Methods of Solving Recurrences

Methods for Solving Recurrences

- **Iteration Method:** Convert the recurrence relation into a summation and solve it using a known series.
- **Recursion-Tree Method:**
  - Convert the recurrence into a tree.
  - Each node represents the cost incurred at the various levels of recursion.
  - Sum up the costs of all levels.
  - Used to “guess” a solution for the recurrence.
- **Master Method**

Iteration Method

Example - 1: Binary Search

$T(n) = c + T(n/2)$ //recurrence relation

$= c + c + T(n/4)$

$= c + c + c + T(n/8)$

//problem is subdivided in each iteration

Assume $n=2^k$ then, $k = \log n$

$T(n) = c + c + c + \ldots \text{ k-times} + T(n/2^k)$

$T(n) = k*c + T(1)$

$T(n) = c \log n$

Complexity of Binary Search= O(logn)

//Ignoring the coefficient

```
BINARY-SEARCH (A, lo, hi, x)
    if (lo > hi)
        return FALSE          //constant time: c1
    mid = (lo+hi)/2           //constant time: c2
    if x = A[mid]
        return TRUE           //constant time: c3
    if ( x < A[mid] )
        BINARY-SEARCH (A, lo, mid-1, x)
        //same problem of size n/2
    if ( x > A[mid] )
        BINARY-SEARCH (A, mid+1, hi, x)
        //same problem of size n/2
```

Mutually Exclusive: Only one of them is called during one iteration

Example - 2: Merge Sort

$T(n) = nc + 2T(n/2)$

//recurrence relation

Assume $n=2^k$ then, $k = \log n$

$= nc + 2(nc/2 + 2T(n/4))$

$= nc + nc + 4(nc/4 + 2T(n/8))$

$= 3nc + 2^3\, T(n/2^3)$

//problem is subdivided in each iteration

Similarly,

$T(n) = knc + 2^k\, T(n/2^k)$

$= cn \log n + n\, T(1)$

$T(n) = O(n \log n)$

//Ignoring the coefficient

```
MergeSort(A, left, right)

if (left < right)                             //constant time: c1

    mid = floor((left + right) / 2)           //constant time: c2

MergeSort(A, left, mid)                       //same problem of size n/2

MergeSort(A, mid+1, right)                    //same problem of size n/2

Merge(A, left, mid, right)                    //time proportional to size of n
```

$$T(n) = 2 * T(n/2) + nc$$

Recursive-Tree Method

Some important summations

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$\sum_{k=1}^{n} k^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1} \quad \text{If } |x| < 1 \text{ then } \lim_{n \to \infty} \sum_{k=0}^{n} x^k = \frac{1}{1-x}$$

$$\text{Harmonic Series} \quad H_n = \sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n} = \ln n + O(1)$$

