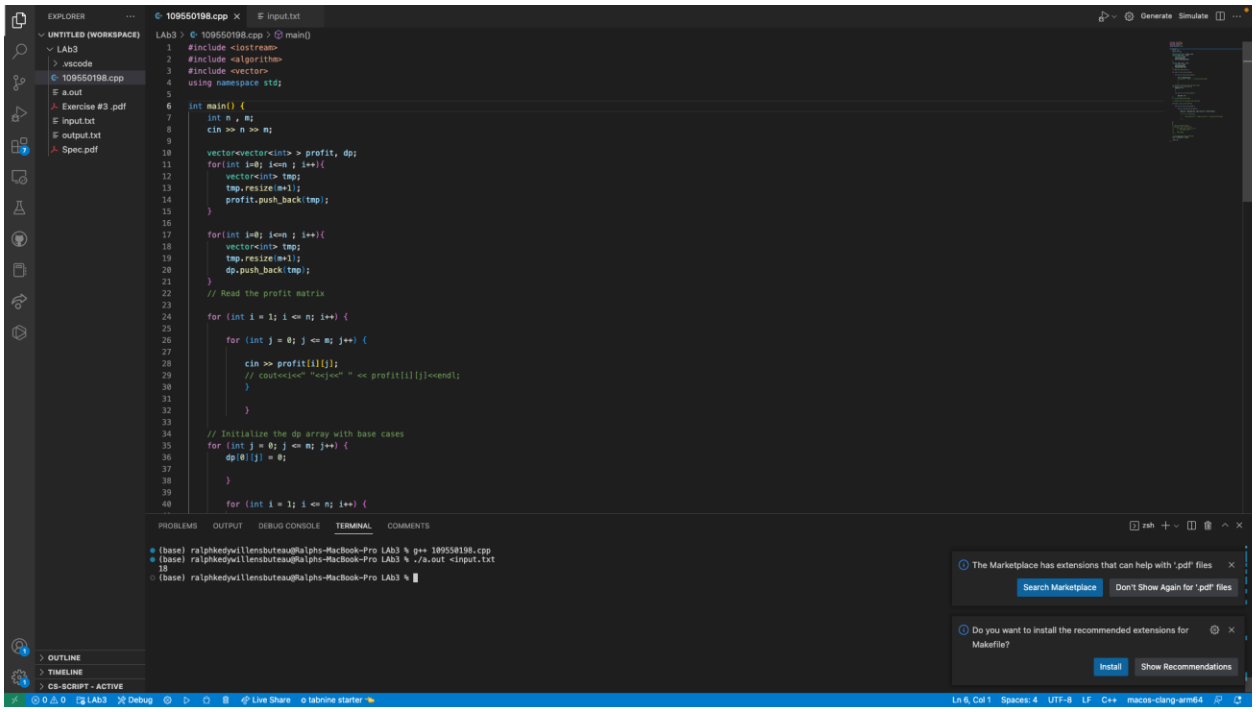
Intro to Algorithm
HW3
Report

109550198

- **<u>Environment (Os, compiler version, IDE)</u>**
  To implement this program, which is a Resource Allocation problem using dynamic programming. The OS used for this program is MacOS, the compiler version is apple clang version 14.0.0, and the IDE is visual studio code for mac.
  - <u>How to run your program</u>
    To run the program, I used the terminal of vs code because I was already in the file directory and typed *g++ 109550198.cpp* . After I clicked enter and typed *./a.out <input.txt* because the program uses standard input and output, I pressed enter. The terminal shows me the result

Here is the screenshot:



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

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

    vector<vector<int> > profit, dp;
    for(int i=0; i<=n ; i++){
        vector<int> tmp;
        tmp.resize(m+1);
        profit.push_back(tmp);
    }

    for(int i=0; i<=n ; i++){
        vector<int> tmp;
        tmp.resize(m+1);
        dp.push_back(tmp);
    }
    // Read the profit matrix

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

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

            cin >> profit[i][j];
            // cout<<i<<" "<<j<<" " << profit[i][j]<<endl;
            }

            }

    // Initialize the dp array with base cases
    for (int j = 0; j <= m; j++) {
        dp[0][j] = 0;

        }

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

```
● (base) ralphkedywillensbuteau@Ralphs-MacBook-Pro LAb3 % g++ 109550198.cpp
● (base) ralphkedywillensbuteau@Ralphs-MacBook-Pro LAb3 % ./a.out <input.txt
18
○ (base) ralphkedywillensbuteau@Ralphs-MacBook-Pro LAb3 % 
```

The Marketplace has extensions that can help with '.pdf' files

Do you want to install the recommended extensions for Makefile?

