Aya Rafe' Daraghma
11924594

DB_Project
The Avengers

Dr. Samer Arandi

Step 1: Divide & Conquer

1- Define the value returned by the function f which we want to optimize

The FUNCTION() computes The maximum total possible power of the weapons to help the Avengers!

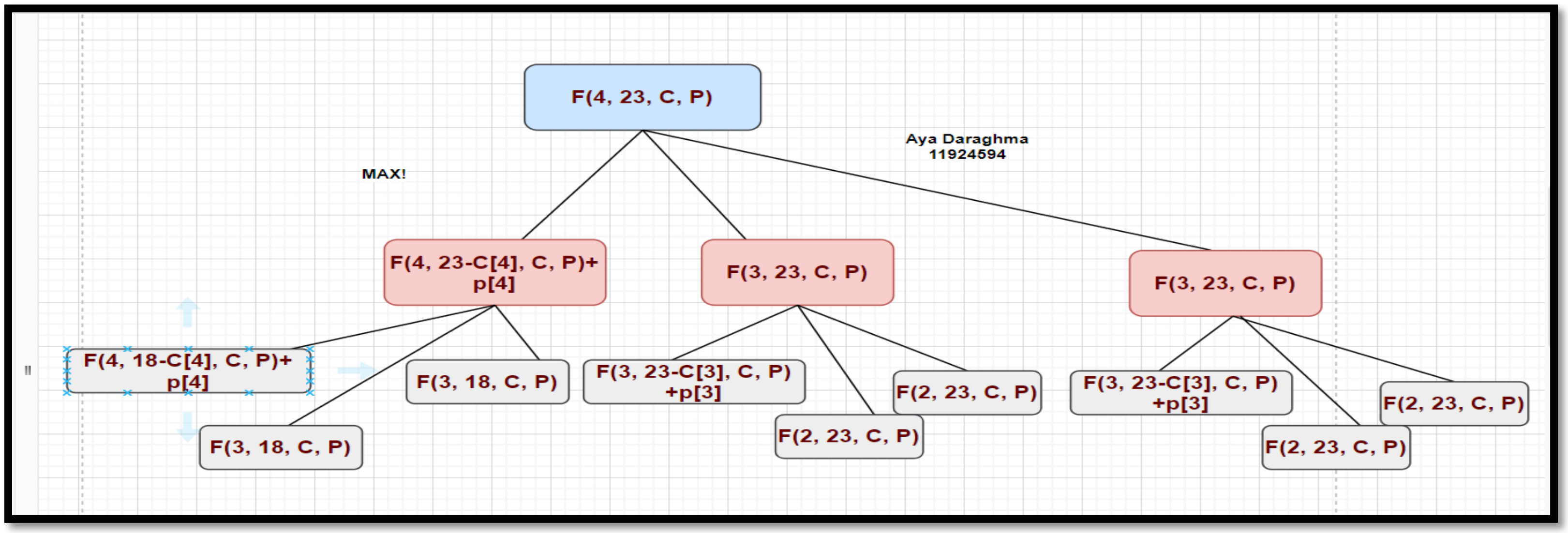
2- Define the parameters which f depends on.

We have 4 parameter in function..
FUNCTION(numof_weap, vib_val, C, P)

- The first parameters is: number of weapons
- The Second Parameter is: The amount of vabrenuim
- The Third Parameter is: the cost for each weapons we have
- The Fourth Parameter is: the power for each weapons we have

