Algorithms and Data Structures
Divide and Conquer, Master theorem

Prof. Dr. Rolf Backofen
Bioinformatics Group / Department of Computer Science
Algorithms and Data Structures, December 2018
Structure

Divide and Conquer
  Concept
  Maximum Subtotal

Recursion Equations
  Substitution Method
  Recursion Tree Method
  Master theorem
    Master theorem (Simple Form)
    Master theorem (General Form)
Concept:

- **Divide** the problem into smaller subproblems
- **Conquer** the subproblems through recursive solving. If subproblems are small enough solve them directly
- **Connect** all subsolutions to solve the overall problem
- **Recursive** application of the algorithm on smaller subproblems
- **Direct** solving of small subproblems
Divide and Conquer
Maximum Subtotal

Input:
- Sequence \( X \) of \( n \) integers

Output:
- Maximum sum of an uninterrupted subsequence of \( X \) and its index boundary

Table: input values

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>31</td>
<td>-41</td>
<td>59</td>
<td>26</td>
<td>-53</td>
<td>58</td>
<td>97</td>
<td>-93</td>
<td>-23</td>
<td>84</td>
</tr>
</tbody>
</table>

Output: Sum: 187, Start: 2, End: 6
Idea:

- Solve the left / right half of the problem recursively
- Combine both solutions into an overall solution
- The maximum is located in the left half \((A)\) or the right half \((B)\)
- The maximum interval can overlap with the border \((C)\)
Principle:

- Small problems are solved directly: \( n = 1 \Rightarrow \max = X[0] \)
- Big problems are decomposed into two subproblems and solved recursively. Subsolutions \( A \) and \( B \) are returned.
- To solve \( C \) we have to calculate \( \text{rmax} \) and \( \text{lmax} \)
- The overall solution is the maximum of \( A, B \) and \( C \)
def maxSubArray(X, i, j):
    if i == j:  # trivial case
        return (X[i], i, i)

    m = (i + j) // 2
    A = maxSubArray(X, i, m)
    B = maxSubArray(X, m + 1, j)

    # rmax and lmax for corner case C
    C1, C2 = rmax(X, i, m), lmax(X, m + 1, j)
    C = (C1[0] + C2[0], C1[1], C2[1])

    # compute solution from results A, B, C
    return max([A, B, C], key=lambda i: i[0])
# Alternative trivial case

def maxSubArray(X, i, j):
    # trivial: only one element
    if i == j:
        return (X[i], i, i)

    # trivial: only two elements
    if i + 1 == j:
        return max([
            (X[i], i, i),
            (X[j], j, j),
            (X[i] + X[j], i, j)
        ], key=lambda item: item[0])

    ... # continue as before
#Implementation max

def max(a, b, c):
    if a > b:
        if a > c:
            return a
        else:
            return c
    else:
        if c > b:
            return c
        else:
            return b
#Alternative implementation max

```python
def max(a, b):
    if a > b:
        return a
    else:
        return b

def maxTripel(a, b, c):
    return max(max(a, b), c)
```
```python
# Implementation left maximum

def lmax(X, i, j):
    maxSum = (X[i], i)
    s = X[i]

    # sum up from the lower index going up
    # (from left to right)
    for k in range(i+1, j+1):
        s += X[k]

        if s > maxSum[0]:
            maxSum = (s, k)

    return maxSum
```
```
#Implementation right maximum

def rmax(X, i, j):
    maxSum = (X[j], j)
    s = X[j]

    # sum up from the upper index going down
    # (from right to left)
    for k in range(j-1, i-1, -1):
        s += X[k]

        if s > maxSum[0]:
            maxSum = (s, k)

    return maxSum
```
## Divide and Conquer

### Maximum Subtotal

#### Table: $l_{max}$ example

<table>
<thead>
<tr>
<th>index</th>
<th>$i$</th>
<th>$i+1$</th>
<th>$\cdots$</th>
<th>$\cdots$</th>
<th>$j-1$</th>
<th>$j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>58</td>
<td>-53</td>
<td>26</td>
<td>59</td>
<td>-41</td>
<td>31</td>
</tr>
<tr>
<td>$sum$</td>
<td>58</td>
<td>5</td>
<td>31</td>
<td>90</td>
<td>49</td>
<td>80</td>
</tr>
<tr>
<td>$l_{max}$</td>
<td>58</td>
<td>58</td>
<td>58</td>
<td>90</td>
<td>90</td>
<td>90</td>
</tr>
</tbody>
</table>

