Midterm Problem 2: Volatile Knapsack

The subproblems can be for weight w’ which is strictly less than W, what is the minimum max volatility to collect total weight exactly w’.

Having this subproblems, we can solve the problem from w’ = 2 up to w’ = W-1, and solve the final problem with all the solutions we got.

To achieve the goal, we can construct a dp array dp[ ], where dp[w] indicates that the minimum max volatility to collect total weight exactly w.

For item i, where $1 \le i \le n$,

$dp[x] = \min(\max(dp[x - w_i], dp[w_i]), \max(v_i, dp[x - w_i]))$.

And we traverse all items to find the minimum dp[x].

The reason why the recurrence relation look like this is that: if we want weight x, then for each item, we can decide whether we want to take this item, if we do, then the dp[x] is $\max(v_i, dp[x - w_i])$, which is the maximum one between volatility of this item and the volatility of the rest weight; if not, then the dp[x] is $\max(dp[x - w_i], dp[w_i])$, which is the maximum value between the two subproblems one with the weight of this item, and one of the rest weight.

This can consider all cases although not all combinations of weight of x is considered, because if there is an best choice which we want to select, then it must choose some of the items, And this time, since we traverse all items, the best choice must be calculated at some places. Which means that we did not miss the best case.

The pseudocode is as follows:

```
$minimum\_Volatile(w[\,], v[\,], n, W)$
    $dp[\,]$ $initialized\ to\ be\ \infty$
    $for\ i\ from\ 1\ to\ n\ do$
        $if\ w[i] = 1\ then$
            $dp[1] = \min(dp[1], v[i])$
    $for\ j\ from\ 2\ to\ W\ do$
        $for\ k\ from\ 1\ to\ n\ do$
            $if\ w[i] > j\ then$
                $continue$
            $dp[j] = \min(\max(dp[j - w[i]], dp[w[i]]), \max(dp[j - w[i]], v[i]))$
```

In the pseudocode, if we are trying to access dp[i] where i is less than 0, we consider that to be ∞ using some if statements.

Since for each dp[j], we can compute it using the entries that has index strictly less than j. Thus we can get the whole dp array inductively.

The running time of this algorithm should be $O(Wn)$, since to get dp[1], we cost O(n), and for each dp[i] with i > 1, we still cost O(n). So totally, from 1 to W, the time we need is O(Wn).