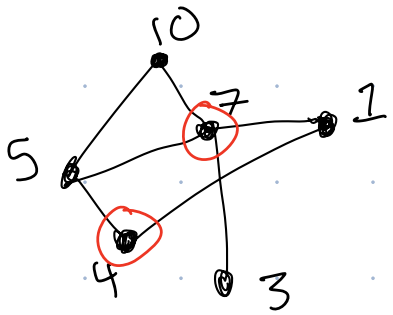
Max Weight Independent Set

Optimization Problem

Input: Graph $G = (V, E)$

Weights $w: V \to \mathbb{Z}^+$



Output: $S \subseteq V$ s.t.

- If $\{u, v\} \in E \quad \neg(u \in S \wedge v \in S)$ } Constraint
- Maximizes $W(S) = \sum_{v \in S} w(v)$

objective function

Applications:

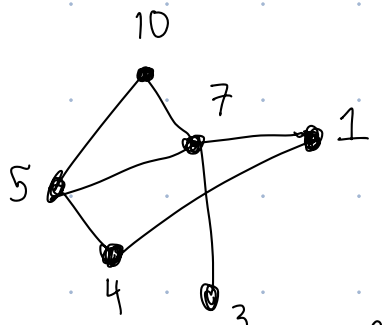
- Cell tower transmission
- Choose franchise locations
- Party invite
- Scheduling
- ~~House robbing~~

General Graph: HARD

Line Graph: Easy, with Dynamic Programming

Max Weight Independent Set (MWIS)

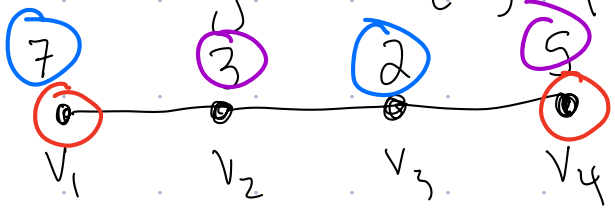
Input: Graph $G = (V, E)$
weights $w : V \to \mathbb{Z}^+$



Output: $S \subseteq V$ s.t.

- If $\{u, v\} \in E$, $\neg(u \in S \wedge v \in S)$ ⇐ "Independent Set Condition" ("Constraint")
  (not ↓, and ↙)
- Maximizes $W(S) = \sum_{v \in S} w(v)$ ⇐ "Weight of S"

What is max weight $W(S)$ for MWIS of this line graph:



A) 0 B) 8 C) 9 **D) 12**

Dynamic Prog. Approach

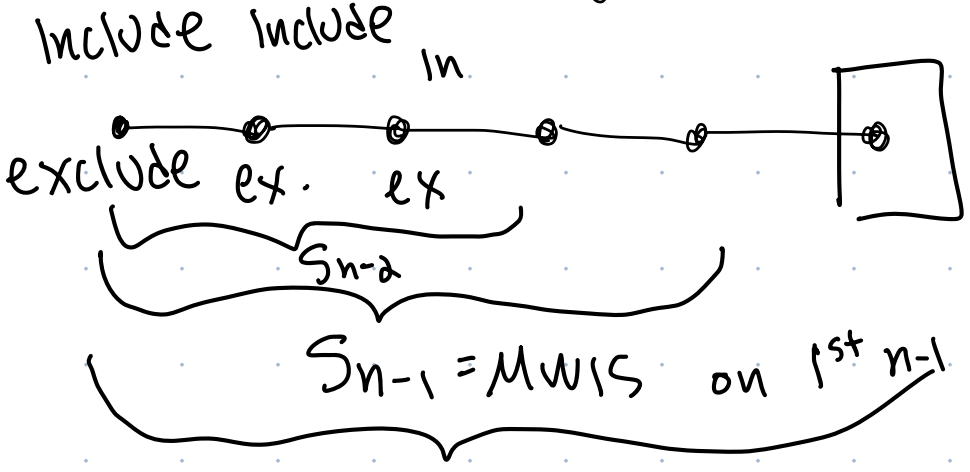
Recall: # of n bit strings with 2 consecutive ones

2 options: 0 or 1

How many strings?

How many strings?

