Design and Analysis of Algorithms

CS3230

Lecture 6
Dynamic Programming

Eric Han

Sanjay Jain

Warut Suksompong

Midterm

- Midterm next week (Mar 13)
- Today's lecture won't be covered on the midterm, but will be covered on the final exam.

Fibonacci Number F(n)

- F(0) = 0
- F(1) = 1
- F(n) = F(n-1) + F(n-2) for n>1

Problem: Given n, m, compute $F(n) \bmod m$
- Recursive algorithm
- Iterative algorithm

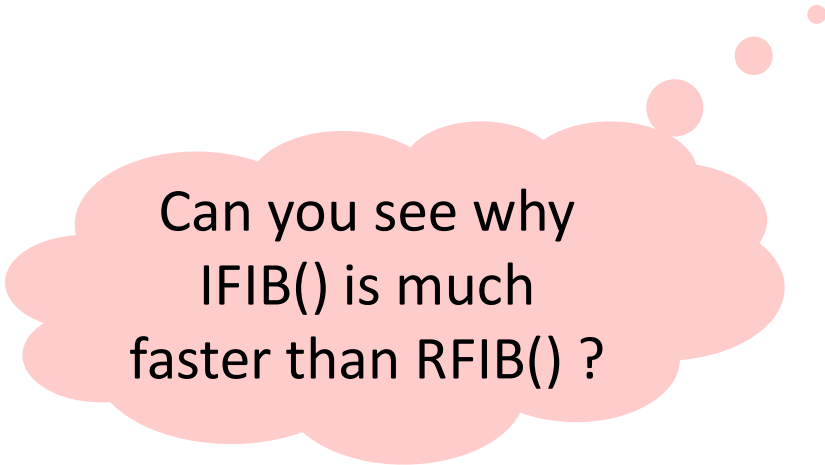# Two algorithms for Fibonacci (mod m)

Recursive Algorithm

RFIB(n,m) {
    if n=0 return 0;
    else if n=1 return 1;
    else return((RFIB(n-1) + RFIB(n-2)) mod m);
}

Iterative Algorithm

IFIB(n,m) {
    if n=0 **return 0**;
    else if n=1 **return 1**;
    else {
        a ← 0; b← 1;
        For(i=2 to n) **do**
        {
        temp ← b;
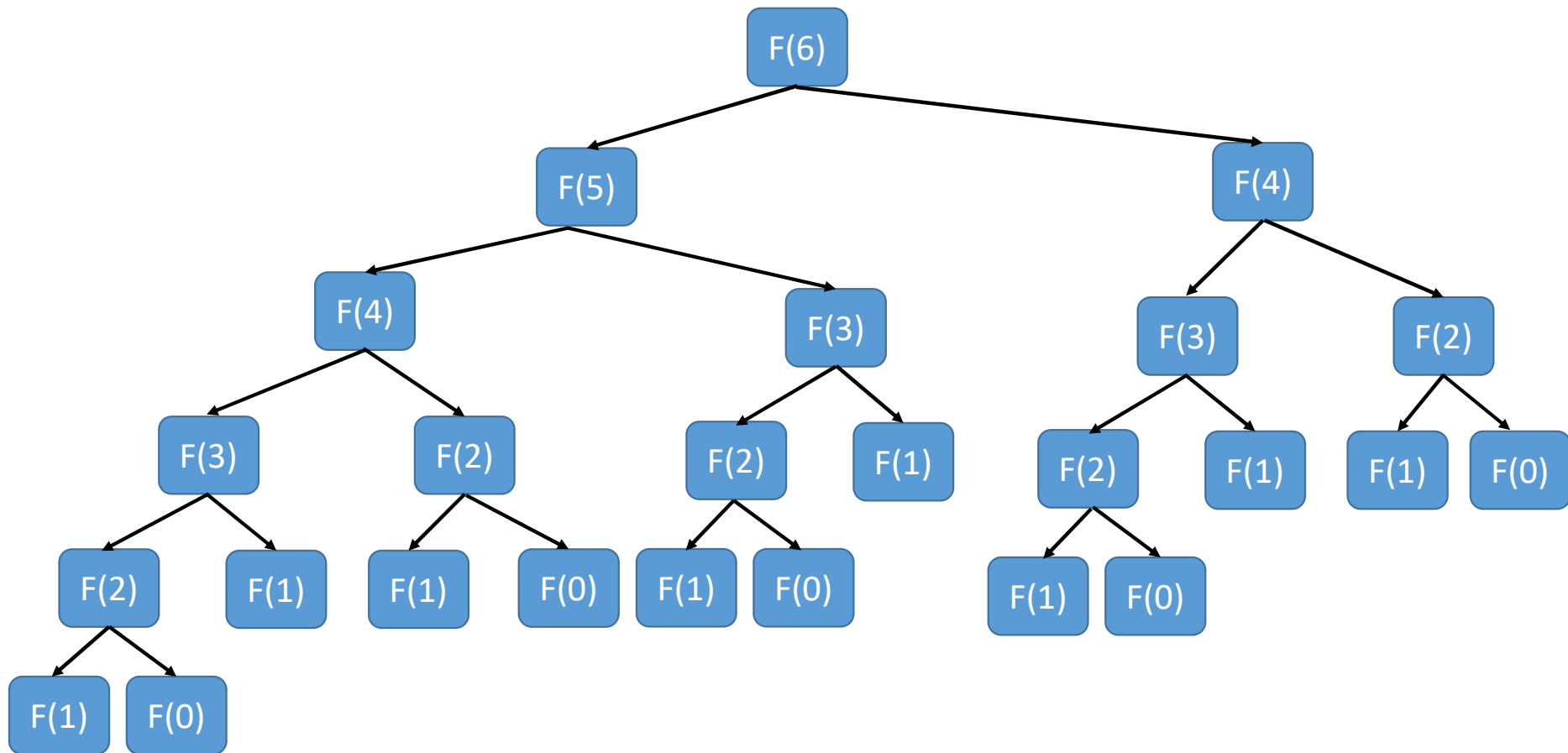   b← (a+b) mod m;
        a← temp; }
    }
    return b;}

Compare two algorithms for F(n) mod m

➤ No. of instructions by recursive algorithm **RFIB**(n,m) is $\geq 2^{(n-2)/2}$ (**exponential in n**)