- The $sum$ and $l_{max}$ are initialized with $X[j]$
- We iterate over $X$ from $i + 1$ to $j$ and update $sum$
- If $sum > l_{max}$, then $l_{max}$ gets updated
Divide and Conquer
Maximum Subtotal

Call with array of four elements
in fact:
maxSubArray(-3,9,-4,7)
maxSubArray(X,0,3)
with X=[-3,9,-4,7]
maxSubArray(-3,9)
maxSubArray(-4,7)
and so on ...
maxSubArray(9)
maxSubArray(-3)
maxSubArray(-4)
maxSubArray(7)

max(A,B,C1+C2)
A=-3
B=9
C1=-3, C2=9
max(A,B,C1+C2)

rmax(-3), lmax(9)
A=-4
B=7
C1=-4, C2=7
max(A,B,C1+C2)

rmax(-4), lmax(7)
A=9
B=7
C1=9, C2=3
max(A,B,C1+C2)

12

in fact: maxSubArray(X,0,3) with X=[-3,9,-4,7]
def maxSubArray(X, i, j):
    if i == j:
        # O(1)
        return (X[i], i, i)
        # O(1)

    m = (i + j) // 2
    # O(1)

    A = maxSubArray(X, i, m)
    # T(n/2)

    B = maxSubArray(X, m + 1, j)
    # T(n/2)

    C1 = rmax(X, i, m)
    # O(n)

    C2 = lmax(X, m + 1, j)
    # O(n)

    C = (C1[0] + C2[0], C1[1], C2[1])
    # O(1)

    return max([A, B, C],
                key=lambda item: item[0])
                # O(1)
Divide and Conquer

Maximum Subtotal - Number of steps $T(n)$

Recursion equation:

$$T(n) = \begin{cases} 
\Theta(1) & n = 1 \\
2 \cdot T(\frac{n}{2}) + \Theta(n) & n > 1
\end{cases}$$

- There exist two constants $a$ and $b$ with:

$$T(n) \leq \begin{cases} 
2 \cdot T(\frac{n}{2}) + b \cdot n & n > 1
\end{cases}$$

- We define $c := \max(a, b)$:

$$T(n) \leq \begin{cases} 
c & n = 1 \\
2 \cdot T(\frac{n}{2}) + c \cdot n & n > 1
\end{cases}$$
Divide and Conquer

Maximum Subtotal - Illustration of $T(n)$

Combining solutions

Solving subproblems

$$T(n)c \cdot n$$

$$T\left(\frac{n}{2}\right)c \cdot \frac{n}{2}$$

$$T\left(\frac{n}{4}\right)$$

$$T\left(\frac{n}{4}\right)$$

$$T\left(\frac{n}{4}\right)$$

$$T\left(\frac{n}{4}\right)$$

$$T\left(\frac{n}{2}\right) = 2 \cdot T\left(\frac{n}{4}\right) + c \cdot \frac{n}{2}$$
Divide and Conquer
Maximum Subtotal - Illustration of $T(n)$

1 node processing $n$ elements
⇒ $c \cdot n$

2 nodes processing $\frac{n}{2}$ elements
⇒ $2c \cdot \frac{n}{2} = c \cdot n$

4 nodes processing $\frac{n}{4}$ elements
⇒ $4c \cdot \frac{n}{4} = c \cdot n$

$2^i$ nodes processing $\frac{n}{2^i}$ elements
⇒ $2^i c \cdot \frac{n}{2^i} = c \cdot n$

$n$ nodes processing 1 element
⇒ $c \cdot n$

Figure: recursion tree method
Divide and Conquer
Maximum Subtotal - Illustration of $T(n)$

Depth:
- Top level with depth $i = 0$
- Lowest level with $2^i = n$ elements

$\Rightarrow i = \log_2 n$

Runtime:
- A total of $\log_2 n + 1$ levels costing $c \cdot n$ each
  The costs of merging the solutions and solving the trivial problems are the same in this case

