Julian Garcia
CSE 344: Hw5
I certify that every answer in this assignment is the result of my own work; that Ihave neither copied off the Internet nor from any one else's work; and I have not shared my answers or attempts at answers with anyone else.

Problem P_1: determine if a given set of positive integers has a subset whose sum is exactly half that of the entire set. For example, the set $S = \{2, 5, 12, 13, 15, 19, 21, 29\}$ with sum = 116 does have such a subset but $\{2, 5, 14, 15, 20, 60\}$ does not.

Problem P_2 (a generalization of the problem P_1): given an array S of size n, the sum of whose elements is M, determine for *every* integer i between 0 and M (inclusive), if i is a sum of some subset of S.

You need to articulate the four steps of a dynamic programming approach for P_2, and thus derive a solution for P_1. However, you are given a solution for the third step (a bottom-up computation) in the form of an algorithm EXISTSSUBSET. Unfortunately, it is *almost* correct! You must reverse-engineer it to discern the first two steps and correct the error in the algorithm.
Specifically, answer the following based on the approach in EXISTSSUBSET.

1. Characterize the structure of an optimal solution for P_2.
   (*Hint:* Suppose you were *given* the solution for a particular integer i in $[0 \cdot \cdot M]$, can you relate it to solution(s) to subproblems, e.g., for $j < i$?)
2. Express the optimal solution (for P_2) recursively in terms of solutions to subproblems.
3. Find and explain the error in the bottom-up computation strategy used by the algorithm EXISTSSUBSET.
4. Correct that error in the algorithm and explain your correction briefly.
5. What is the time complexity of the given algorithm in terms of n and M? What about your corrected algorithm?
6. Given a correct $A[0 \cdot \cdot M]$ array, how would you detect the elements of a subset of S whose sum is i for any i (when there is such a subset)?
7. Use your corrected algorithm to solve for problem P_1.

```
EXISTSSUBSET(S, M)
1   ▷ Input: S[1 · ·n] nonempty array of positive integers representing a set
2   ▷ Input: M: sum of the elements of S
3   ▷ Output: A[0 · · · M]: zero-indexed boolean array; A[i] = TRUE iff ∃S′ : (S′ ⊆ S) ∧ sum(S′) = i
4   n ← S.size
5   A.startindex = 0
6   A.size = M + 1
7   A[0] ← TRUE
8   for i ← 1 to M do
9       A[i] ← FALSE
10  for i ← 1 to n do
11      for j ← 0 to M do
12          if A[j] = TRUE then
13              A[j + S[i]] ← TRUE
14  return A
```

1. To begin to solve this I drew a matrix of what the solution would look like for a set {2,3,7,8,10} up to an i of 10

	0	1	2	3	4	5	6	7	8	9	10	11
2	T	F	T	F	F	F	F	F	F	F	F	F
3	T	F	T	T	F	T	F	F	F	F	F	F
7	T	F	T	T	F	T	F	T	F	T	T	F
8	T	F	T	T	F	T	F	T	T	T	T	T
10	T	F	T	T	F	T	F	T	T	T	T	T

From constructing this, I could tell that after being given a particular solution for i, you can see that every row in the array up to i will be taken from the previous answers. So, for example, every tuple of row 2 (where the array value is 3) is identical to every tuple of row 3 (where the array value is 7) up to an i of 6, since 7 > 6. Every value afterwards can be calculated by going up one row, and subtracting the column number by the element.

So, to construct this optimally, we'd need a 2D boolean array, and we'd need to copy over the last row's answers (for whether i can be a sum of subsets) up to the current element, then answers for the i's beyond will be calculated by subtracting the current element from i and using the value of the result to see if the current tuple is true or false.

2. So we will create a 2D boolean array. The state subsum[j][i] will be true if there exists a subset of elements from A[0….j] with sum value = 'i'. Subsum will be true if we're looking at A[0] since the sum of the empty set is 0. The general approach for the problem is:
   if A[0]
   true
   if (A[j-1] > i)

```
        subsum[j][i] = subsum[j-1][i]
else
        subsum[j][i] = subsum[j-1][i] OR subsum[j-1][i-A[j-1]]
```

So if the current element has a value greater than the current sum value, we'll copy the answer for previous cases
And if the current sum value is greater than the 'jth' element we will see if any of previous states have already experienced the sum='i' OR any previous states experienced a value 'i – A[j]' which will solve our problem.

3. The main correction I think this algorithm needs comes in lines 12 and 13, since the truth assignment seems to run continually.
4. I would change lines 12 and 13 to:
   ```
   12:   if A[ j - S[ i ] ] = TRUE then
   13:       A[ j ]  = TRUE
   ```

5. The time complexity of the given algorithm is O(n*m), and it would still retain the same complexity after my correction.
6. You would detect those elements by how the previous row is built, since each rows truths would be built on which numbers are found to be sums, by the next i, you'd have to follow the trace of T's down to see which elements built the sum you're looking for.
7. HALFSUBSET(S,M)
   1. Input:S[1⋯n]nonempty array of positive integers representing a set
   2. Input:M: sum of the elements of S
   3. Output:A[0⋯M]: zero-indexed boolean array;A[i]= TRUE iff ∃ S′:(S′⊆S)∧sum(S′) =i
   4 n ←S.size
   5 A.startindex= 0
   6. M = M / 2
   7 A.size=M + 1
   8 A[0]←TRUE
   9 for i←1 to M do
   10 A[i]←FALSE
   11 for i←1 to n do
   12 for j← 0 to M do
   13 i if A[j - S[i]] = TRUE then
   14 A[j] = TRUE
   15 return A