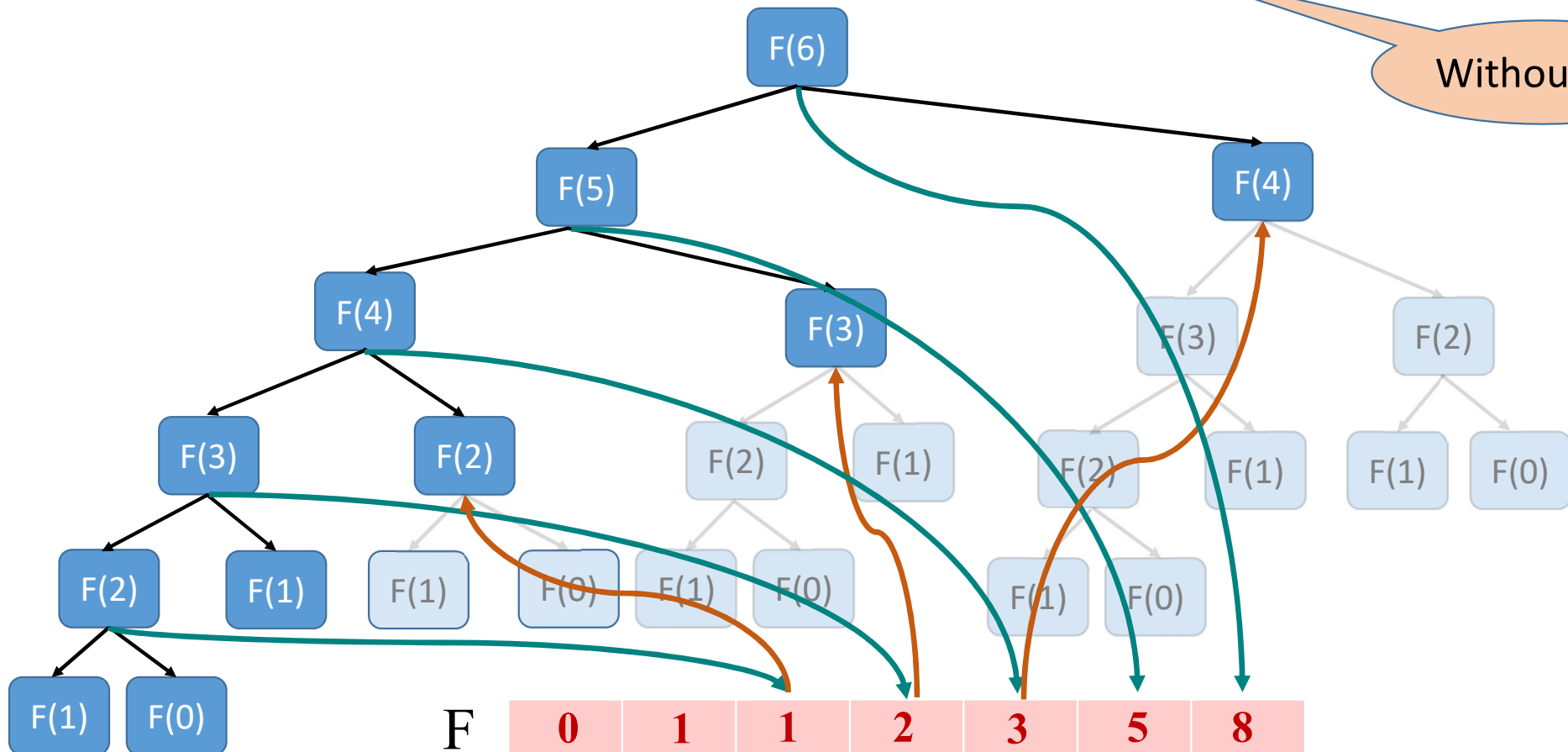➤ No. of instructions by iterative algorithm **IFIB**(n,m) is $\approx 5n$ (**linear in n**)

Can you see why
IFIB() is much
faster than RFIB() ?

Recursion tree for F(n)

Pruning recursion tree by memoization

Longest Common Subsequence

Applications in Computational Biology, Text Processing and many more
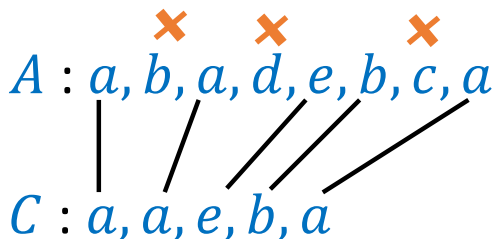
What is a subsequence?

Sequence $A : a_1, a_2, ..., a_n$

Can be stored in an array $A[1..n]$, $A[j]$: ? a_j $A[1..k]$: ? $a_1, a_2, ..., a_k$

Definition: C is said to be a **subsequence** of A if we can obtain C by removing zero or more elements from A.

Example: $A : a, b, a, d, e, b, c, a$

$C : a, a, e, b, a$

A more formal definition:

C is a **subsequence** of A if there exist k integers $1 \leq i_1 < \cdots < i_k \leq n$ such that
for all $1 \leq j \leq k$: $C[j] = A[i_j]$ (and C has length k)

Longest Common Subsequence - Definition

Given : Two sequences $A[1..n]$ and $B[1..m]$,

Aim : To compute a (not "the") <u>longest</u> sequence C such that

C is a subsequence of both A and B

Example: A : $a\ a\ s\ b\ d\ e\ s\ c\ b\ d$

B : $a\ c\ b\ s\ d\ c\ d\ e\ b$

Answer: $a\ s\ d\ e\ b$

Question : How to compute a LCS of A and B efficiently.

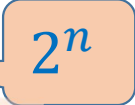# Finding LCS: Trivial Brute-Force Solution

Given : Two sequences $A[1..n]$ and $B[1..m]$

$$A : a_1, a_2, ..., a_n$$
$$B : b_1, b_2, ..., b_m$$

- Check <u>all the possible subsequences</u> of A to see if it is <u>also a subsequence</u> of B, and then output a <u>longest</u> one.

Analysis:

- Checking whether a particular subsequence of A is a subsequence of B takes $O(m)$ time.

- How many possible subsequences of A are there? 2^n

  (Each bit-vector of length n determines a distinct subsequence)

- So total time = $O(m2^n)$

Can we do better?

Finding LCS: Recursive Formulation

Given : Two sequences $A[1..n]$ and $B[1..m]$

$$A : a_1, a_2, ..., a_n$$
$$B : b_1, b_2, ..., b_m$$

Notation for recursive formulation:

LCS(i, j): Longest common subsequence of $A[1..i]$ and $B[1..j]$

Aim : To express **LCS**(i, j) recursively.

Base Cases:

LCS$(i, 0)$ = ε (empty sequence)   for all i

LCS$(0, j)$ = ε (empty sequence)   for all j

Since one of the sequences is **empty**

Recursive Formulation of LCS(n,m)

$a_1 \quad a_2 \quad a_3 \qquad \ldots \qquad a_{n-1} \; a_n$

$b_1 \quad b_2 \; \ldots \qquad\qquad b_{m-1} b_m$

How does LCS(n,m) look like when $a_n = b_m$?

$a_1 \quad a_2 \quad a_3 \quad \ldots \quad a_{n-1} \; a_n$

$b_1 \quad b_2 \; \ldots \quad b_{m-1} \, b_m$
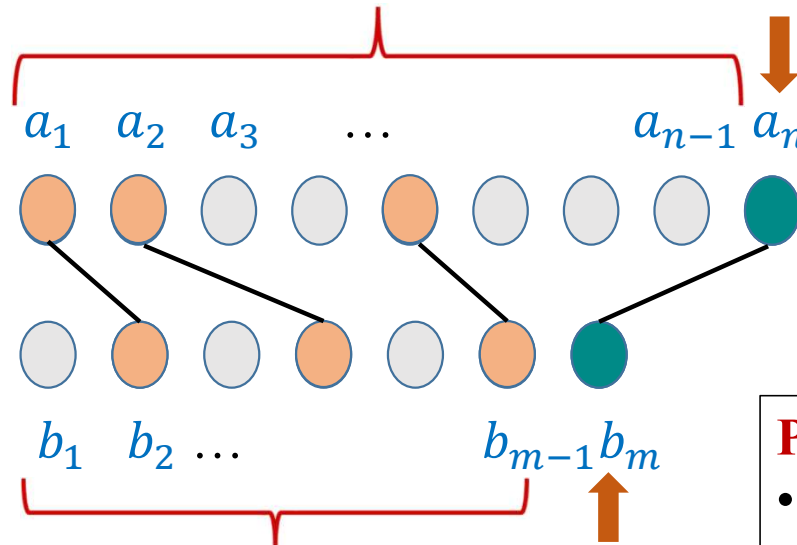
Intuition: **LCS**(n,m) should terminate with a_n

Lemma: If $a_n = b_m$ then
$$\mathbf{LCS}(n,m) = \mathbf{LCS}(n-1,m-1) \; :: \; a_n$$