To create a DP alg, (often) need to conceptualize the optimal solution as a sequence of choices

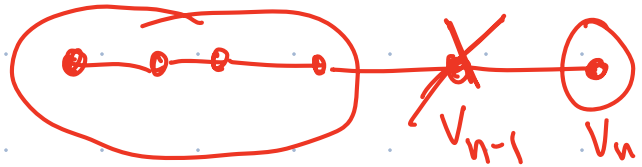


Final choice:

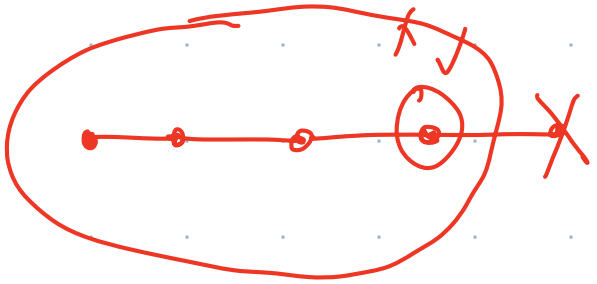
$V_n \in S$

$V_n \notin S$

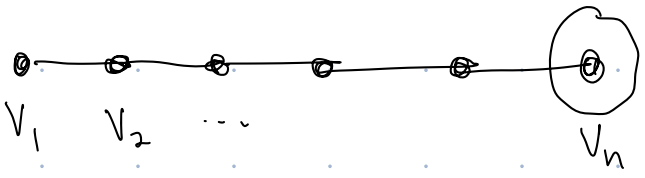
Dynamic Prog. Approach



or



MWIS



← 2 cases: $v_n \in S$ or $v_n \notin S$

Let's call S_i the MWIS of first i vertices

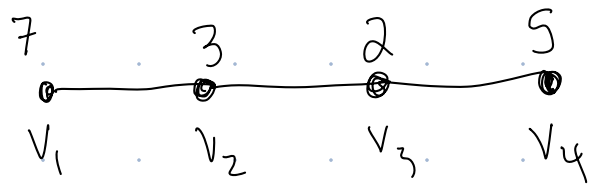
- $v_n \in S$, then $S_n = S_{n-2} \cup \{v_n\}$ Options: S_{n-1}, $S_{n-2} \cup \{v_n\}$
- $v_n \notin S$, then $S_n = S_{n-1}$ $S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$

Name, pronouns, best thing watched/listened to recently

Write pseudocode for brute force approach, analyze runtime

Brainstorm greedy, divide + conquer approaches

Brute Force $O(n2^n)$




```
MWIS(G = (V, E), w)
max S ← ∅
max W ← 0
For each set S ⊆ V:                    O(2^n)
    If S is I.S.:
        w ← W(S)
        if w > max W then
            max S ← S
            max W ← w
Return max S
```

```
S is I.S.:
For i ← 1 to n − 1:                    O(n)
    If v_i ∈ S and v_{i+1} ∈ S:
        Return False
Return true
```

```
W(S):                                  O(n)
W ← 0
For i ← 1 to n
    If v_i ∈ S
        W ← W + w(v_i)
return W
```

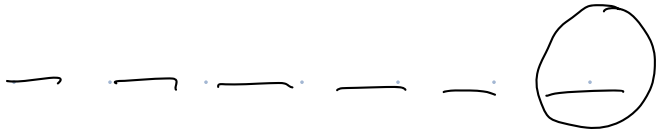
0 1 1 0

0 0 0 1

0 0 1 0

Dynamic Prog. Approach

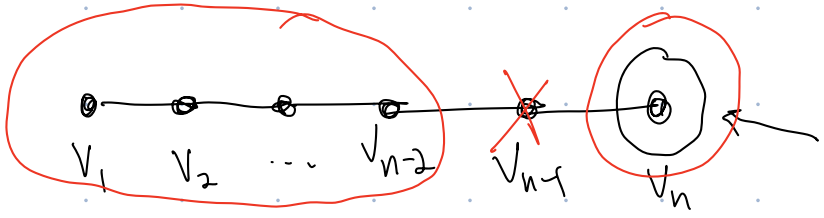
Recall: # of n bit strings with 2 consecutive ones



← 2 cases, 0 or 1

↙ T(n-1...) .... ↓ T(n-2)...