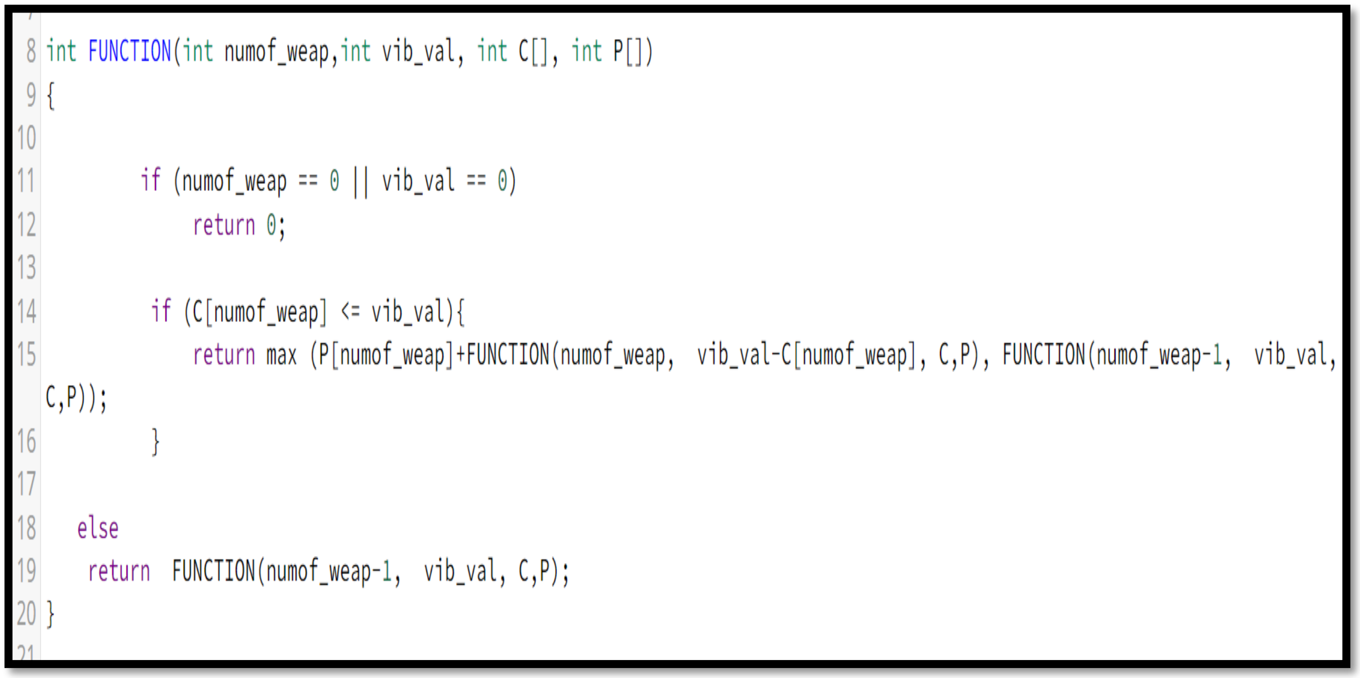
3- Draw the recursion tree for f using the values from the example above.

From the bellow recursion tree we noticed that it describe the problem clearly and divide it to sub problems each time..



4- **Write the recursive (divide and conquer) code to solve the question.**

in this funtion we divide problem to subproblems by call the functions recursively inside the function many times but this way of coding inefficient because each time we need any value for any subproblem we must calculate it so we go to solve this problem by using Dynamic Programming.



```
int FUNCTION(int numof_weap,int vib_val, int C[], int P[])
{

        if (numof_weap == 0 || vib_val == 0)
            return 0;

         if (C[numof_weap] <= vib_val){
            return max (P[numof_weap]+FUNCTION(numof_weap,  vib_val-C[numof_weap], C,P), FUNCTION(numof_weap-1,  vib_val,
C,P));
         }

   else
    return  FUNCTION(numof_weap-1,  vib_val, C,P);
}
```

Step 2: Dynamic Programming

In this part, we will use dynamic programming to solve THE AVENGERS problem as we know solving problems by divide & conquer is inefficient because we have to solve subproblems many times but by using dynamic programming each time we solve a subproblem we store the answer inside a table and use it later if we need