Proof Idea:
- LCS(n,m) must terminate with the same symbol as a_n; otherwise we could **extend the solution** by concatenating a_n
- Observe, it is fine to **match** a_n with b_m

How does LCS(n,m) look like when $a_n = b_m$?
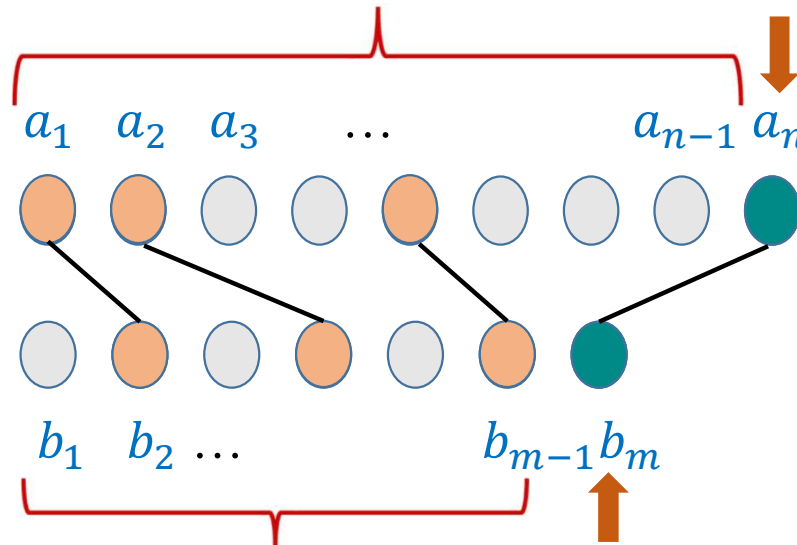
Proof: (Proof by contradiction)

- If the last symbol in $S = \text{LCS}(n,m)$ is <u>not the same</u> as $a_n (= b_m)$, then that <u>last symbol must be part of</u> $a_1, \cdots, a_{n-1}$ and $b_1, \cdots, b_{m-1}$.

- So, S is actually a subsequence of $a_1, \cdots, a_{n-1}$ and $b_1, \cdots, b_{m-1}$.

How does LCS(n,m) look like when $a_n = b_m$?

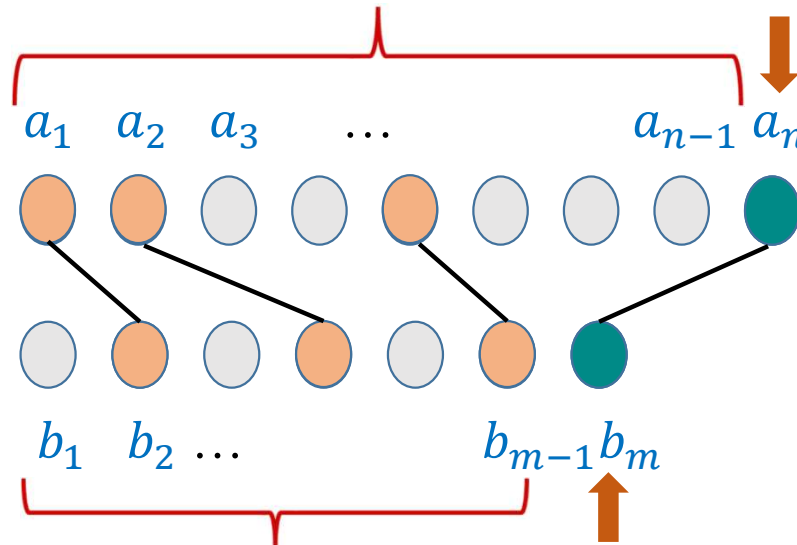This type of argument is also referred to as **cut-and-paste** argument

Proof: (Proof by contradiction)
- Now we can append a_n with S (i.e., $S :: a_n$) and get a subsequence of <u>length one more</u>
- Observe, it is fine to **match** a_n with b_m
- Thus S cannot be the largest subsequence of $a_1, \cdots, a_n$ and $b_1, \cdots, b_m$ (Contradiction)
- So we conclude **LCS**(n, m) = **LCS**$(n-1, m-1) :: a_n$

How does LCS($\boldsymbol{n,m}$) look like when $\boldsymbol{a_n \neq b_m}$?
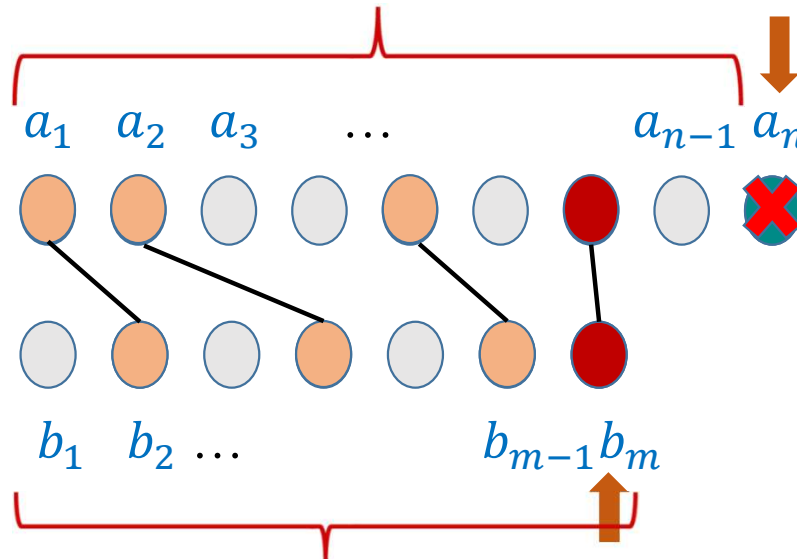
$a_1 \quad a_2 \quad a_3 \quad \ldots \quad a_{n-1} \; a_n$

$b_1 \quad b_2 \; \ldots \quad b_{m-1} b_m$

Intuition: Either a_n or b_m <u>is not</u> the last symbol of **LCS**(n,m)

Observation: If a_n <u>is not</u> the last symbol of **LCS**(n,m): **LCS**(n,m) = **LCS**($n-1,m$)

How does LCS(n,m) look like when $a_n \neq b_m$?

Intuition: Either a_n or b_m is not the last symbol of **LCS**(n,m)
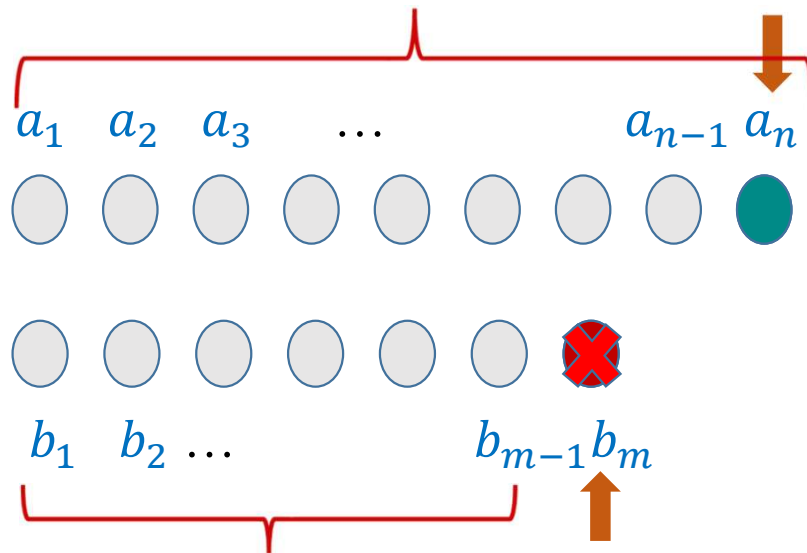
Observation: If a_n is not the last symbol of **LCS**(n,m): $\mathbf{LCS}(n,m) = \mathbf{LCS}(n-1,m)$

