How many ways are there to tile a $2 \times n$ board with dominoes?

How many ways to tile a $2 \times 7$ board

With these?
Two ways to fill the final column:

Tile\( (n) \) = Tile\( (n - 1) \) + Tile\( (n - 2) \)

Tile\( (0) \) = Tile\( (1) \) = 1
Homework

• Dynamic Programming Exercise Set 1 due Monday, October 12
  • Dynamic programming
  • Programming assignment (Java/Python)

• Hashing/Sorting Unit Quiz released Monday, October 12
Maximum Sum Subarray Problem

<table>
<thead>
<tr>
<th>5</th>
<th>8</th>
<th>-4</th>
<th>3</th>
<th>7</th>
<th>-15</th>
<th>2</th>
<th>8</th>
<th>-20</th>
<th>17</th>
<th>8</th>
<th>-50</th>
<th>-5</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Maximum sum contiguous subarray (MSCS) problem:
find the largest contiguous subarray that maximizes the sum of the values
Maximum Sum Contiguous Subarray (MSCS) problem:
find the largest contiguous subarray that maximizes the sum of the values
Divide and Conquer $\Theta(n \log n)$

<table>
<thead>
<tr>
<th>5</th>
<th>8</th>
<th>-4</th>
<th>3</th>
<th>7</th>
<th>-15</th>
<th>2</th>
<th>8</th>
<th>-20</th>
<th>17</th>
<th>8</th>
<th>-50</th>
<th>-5</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

- Divide in half
- Recursively solve on left
- Recursively solve on right

19
25
Divide and Conquer $\Theta(n \log n)$

Largest sum that ends here + Largest sum that starts here

Divide in half

Recursively solve on left

Recursively solve on right

Combine: Find largest sum that spans the cut
Divide and Conquer $\Theta(n \log n)$

Largest sum that ends here + Largest sum that starts here

Divide in half

Recursively solve on left

Recursively solve on right

Combine: Find largest sum that spans the cut

$T(n) = 2T(n/2) + \Theta(n) \in \Theta(n \log n)$
Unbalanced Divide and Conquer

**Divide**

- Make a subproblem of all but the last element
Unbalanced Divide and Conquer

Divide

- Make a subproblem of all but the last element

Conquer

- Find best subarray on the left ($BSL(n - 1)$)
- Find the best subarray ending at the divide ($BED(n - 1)$)

<table>
<thead>
<tr>
<th>5</th>
<th>8</th>
<th>-4</th>
<th>3</th>
<th>7</th>
<th>-15</th>
<th>2</th>
<th>8</th>
<th>-20</th>
<th>17</th>
<th>8</th>
<th>-50</th>
<th>-5</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Best subarray ending at the divide is empty
Unbalanced Divide and Conquer

Divide
- Make a subproblem of all but the last element

Conquer
- Find best subarray on the left ($BSL(n - 1)$)
- Find the best subarray ending at the divide ($BED(n - 1)$)

Combine
- Find the best subarray that “spans the divide” and output best among all candidates
Unbalanced Divide and Conquer

Best subarray that spans divide must include last element: $BED(n)$

- $BED(n) = \max(BED(n - 1) + arr[n], 0)$

Best subarray must either include or exclude the last element

- $BSL(n) = \max(BSL(n - 1), BED(n))$

<table>
<thead>
<tr>
<th>5</th>
<th>8</th>
<th>-4</th>
<th>3</th>
<th>7</th>
<th>-15</th>
<th>2</th>
<th>8</th>
<th>-20</th>
<th>17</th>
<th>8</th>
<th>-50</th>
<th>-5</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Combine

- Find the best subarray that “spans the divide” and output best among all candidates
Unbalanced Divide and Conquer

Divide
• Make a subproblem of all but the last element

Conquer
• Find best subarray on the left ($BSL(n - 1)$)
• Find the best subarray ending at the divide ($BED(n - 1)$)

Combine
• New best subarray ending at the divide:
  • $BED(n) = \max(BED(n - 1) + arr[n], 0)$
• New best on the left:
  • $BSL(n) = \max(BSL(n - 1), BED(n))$

If we compute $BED(n - 1)$ and $BSL(n - 1)$, then Combine is constant-time!
Unbalanced Divide and Conquer

\[
BED(n) = \max(BED(n - 1) + arr[n], 0)
\]
\[
BSL(n) = \max(BSL(n - 1), BED(n))
\]
Unbalanced Divide and Conquer

\[ B_{ED}(n) = \max(B_{ED}(n - 1) + arr[n], 0) \]
\[ B_{SL}(n) = \max(B_{SL}(n - 1), B_{ED}(n)) \]
Unbalanced Divide and Conquer

\[ \text{BED}(n) = \max(\text{BED}(n-1) + \text{arr}[n], 0) \]
\[ \text{BSL}(n) = \max(\text{BSL}(n-1), \text{BED}(n)) \]
Unbalanced Divide and Conquer

\[
BED(n) = \max(BED(n - 1) + arr[n], 0)
\]

\[
BSL(n) = \max(BSL(n - 1), BED(n))
\]
**Unbalanced Divide and Conquer**

- **Divide**
  - Find largest sum ending at the cut
  - Recursively solve on left

\[
BED(n) = \max(BED(n - 1) + arr[n], 0)
\]
\[
BSL(n) = \max(BSL(n - 1), BED(n))
\]
Unbalanced Divide and Conquer

\[ BED(n) = \max(BED(n - 1) + \text{arr}[n], 0) \]
\[ BSL(n) = \max(BSL(n - 1), BED(n)) \]
Unbalanced Divide and Conquer