$T(n) = c \cdot n \log_2 n + c \cdot n \in \Theta(n \log n)$
Summary:

- Direct solution is slow with $\mathcal{O}(n^3)$
- Better solution with incremental update of sum was $\mathcal{O}(n^2)$
- Divide and conquer approach results in $\mathcal{O}(n \log n)$
- There is an approach running in $\mathcal{O}(n)$, under the assumption that all subtotals are positive
Divide and Conquer

Maximum Subtotal

point of last negative subtotal

\[ t_{\text{Max}} \quad r_{\text{Max}} \]

Figure: scanning the array in linear time
```python
#Implementation - linear runtime

def maxSubArray(X):
    # sum, start index
    rMax, irMax = 0, 0  # current maximum
    tMax, itMax = 0, 0  # total maximum

    for i in range(len(X)):
        if rMax == 0:
            irMax = i
        rMax = max(0, rMax + X[i])

        if rMax > tMax:
            tMax, itMax = rMax, irMax

    return (tMax, itMax)
```
Recursion equation:

Runtime description for recursive functions:

\[
T(n) = \begin{cases} 
  f_0(n) & n = n_0 \\
  a \cdot T\left(\frac{n}{b}\right) + f(n) & n > n_0
\end{cases}
\]

- trivial case for \(n_0\)
- solving of \(a\) subproblems with reduced input size \(\frac{n}{b}\)
- slicing and splicing of subsolutions
Recursion equation:

- Runtime description for recursive functions:

\[
T(n) = \begin{cases} 
  f_0(n) & n = n_0 \\
  a \cdot T\left(\frac{n}{b}\right) + f(n) & n > n_0
\end{cases}
\]

- \( n_0 \) is usually small, \( f_0(n_0) \in \Theta(1) \)
- Usually, \( a > 1 \) and \( b > 1 \)
- Dependent on the strategy of solving \( T(n) \) \( f_0 \) is ignored
- \( T(n) \) is only defined for integers of \( \frac{n}{b} \), which is often ignored in benefit of a simpler solution
Substitution Method:

- Guess the solution and prove it with induction
- Example:

\[
T(n) = \begin{cases} 
1 & n = 1 \\
2 \cdot T\left(\frac{n}{2}\right) + n & n > 1
\end{cases}
\]

- Assumption: \( T(n) = n + n \cdot \log_2 n \)
Induction:

- Induction basis (for $n = 1$): $T(1) = 1 + 1 \cdot \log_2 1 = 1$
- Induction step (from $\frac{n}{2}$ to $n$):

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$$

\[= 2 \cdot \left(\frac{n}{2} + \frac{n}{2} \cdot \log_2 \frac{n}{2}\right) + n\]
\[= 2 \cdot \left(\frac{n}{2} + \frac{n}{2} \cdot (\log_2 n - 1)\right) + n\]
\[= n + n \log_2 n - n + n\]
\[= n + n \log_2 n\]
Substitution Method:

- Alternative assumption
- Example:

\[
T(n) = \begin{cases} 
1 & n = 1 \\
2 \cdot T \left( \frac{n}{2} \right) + n & n > 0 
\end{cases}
\]

- Assumption: \( T(n) \in O(n \log n) \)
- Solution: Find \( c > 0 \) with \( T(n) \leq c \cdot n \log_2 n \)
Recursion Equations
Substitution Method

**Induction:**
- Solution: Find $c > 0$ with $T(n) \leq c \cdot n \log_2 n$
- Induction step (from $\frac{n}{2}$ to $n$):

\[
T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n \\
\leq 2 \cdot \left( c \cdot \frac{n}{2} \log_2 \frac{n}{2} \right) + n \\
= c \cdot n \log_2 n - c \cdot n \log_2 2 + n \\
= c \cdot n \log_2 n - c \cdot n + n \\
\leq c \cdot n \log_2 n, \quad c \geq 1
\]
Recursion tree method:

- Can be used to make assumptions about the runtime
- Example:

\[
T(n) = 3 \cdot T\left(\frac{n}{4}\right) + \Theta(n^2) \leq 3 \cdot T\left(\frac{n}{4}\right) + c \cdot n^2
\]
Recursion Equations