Observation: If b_m is not the last symbol of **LCS**(n,m): $\mathbf{LCS}(n,m) = \mathbf{LCS}(n,m-1)$

How does LCS($\boldsymbol{n,m}$) look like when $\boldsymbol{a_n \neq b_m}$?

$$a_1 \quad a_2 \quad a_3 \quad \ldots \quad a_{n-1} \; a_n$$

$$b_1 \quad b_2 \; \ldots \quad b_{m-1} b_m$$
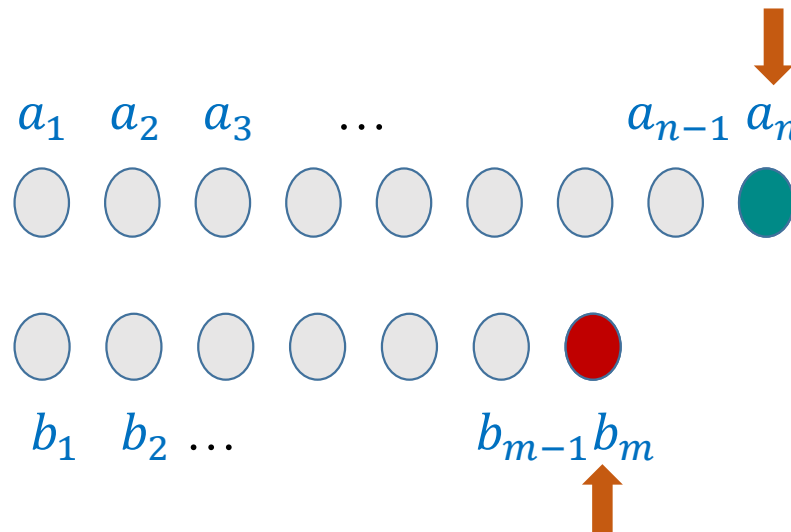
Intuition: Either a_n or b_m <u>is not</u> the last symbol of **LCS**(n,m)

Lemma: If $a_n \neq b_m$ then
LCS(n,m) is either **LCS**($n-1,m$) or **LCS**($n,m-1$)

Finding LCS: Recursive Formulation

Base Case:

LCS$(i, 0)$ = ε (empty sequence)   for all i

LCS$(0, j)$ = ε (empty sequence)   for all j

General Case:

If $a_n = b_m$ then   **LCS**(n,m) =   **LCS**$(n-1, m-1)$ :: a_n

If $a_n \neq b_m$ then   **LCS**(n,m) =   *larger* of **LCS**$(n, m-1)$ or **LCS**$(n-1, m)$
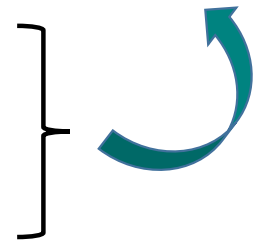
Simplified Problem: Find the length of LCS

Let $L(n,m)$: Length of LCS of $A[1..n]$ and $B[1..m]$

$L(n,m)$ = 0 if n or m is 0

Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems

Finding LCS: Recursive Formulation

Base Case:

LCS$(i, 0) = \emptyset$ for all i

LCS$(0, j) = \emptyset$ for all j

General Case:

If $a_n = b_m$ then   **LCS**$(n,m) =$ **LCS**$(n-1, m-1)$ :: a_n

If $a_n \neq b_m$ then   **LCS**$(n,m) =$ *larger* of **LCS**$(n, m-1)$ or   **LCS**$(n-1, m)$

Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems

Simplified Problem: Find the length of LCS

Let $L(n,m)$: Length of LCS of $A[1..n]$ and $B[1..m]$

$L(n,m) = 0$ if n or m is 0

If $a_n = b_m$ then $L(n,m)$ $=$ $L(n-1, m-1) + 1$

If $a_n \neq b_m$ then $L(n,m)$ $=$ **Max**$(L(n, m-1), L(n-1, m))$

Recursive algorithm for $L(n,m)$

$L(n,m)$
{   If ($n = 0$ or $m = 0$)
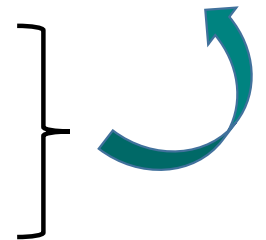
       return 0;

  Else

  {   If $a_n = b_m$ then

        return ($L(n-1,m-1) + 1$);

    Else

    { $l_1 \leftarrow L(n-1,m)$;

        $l_2 \leftarrow L(n,m-1)$;

        return **Max**(l_1, l_2);

    }

  }

}

$T(n,m)$: Worst case running time of $L(n,m)$

$$T(n,m) = T(n-1,m) + T(n,m-1)$$

An exercise from discrete math:
$$T(n,m) \geq \binom{n+m}{n} > 2^n \text{ (assuming } m \approx n)$$
Exponential !!

😞

But why ?
Let us explore

Recursive algorithm for $L(n,m)$

$L(n,m)$
{   If ($n = 0$ or $m = 0$)

      return 0;

  Else

  {   If $a_n = b_m$ then

      return $(L(n - 1, m - 1) + 1)$;

  Else

    { $l_1 \leftarrow L(n - 1, m)$;

      $l_2 \leftarrow L(n, m - 1)$;

      return **Max**(l_1, l_2);

    }

  }

}

$L(n,m)$

$L(n - 1, m)$ $L(n, m - 1)$

$L(n - 1, m - 1)$

$L(n - 2, m)$ $L(n - 1, m - 1)$ $L(n, m - 2)$

- Solving same sub-problem multiple times!!
- But how many **distinct** sub-problems are there?
- Only $(n + 1) \times (m + 1)$

Overlapping subproblems
A recursive solution contains a "small" number of distinct subproblems repeated many times

Recursive algorithm for $L(n,m)$

$L(n,m)$
{   If ($n = 0$ or $m = 0$)

    return 0;

 Else

 {   If $a_n = b_m$ then

    return ($L(n-1,m-1) + 1$);

 Else

  { $l_1 \leftarrow L(n-1,m)$;

    $l_2 \leftarrow L(n,m-1)$;

    return **Max**(l_1, l_2);

