Computer Science
Spring 2021 Lecture 15:
Dynamic Programming:
Subset Sum

The Subset Sum Problem

T: target

Problem Statement: Given a set S of n positive integers, as well as a positive integer T, determine if there is a subset of S that sums to exactly T.

Example 1: $S = \{2, 3, 4\}$, T = 6, answer is "yes" **Example 2**: $S = \{2, 3, 5\}$, T = 6, answer is "no"

Subset Sum: recursive solution

As with any dynamic programming problem

- ► Try a recursive approach first
- Find a tautology, then list decisions

Recursive solution, attempt two

```
Sub(n,T): "does a subset of S[1...n] add to T?

// Tautology: if yes, S[n] is used or it is not

// if_no = Sub(n-1, T)

// if_yes = Sub(n-1, T-S[n])

// Now the code:

\longrightarrow if O==T return true, // \times \in \emptyset

else if O==n return false,

else return Sub(n-1,T) ||

(T \ge S[n]) & M Sub(n-1,T-S[n])
```

```
Subset Sum: iterative solution

dow a subset of S[i..i] add to j?

SubsetSum(i,j) // recursive for reference

if 0 = j then

return true

else if 0 = i then

return SubsetSum(i-1,j) OR

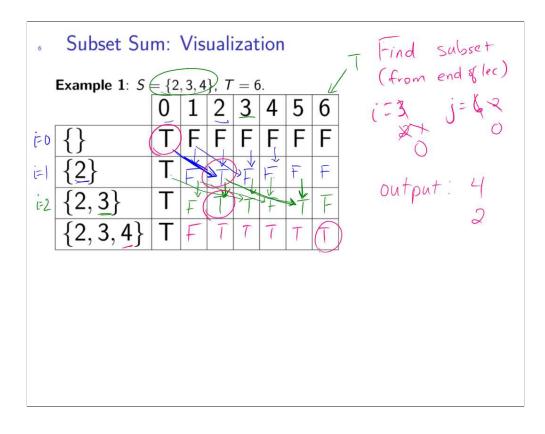
S[i] = 0 and SubsetSum(i-1,i-1) OR

S[i] = 0 and SubsetSum(i-1,i-1) Or

S[i] = 0 to S[i] = 0 to S[i] = 0

for i=0 to S[i] = 0

Sub S[i] = 0
```



Subset Sum: Running Time

```
SubsetSum(S[1...n], T) // iterative
  for i = 0 ... n do
     SUB[i, 0] = true
     SUB[0,j] = false \Theta(T)
  for j = 1 \dots T do
  for i = 1 \dots n do
     for j = 1 \dots T do
        Fill in SUB[i,j] in \mathcal{O}(1)
  return SUB[n, T]
```

▶ What is the running time of Subset Sum?

$$\Theta(nT)$$

Subset Sum: Running Time O(nT): Pseudo-polynomial SubsetSum(S[1...n], T) // iterative Size of input: n, log2T for $i = 0 \dots n$ do SUB[i, 0] = truefor $j = 1 \dots T$ do On (logT) 100) Would be polynomial SUB[0, j] = falsefor $i = 1 \dots n$ do for $j = 1 \dots T$ do

► Suppose we double the size of S, but leave T alone. Will your algorithm scale well? <

Fill in SUB[i,j] in $\mathcal{O}(1)$

return SUB[n, T]

▶ Suppose we double the **size** of T, but leave S alone. Will your algorithm scale well?

Subset Sum: Find the Subset

```
SubsetSum(S[1...n], T) // iterative

for i = 0 ... n do

SUB[i, 0] = true

for j = 1 ... T do

SUB[0, j] = false

for i = 1 ... n do

for j = 1 ... T do

Fill in SUB[i, j] in O(1)

if SUB[n, T] is true then

if n \in I is true then

if n \in I is true then

output S[i]

Output S[i]

Output S[i]
```