MWIS

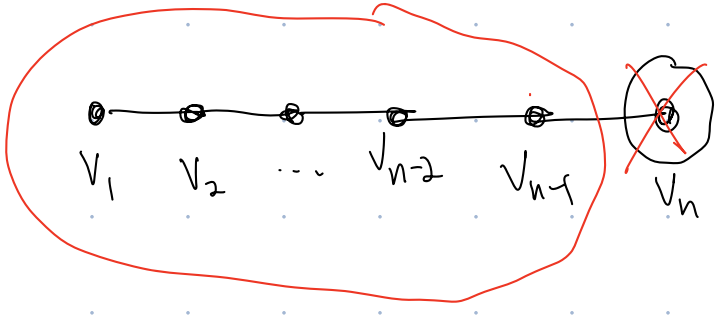


← 2 cases: $v_n \in S$ or $v_n \notin S$

Let's call S_i the MWIS of first i vertices

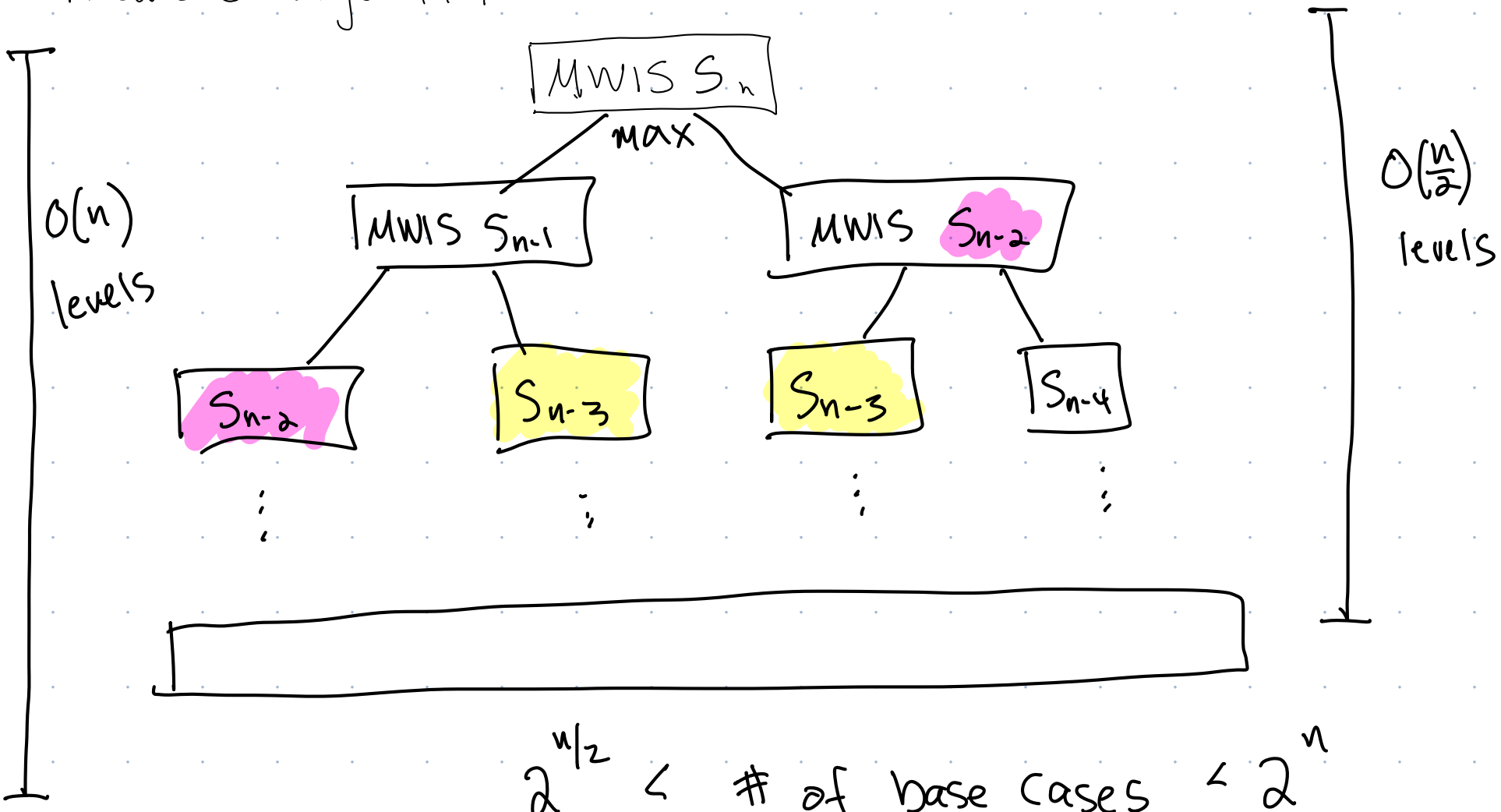
- $v_n \in S$, then $S_n = S_{n-2} \cup \{v_n\}$
- $v_n \notin S$, then $S_n = S_{n-1}$