Divide
- Find largest sum ending at the cut
  - $BED(n) = \max(BED(n - 1) + arr[n], 0)$
  - $BSL(n) = \max(BSL(n - 1), BED(n))$

Recursively solve on left
- $BSL(n) = \max(BSL(n - 1), BED(n))$
Unbalanced Divide and Conquer

Find largest sum ending at the cut

Recursively solve on left

\[ T(n) = T(n - 1) + \Theta(1) \in \Theta(n) \]

\[ \text{BED}(n) = \max(\text{BED}(n - 1) + \text{arr}[n], 0) \]

\[ \text{BSL}(n) = \max(\text{BSL}(n - 1), \text{BED}(n)) \]
Was Unbalanced Better?

Old:

\[ T(n) = 2T(n/2) + \Theta(n) \in \Theta(n \log n) \]

- We split into 2 problems of size \( n/2 \)
- Linear time combine (to find arrays that span the cut)

New:

\[ T(n) = T(n - 1) + T(1) + \Theta(1) \in \Theta(n) \]

- We split into 2 problems of size \( n - 1 \) and 1
- Constant time combine
Another Look at the Recursion

Divide

Find largest sum ending at the cut

Recursively solve on left

\[ \text{BED}(n) = \max(\text{BED}(n - 1) + \text{arr}[n], 0) \]
\[ \text{BSL}(n) = \max(\text{BSL}(n - 1), \text{BED}(n)) \]
Another Look at the Recursion

\[ \text{BED}(n) = \max(\text{BED}(n - 1) + \text{arr}[n], 0) \]

\[ \text{BSL}(n) = \max(\text{BSL}(n - 1), \text{BED}(n)) \]
Another Look at the Recursion

Divide

Find largest sum ending at the cut

Recursively solve on left

\[ \text{BED}(n) = \max(\text{BED}(n - 1) + \text{arr}[n], 0) \]
\[ \text{BSL}(n) = \max(\text{BSL}(n - 1), \text{BED}(n)) \]
Observation: No need to recurse! Just maintain two numbers and iterate from 1 to $n$: best value so far, best value ending at current position

\[
BED(n) = \max(BED(n - 1) + arr[n], 0)
\]

\[
BSL(n) = \max(BSL(n - 1), BED(n))
\]
Tiling Dominoes

How many ways are there to tile a $2 \times n$ board with dominoes?

How many ways to tile a $2 \times 7$ board

With these?
Tiling Dominoes

Two ways to fill the final column:

Tile\( (n) \) = Tile\( (n - 1) \) + Tile\( (n - 2) \)

Tile\( (0) = \) Tile\( (1) = 1 \)

0 1 2 3 4 5 6 7 8 13 21
How to compute $\text{Tile}(n)$?

```python
def tile(n):
    if n < 2:
        return 1
    return tile(n-1) + tile(n-2)
```

Problem?
Recursion Tree

Runtime: $\Omega(2^n)$
Recursion Tree

Runtime: $\Omega(2^n)$
But lots of redundant calls...

We only computed $n$ distinct values
Computing Tile($n$) with Memory ("Top Down")

initialize array $M$ of size $n$

tile($n$):
    if $n < 2$:
        return 1
    if $M[n]$ is filled:
        return $M[n]$
    $M[n] = \text{tile}(n-1) + \text{tile}(n-2)$
    return $M[n]$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$M[n]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Computing Tile(n) with Memory ("Top Down")

initialize array M of size n

\[
\text{tile}(n) : \\
\text{if } n < 2 : \\
\quad \text{return } 1 \\
\text{if } M[n] \text{ is filled:} \\
\quad \text{return } M[n] \\
M[n] = \text{tile}(n-1) + \text{tile}(n-2) \\
\text{return } M[n]
\]

Runtime: \(\Theta(n)\)

Bottom-Up: Can also iterate through \(M\) and fill in entries sequentially
Dynamic Programming

Requires **optimal substructure**
- Solution to **larger** problem contains the solutions to **smaller** ones ("overlapping subproblems")

**General idea:** Identify **recursive** structure of the problem and express solution to **larger** instances in terms of solutions to **smaller** instances

\[
\begin{align*}
\text{n - 1} \\
\text{\cellcolor{green!50}}
\end{align*}
\]

\[
\begin{align*}
\text{n - 2} \\
\text{\cellcolor{green!50} \cellcolor{yellow!50}}
\end{align*}
\]