Announcement: Homework 1 Solutions released on Piazza
Homework 2 Due: September 16th, 2019 @ 11:59PM on Gradescope
Recap: Master Theorem

- Derived by analyzing recursion tree

\[
T(n) = \begin{cases} 
1000000 & \text{if } n \leq 100 \\
 a \times T\left(\frac{n}{b} + 1\right) + n^c & \text{otherwise}
\end{cases}
\]

Obtain Geometric Series:

\[
T(n) = \Theta \left(n^c \sum_{i=1}^{\log_b n} \left(\frac{a}{b^c}\right)^i\right)
\]

- Key Ratio: \(a/b^c\)

\[
\log_b a \geq c \iff \frac{a}{b^c} \geq 1
\]

- Case 1: \(\left(\frac{a}{b^c}\right) < 1\)

\[
T(n) = \Theta(n^c)
\]

- Case 2: \(\left(\frac{a}{b^c}\right) = 1\)

\[
T(n) = \Theta(n^c \log n)
\]

- Case 3: \(\left(\frac{a}{b^c}\right) > 1\)

\[
T(n) = \Theta(n^{\log_b a})
\]
Other Form of Master Theorem

- What if \textit{MergeCost} is not exactly \( f(n) = n^c \)?
  
  - \( f(n) = n \log n \) or
  - \( f(n) = n^{\frac{1}{10} + \log b a} / \log(n) ? \)

\[
T(n) = \begin{cases} 
O(1) & \text{if } n \leq 100 \\
\alpha \times T \left(\frac{n}{b} + 50\right) + f(n) & \text{otherwise} 
\end{cases}
\]

Assume \( f(n) \geq 0 \)

\begin{align*}
\text{Case 1: } & f(n) = \Omega(n^{\varepsilon + \log_b a}) & T(n) = \Theta(f(n)) \\
\text{Case 2: } & f(n) = \Theta(n^{\log_b a \log^{k+1} n}) & (\text{assumes } k \geq 0) \\
\text{Case 3: } & f(n) = O(n^{\log_b a - \varepsilon}) & T(n) = \Theta(n^{\log_b a})
\end{align*}
Clicker Question

\[
T(n) \leq \begin{cases} 
1000000 & \text{if } n \leq 100 \\
a \times T\left(\frac{n}{b} + 50\right) + f(n) & \text{otherwise}
\end{cases}
\]

Assume \( f(n) \geq 0 \)

Case 1: \( f(n) = \Omega(n^{\varepsilon + \log b\ a}) \)

\[
T(n) = \Theta(f(n))
\]

Case 2: \( f(n) = \Theta(n^{\log b\ a \ log^k n}) \)

\[
T(n) = \Theta(n^{\log b\ a \ log^{k+1} n})
\]

(assumes \( k \geq 0 \))

Case 3: \( f(n) = o(n^{\log b\ a - \varepsilon}) \)

\[
T(n) = \Theta(n^{\log b\ a})
\]

Suppose that \( f(n) = n^{10+\log b\ a} / \log(n) \) above what is \( T(n) \)?

A. \( \Theta(f(n)) \)

B. \( \Theta(n^{\log b\ a \ log^{k+1} n}) \)

C. \( T(n) = \Theta(n^{\log b\ a}) \)

D. More info required
Suppose that \( f(n) = n^{\frac{1}{10} + \log_b a} / \log(n) \) above what is \( T(n) \)?

A. \( \Theta(f(n)) \)

B. \( \Theta(n^{\log_b a \log^{k+1} n}) \)

C. \( T(n) = \Theta(n^{\log_b a}) \)

D. More info required
Other Types of Recurrences

- $T(n) = T(n - 1) + 1$  \hspace{1cm} (Unroll: $T(n) = \Theta(n)$)
  
  \begin{align*}
  T(n) &= T(n - 1) + 1 = T(n - 2) + 1 + 1 \\
  &= T(n - 3) + 1 + 1 + 1 = \cdots = T(n - k) + k \\
  &= T(1) + n - 1
  \end{align*}

- $T(n) = 2 \times T(n - 10)$  \hspace{1cm} (Exponential)

  \begin{align*}
  T(n) &= 2T(n - 10) = 2 \left( 2T(n - 20) \right) = 4T(n - 20) \\
  &= 8T(n - 30) = \cdots = 2^iT(n - 10i) \\
  &= 2^{\frac{n}{10} - 1}T(10) = \Theta \left( 2^{\frac{n}{10}} \right)
  \end{align*}