Recurrence Tree Method

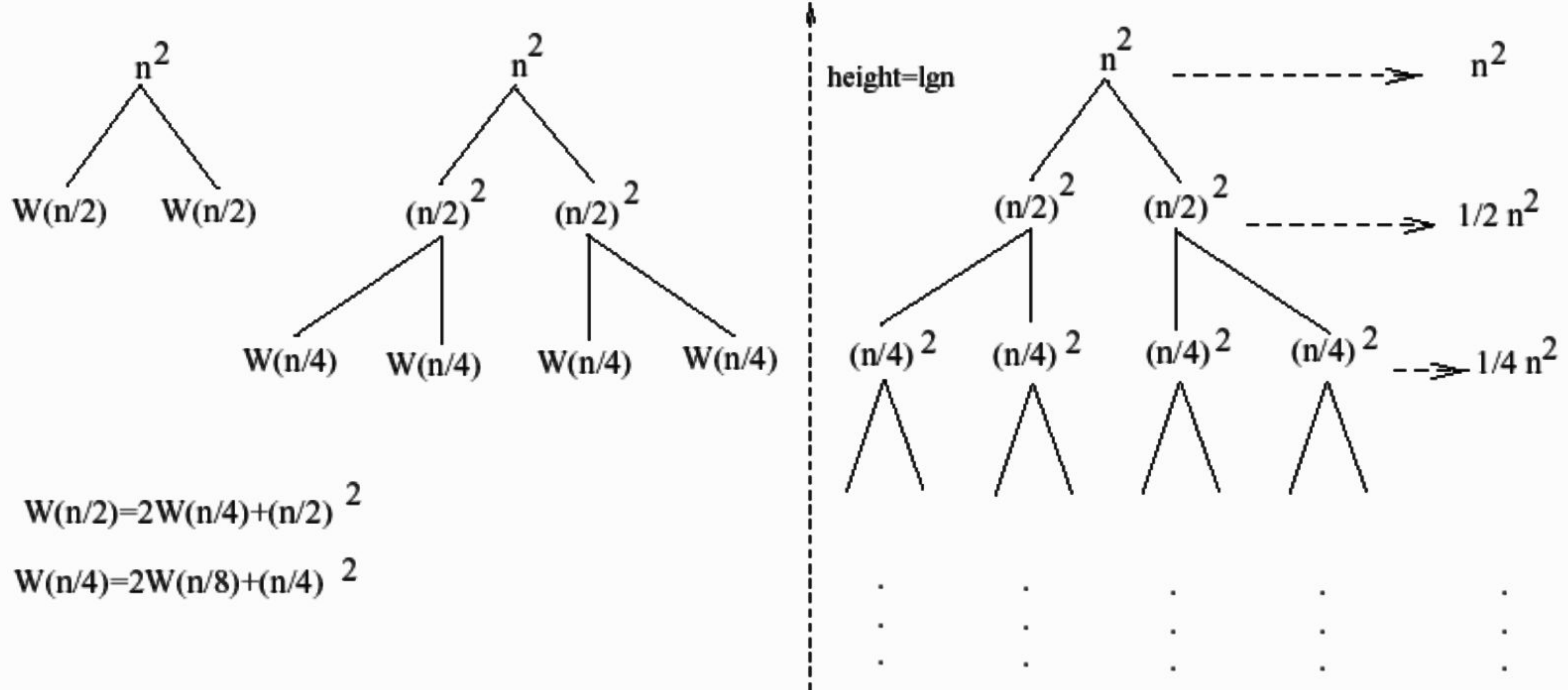
1. Draw a recurrence tree
2. Calculate the time taken by every level of tree.
3. Sum the work done at all levels.

To draw the recurrence tree:

- Start from the given recurrence
- Keep drawing till we find a pattern among levels.
- The pattern is typically a arithmetic or geometric series.

Example - 1: Recursive-Tree

$$W(n) = 2\,W(n/2) + n^2$$



Example - 1: Finding the complexity

- Sub-problem size at level i = $n/2^i$
- At level i: Cost of each node = $(n/2^i)^2$

  where the number of nodes = 2^i
- Total cost =2^i x $(n/2^i)^2$ = $(n^2/2^i)$
- If h is the height of the tree, $n/2^h = 1$, h = log n
- Total cost at all levels:

$$W(n) = \sum_{i=0}^{\lg n} \frac{n^2}{2^i}$$

$$= n^2 \sum_{i=0}^{\lg n} \left(\frac{1}{2}\right)^i$$

$$\le n^2 \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

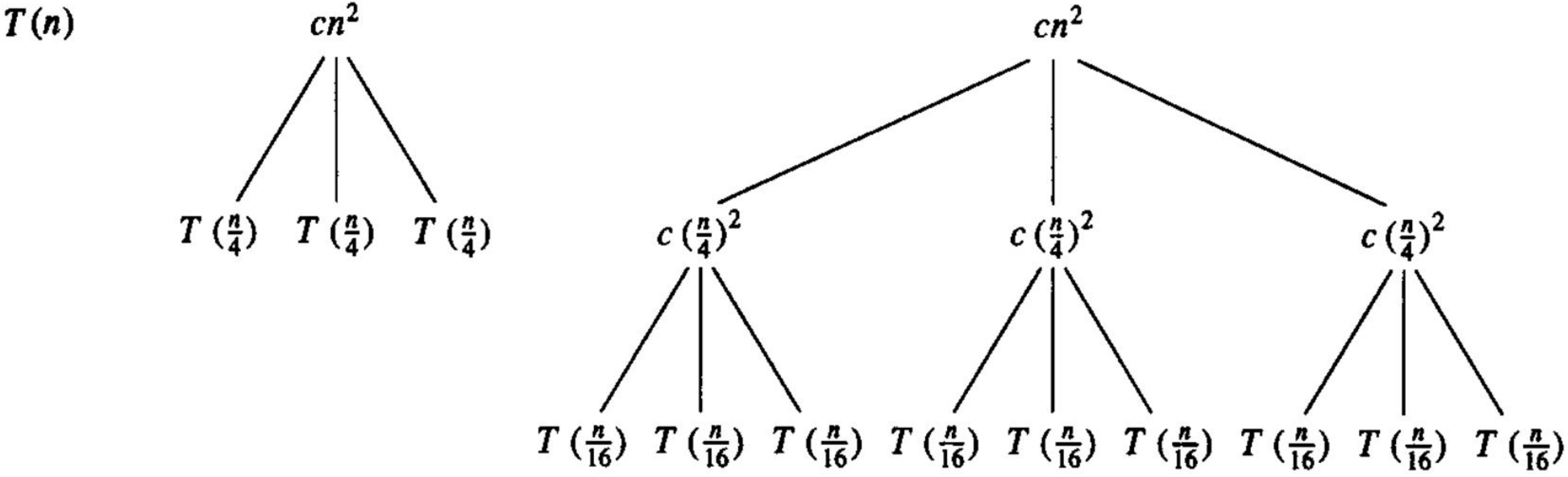
$$= n^2 \frac{1}{1 - \frac{1}{2}}$$

$$= 2n^2$$

$W(n) = O(n^2)$

Example - 2:

$T(n) = 3\,T(n/4) + cn^2$



What’s going on ...

$c8^2$

$c(8/2)^2$ $c(8/2)^2$ $c(8/2)^2$

$c(8/4)^2$ $c(8/4)^2$ $c(8/4)^2$

T(1) T(1) T(1)

- N = 8 to start with
- $8 = 2^3$
- Now it goes well till 3-1 levels
- At the last level whole bunch of T(1)s get generated
- How many T(1)
  - 3 raised to $\log_2 8$
- Treatment for last level is different
- Apply logarithmic power exchange

Example - 2: Finding the complexity

- Sub-problem size at level i = $n/4^i$
- At level i: Cost of each node = $c(n/4^i)^2$

  where the number of nodes = 3^i
- Total cost = $cn^2(3/16)^i$
- If h is the height of the tree, $n/4^h = 1$, $h = \log_4 n$
- Total cost at all levels: (last level has $3^{\log_4 n}$ nodes)

$$T(n) = \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$\leq \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$= \frac{1}{1-\frac{3}{16}} cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$= O(n^2)$$

$T(n) = O(n^2)$

Logarithmic power change proof

https://math.stackexchange.com/questions/453918/proof-of-logarithm-power-change

Practice Question

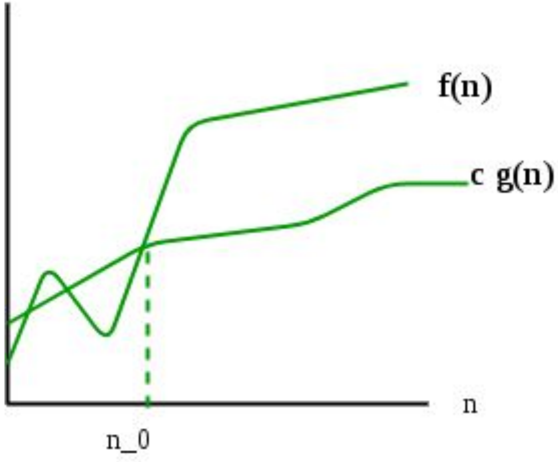
Q1. Draw the Recursion Tree for Merge Sort

Recurrence Relation: $T(n) = 2 * T(n/2) + nc$

Formative Assessment!

Theta and Omega

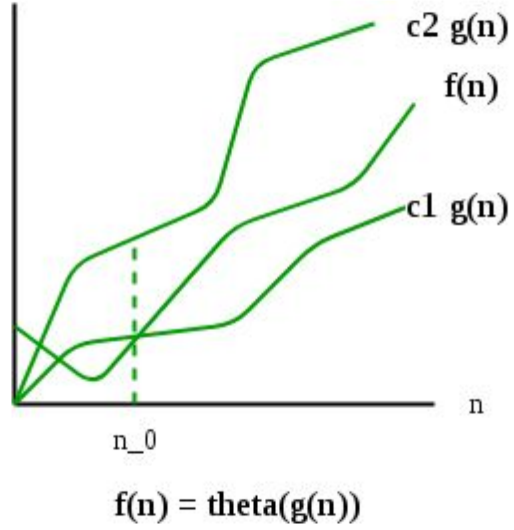
Big Omega



f(n) = Omega(g(n))

Ω (g(n)) = {f(n): there exist positive constants c and n_0 such that $0 \leq c*g(n) \leq f(n)$ for all $n \geq n_0$}.

