CSE548 & AMS542: Analysis of Algorithms, Fall 2017 **Lecturer:** Rezaul A Chowdhury
Topic: Divide-and-Conquer Algorithms: The Master Theorem **Date:** 09-20-2017
Scribe: Saish Sali

5.1 A useful recurrence

The following recurrence represents the running time of a recursive divide and conquer algorithm on a input of size n

$$T(n) = \begin{cases} \Theta(1), & if \quad n \leq 1 \\ aT(\frac{n}{b}) + f(n), & otherwise \end{cases}$$

where,

a ≥1 is the number of sub problems,

b ≥1 is the factor by which the size of the input is reduced,

f(n) is the cost of division and merging and should be positive,

$T(\frac{n}{b})$ is a subproblem,

$aT(\frac{n}{b})$ is the cost of solving a subproblem recursively,

n is a power of b

Consider the following recurrences:

Karatsubas Algorithm: $T(n) = 3T(\frac{n}{2}) + \Theta(n)$

Here, a = 3, b = 2 and f(n) = $\Theta(n)$
There are 3 subproblems each of size $\frac{n}{2}$ and $\Theta(n)$ is the cost of addition and shift operations

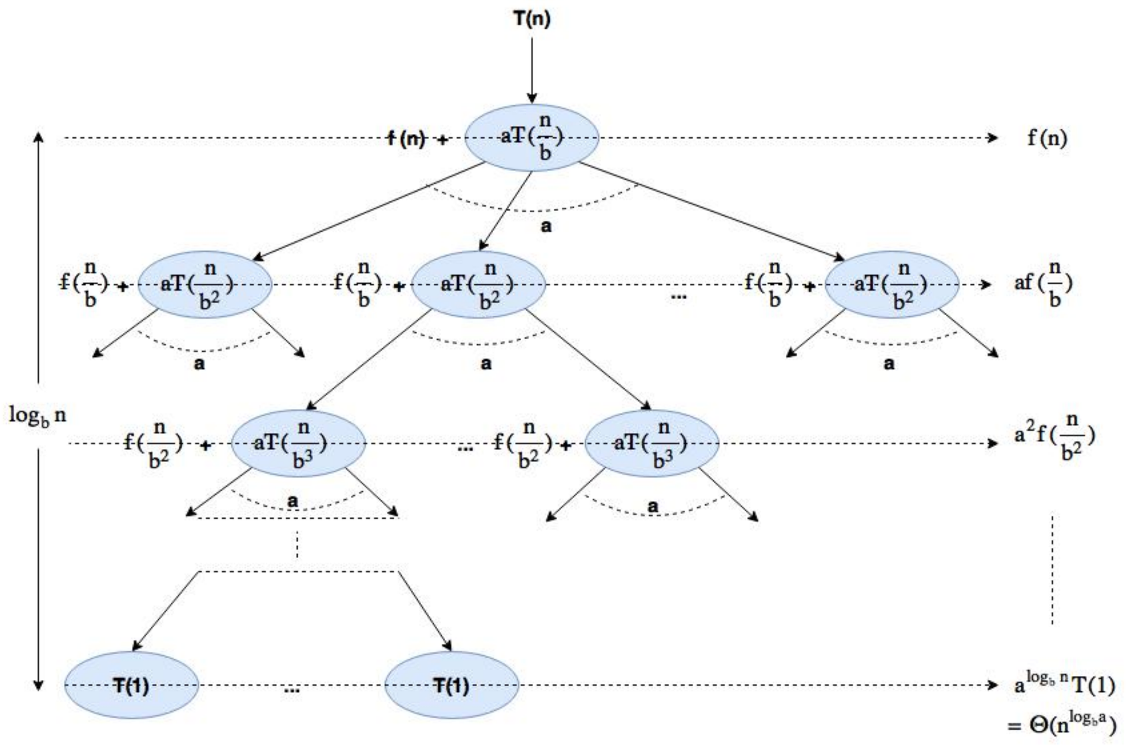
Strassens Algorithm: $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$

Here, a = 3, b = 2 and f(n) = $\Theta(n)$
There are 7 subproblems each of size $\frac{n}{2}$ and $\Theta(n)$ is the cost of addition of matrices

Fast Fourier Transform: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

Here, a = 3, b = 2 and f(n) = $\Theta(n)$
There are 2 subproblems each of size $\frac{n}{2}$ and $\Theta(n)$ is the cost of addition and subtraction of odd and even coefficients

5.2 How the Recurrence Unfolds



At recursion level 1, we have f(n) as the cost of dividing and merging and $aT(\frac{n}{b})$ is being solved recursively. There are 'a' branches coming out of it, each getting an input of size $\frac{n}{b}$. Each of them in turn has $T(\frac{n}{b})$ as the cost of dividing and merging and makes 'a' recursive calls with input of size $\frac{n}{b^2}$. This continues until we reach a problem of size 1.

