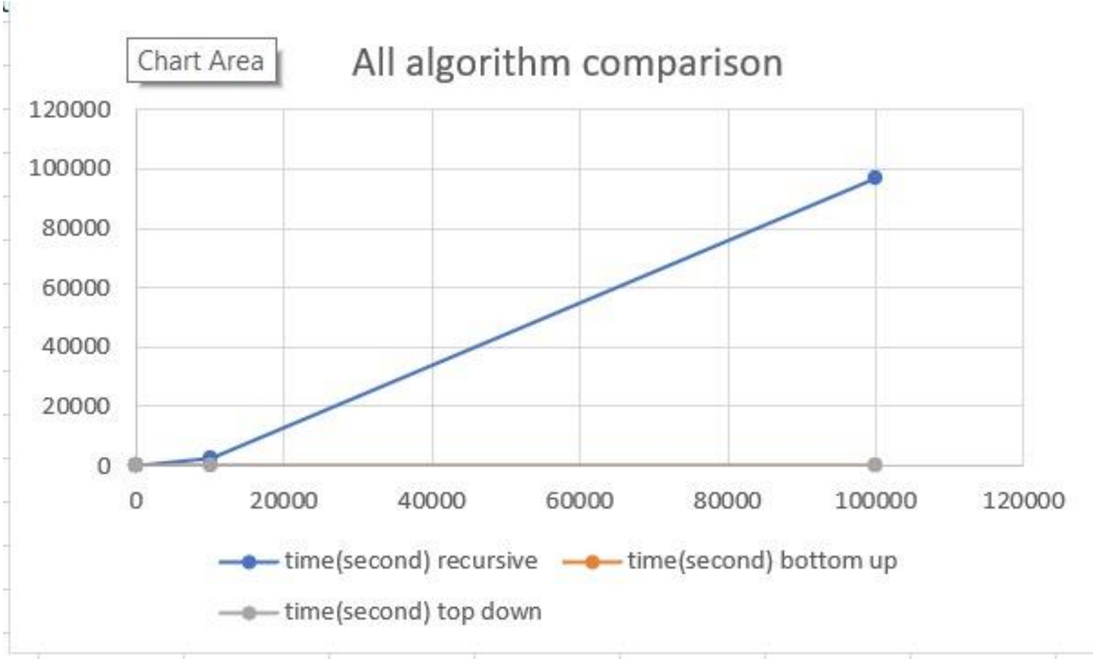
Bottome_up_algorithm:



Top-Down algorithm:



Comparing all the algorithms



As here compared

The dynamic programming approach decreases the time complexity drastically

As in ppt we got time complexity of the recursive algorithm

In Exponential time which is reduced to kn^2 times

Reflects in the above graphs also

Which implies the utility of

Solving the problem with dynamic programming approach.