1. **Input:** *data*: array of integers
   Input: *t*: target value
   Input: S: subset of *data* that sums to *t*
   Output: Verifies S is a subset sum

```
Algorithm: SSD
Check S ⊆ data        // O(n)
Set sum to 0          // O(1)
for s in S            // n iter. × O(1) = O(n)
   Add s to sum       // O(1)
end
if sum is equal to t  // O(1)
   return true        // O(1)
else
   return false       // O(1)
end
```

   Since SSD runs in polynomial time **O(n + n)**, SSD ∈ NP.

2. **Input:** data: array of integers
   Input: n: size of data
   Input: t: target value
   Output: S: subset of data that sums to t, or ; if no such set exists

```
Algorithm: SSReduction
if ~SSD(data, t) then                       // O(S(n))
   return ∅                                 // O(1)
end
sub = 0                                     // O(1)
top = n                                     // O(1)
while sub < top do                          // n iter. × O(S(n))
   if ~SSD(data[1..(top - 1)], t) then          // O(S(n))
      /* Add data[top] to subset */
      sub = sub + 1                         // O(1)
      Swap data[top] and data[sub]          // O(1)
   else
      /* Delete data[top] from array */
      top = top - 1                         // O(1)
   end
end
return data[1..sub]                         // O(1)
```

The worst-case complexity of SS is **`O(S(n) + nS(n))`** if the worst-case complexity of SSD is **`O(S(n))`**.