Search Marketplace
Don't Show Again for '.pdf' files
Install
Show Recommendations
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS
Ln 6, Col 1 Spaces: 4 UTF-8 LF C++ macos-clang-arm64

- Results
  - Method or solutions

<u>Explanations</u>
Here is a step-by-step process for using dynamic programming to solve the problem:

1. Create a 2D array with the dimensions dp[i][j] and dp[n+1] [m+1] to indicate the maximum profit that can be made by allocating j resources to the first i projects.
2. Start the array with common scenarios:
   • For all j, dp[0][j] = 0.
   • For all i dp[i][0] = 0.

3. Repeat the process for the resources and projects:
   Considering allocating j resources to each project i.

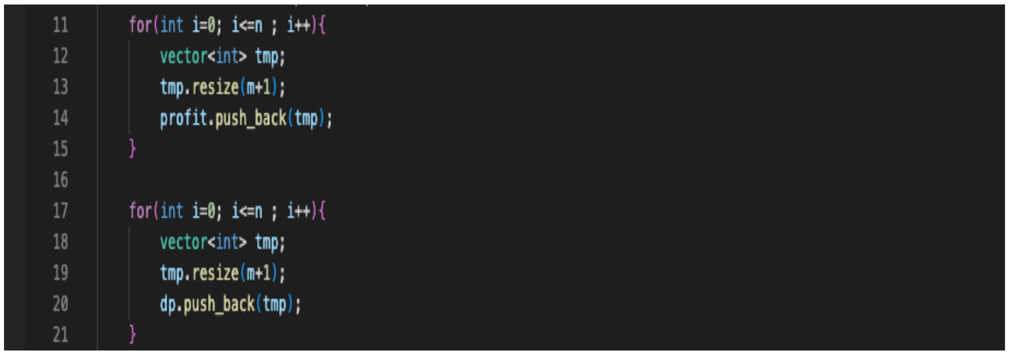
- Use the formula profit (i, j) to determine the profit that may be obtained by allocating j resources to project i.
- Use the formula dp[i][j] = max(dp[i][j], dp[i-1] [j-k] + profit (i, k)) to update the maximum profit for project i by adding the profit to the maximum profit gained from the preceding projects. where k varies between 0 and j.

4. The maximum profit that may be made by allocating resources to the projects after the iteration is finished is dp[n][m].

- **analyze the running time of your algorithm**

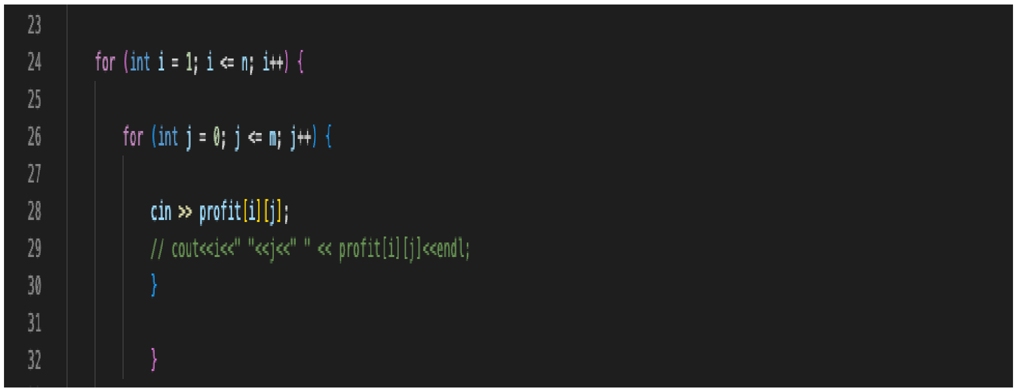
Here are some functions in the code. Each function has its own time complexity.

Step1: The time complexity of this code O(m*n)



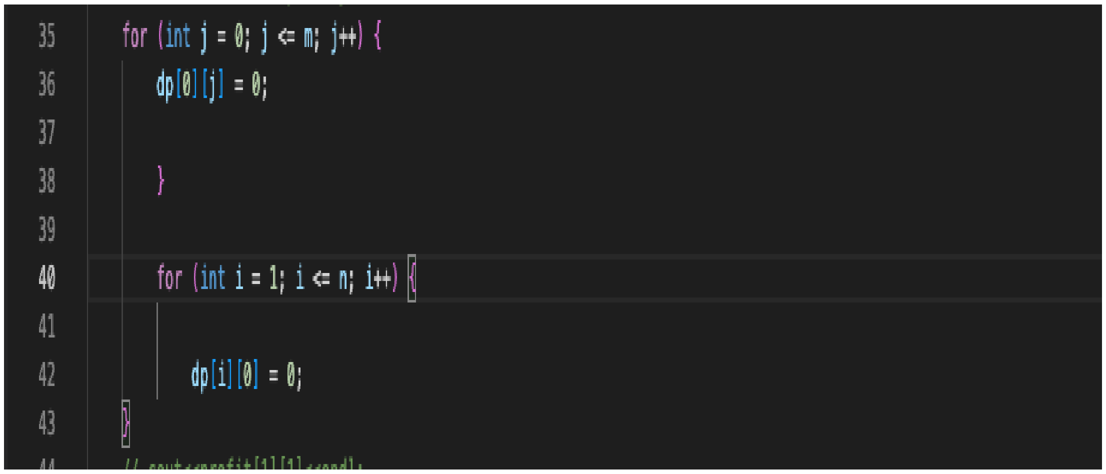
```
for(int i=0; i<=n ; i++){
    vector<int> tmp;
    tmp.resize(m+1);
    profit.push_back(tmp);
}

for(int i=0; i<=n ; i++){
    vector<int> tmp;
    tmp.resize(m+1);
    dp.push_back(tmp);
}
```