Options:
S_{n-1}, S_{n-2}, $S_{n-1} \cup \{v_n\}$
$S_{n-2} \cup \{v_n\}$, $S_{n-2} \cup \{v_{n-1}\}$



$$S_n = \begin{cases} \text{Max Weigh} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ \text{Base case} \end{cases}$$

Recursive Algorithm:

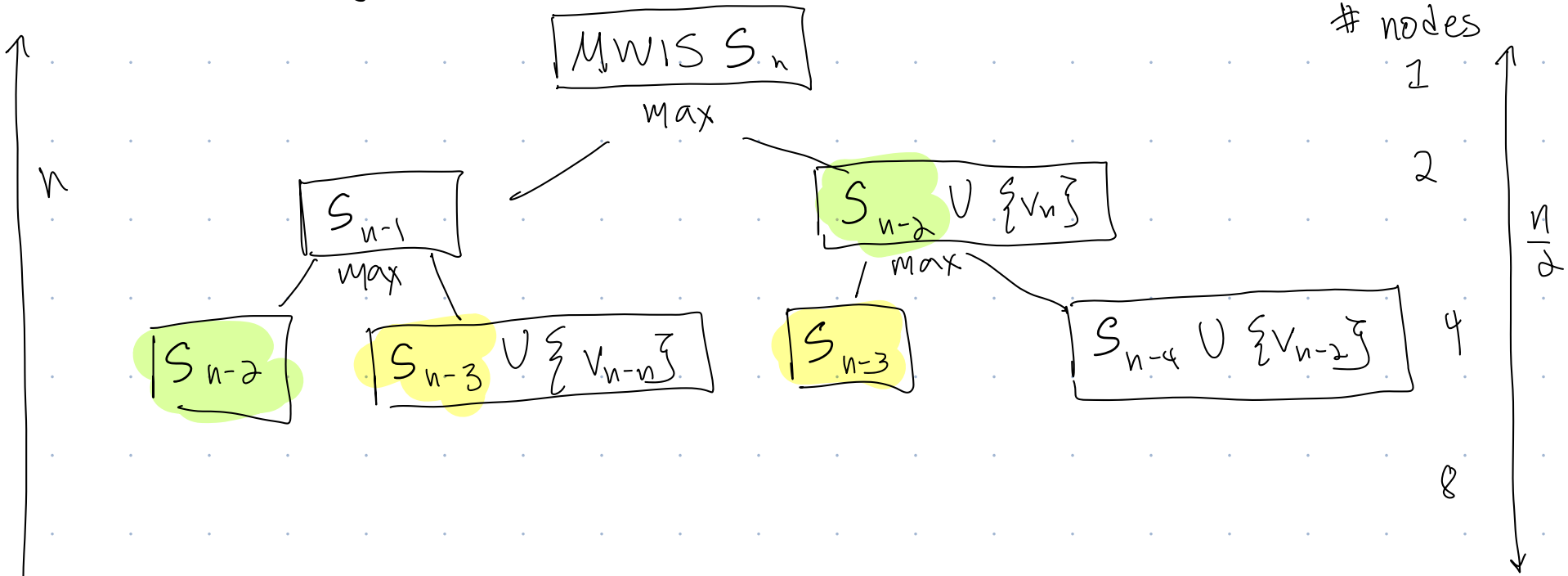


$2^{n/2} <$ # of base cases $< 2^n$

$$S_n = \begin{cases} \max\text{weight} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} & \leftarrow \text{Only 2 possible options, take larger!} \\ \text{base case} \end{cases}$$

★ And base case! Later...