  }

 }

}

$L(n,m)$

$L(n-1,m)$ $L(n,m-1)$

$L(n-1,m-1)$

$L(n-2,m)$ $L(n-1,m-1)$ $L(n,m-2)$
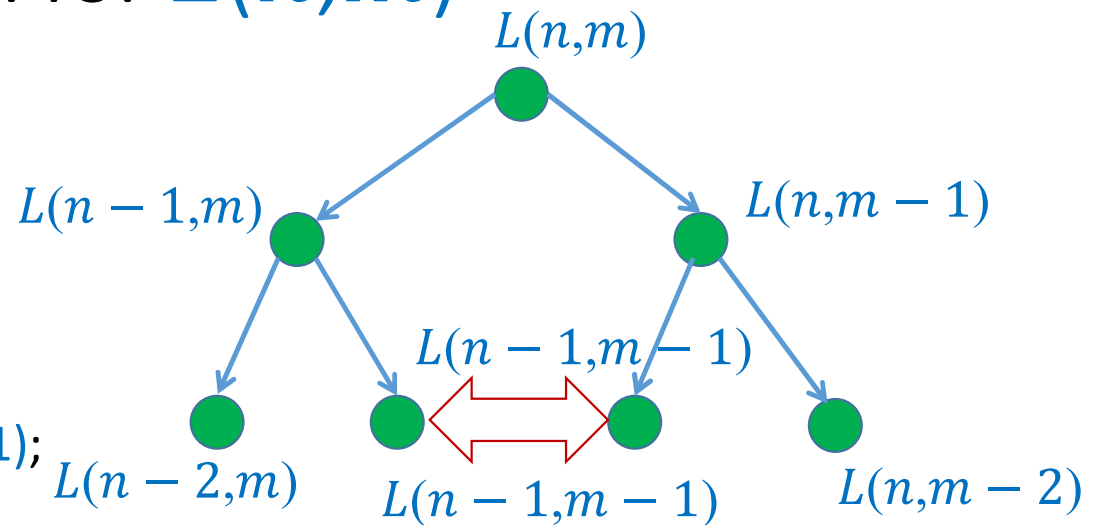
- Solving same sub-problem multiple times!!
- But how many sub-problems are there?
- Only $(n+1) \times (m+1)$
- Can we compute them efficiently?
- Get inspiration from algorithm for Fibonacci number!
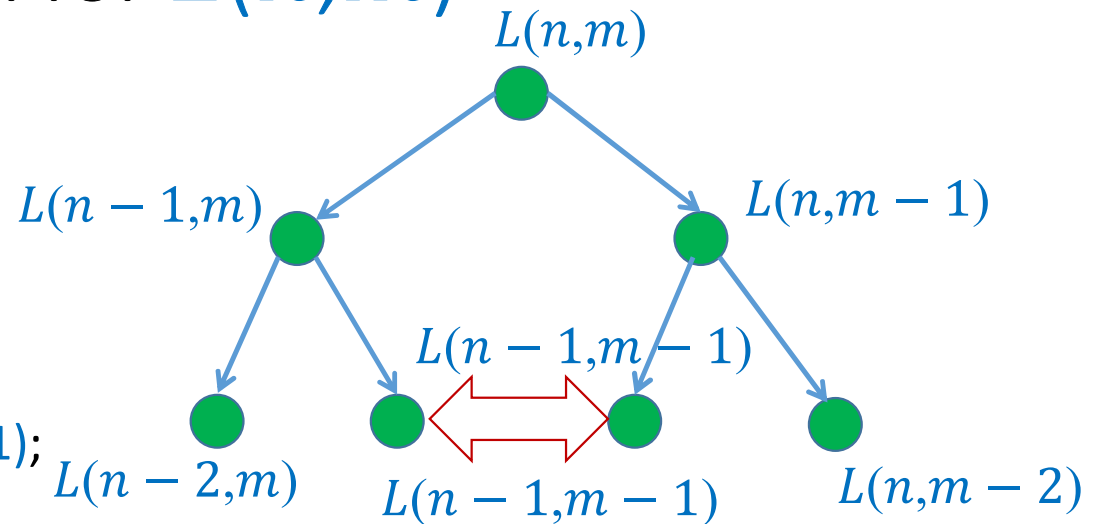
Recursive algorithm for $L(n,m)$

$L(n,m)$
```
{   If (n = 0 or m = 0)

        return 0;

   Else

   {    If a_n = b_m then

            return (L(n − 1, m − 1) + 1);

      Else

      {       l_1 ← L(n − 1, m) ;
              l_2 ← L(n, m − 1) ;
           return Max(l_1, l_2);

      }

   }

}
```

$T[i,j] = L(i,j)$

m 0						
0						
0						
0						
0						
1 0						
0 0	0	0	0	0	0	0

0   1   ... n

T

Dynamic Programming algorithm for $L(n,m)$

$L(n,m)$

{   **For** $(i = 0$ to $n)$ $T[i,0] \leftarrow 0$;

   For $(j = 0$ to $m)$ $T[0,j] \leftarrow 0$;

   For $(i = 1$ to $n)\{$

      For $(j = 1$ to $m)\{$

         If $a_i = b_j$ then

            $T[i,j] \leftarrow T[i-1, j-1] +1$;

         Else {

            $l_1 \leftarrow T[i-1,j]$;

            $l_2 \leftarrow T[i,j-1]$;

            $T[i,j] \leftarrow$ **Max**(l_1, l_2);

      } } }

}

$T[i,j] = L(i,j)$

Dynamic Programming algorithm for $L(n,m)$

$L(n,m)$

```
{   For (i = 0 to n)  T[i,0] ← 0;
    For (j = 0 to m)  T[0,j] ← 0;
    For (i = 1 to n){
        For (j = 1 to m){
            If a_i = b_j then
                T[i,j] ← T[i−1,j−1] +1;
            Else {
                l_1 ← T[i−1,j] ;
                l_2 ← T[i,j−1] ;
                T[i,j] ← Max(l_1, l_2);
            } } }
}
```

$T[i,j] = L(i,j)$

Time per table entry = $O(1)$
Total time = $O(nm)$
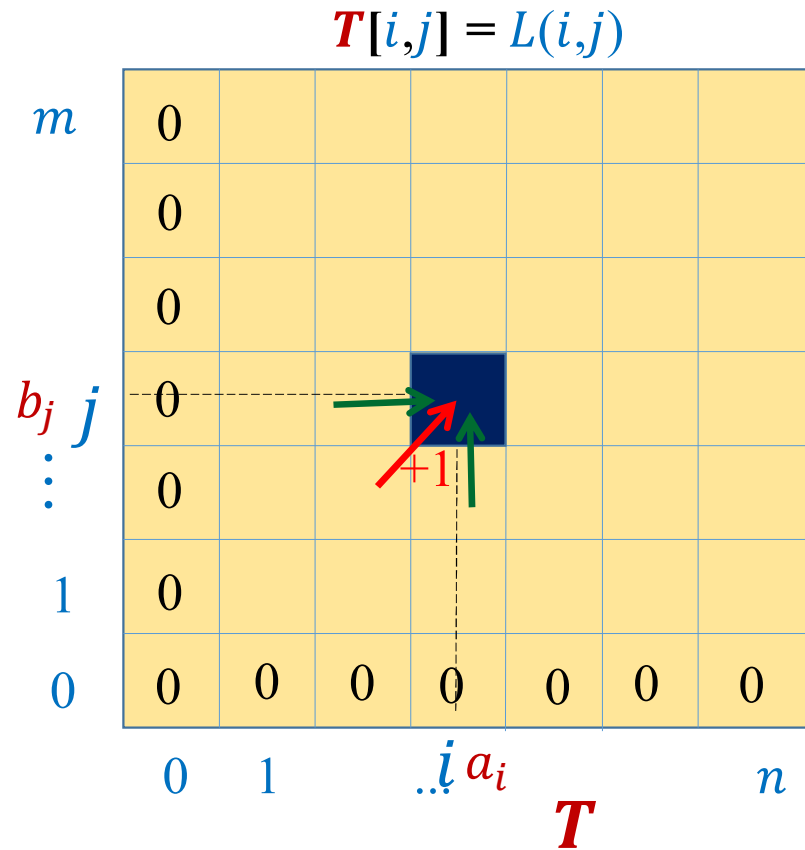
Dynamic Programming algorithm for $L(n,m)$

$L(n,m)$
{ **For** $(i = 0$ to $n)$ $T[i,0] \leftarrow 0;$

  For $(j = 0$ to $m)$ $T[0,j] \leftarrow 0;$

  For $(i = 1$ to $n)\{$

    For $(j = 1$ to $m)\{$

      If $a_i = b_j$ **then**

        $T[i,j] \leftarrow T[i-1,j-1] +1;$

      Else {

        $l_1 \leftarrow T[i-1,j] ;$

        $l_2 \leftarrow T[i,j-1] ;$

        $T[i,j] \leftarrow$ **Max**$(l_1, l_2);$

    } } }

}

$T[i,j] = L(i,j)$

	a	d	c	d	s	a
a	0	1	2	2	3	3	4
d	0	1	2	2	3	3	3
c	0	1	1	2	2	2	3
a	0	1	1	1	2	2	3
d	0	0	1	1	2	2	2
d	0	0	1	1	1	1	1
	0	0	0	0	0	0	0

T

Dynamic Programming algorithm for $L(n,m)$

$L(n,m)$

{ **For** $(i = 0$ to $n)$ $T[i,0] \leftarrow 0$;

   For $(j = 0$ to $m)$ $T[0,j] \leftarrow 0$;

   For $(j = 1$ to $m)\{$

      For $(i = 1$ to $n)\{$

        If $a_i = b_j$ then

           $T[i,j] \leftarrow T[i-1,j-1] +1$;

       Else {

          $l_1 \leftarrow T[i-1,j]$;

          $l_2 \leftarrow T[i,j-1]$;

          $T[i,j] \leftarrow$ **Max**(l_1, l_2);

      } } }

}

Note, you need to store the table T. So space requirement is $O(mn)$.