Step 2: The time complexity of this code O(m*n)



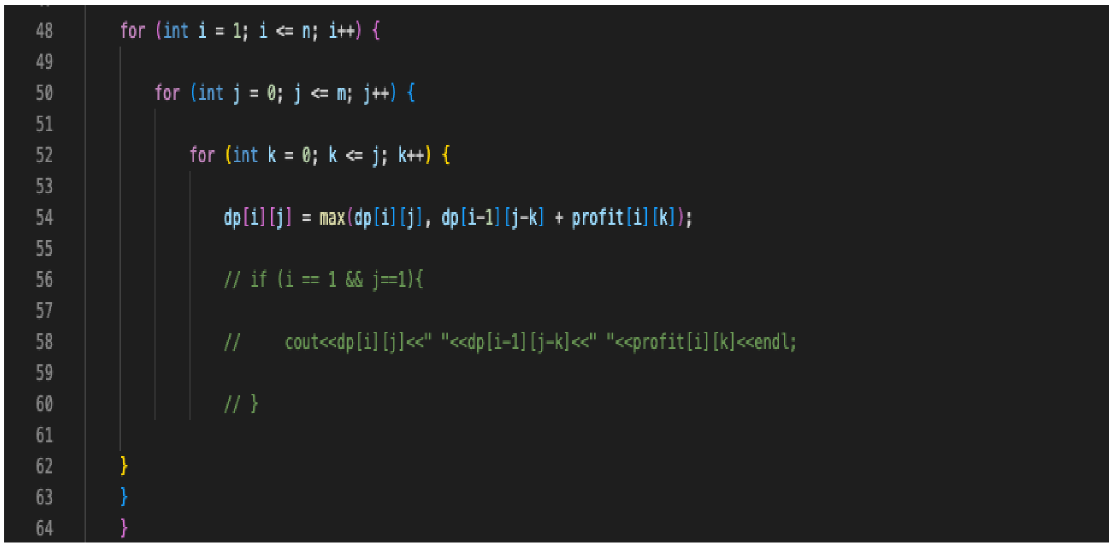
```

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

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

        cin >> profit[i][j];
        // cout<<i<<" "<<j<<" " << profit[i][j]<<endl;
        }

        }
```

Step 3: The time complexity of this code O(m+n)



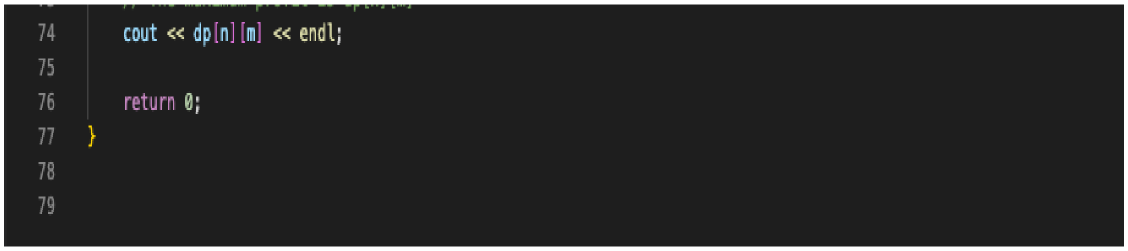
```
    for (int j = 0; j <= m; j++) {
        dp[0][j] = 0;

        }

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

            dp[i][0] = 0;
    }
    // cout<<profit[1][1]<<endl;
```

Step 4: The time complexity of this code $(n*m^2)$



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

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

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

                dp[i][j] = max(dp[i][j], dp[i-1][j-k] + profit[i][k]);

                // if (i == 1 && j==1){

                //     cout<<dp[i][j]<<" "<<dp[i-1][j-k]<<" "<<profit[i][k]<<endl;

                // }

    }
    }
    }
```

Step 5: The time complexity of this code O (1)



```
    cout << dp[n][m] << endl;

    return 0;
}
```

- Explanation
  The time complexity of the Resource allocation problem using dynamic programming from step 1 to step 5:

  $T(N)= O(m*n) + O(m*n) + O(m+n) + O(n*m^2) +O(1)$

Simplification :
$T(N) = O(n*m^2)$

Final answer :

The time complexity is $O(n*m^2)$

- **Anything you want to share**

Reference URL: https://www.youtube.com/watch?v=P6XopZDtNCM