Algorithm:

```
Function checker(index,enemies[],current_power,behind_power,recharge,skip,initial_power)

//Base condition
if(index==enemies.size()) return true;

op1=0,op2=0,op3=0

if(current_power>=behind_power)
     current_power-=behind_power
     behind=0
else
    Return false

//case 1 : Skip
if (skip>0)
   op1=checker(index+1,enemies,current_power,behind_power,recharge,skip-1,initial_power)

// case 2: Fight
if(current_power>=enemies[index]){

     newBehind_power=behind_power
      if(index==2 or index==6)
              newBehind_power=enemies[index]/2;

op2=checker(index+1,enemies,current_power-enemies[index],newBehind_power,recharge,skip,
initial_power)

}

//case 3:recharge

if(recharge>0 and current_power<initial_power)
     op3=checker(index,enemies,initial_power,behind_power,recharge,skip,initial_power)

Return op1 or op2 or op3
```

Complete CODE:

```
#include <bits/stdc++.h>
using namespace std;

bool checker(int index, vector<int>& enemies, int current_power, int
behind_power, int recharge, int skip, int initial_power) {

    // Base condition

    if (index == enemies.size()) return true;

    bool op1 = false, op2 = false, op3 = false;


     // Handling regenerated power of k3 and k7

    if (current_power >= behind_power) {

        current_power -= behind_power;
        behind_power = 0; // Reset behind power

    } else {
        return false; // Attacked by k3 or k7
    }


    // Case 1: Skip the circle

    if (skip > 0) {
        op1 = checker(index + 1, enemies, current_power, behind_power,
recharge, skip - 1, initial_power);
    }



    // Case 2: Fight with the enemy
    if (current_power >= enemies[index]) {

        int new_behind_power = behind_power;

        if (index == 2 || index == 6) {

            new_behind_power = enemies[index] / 2; // Set regenerated
power of k3 and k7
        }

        op2 = checker(index + 1, enemies, current_power - enemies[index],
new_behind_power, recharge, skip, initial_power);
    }


  // Case 3: Recharge

    if (recharge > 0 && current_power < initial_power) {

        op3 = checker(index , enemies, initial_power, behind_power,
recharge - 1, skip, initial_power);
    }

    // Return true if any of the cases work
    return op1 || op2 || op3;
}





int main() {
    vector<int> enemies(11);

    for(int i=0;i<11;i++){
        cout<<"Enter the power of enemy "<<i<<"\n";
        cin>>enemies[i];
    }
    int initial_power,recharge,skip;
    cout<<"Enter Initial power of Abhimanyu\n";
    cin>>initial_power;

    cout<<"Enter number of times to recharge\n";
    cin>>recharge;

    cout<<"Enter number of times to skip\n";
    cin>>skip;

    bool result = checker(0, enemies, initial_power, 0,
recharge,skip,initial_power);

    if (result) {
        cout << "Abhimanyu Survived!" << endl;
    } else {
        cout << "Abhimanyu was defeated!" << endl;
    }


    return 0;
}
```

Explanation for the Algorithm
The recursive function (checker) handles three scenarios:

1)Skip the battle.
2)Fight the enemy if power is sufficient.
3)Recharge to restore his power to the initial value.
 Abhimanyu can choose one of the above 3 option and proceed the fight
If the function returns true if Abhimanyu successfully crosses all circles, otherwise false.

Test cases with sample output

Test case: 1

Enemies: `[10, 15, 20, 12, 18, 25, 22, 16, 10, 8, 14]`
Initial Power: 120
Recharge Count: 2

Skip Count: 2

Output : `Abhimanyu Survived!`

Test case: 2

Enemies: `[20, 25, 30, 18, 15, 12, 10, 22, 28, 35, 40]`
Initial Power: 70
Recharge Count: 1
Skip Count: 2