Height of the tree (Number of levels of recursion): $\log_b n$
Number of leaves: (Branching factor) ^ (Height of the tree) = $a^{\log_b n}$

Cost of actual work = Number of leaves $\times$ Cost of solving problem of size 1
= $a^{\log_b n}T(1)$
= $a^{\log_b n}\Theta(1)$
= $\Theta(a^{\log_b n})$
= $\Theta(n^{\log_b a})$

Cost of division and merging = $\sum_{i=1}^{\log_b n-1} a^i f(\frac{n}{b^i})$

The solution of recurrence depends on the relative values of cost of actual work and the cost of division and merging. There are 3 cases:

(1) Cost of actual work dominates cost of division and merging
(2) Cost of division and merging dominates cost of actual work
(3) Neither cost of division and merging nor cost of actual work dominates

5.2.1 How the Recurrence Unfolds: Case 1

Cost of division and merging is polynomially smaller than the cost of actual work done at the leaves, then the cost of actual work dominates and is the solution of the recurrence.

The sum geometrically increases level by level and the last level dominates.

$f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant ϵ ¿ 0

$$\boldsymbol{T(n) = \Theta(n^{\log_b n})}$$

Proof:

$$T(n) = \Theta(n^{log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(\tfrac{n}{b^j})$$

Given: $f(n) = \Theta(n^{\log_b a - \epsilon})$

Let $g(n) = \sum_{j=0}^{\log_b n-1} a^j f(\tfrac{n}{b^j})$

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j \Theta((\tfrac{n}{b^j})^{\log_b a - \epsilon})$$

$$g(n) = \Theta(\sum_{j=0}^{\log_b n-1} a^j (\tfrac{n^{\log_b a-\epsilon}}{(b^j)^{\log_b a-\epsilon}}))$$

$$g(n) = \Theta(n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} a^j (\tfrac{b^{j^{\epsilon}}}{(b^j)^{\log_b a}}))$$

$$g(n) = \Theta(n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} a^j (\tfrac{b^{j^{\epsilon}}}{(b^{\log_b a})^j}))$$

$$g(n) = \Theta(n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} a^j (\tfrac{b^{j^{\epsilon}}}{a^j}))$$

$$g(n) = \Theta(n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} (b^j)^{\epsilon})$$

$$g(n) = \Theta(n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} (b^{\epsilon})^j)$$

$g(n) = \Theta(n^{\log_b a - \epsilon} \frac{(b^\epsilon)^{\log_b n} - 1}{b^\epsilon - 1})$

Since $b^\epsilon - 1$ is a constant,

$g(n) = \Theta(n^{\log_b a - \epsilon}((b^\epsilon)^{log_b n} - 1))$

$g(n) = \Theta(n^{\log_b a - \epsilon}(b^\epsilon)^{log_b n})$

$g(n) = \Theta(n^{\log_b a - \epsilon}(b^{log_b n})^\epsilon)$

$g(n) = \Theta(n^{\log_b a - \epsilon} n^\epsilon)$

$g(n) = \Theta(n^{\log_b a})$

Since $T(n) = \Theta(n^{\log_b a}) + g(n)$

$T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a})$

Hence, $\mathbf{T(n) = \Theta(n^{\log_b a})}$

5.2.2 How the Recurrence Unfolds: Case 2

Cost of division and merging is within polylog factor of the number of leaves.

The sum arithmetically increases level by level and no level dominates.