Big Theta



$\Theta(g(n))$ = {f(n): there exist positive constants c1, c2 and n_0 such that $0 \le c1*g(n) \le f(n) \le c2*g(n)$ for all $n \ge n_0$}

Master Method

Master Method

The master theorem concerns recurrence relations of the form:

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\, b > 1$$

In the application to the analysis of a recursive algorithm, the constants and function take on the following significance:

- n is the size of the problem.
- a is the number of subproblems in the recursion.
- n/b is the size of each subproblem. (Here it is assumed that all subproblems are essentially the same size.)
- $f(n)$ is the cost of the work done outside the recursive calls, which includes the cost of dividing the problem and the cost of merging the solutions to the subproblems.

Case 1

Generic form

If $f(n) = \Theta\left(n^{c}\right)$ where $c < \log_b a$ (using Big O notation)

then:

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

Example

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

As one can see from the formula above:

$$a = 8,\ b = 2,\ f(n) = 1000n^2, \text{ so}$$
$$f(n) = \Theta\left(n^c\right), \text{ where } c = 2$$

Next, we see if we satisfy the case 1 condition:

$$\log_b a = \log_2 8 = 3 > c.$$

It follows from the first case of the master theorem that

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^3\right)$$

(indeed, the exact solution of the recurrence relation is $T(n) = 1001n^3 - 1000n^2$, assuming $T(1) = 1$).

Case 1

Generic form

If $f(n) = \Theta\left(n^c\right)$ where $c < \log_b a$ (using Big O notation)

then:

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

Case 2

Generic form

If it is true, for some constant $k \geq 0$, that:

$$f(n) = \Theta\left(n^c \log^k n\right) \text{ where } c = \log_b a$$

then:

$$T(n) = \Theta\left(n^c \log^{k+1} n\right)$$

Example

$$T(n) = 2T\left(\frac{n}{2}\right) + 10n$$

As we can see in the formula above the variables get the following values:

$$a = 2,\ b = 2,\ c = 1,\ f(n) = 10n$$

$$f(n) = \Theta\left(n^c \log^k n\right) \text{ where } c = 1, k = 0$$

Next, we see if we satisfy the case 2 condition:

$$\log_b a = \log_2 2 = 1, \text{ and therefore, yes, } c = \log_b a$$

So it follows from the second case of the master theorem:

$$T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right) = \Theta\left(n^1 \log^1 n\right) = \Theta\left(n \log n\right)$$

Thus the given recurrence relation $T(n)$ was in $\Theta(n \log n)$.

(This result is confirmed by the exact solution of the recurrence relation, which is $T(n) = n + 10n \log_2 n$, assuming $T(1) = 1$.

Case 2

Generic form

If it is true, for some constant $k \geq 0$, that:

$$f(n) = \Theta\left(n^c \log^k n\right) \text{ where } c = \log_b a$$

then:

$$T(n) = \Theta\left(n^c \log^{k+1} n\right)$$

Case 3

Generic form

If it is true that:

$$f(n) = \Theta\left(n^c\right) \text{where } c > \log_b a$$

then:

$$T\left(n\right) = \Theta\left(f(n)\right)$$

Example

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

As we can see in the formula above the variables get the following values:

$$a = 2,\ b = 2,\ f(n) = n^2$$
$$f(n) = \Theta\left(n^c\right), \text{ where } c = 2$$

Next, we see if we satisfy the case 3 condition:

$$\log_b a = \log_2 2 = 1, \text{ and therefore, yes, } c > \log_b a$$

So it follows from the third case of the master theorem:

$$T\left(n\right) = \Theta\left(f(n)\right) = \Theta\left(n^2\right).$$

Thus the given recurrence relation $T(n)$ was in $\Theta(n^2)$, that complies with the $f(n)$ of the original formula.

(This result is confirmed by the exact solution of the recurrence relation, which is $T(n) = 2n^2 - n$, assuming $T(1) = 1$.)

Case 3

Generic form

If it is true that:

$$f(n) = \Theta\left(n^c\right) \text{where } c > \log_b a$$

then:

$$T\left(n\right) = \Theta\left(f(n)\right)$$