Exercise:

- How can you reduce the space requirement to $O(\min\{m, n\})$?

- Can you modify the algorithm so that you can output a LCS?

Dynamic Programming algorithm paradigm

- Expressing the solution recursively

- Overall there are only polynomial number of subproblems

- But there is a huge overlap among the subproblems. So the recursive algorithm takes exponential time (solving same subproblem multiple times)

- So we compute the recursive solution iteratively in a bottom-up fashion (like in the case of Fibonacci numbers). This avoids wastage of computation and leads to an efficient implementation

Knapsack Problem

Knapsack Problem
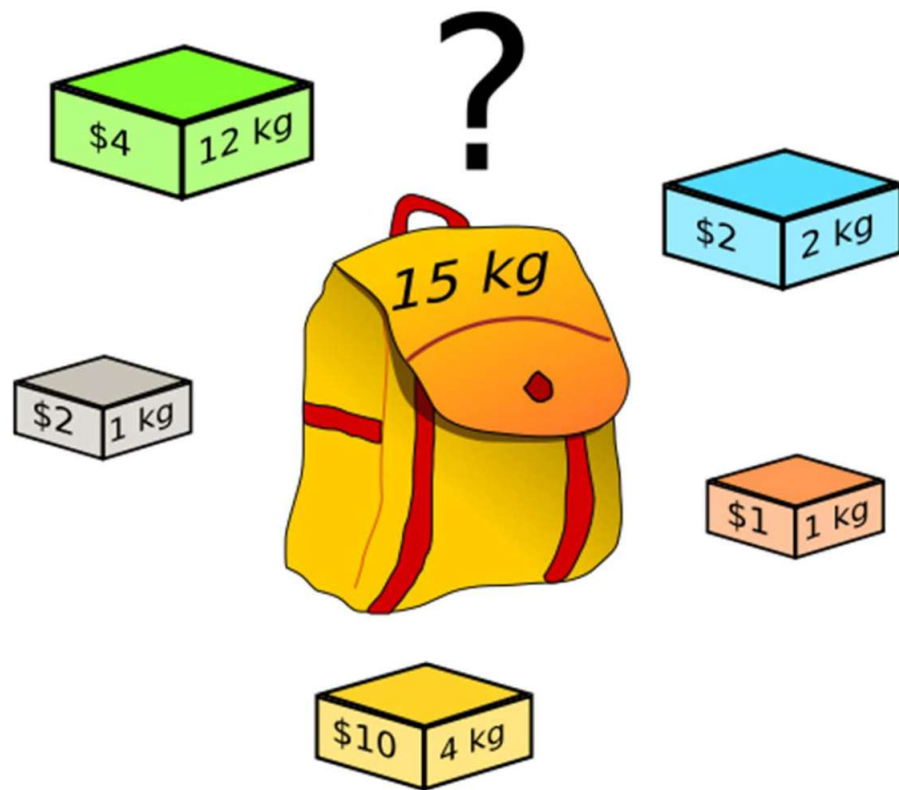
What is the maximum value you can get?

Formal Definition

KNAPSACK

Input:
$(w_1, v_1), (w_2, v_2), \ldots, (w_n, v_n)$, and W

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes

$\Sigma_{i \in S} v_i$ such that $\Sigma_{i \in S} w_i \leq W$

2^n subsets, so naïve algorithm is too costly!

Dynamic Programming

Problem: $(w_1, v_1), \ldots, (w_n, v_n), W$
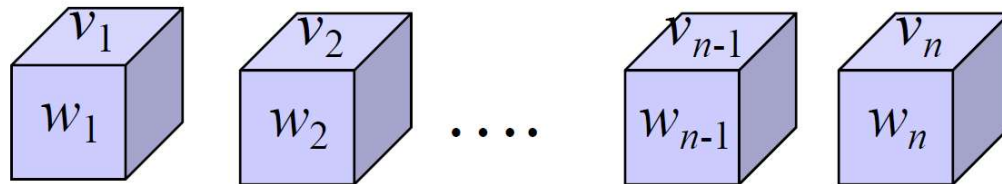
Is there optimal substructure?

Dynamic Programming

Case 1: Item n (the last one) is taken

Have optimal solution to subproblem
defined by $(w_1, v_1), \ldots, (w_{n-1}, v_{n-1}), W-w_n$
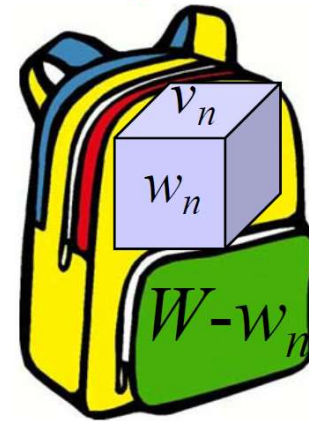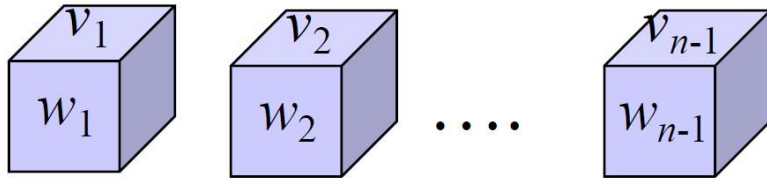
Dynamic Programming

Case 2: Item n (the last one) is **not** taken

Have optimal solution to subproblem
defined by $(w_1, v_1), \ldots, (w_{n-1}, v_{n-1}), W$
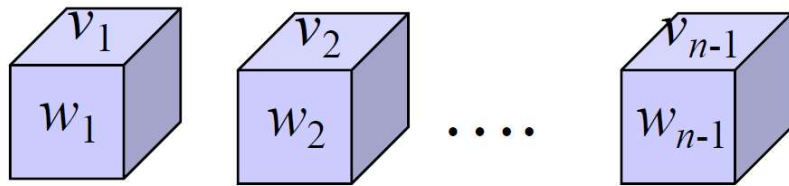
Otherwise, by *cut and paste*
argument, we can get a better solution

Recursive Solution

Let $m[i, j]$ be the maximum value that can be obtained using:

- a subset of items in $\{1, 2, \ldots, i\}$

- with total weight no more than j

$$m[i, j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \\ \max\{m[i-1, j-w_i] + v_i, m[i-1, j]\}, & \text{if } w_i \leq j \\ m[i-1, j], & \text{otherwise} \end{cases}$$

Pseudocode

$\textsc{Knapsack}(v, w, W)$:

    for $j = 0, \dots, W$:

        $m[0, j] \leftarrow 0$

    for $i = 1, \dots, n$:

        $m[i, 0] \leftarrow 0$

    ⟨*Recursive cases*⟩

    return $m[n, W]$

Pseudocode

$\langle Recursive\ cases \rangle$

for $i = 1, \dots, n$:

    for $j = 0, \dots, W$:

        if $j \geq w[i]$:

            $m[i, j] \leftarrow \max(m[i-1, j-w[i]] + v[i], m[i-1, j])$

        else:

            $m[i, j] \leftarrow m[i-1, j]$

$v[1..5] = \{4, 2, 10, 1, 2\}$

$w[1..5] = \{12, 1, 4, 1, 2\}$

$W = 15$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	j
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0																
2	0																
3	0																
4	0																
5	0																

i

$$m[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{m[i-1, j - w_i] + v_i, m[i-1, j]\} & \text{if } w_i \leq j \\ m[i-1, j] & \text{otherwise} \end{cases}$$

$v[1..5] = \{4, 2, 10, 1, 2\}$

$w[1..5] = \{12, 1, 4, 1, 2\}$

$W = 15$
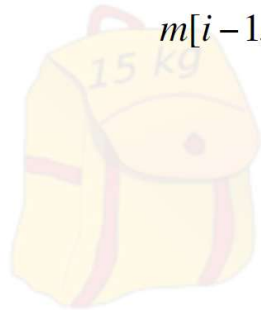
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	j
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	4				
2	0																
3	0																
4	0																
5	0																

i

$$m[i,j'] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{m[i-1,j-w_i]+v_i, m[i-1,j']\} & \text{if } w_i \leq j \\ m[i-1,j] & \text{otherwise} \end{cases}$$

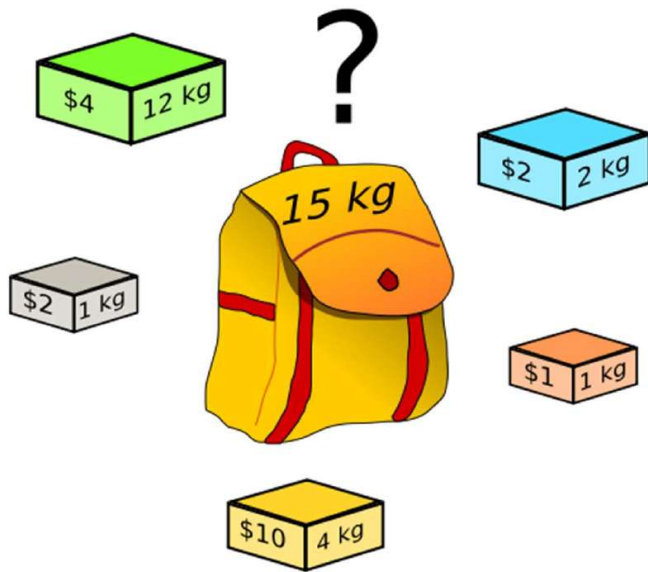$v[1..5] = \{4, 2, 10, 1, 2\}$

$w[1..5] = \{12, 1, 4, 1, 2\}$

$W = 15$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	j
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4	
2	0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6	
3	0	2	2	2	10	**12**											
4	0																
5	0																

i

42

$$m[i,j\,'] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{m[i-1,j-w_i]+v_i, m[i-1,j\,']\} & \text{if } w_i \leq j \\ m[i-1,j\,'] & \text{otherwise} \end{cases}$$

$v[1..5] = \{4, 2, 10, 1, 2\}$

$w[1..5] = \{12, 1, 4, 1, 2\}$

$W = 15$

i\j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6
3	0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12
4	0	2	3	3	10	**12**										
5	0															

$$m[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{m[i-1, j - w_i] + v_i, m[i-1, j]\} & \text{if } w_i \leq j \\ m[i-1, j] & \text{otherwise} \end{cases}$$
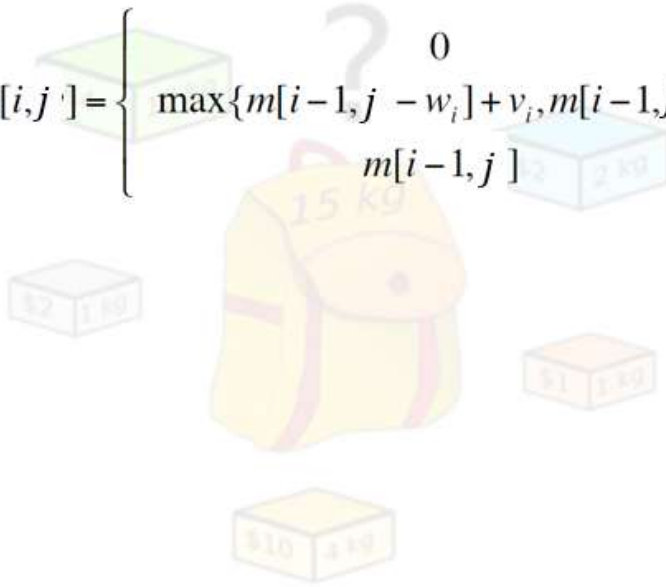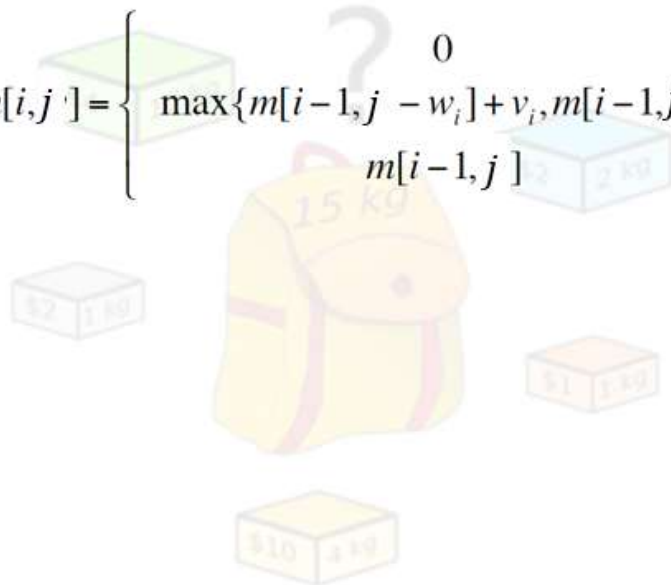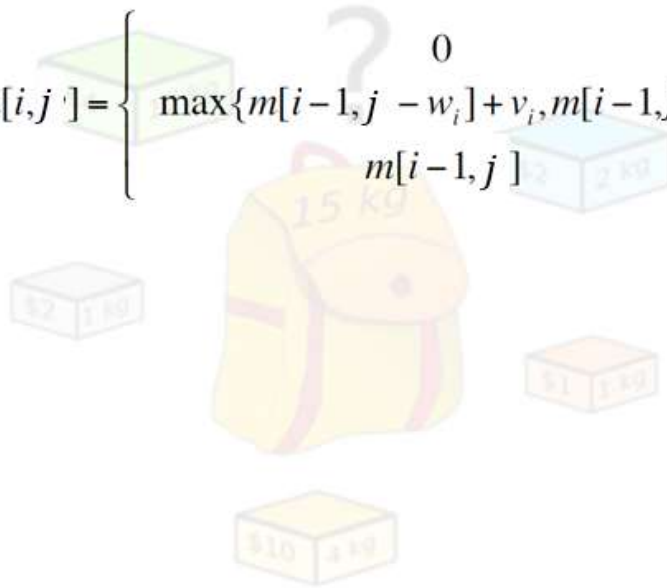
$v[1..5] = \{4, 2, 10, 1, 2\}$

$w[1..5] = \{12, 1, 4, 1, 2\}$

$W = 15$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	j
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4	
2	0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6	
3	0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12	
4	0	2	3	3	10	12	13	13	13	13	13	13	13	13	13	13	
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	**15**	

i

Pseudocode

⟨*Recursive cases*⟩

for $i = 1, \ldots, n$:

      for $j = 0, \ldots, W$:

            if $j \geq w[i]$:

$$m[i, j] \leftarrow \max(m[i-1, j-w[i]] + v[i], m[i-1, j])$$

          else:

$$m[i, j] \leftarrow m[i-1, j]$$

> Time per table entry = $\boldsymbol{O(1)}$
> Total time = $\boldsymbol{O(nW)}$

Example: Changing Coins

We have n cents and need to get change in terms of denominations $d_1, d_2, \ldots, d_k$. The goal is to use the fewest total number of coins.