Only constant reduction in input size

Two branches
Other Recurrences

- \( T(n) = T(n-1) + T(n-3) \) (Exponential)

  Two branches  Only constant reduction in input size

  \[ T(n) = \Theta(c^n) \]

- Unrolling gets messy fast! How to find \( c \)? [Trick]
  Assume \( T(n) = c^n \) for some \( c \)
  \[ c^n = T(n) = T(n-1) + T(n-3) = c^{n-1} + c^{n-3} \]
  \[ \rightarrow c^n = c^{n-1} + c^{n-3} \]
  \[ \rightarrow c^3 = c^2 + 1 \]
  \[ \rightarrow c \approx 1.46577 \]
  (Root of Characteristic Equation)

Must verify solution by induction
Other Recurrences

\[ T(n) = T(n - 1) + T(n - 3) \quad \text{(Exponential)} \]

Two branches

Only constant reduction in input size

\[ c^n = c^{n-1} + c^{n-3} \rightarrow c^3 = c^2 + 1 \]

\[ \rightarrow c \approx 1.46577 \quad \text{(Root of Characteristic Equation)} \]

**Claim:** \( T(n) \leq kc^n \) (pick \( k \) s.t. \( T(0) < k \))

**Inductive Step:**
\[ T(n) = T(n - 1) + T(n - 3) \]
\[ \leq k(c^{n-1} + c^{n-3}) \quad \text{(IH)} \]
\[ = k(c^n) \quad \text{(Choice of } c \text{)} \]
Other Recurrences

- MergeSort with Uneven Split: Split L into A, B of sizes \( \frac{n}{4} \) and \( \frac{3n}{4} \).

\[
T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n
\]

A bit harder to Analyze with recursion tree

\[
T(n) = \Theta(n \log n)
\]
Another Unbalanced Recurrence

- **Geometric Series**
  \[ T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + n \]

- **Claim:** \( T(n) = \Theta(n) \)

\[ T(n) \leq n \sum_{i=0}^{\infty} \left(\frac{19}{20}\right)^i \]
Divide and conquer algorithms

- Mergesort
- Quicksort
- Binary Search
- Linear-time selection
- Skyline Problem
- Maximum Subarray
- Counting inversions
Maximum Subarray Problem

Given an array A of n (positive and negative) numbers, find the contiguous subarray whose sum has the largest value.

10 5 -20 5 12 -6 33 6 2 -52 6 45 3 -4

Brute Force

• For every pair i and j, i≤j, compute the sum from A(i) to A(j). Remember the pair resulting in the maximum.
• How many pairs? O(n^2)
• Using previously computed values, the total running time is O(n^2) (n^3 is excessive brute force)
Aim for O(n) or O(n log n)?

Divide and conquer?

- Split the problem into two halves and solve each recursively
- Combine two solutions to produce the final answer

**Figure 4.4** (a) Possible locations of subarrays of $A[low..high]$: entirely in $A[low..mid]$, entirely in $A[mid + 1..high]$, or crossing the midpoint $mid$. (b) Any subarray of $A[low..high]$ crossing the midpoint comprises two subarrays $A[i..mid]$ and $A[mid + 1..j]$, where $low \leq i \leq mid$ and $mid < j \leq high$. 
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>-20</td>
<td>21</td>
<td>12</td>
<td>-18</td>
<td>12</td>
</tr>
<tr>
<td>-20</td>
<td>21</td>
<td>-18</td>
<td>12</td>
<td>5</td>
<td>3</td>
<td>-4</td>
</tr>
</tbody>
</table>

32 46
Combining the answers from the two subproblems

- The maximum subarray can be in one the two halves (easy case) or it can cross the midpoint
- To determine the maximum subarray crossing the midpoint, we can compute the two maximum subarrays “anchored” at the midpoint.
  - Running Sum: sweep left (resp. right) from midpoint
    - keeping track of max
    - Requires time $O(n)$ for merge step

Results in $T(n) = 2T(n/2) + cn$

Gives an $O(n \log n)$ time algorithm
A Faster O(n) time solution

**Idea:** Each recursive call returns *additional information* to make merge step easier

- **Optimal Solution:** \((i,j)\) and value \(v = \sum_{x=i}^{j} A[x]\)
- **Total Sum:** \(T = \sum_{x=1}^{n} A[x]\)
- **\(i_{end}\) maximizing value** \(v_{end} = \sum_{x=i_{end}}^{n} A[x]\)
- **\(j_{begin}\) maximizing value** \(v_{begin} = \sum_{x=1}^{j_{begin}} A[x]\)

\[
\begin{array}{cccccccc}
10 & 5 & -20 & \color{green}{21} & 12 & -18 & 12 & 5 & -50 \\
\end{array}
\]