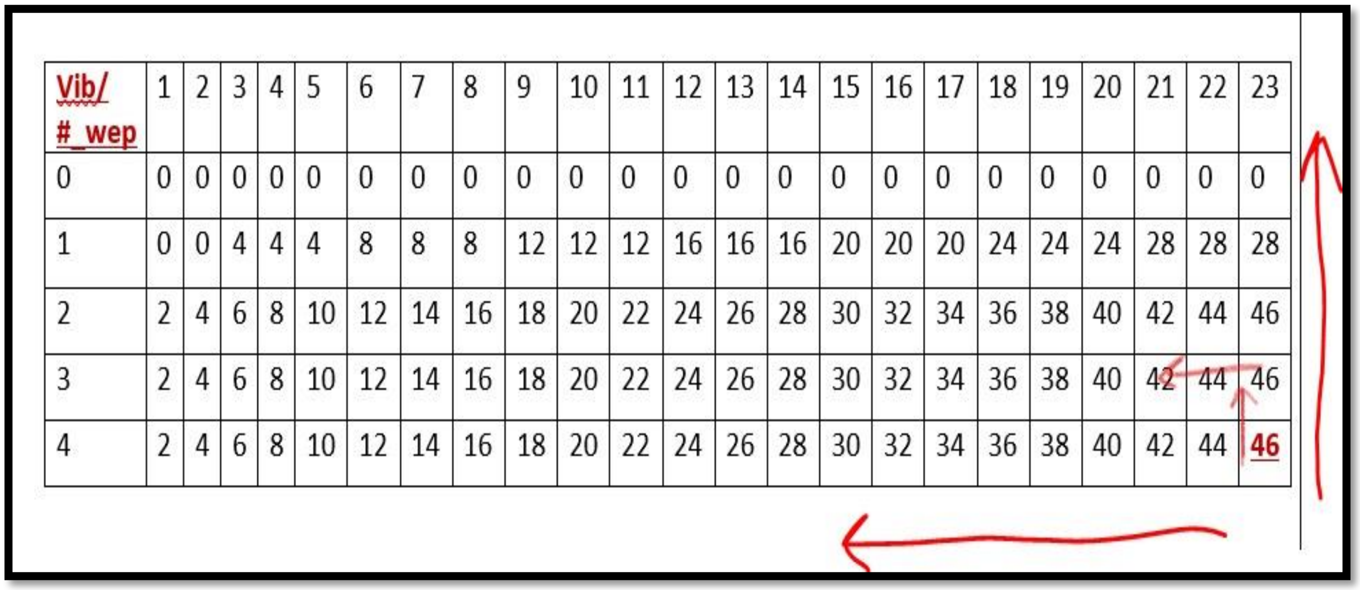
5- **Draw the table and determine the dependencies between the table cells.**

Row for -> number of weapons
Colom for -> Amount of Vibranium

Vib/ #_wep	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	4	4	4	8	8	8	12	12	12	16	16	16	20	20	20	24	24	24	28	28	28
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
3	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
4	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	**46**

6- Determine the direction of movement within the table.

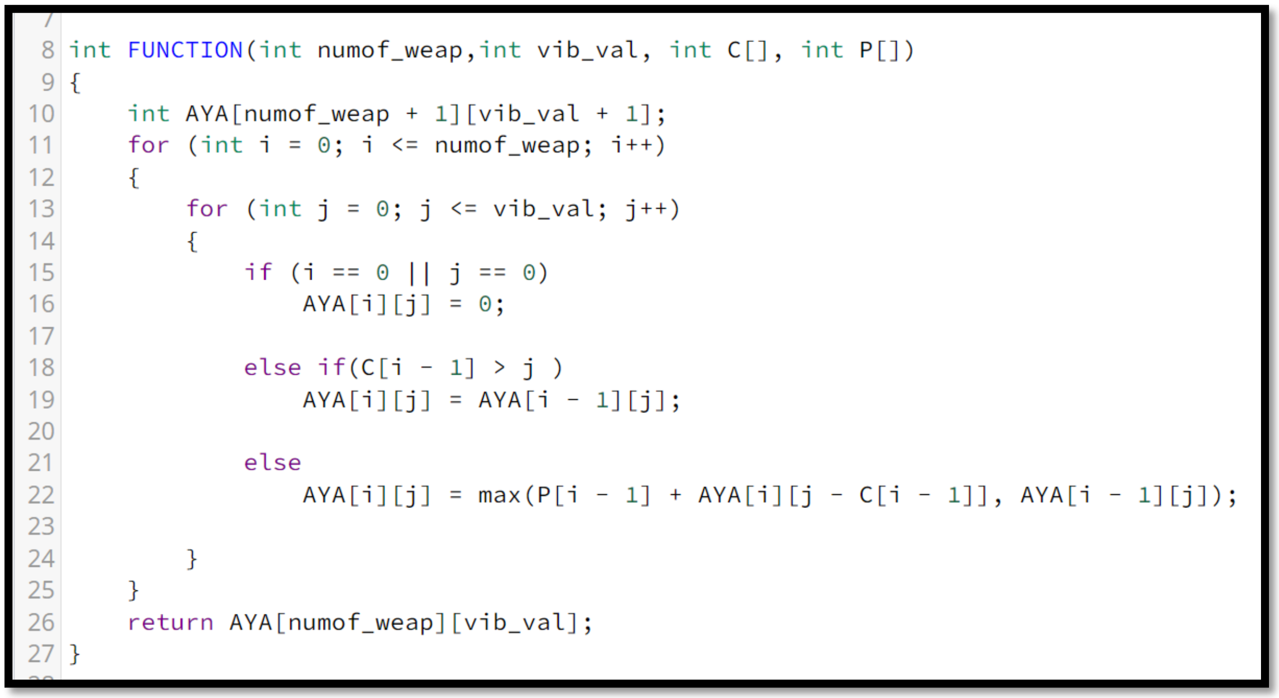
The figure below shows the direction of movement..