Recursion Tree Method

\[ T(n) = 3 \cdot T\left(\frac{n}{4}\right) + c \cdot n^2 \]

**Figure:** recursion tree of example
Recursion Equations

Recursion Tree Method

Figure: levels of the recursion tree
Costs of connecting the partial solutions: (excludes the last layer)

- Size of partial problems on level $i$: $s_i(n) = \left(\frac{1}{4}\right)^i \cdot n$
- Costs of partial problems on level $i$:

$$T_{ip}(n) = c \cdot \left(\left(\frac{1}{4}\right)^i \cdot n\right)^2$$

- Number of partial problems on level $i$: $n_i = 3^i$
- Costs on level $i$:

$$T_i(n) = 3^i \cdot c \cdot \left(\left(\frac{1}{4}\right)^i \cdot n\right)^2 = \left(\frac{3}{16}\right)^i \cdot c \cdot n^2$$
Costs of solving partial solutions: (only the last layer)

- Size of partial problems on the last level: \( s_{i+1}(n) = 1 \)
- Costs of partial problem on the last level: \( T_{i+1_p}(n) = d \)
- With this the depth of the tree is:

\[
\left( \frac{1}{4} \right)^i \cdot n = 1 \implies n = 4^i \implies i = \log_4 n
\]

- Number of partial problems on the last level:

\[
n_{i+1} = 3^{\log_4 n} = n^{\log_4 3} \leftarrow \text{next slide}
\]

- Costs on the last level: \( T_{i+1}(n) = d \cdot n^{\log_4 3} \)
Transforming $3^{\log_4 n}$ using general log rules

$$
\log_4 n = \log_4 \left( 3^{\log_3 n} \right) \quad \text{using } n = 3^{\log_3 n}
$$

$$
= \log_3 n \cdot \log_4 3 \quad \text{using } \log_a b^c = b \cdot \log_a b
$$

This proves the general log rule $\log_b c = \log_a c \cdot \log_b a$

Now the whole expression:

$$
3^{\log_4 n} = 3^{\log_3 n \cdot \log_4 3} \quad \text{using reformulation above}
$$

$$
= \left( 3^{\log_3 n} \right)^{\log_4 3} \quad \text{using } x^{a \cdot b} = (x^a)^b
$$

$$
= n^{\log_4 3}
$$

This term will recur in the master theorem
Total costs:

- Costs of level $i$: $T_i(n) = \left(\frac{3}{16}\right)^i \cdot c \cdot n^2$

- Costs of last level: $T_{i+1}(n) = d \cdot n^\log_4 3$

\[
T(n) = \sum_{i=0}^{(\log_4 n)-1} \left(\frac{3}{16}\right)^i \cdot c \cdot n^2 + d \cdot n^\log_4 3 \in O(n^2)
\]

- geometric series, constant
- even with infinite elements

- $\log_4 3 < 1$, grows a lot slower than $n^2$

Here: The costs of connecting the partial problems dominate
**Geometric progression:**
Quotient of two neighboring sequence parts is constant

\[ 2^0, 2^1, 2^2, \ldots, 2^k \]

**Geometric series:**
The series (cumulative sum) of a geometric sequence

For \(|q| < 1|:

\[
\sum_{k=0}^{\infty} a_0 \cdot q^k = \frac{a_0}{1 - q} \Rightarrow \text{constant}
\]
Proof of $O(n^2)$:

- We know:

$$T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$$

$$\leq 3T\left(\frac{n}{4}\right) + c \cdot n^2$$

- Assumption: $T(n) \in O(n^2)$, so there exists a $k > 0$ with

$$T(n) \leq k \cdot n^2$$
Proof of $\Theta(n^2)$:

- Presumption: $T(n) \in \Theta(n^2)$, so there exists a $k > 0$ with
  \[ T(n) < k \cdot n^2 \]