Example: If denominations are 25c, 10c, and 1c, then solution for $n = 30c$ should be 10c+10c+10c.

Let $M[j]$ be the fewest number of coins needed to change j cents. Write a recursive formula for $M[j]$ in terms of $M[i]$ with $i < j$.

Example: Changing Coins

Optimal substructure: Suppose $M[j] = t$, meaning that
$$j = d_{i_1} + d_{i_2} + \cdots + d_{i_t}$$
for some $i_1, \ldots, i_t \in \{1, \ldots, k\}$. Then, if $j' = d_{i_1} + d_{i_2} + \cdots + d_{i_{t-1}}$, $M[j'] = t - 1$, because otherwise if $M[j'] < t - 1$, by **cut-and-paste** argument, $M[j] < t$.

$$M[j] = \begin{cases} 1 + \min_{i \in [k]} M[j - d_i], & j > 0 \\ 0, & j = 0 \\ \infty, & j < 0 \end{cases}$$

Example: Changing Coins

Using the above, derive a DP algorithm to compute the minimum number of coins of denomination $d_1, \ldots, d_k$ needed to change n cents.

Example: Changing Coins

$\textsc{Num-coints-dp}(n, d)$:

    for $j = 0, \dots, n$:

        $M[j] \leftarrow \infty$

    $M[0] \leftarrow 0$

    for $j = 1, \dots, n$:

        for $i = 1, \dots, k$:

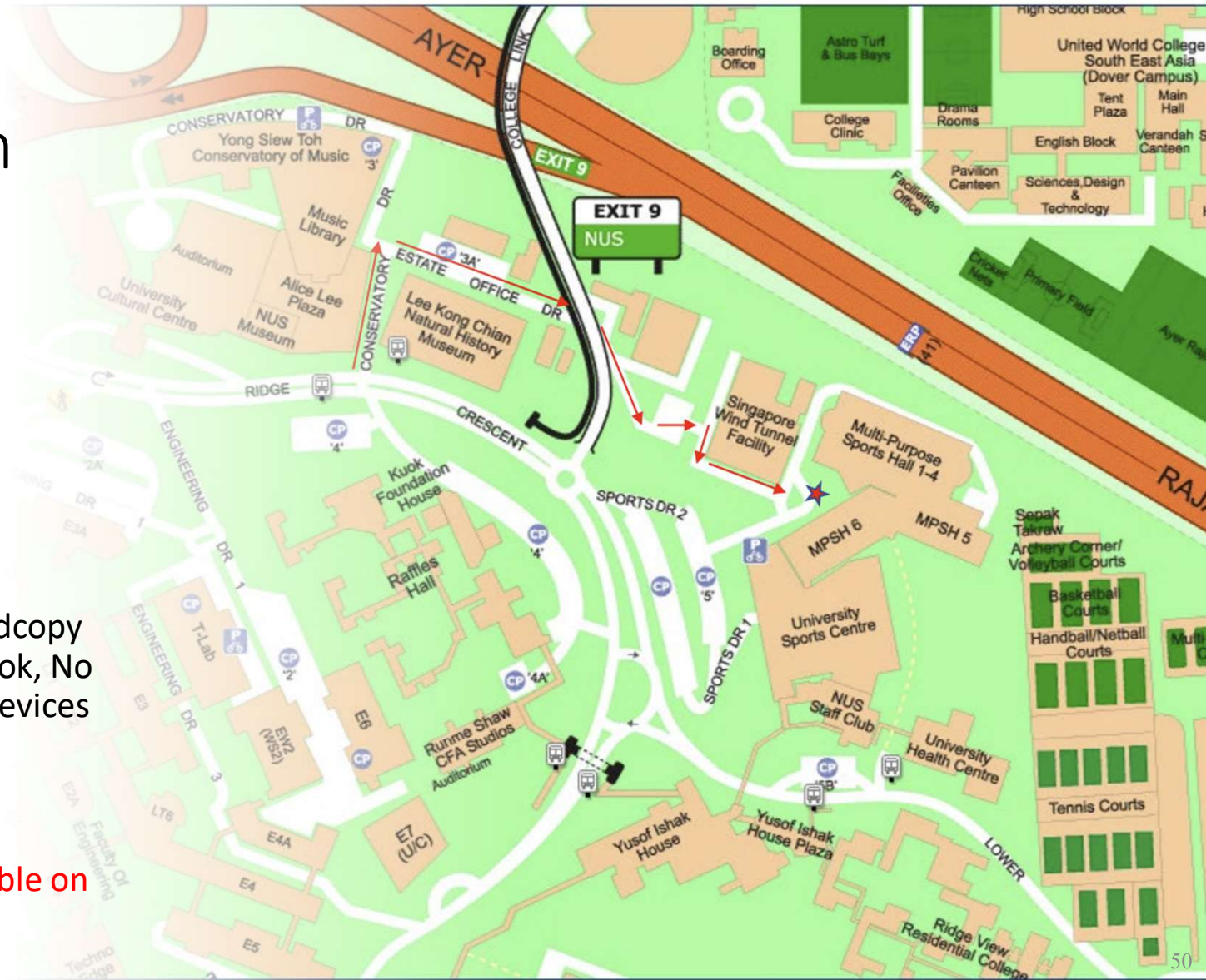   **if** $(j - d_i \geq 0) \wedge (M[j - d_i] + 1 < M[j])$:

                $M[j] \leftarrow M[j - d_i] + 1$

    return $M[n]$

Running time = $\boldsymbol{O(nk)}$

Midterm exam information

- 13/03/2025 (THU)

- MPSH 1B

- Arrive at venue by 14:00
  Exam starts: ~14:10
  Exam duration: 80 min

- Mode of assessment: Hardcopy (Pen and Paper), Open book, No calculators or electronic devices

- Bring 2B pencils

- Scratch paper

- Seating assignment available on Canvas

Some Information about Midterm Exam

- Covers all lectures before the recess week and the corresponding assignments and tutorials (including the one on March 3rd/4th).

- 10 MCQ (1.5 marks each) = 15 marks

- 3 proof questions (10 marks each) = 30 marks

- First proof question broken down into 2 related parts (5 marks each)

- Write only in the boxes/space provided for answers

- Second and third proof questions are single-part

- Leaving a proof question part blank will result in 1 mark (so leaving the whole exam blank will result in 4 marks)

- However, a totally wrong/irrelevant/nonsense answer will receive 0 marks

Acknowledgement

- The slides are modified from previous editions of this course and similar courses elsewhere.
- List of credits:
  - Surender Baswana
  - Arnab Bhattacharyya
  - Diptarka Chakraborty
  - Yi-Jun Chang
  - Erik Demaine
  - Steven Halim
  - Sanjay Jain
  - Wee Sun Lee
  - Charles Leiserson
  - Hon Wai Leong
  - Warut Suksompong
  - Wing-Kin Sung