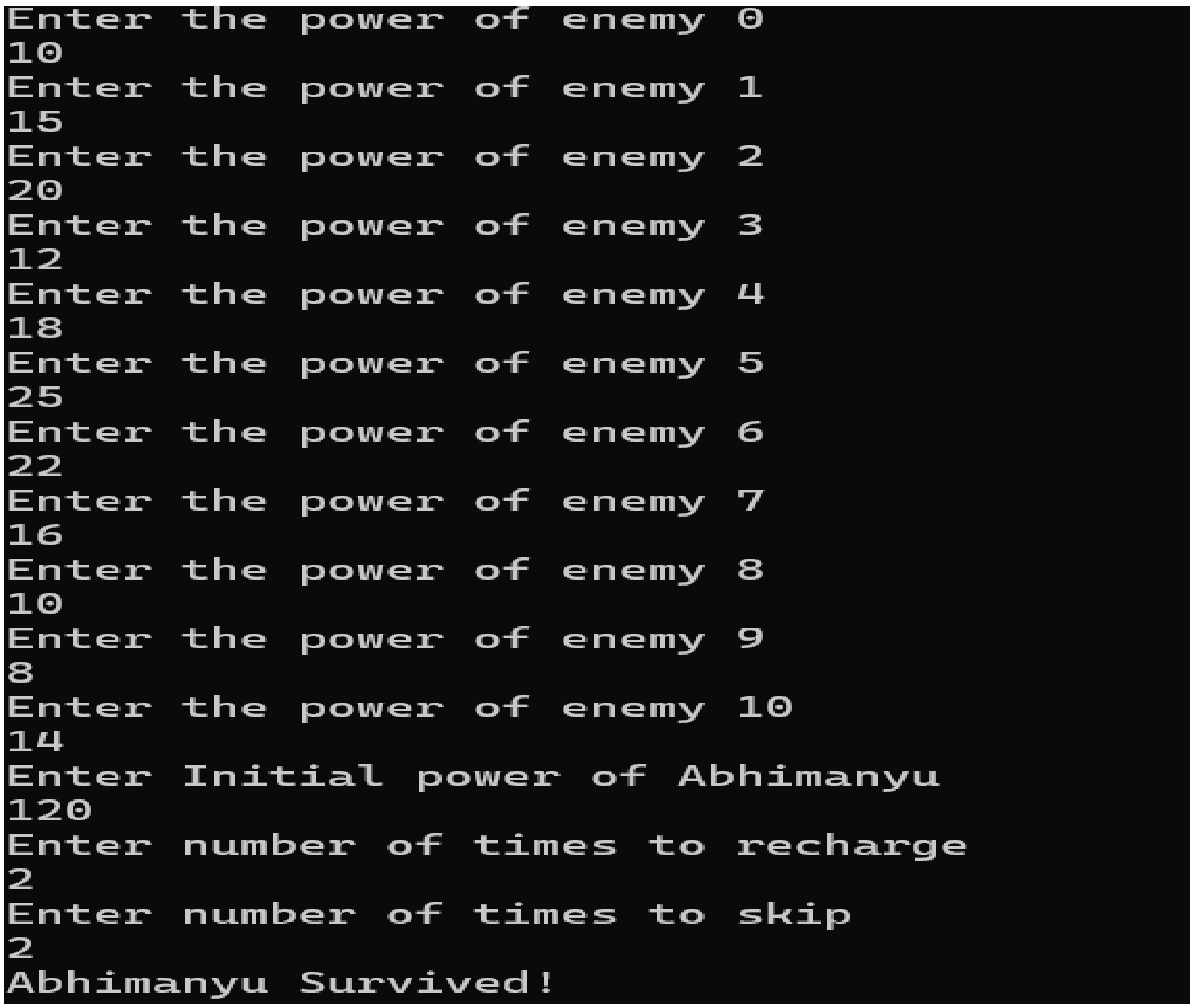
Output : `Abhimanyu was defeated!`

Test case: 3

Enemies: `[20, 25, 30, 35, 10, 5, 8, 22, 30, 50, 18]`
Initial Power: 80
Recharge Count: 1
Skip Count: 3