Vib/ #_wep	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	4	4	4	8	8	8	12	12	12	16	16	16	20	20	20	24	24	24	28	28	28
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
3	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
4	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	**46**

7- Write the Dynamic programming code which fills the table(s).

Here if we need any values from any previous subproblems it will be already calculated and stored inside (AYA[]) array and we won't have to calculate it another time..



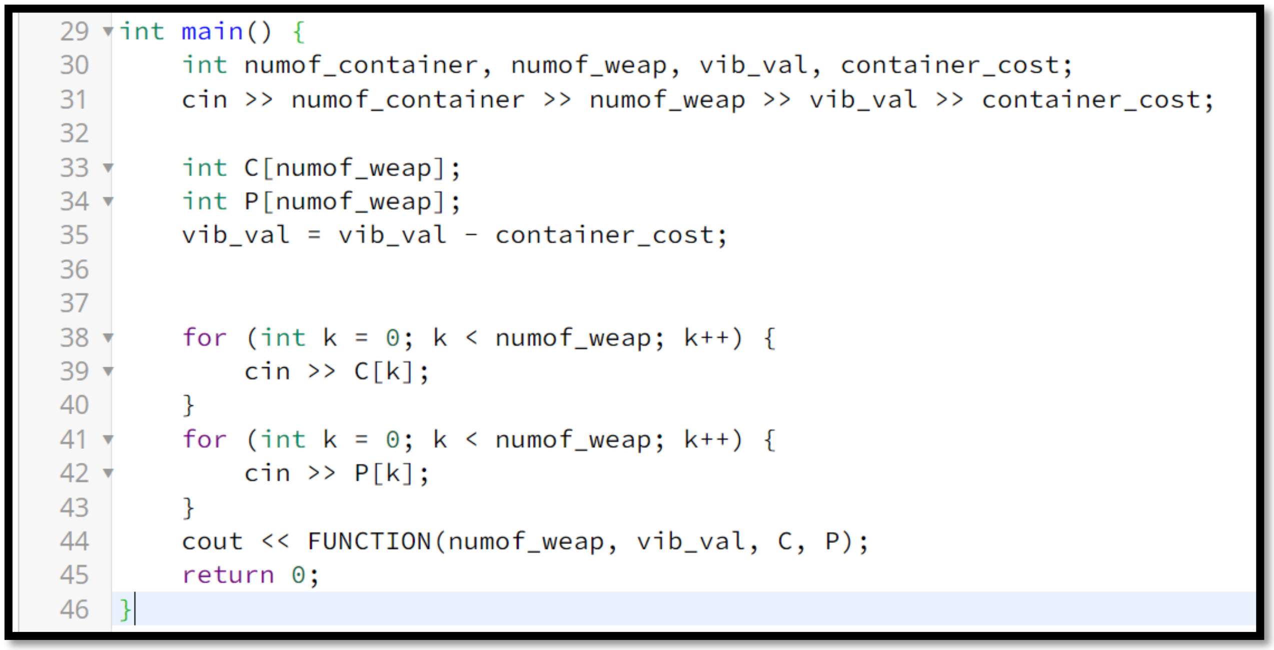
```cpp
int FUNCTION(int numof_weap,int vib_val, int C[], int P[])
{
    int AYA[numof_weap + 1][vib_val + 1];
    for (int i = 0; i <= numof_weap; i++)
    {
        for (int j = 0; j <= vib_val; j++)
        {
            if (i == 0 || j == 0)
                AYA[i][j] = 0;

            else if(C[i - 1] > j )
                AYA[i][j] = AYA[i - 1][j];

            else
                AYA[i][j] = max(P[i - 1] + AYA[i][j - C[i - 1]], AYA[i - 1][j]);

        }
    }
    return AYA[numof_weap][vib_val];
}
```

and this is the main..