$f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant k ≥ 0

$$\boldsymbol{T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)}$$

Proof:

$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j})$

Given: $f(n) = \Theta(n^{\log_b a} \log^k n)$ for k ≥ 0

Let $g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j})$

$g(n) = \Theta(\sum_{j=0}^{\log_b n - 1} a^j (\frac{n}{b^j})^{\log_b a} \log^k(\frac{n}{b^j}))$

$g(n) = \Theta(\sum_{j=0}^{\log_b n - 1} a^j (\frac{n^{\log_b a}}{(b^j)^{\log_b a}}) \log^k(\frac{n}{b^j}))$

$g(n) = \Theta(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} a^j (\frac{1}{(b^j)^{\log_b a}}) \log^k(\frac{n}{b^j}))$

$g(n) = \Theta(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} a^j (\frac{1}{(b^{\log_b a})^j}) \log^k(\frac{n}{b^j}))$

$$g(n) = \Theta(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} a^j (\tfrac{1}{a^j}) \log^k (\tfrac{n}{b^j}))$$

$$g(n) = \Theta(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \log^k (\tfrac{n}{b^j}))$$

$$g(n) = \Theta(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} (\log n - \log(b^j))^k)$$

$$g(n) = \Theta(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} (\log n - j \log b)^k)$$

$$\text{Consider: } \sum_{j=0}^{\log_b n - 1} (\log n - j \log b)^k$$

$$\leq \sum_{j=0}^{\log_b n - 1} \log^k n \quad \text{(Upper bound)}$$

$$\leq \log^k n \times \log_b n$$

$$= \mathcal{O}(\log^{\mathbf{k+1}} \mathbf{n}) \tag{1}$$

$$\text{Consider: } \sum_{j=0}^{\log_b n - 1} (\log n - j \log b)^k$$

The value of the equation above decreases with increase in j. To calculate lower bound, we need to figure out a value (at larger values of j) which is proportional to $\log n$

At j = $log_b n - 1$, we get the value as

$$= (\log n - \log b^j)^k$$

$$= (\log n - \log b^{\log_b n - 1})^k$$

$$= (\log n - \log(\tfrac{b^{\log_b n}}{b}))^k$$

$$= (\log n - \log(\tfrac{n}{b}))^k$$

$$= (\log n - \log n - \log b)^k$$

$= (\log b)^k$ (Not considered since it is not proportional to $\log n$)

$$\text{Consider: } \sum_{j=0}^{\log_b n - 1} (\log n - j \log b)^k$$

$$\geq \sum_{j=0}^{\frac{1}{2}\log_b n} (\log n - j \log b)^k \quad \text{(Lower bound)}$$

$$\geq \sum_{j=0}^{\frac{1}{2}\log_b n} (\tfrac{1}{2} \log n)^k$$

$$\geq (\tfrac{1}{2} \log n)^k \times \tfrac{1}{2} \log_b n$$

$$= \mathbf{\Omega}(\log^{\mathbf{k+1}} \mathbf{n}) \tag{2}$$

From (1) and (2), we can say that the tight bound is $\mathbf{\Theta}(\log^{\mathbf{k+1}} \mathbf{n})$

$$\text{Hence, } \mathbf{T(n)} = \mathbf{\Theta}(\mathbf{n}^{\log_\mathbf{b} \mathbf{a}} \lg^{\mathbf{k+1}} \mathbf{n})$$

5.2.3 How the Recurrence Unfolds: Case 3

Cost of division and merging is polynomially larger than the cost of actual work done at the leaves, then the cost of division and merging dominates and is the solution of the recurrence.

The sum geometrically decreases level by level and the first level dominates.

$f(n) = \Omega(n^{\log_b a+\epsilon})$ and $af(\frac{n}{b}) \leq cf(n)$ for constants $\epsilon > 0$ and $c < 1$

$$\boldsymbol{T(n) = \Theta(f(n))}$$

Proof:

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(\tfrac{n}{b^j})$$

Given: $f(n) = \Omega(n^{\log_b a+\epsilon})$ for $\epsilon > 0$

Let $g(n) = \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})$

$af(\frac{n}{b}) \leq cf(n)$ for $c < 1$

Smallest value of $f(n) = n^{\log_b a+\epsilon}$

$af(\frac{n}{b}) = a(\frac{n}{b})^{\log_b a+\epsilon}$

$af(\frac{n}{b}) = a(\frac{n^{\log_b a+\epsilon}}{b^{\log_b a+\epsilon}})$

$af(\frac{n}{b}) = a(\frac{f(n)}{b^{\log_b a+\epsilon}})$

$af(\frac{n}{b}) = a(\frac{f(n)}{ab^\epsilon})$

$af(\frac{n}{b}) = \frac{f(n)}{b^\epsilon}$

$af(\frac{n}{b}) = f(n)b^{-\epsilon}$

$af(\frac{n}{b}) = b^{-\epsilon}f(n)$

$af(\frac{n}{b}) \leq cf(n)$ for $b^{-\epsilon} < c < 1$

Consider: $af(\frac{n}{b}) \leq cf(n)$ for $c < 1$

Consider: $f(\frac{n}{b}) \leq \frac{c}{a}f(n)$

$f(\frac{n}{b^2}) \leq \frac{c}{a}f(\frac{n}{b})$

$f(\frac{n}{b^2}) \leq \frac{c}{a}f(\frac{n}{b}) \leq (\frac{c}{a})^2 f(n)$

$a^2 f(\frac{n}{b^2}) \leq c^2 f(n)$

where $a^2 f(\frac{n}{b^2})$ is the cost of solving a subproblem at level 2

Hence, $\mathbf{a^j f(\frac{n}{b^j}) \leq c^j f(n)}$

Consider $g(n) = \sum\limits_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})$

$$\leq \sum_{j=0}^{\log_b n-1} c^j f(n) \quad \text{(Upper bound)}$$

$$\leq f(n) \sum_{j=0}^{\log_b n-1} c^j$$

$$\leq f(n) \sum_{j=0}^{\infty} c^j$$

where $\sum\limits_{j=0}^{\infty} c^j$ is a geometric series

$$g(n) \leq \frac{f(n)}{1-c}$$

$$\mathbf{g(n)} = \mathcal{O}(\mathbf{f(n)}) \tag{1}$$

Since g(n) includes f(n) and other non negative terms,

$$\mathbf{g(n)} = \boldsymbol{\Omega}(\mathbf{f(n)}) \tag{2}$$

From (1) and (2), we can conclude

$$\mathbf{g(n)} = \boldsymbol{\Theta}(\mathbf{f(n)})$$

Here, first level dominates, therefore

$$\mathbf{T(n)} = \boldsymbol{\Theta}(\mathbf{f(n)})$$

5.3 The Master Theorem

$$T(n) = \begin{cases} \Theta(1), & if \quad n \leq 1 \\ aT(\frac{n}{b}) + f(n), & otherwise (a \geq 1, b \geq 1) \end{cases}$$

Case 1: $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$

$$\boldsymbol{T(n) = \Theta(n^{\log_b n})}$$

Case 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant k ≥ 0

$$\boldsymbol{T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)}$$

Case 3: $f(n) = \Omega(n^{\log_b a+\epsilon})$ and $af(\frac{n}{b}) \leq cf(n)$ for constants $\epsilon > 0$ and $c < 1$

$$\boldsymbol{T(n) = \Theta(f(n))}$$

5.4 Example Applications of Master Theorem

Example 1: $T(n) = 3T(\frac{n}{2}) + \Theta(n)$

Here,

a = 3,

b = 2,

f(n) = $\Theta(n)$,

Number of leaves = $n^{\log_2 3}$,

The number of leaf nodes dominates f(n)

Master Theorem Case 1: $T(n) = \Theta(n^{\log_2 3})$

Example 2: $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$

Here,

a = 7,

b = 2,

f(n) = $\Theta(n^2)$,

Number of leaves = $n^{\log_2 7}$,

The number of leaf nodes dominates f(n)

Master Theorem Case 1: $T(n) = \Theta(n^{\log_2 7})$

Example 3: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

Here,

a = 2,

b = 2,

f(n) = $\Theta(n)$,

Number of leaves = $n^{\log_2 2}$ = n,

f(n) is within polylog factor of the number of leaf nodes

Master Theorem Case 2: $T(n) = \Theta(n \log n)$

Assuming that we have an infinite number of processors, and each recursive call in example 2 above can be executed in parallel:

Example 4: $T(n) = T(\frac{n}{2}) + \Theta(n^2)$

Here,

a = 1,

b = 2,

f(n) = $\Theta(n^2)$,

Number of leaves = $n^{\log_2 1} = n^0 = 1$,

f(n) dominates the number of leaf nodes

Master Theorem Case 3: $T(n) = \Theta(n^2)$