- Substitution method:
  \[
  T(n) \leq 3 \cdot T\left(\frac{n}{4}\right) + c \cdot n^2 \\
  \leq 3k \cdot \left(\frac{n}{4}\right)^2 + c \cdot n^2 \\
  = \frac{3}{16} k \cdot n^2 + c \cdot n^2 \\
  \leq k \cdot n^2 
  \quad \text{for } k \geq \frac{16}{13} c
  \]
Master theorem:

- Solution approach for a recursion equation of the form:

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + f(n), \quad a \geq 1, b > 1 \]

- \( T(n) \) is the runtime of an algorithm …
  - … which divides a problem of size \( n \) in \( a \) partial problems
  - … which solves each partial problem recursively with a runtime of \( T \left( \frac{n}{b} \right) \)
  - … which takes \( f(n) \) steps to merge all partial solutions
Master theorem:

- In the examples we have seen that …
  - Either the runtime of connecting the solutions dominates
  - Or the runtime of solving the problems dominates
  - Or both have equal influence on runtime

**Simple form:** Special case with runtime of connecting the solutions $f(n) \in O(n)$
Recursion Equations
Master theorem (Simple Form)

Simple form:

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n, \quad a \geq 1, b > 1, c > 0 \]

Is any \( f(n) \) in general form

- This yields a runtime of:

\[
T(n) = \begin{cases} 
\Theta\left(n^{\log_b a}\right) & \text{if } a > b \\
\Theta(n \log n) & \text{if } a = b \\
\Theta(n) & \text{if } a < b 
\end{cases}
\]
Recursion Equations
Master theorem (Simple Form)

Case 1: $a > b$
- Three partial problems with $\frac{1}{2}$ the size
- Solving the partial problems dominates (last layer, leaves)
- Runtime of $\Theta(n^{\log_b a})$

Figure: simple recursion equation with $a = 3, b = 2$
Recursion Equations

Master theorem (Simple Form)

Case 2: $a = b$

- Two partial problems with $\frac{1}{2}$ the size
- Each layer has equal costs, $\log n$ layers
- Runtime of $\Theta(n \log n)$
Recursion Equations
Master theorem (Simple Form)

Figure: simple recursion equation with $a = 2, b = 3$

Case 3: $a < b$
- Two partial problems with $\frac{1}{3}$ the size
- Connecting all partial solutions dominates (first layer, root)
- Runtime of $\Theta(n)$
Recursion Equations
Master theorem (Simple Form)

For a recursion equation like

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n, \quad a \geq 1, b > 1, c > 0 \]

\[ T(n) = \begin{cases} 
\Theta(n \log_b a) & \text{if } a > b \\
\Theta(n \log_b n) & \text{if } a = b \\
\Theta(n) & \text{if } a < b 
\end{cases} \]

Proof with \textit{geometric series}: Number of operations per layer grows / shrinks by constant factor \( \frac{a}{b} \)
Recursion Equations
Master theorem (General Form)

Master theorem (general form):

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + f(n), \quad a \geq 1, b > 1 \]

- **Case 1:** \( T(n) \in \Theta(n^{\log_b a}) \) if \( f(n) \in \Theta(n^{\log_b a - \epsilon}), \epsilon > 0 \)
  
  Solving the partial problems dominates (last layer, leaves)

- **Case 2:** \( T(n) \in \Theta(n^{\log_b a \log n}) \) if \( f(n) \in \Theta(n^{\log_b a}) \)
  
  Each layer has equal costs, \( \log_b n \) layers
Master theorem (general form):

- **Case 3**: $T(n) \in \Theta(f(n))$ if $f(n) \in \Omega(n^{\log_b a+\varepsilon})$, $\varepsilon > 0$

  Connecting all partial solutions in first layer (root) dominates

**Regularity condition:**

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n), \quad 0 \leq c \leq 1,$$

$$n > n_0$$
Case 1 - Example: \( T(n) \in \Theta(n^{\log_b a}) \) if \( f(n) \in O(n^{\log_b a - \varepsilon}), \varepsilon > 0 \)

Solving the partial problems dominates (last layer, leaves)

\[ T(n) = 8 \cdot T\left(\frac{n}{2}\right) + 1000 \cdot n^2 \]