\(v_{begin} = 27 \quad v = 32 \quad v_{end} = 5\)
A Faster O(n) time solution

Merge in Constant Time: Suppose A was split into L and R

- Three possibilities for (i,j): \((i^L, j^L), (i^R, j^R), (i^L_{end}, j^L_{begin})\)
  - Case 1: Opt in L
  - Case 2: Opt in R
  - Case 3: Opt crosses L and R

\[
\begin{align*}
    v^L_{begin} &= 27 \\
    v^L &= 32 \\
    v^L_{end} &= 7 \\
    v^R_{begin} &= 46 \\
    v^R &= 52 \\
    v^R_{end} &= 52
\end{align*}
\]

\[
v = \max\{v^L, v^R, v^L_{end} + v^R_{begin}\} = v^L_{end} + v^R_{begin} = 53\]

\[
\rightarrow (i, j) = (i^L_{end}, j^L_{begin})
\]
A Faster O(n) time solution

**Merge in Constant Time:** Suppose A was split into L and R

- Still Needs to Compute Extra Values
- **Update Total:** \( T = T^L + T^R \)
- **\( i_{\text{end}} \) maximizing value** \( v_{\text{end}} = \sum_{x=i_{\text{end}}}^{n} A[x] \)
  - Case 1: \( i_{\text{end}} = i^R_{\text{end}} \) (interval in R)
  - Case 2: \( i_{\text{end}} = i^L_{\text{end}} \) (interval crosses L)

\[
\begin{align*}
10 & \ 5 \ -20 \ & 21 \ & 12 \ & -18 \ & 12 \ & 5 \ & -50 \ & 2 \ & 5 \ & -5 \ & 50 \ & 1 \ & -2 \ & -80 \ & 44 \ & 1 \ & 1 \ & 6
\end{align*}
\]

\( v^L_{\text{end}} = 7 \quad v^R_{\text{end}} = 52 \)

\[
\begin{align*}
v^R &= \max\{v^R_{\text{end}}, T^R + v^L_{\text{end}}\} = v^R_{\text{end}} = 52 \Rightarrow i_{\text{end}} = i^R_{\text{end}}
\end{align*}
\]

For any merges higher in the recursion tree!
A Faster O(n) time solution

Merge in Constant Time: Suppose A was split into L and R

- Merge Needs to Compute Extra Values
- \( T = T^L + T^R \)

- \( j_{\text{begin}} \) maximizing value \( v_{\text{begin}} = \sum_{x=1}^{j_{\text{begin}}} A[x] \)
  
  - Similar to computing \( i_{\text{end}} \)
  
  - Case 1: \( j_{\text{begin}} = j^L_{\text{begin}} \) (interval in L)
  
  - Case 2: \( j_{\text{begin}} = j^R_{\text{begin}} \) (interval crosses R)

\[
\begin{array}{cccccccccc}
10 & 5 & -20 & 21 & 12 & -18 & 12 & 5 & -50 & 2 & 5 \\
\end{array}
\]

\[
\begin{array}{ccccccccc}
-5 & 50 & 1 & -2 & -80 & 44 & 1 & 1 & 6 \\
\end{array}
\]

\[
v^L_{\text{begin}} = 27, \quad v^R_{\text{begin}} = 46
\]

\[
v^L = \max\{v^L_{\text{begin}}, T^L + v^R_{\text{begin}}\} = T^L + v^R_{\text{begin}} = -16 + 46 = 30
\]

\[
\Rightarrow j_{\text{begin}} = j^R_{\text{begin}}
\]
Summary: A Faster $O(n)$ time solution

**Idea:** Each recursive call returns *additional information* to make merge step easier

**Constant Time Merge:**

\[ T(n) = 2T \left( \frac{n}{2} \right) + 1 \]

**Master Theorem** ($a=2$, $b=2$, $c=0$): $T(n) = \Theta(n)$
Summary: Maximum Subarray problem

Given an array A of n (positive and negative) numbers, find the contiguous subarray whose sum has the largest value.

\[
10 \ 5 \ -20 \ 5 \ 12 \ -6 \ 33 \ 6 \ 2 \ -52 \ 6 \ 45 \ 3 \ -4
\]

- *All pairs: O(n^2)*
- *D&C with simple combine step: O(n \log n)*
- *D&C with extra information: O(n)*
- *Simple non-D&C solution: O(n)*