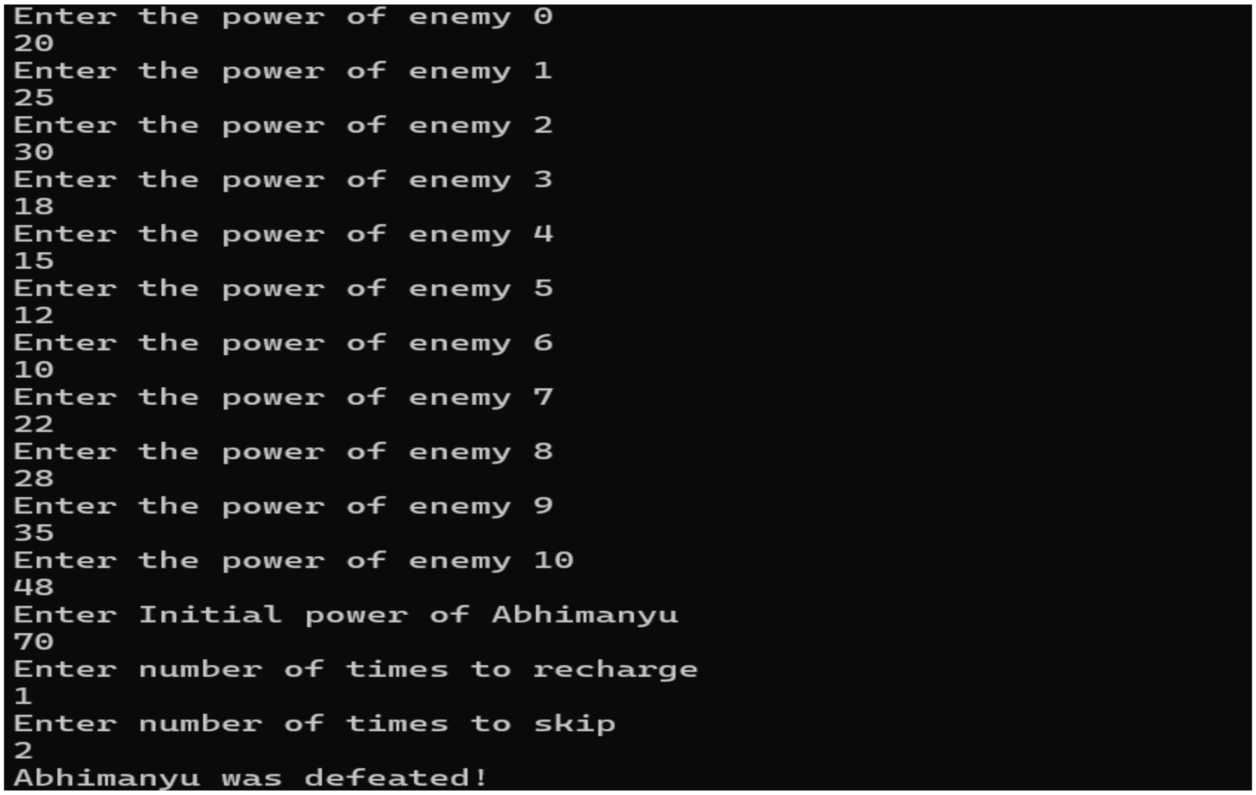
Output : `Abhimanyu Survived!`

Output for test case 1:



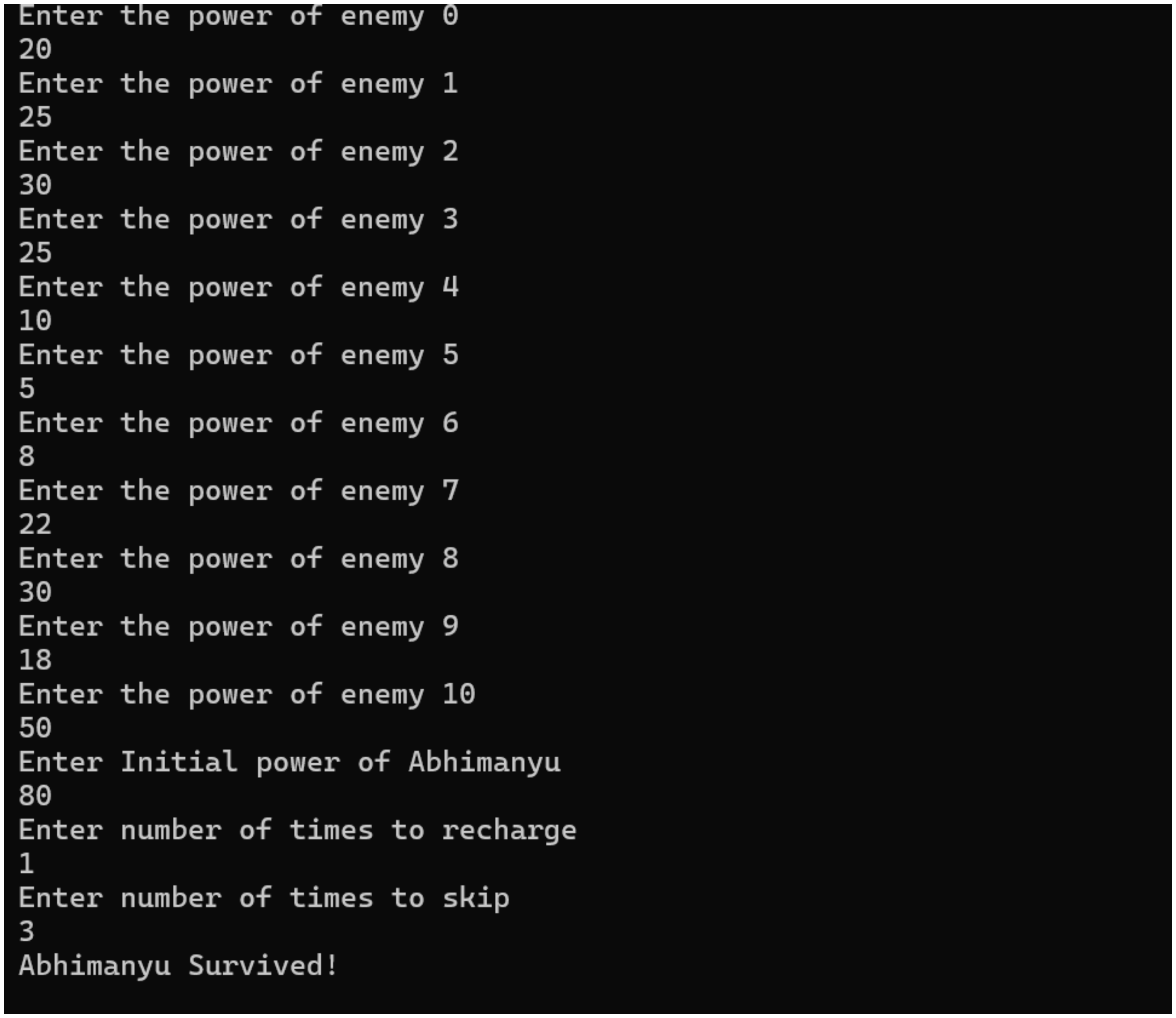
```
Enter the power of enemy 0
10
Enter the power of enemy 1
15
Enter the power of enemy 2
20
Enter the power of enemy 3
12
Enter the power of enemy 4
18
Enter the power of enemy 5
25
Enter the power of enemy 6
22
Enter the power of enemy 7
16
Enter the power of enemy 8
10
Enter the power of enemy 9
8
Enter the power of enemy 10
14
Enter Initial power of Abhimanyu
120
Enter number of times to recharge
2
Enter number of times to skip
2
Abhimanyu Survived!
```

Output for test case 2:



```
Enter the power of enemy 0
20
Enter the power of enemy 1
25
Enter the power of enemy 2
30
Enter the power of enemy 3
18
Enter the power of enemy 4
15
Enter the power of enemy 5
12
Enter the power of enemy 6
10
Enter the power of enemy 7
22
Enter the power of enemy 8
28
Enter the power of enemy 9
35
Enter the power of enemy 10
48
Enter Initial power of Abhimanyu
70
Enter number of times to recharge
1
Enter number of times to skip
2
Abhimanyu was defeated!
```

Output for test case 3:



```
Enter the power of enemy 0
20
Enter the power of enemy 1
25
Enter the power of enemy 2
30
Enter the power of enemy 3
25
Enter the power of enemy 4
10
Enter the power of enemy 5
5
Enter the power of enemy 6
8
Enter the power of enemy 7
22
Enter the power of enemy 8
30
Enter the power of enemy 9
18
Enter the power of enemy 10
50
Enter Initial power of Abhimanyu
80
Enter number of times to recharge
1
Enter number of times to skip
3
Abhimanyu Survived!
```