- \( a = 8, \ b = 2, \ f(n) = 1000 \cdot n^2, \ \log_b a = \log_2 8 = 3 \)

\[ f(n) \in O(n^{3-\varepsilon}) \Rightarrow T(n) \in \Theta(n^3) \]

\[ n^3 \text{ leaves} \]

\[ T(n) = 9 \cdot T\left(\frac{n}{3}\right) + 17 \cdot n \]

- \( a = 9, \ b = 3, \ f(n) = 17 \cdot n, \ \log_b a = \log_3 9 = 2 \)

\[ f(n) \in O(n^{2-\varepsilon}) \Rightarrow T(n) \in \Theta(n^2) \]

\[ n^2 \text{ leaves} \]
Recursion Equations
Master theorem (General Form) - Case 2

Case 2: $T(n) \in \Theta(n^{\log_b a} \log n)$ if $f(n) \in \Theta(n^{\log_b a})$
Each layer has equal costs, $\log n$ layers

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 10 \cdot n$
  
  $a = 2, \ b = 2, \ f(n) = 10 \cdot n, \ \log_b a = \log_2 2 = 1$

  $f(n) \in \Theta(n^{\log_2 2}) \Rightarrow T(n) \in \Theta(n \log n)$ $\quad n^1 \text{ leaves}$

- $T(n) = T\left(\frac{2n}{3}\right) + 1$

  $a = 1, \ b = \frac{3}{2}, \ f(n) = 1, \ \log_b a = \log_{3/2} 1 = 0$

  $f(n) \in \Theta(n^{\log_{3/2} 1}) \Rightarrow T(n) \in \Theta(n^0 \log n) = \Theta(\log n)$ $\quad n^0 \text{ leaves = 1 leaf}$
Case 3: \( T(n) \in \Theta(f(n)) \) if \( f(n) \in \Omega(n^{\log_b a + \epsilon}) \), \( \epsilon > 0 \)

Connecting all partial solutions in first layer (root) dominates

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

\[a = 2, \ b = 2, \ f(n) = n^2, \ \log_b a = \log_2 2 = 1\]

\[f(n) \in \Omega(n^{1+\epsilon})\]
Case 3: \( T(n) \in \Theta(f(n)) \) if \( f(n) \in \Omega(n^{\log_b a + \varepsilon}) \), \( \varepsilon > 0 \)

Connecting all partial solutions in first layer (root) dominates

- \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n^2 \)
- \( f(n) \in \Omega(n^{1+\varepsilon}) \)

Check if regularity condition also holds:

\[
a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)
\]

\[
2 \cdot \left(\frac{n}{2}\right)^2 \leq c \cdot n^2 \quad \Rightarrow \quad \frac{1}{2} \cdot n^2 \leq c \cdot n^2 \quad \Rightarrow \quad c \geq \frac{1}{2}
\]

\( \Rightarrow \) \( T(n) \in \Theta(n^2) \)


**Master theorem:**

- Not always applicable: \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n \log n \)

\[
a = 2, \ b = 2, \ f(n) = n \log n, \ \log_b a = \log_2 2 = 1
\]

\[
n^1 \text{ leaves}
\]

- **Case 1:** \( f(n) \notin O(n^{1-\varepsilon}) \)
- **Case 2:** \( f(n) \notin \Theta(n^1) \)
- **Case 3:** \( f(n) \notin \Omega(n^{1+\varepsilon}) \)

\( n \log n \) is *asymptotically* larger than \( n \), but not *polynomial* larger
Recursion Equations

Master theorem - Summary

Master theorem:

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \]

- Three cases depending on the dominance of the terms
- **Case 1:** Solving the partial problems is *polynomial* bigger than merging all solutions
  \[ T(n) \in \Theta(n^{\log_b a}), \quad T(n) \in \Theta(\text{number of leaves}) \]
- **Case 2:** Each layer has equal costs
  \[ T(n) \in \Theta(n^{\log_b a \log n}), \quad \log n \text{ layers} \]
- **Case 3:** Connecting all partial solutions is *polynomial* bigger than solving all partial problems
  \[ T(n) \in \Theta(f(n)) \]
Further Literature

General


Further Literature

- **Master theorem**

  [Wik] Master theorem
  https://en.wikipedia.org/wiki/Master_theorem