Recursive Algorithm:



$$2^n > \#\text{ of nodes in tree} > 2^{n/2}$$

Each node requires $\Omega(1)$ time $\rightarrow$ $\Omega(2^{n/2})$ algorithm

($\Omega(1)$: asymptotic lower bound)

Bad!

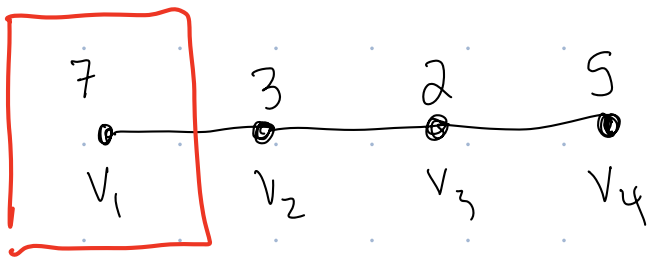
How many unique subproblems are there?

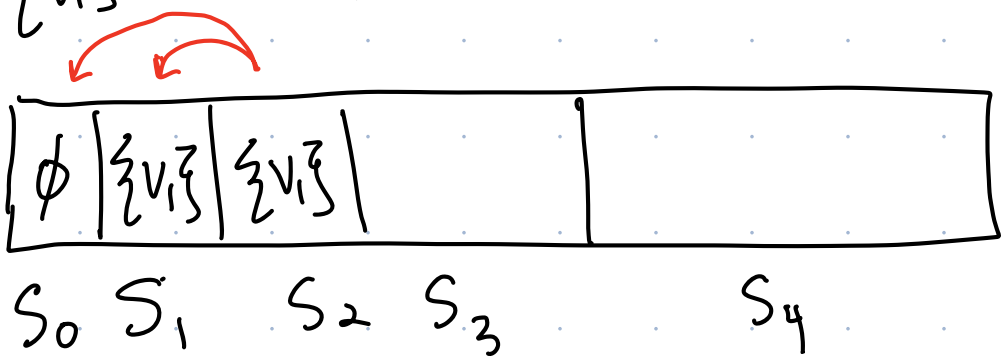
A) $\sqrt{n}$ B) $\frac{n}{2}$ C) n D) n^2

$S_n, S_{n-1}, S_{n-2} \cdots S_1$

Dynamic Programming Idea: Store subproblems in an array + look up

$$S_n = \begin{cases} \max\text{wt} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ \phi \text{ if } n = 0 \\ \{v_1\} \quad n = 1 \end{cases}$$



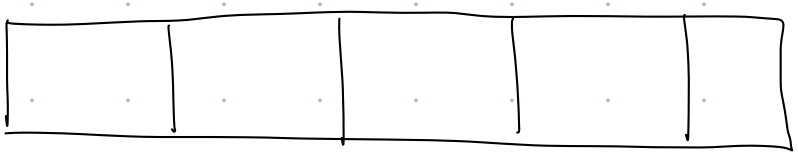


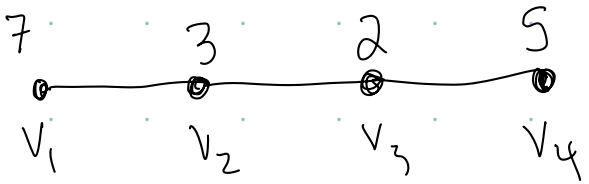
Trick: Start at smallest subproblem and go up

Trick: Create a size 0 problem

$$S_n = \begin{cases} n>1: \max_{wt} \begin{cases} S_{n-1} \\ S_{n-2} \cup \{v_n\} \end{cases} \\ \phi \text{ if } n=0 \\ \{v_1\} \text{ if } n=1 \end{cases}$$

array A





MWIS on a Line (G, w):

// Create array of objective function values

1.

2.

3.

4.

// Determine optimal set

ex:

A

S = { }

Runtime:

7. // Work backwards through FOR loop code

While $i \geq$ [ ] // include vertex v_i?

if A[i] = A[i-1]:

[ ]

else

[ ]

$A[i] \leftarrow \max\{A[i-1], A[i-2] + w(v_i)\}$

8 // Base case(s)

If i == [ ]

[ ]

9 Return S

Proof: Explanation of recurrence relation

Why "dynamic programming"