Makeup Assignment Solution

1. **Demo Solution:**

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    int N, T;
    cin >> N >> T;

    vector<int> marks(N), times(N);

    // Input marks and times
    for (int i = 0; i < N; i++) {
        cin >> marks[i];
    }
    for (int i = 0; i < N; i++) {
        cin >> times[i];
    }

    // DP array to store max marks possible for each time
    vector<int> dp(T + 1, 0);

    // Normal DP without doubling any marks
    for (int i = 0; i < N; i++) {
        for (int j = T; j >= times[i]; j--) {
            dp[j] = max(dp[j], dp[j - times[i]] + marks[i]);
        }
    }

    // Now try doubling each question's marks
    int result = 0;
    for (int i = 0; i < N; i++) {
        vector<int> temp_dp = dp; // Copy current DP array

        // Simulate doubling the marks of question i
        for (int j = T; j >= times[i]; j--) {
            temp_dp[j] = max(temp_dp[j], dp[j - times[i]] + 2 * marks[i]);
        }

        // Update the result to the best possible score
        result = max(result, *max_element(temp_dp.begin(), temp_dp.end()));
    }

    cout << result << endl;

    return 0;
}
```

2. **Demo solution:**

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    int T;  // Number of test cases
    cin >> T;

    while (T--) {
        int N, M;
        cin >> N >> M;

        vector<int> V(N), P(N);  // Vitamins and prices

        for (int i = 0; i < N; i++) {
            cin >> V[i] >> P[i];
        }

        // DP array to store max vitamins possible for each price without spell
        vector<int> dp(M + 1, 0);

        // Normal DP calculation without using the spell
        for (int i = 0; i < N; i++) {
            for (int j = M; j >= P[i]; j--) {
                dp[j] = max(dp[j], dp[j - P[i]] + V[i]);
            }
        }

        // The result without using the spell
        int result = *max_element(dp.begin(), dp.end());

        // Now try applying the spell to each apple and restart the DP
calculation for that case
```

```
        for (int i = 0; i < N; i++) {
            int halved_price = P[i] / 2;

            // Create a new DP array from scratch, simulating halved price for
the ith apple
            vector<int> temp_dp(M + 1, 0);

            // Fill DP considering the ith apple has halved price
            for (int k = 0; k < N; k++) {
                int current_price = (k == i) ? halved_price : P[k]; // Use halved price
for the ith apple
                for (int j = M; j >= current_price; j--) {
                    temp_dp[j] = max(temp_dp[j], temp_dp[j - current_price] + V[k]);
                }
            }

            // Update the result with the maximum value after applying the spell
            result = max(result, *max_element(temp_dp.begin(), temp_dp.end()));
        }

        // Output the final result
        cout << result << endl;
    }

    return 0;
}
```