```
goodwill(S, B, F):
      Initialize 2-D array X with dimension (S+1)*(B+1), indexed [0...S, 0...B].
      Assign each block with initial value of 0.  // base case
      for i = 1 to F do
            for s = S down to s_i do
                  for b = B down to b_i do
                        // recurrence expression
                        if X[s-s_i, b-b_i] + g_i > X[s, b] then
                              X[s, b] = X[s-s_i, b-b_i] + g_i
```

Each time the recurrence expression is evaluated, we are trying to know if project i will provide a higher goodwill points. And the sub-problem is finding the maximum goodwill for $\boldsymbol{S-s}$ students and $\boldsymbol{B-b}$ buses, with projects being the subset of F without project i, recursively.

After the first for-loop (i = 1…F) is finished, we have a table of maximum number of goodwill points given the input. To find the actual project that we have to do, we need to backtrack from X[S, B]

```
      R <- empty set
      f <- set of integers from 1 to F
      s <- S
      b <- B
      while X[s,b] > 0:
            for i in f do
                  if X[s-s_i, b-b_i] == X[s,b] - g_i then
                        s = s-s_i
                        b = b-b_i
                        f.remove(i)
                        R.add(i)
      return R
```

R is the set of integers that represent the projects we need to do to achieve maximum goodwill units.

The total runtime of this algorithm is O(FBS).

The first part of the algorithm, filling the table with nested for-loops takes O(FBS) time.

Assuming each project requires at least 1 student or 1 bus, in other words, there is no “free” projects. Then each backtracking step will decrement ***s*** and/or ***b*** by at least 1, meaning that the while loop in the backtracking part will execute at most **(B+S)** iterations. Making the time complexity of the second part to be O(F * (B+S)) which is smaller than O(FBS) when the value of B and S is large.