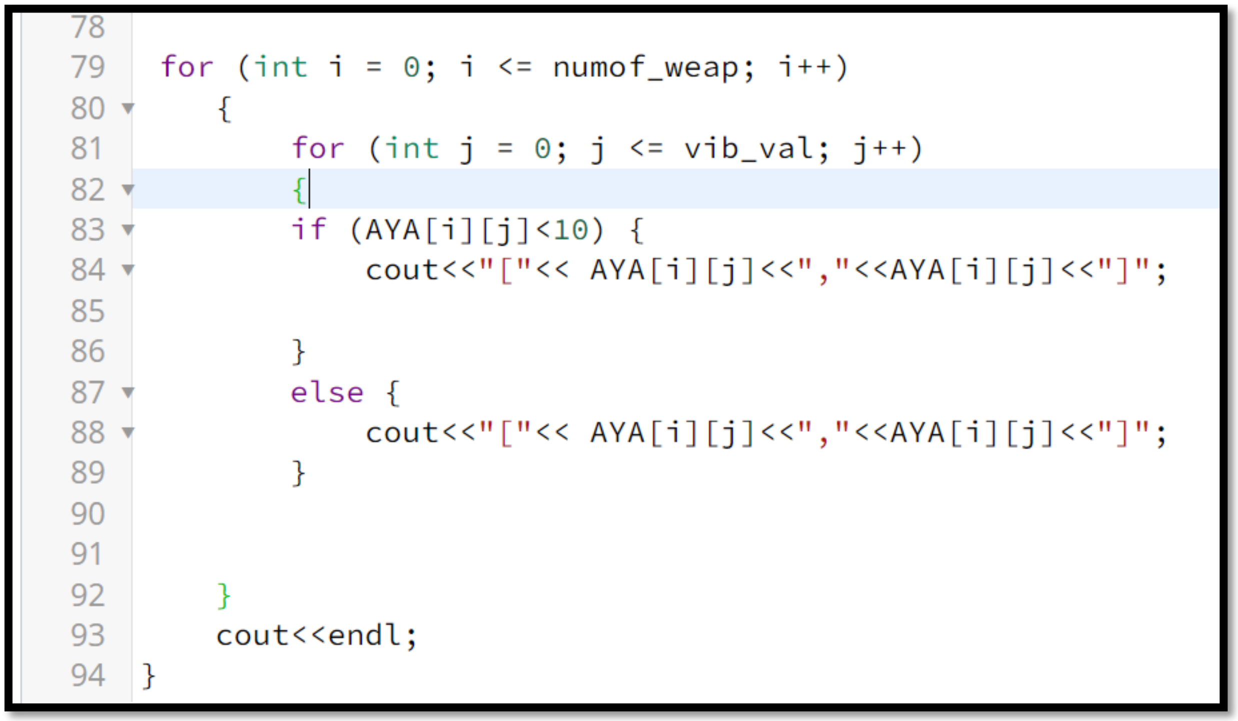


```cpp
int main() {
    int numof_container, numof_weap, vib_val, container_cost;
    cin >> numof_container >> numof_weap >> vib_val >> container_cost;

    int C[numof_weap];
    int P[numof_weap];
    vib_val = vib_val - container_cost;


    for (int k = 0; k < numof_weap; k++) {
        cin >> C[k];
    }
    for (int k = 0; k < numof_weap; k++) {
        cin >> P[k];
    }
    cout << FUNCTION(numof_weap, vib_val, C, P);
    return 0;
}
```

8- **<u>the code that will print the sequence of moves that go you the solution.</u>**



```
for (int i = 0; i <= numof_weap; i++)
    {
        for (int j = 0; j <= vib_val; j++)
        {
        if (AYA[i][j]<10) {
            cout<<"["<< AYA[i][j]<<","<<AYA[i][j]<<"]";

        }
        else {
            cout<<"["<< AYA[i][j]<<","<<AYA[i][j]<<"]";
        }


    }
    cout<<endl;
}
```