Recurrences

- While sum play a role in analysis of iterations, recursive programs requires analysis of recurrences.
- A recurrence is expressed by itself using smaller parameteric values.
- Merge sort for example,

$$T(n) = \begin{cases} \Theta(1), \text{ if } n = 1 \\ 2T(n/2) + \Theta(n), \text{ if } n \geq 1 \end{cases}$$

  We know $T(n) = \Theta(n \log n)$.
- But how? – Substitution method involves
  – Make a good guess (?) then apply induction to find the constants and show that solution works.

Recurrences

- Consider the recurrence $T(n) = 2T(\lfloor n/2 \rfloor) + n$. Assume solution to be $O(n \log n)$. So

$$
\begin{aligned}
T(n) &\leq 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor) + n \\
&\leq cn \log(n/2) + n \\
&\leq cn \log n - cn \log 2 + n \\
&\leq cn \log n - cn + n \\
&\leq cn \log n, \text{ for } c \geq 1
\end{aligned}
$$

- Use mathematical induction to show that the solution holds.

Recurrences

- The constant c must satisfy the boundary conditions.
- Suppose we choose c to be large, but how large?
- If $T(1) = 1$, then as $1 \log 1 = 0$, even if we let c to be very large the solution does not satisfy boundary.
- Also c cannot be arbitrarily large.
- Since asymptotic time complexity is needed, it suffices to let conditions hold when $n \geq n_0$, where n_0 is a fixed positive number.
- So, $T(k)$ for some $k \geq 2$ can be considered as boundary conditions for induction.
- From recurrence $T(2) = 4$, $T(3) = 5$.
- Letting $c \geq 2$ implies

$$T(2) \leq c \times 2 \log 2, \text{ and } T(3) \leq c \times 3 \log 3,$$

thus, c satisfies boundary conditions.

What is a Good Guess

- Relies on intuition.
- For example,

$$T(n) = 2T(\lfloor n/2 \rfloor + 18) + n$$

- We can again assume the solution to be $O(n \log n)$.
  - Notice that the problem is divided into smaller problems each of size nearly half of input size, assuming n to be large.
  - So excess of 18 in problem size compared to earlier recurrence does not have any effect as far as asymtotic running time.

Proving Good Guess

- Consider

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

- Guess is $O(n)$. But when we try to proove correctness

$$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 = cn + 1$$

does not imply that $T(n) \leq cn$ for any choice of c.

Proving Good Guess

- We are off by a constant factor. But induction does not work if the inductive hypothesis is not proved exactly.
- To overcome it, substract a constant from the guess, say b, then try to prove solution $T(n) \leq cn - b$.

  $T(n) \leq c\lfloor n/2 \rfloor - b + c\lceil n/2 \rceil - b + 1 = cn - 2b + 1$

  Implies

  $$T(n) \leq cn - b, \text{ as long as } b \geq 1$$

- Now c can be chosen to handle boundary conditions.
- Similar problems also arise when we try to prove $T(n) = 2T(\lfloor n/2 \rfloor) + n$ as $O(n)$ as follows.

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor) + n \\ T(n) &\leq cn + n = O(n) \end{aligned}$$

Changing Variables

- Consider $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n$.
- Change $n = 2^m$, then $T(2^m) = 2T(\lfloor 2^{m/2}\rfloor) + m$.
- Rename $T(2^m)$ as $S(m)$, then the recurrence relation becomes

$$S(m) = 2S(m/2) + m$$

- So solution becomes $S(m) = O(m \log m) = O(\log n \log\log n)$.

Method of Iteration

- Simply go on iterating the recurrence down to base case.
- Consider, the iteration

$$\begin{aligned} T(n) &= n + 3T(\lfloor n/4 \rfloor) \\ &= n + 3(n/4 + 3T(\lfloor n/16 \rfloor)) \\ &= n + 3n/4 + 9(n/16 + 3T(\lfloor n/64 \rfloor)) \\ &= n + 3n/4 + 9n/16 + 27T(\lfloor n/64 \rfloor) \end{aligned}$$

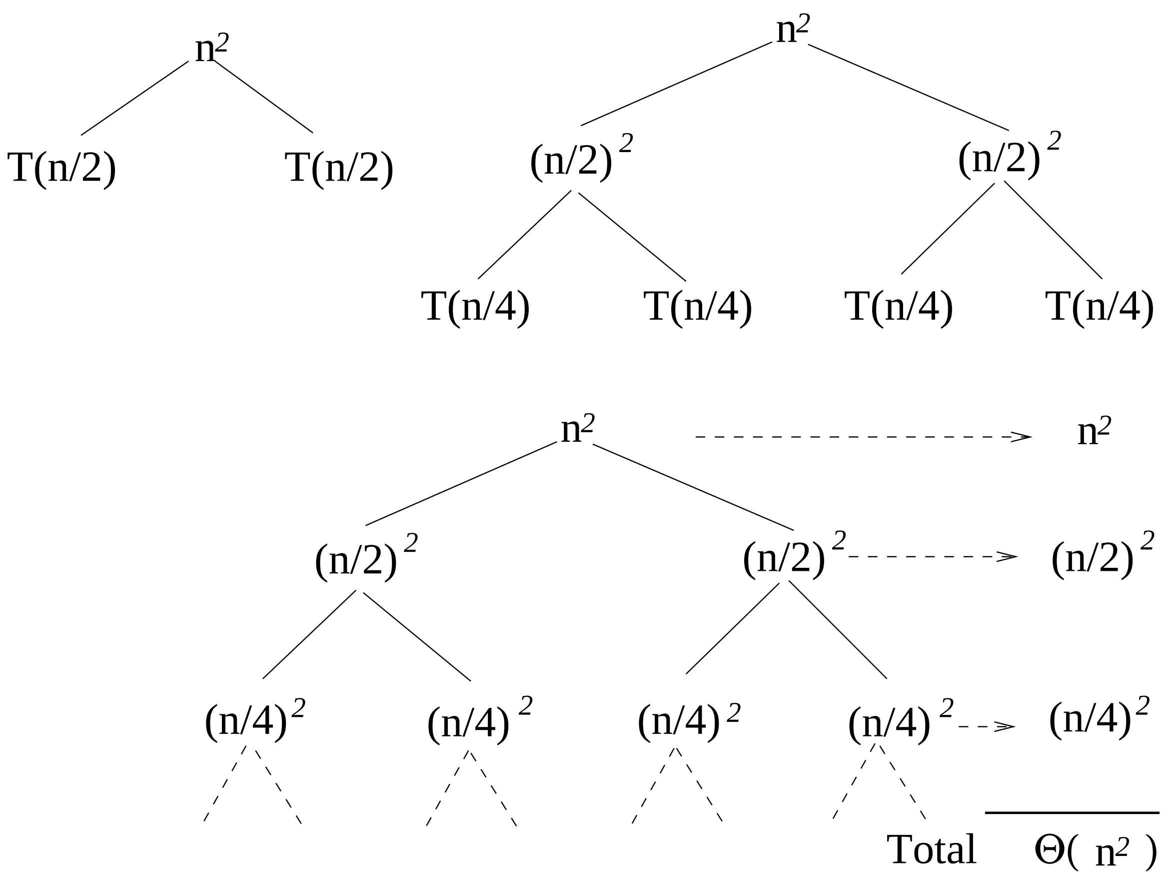
- Obviously the iteration can continue till $\lfloor n/4^i \rfloor = 1$ or when i exceeds $\log_4 n$ times.
- Using $\lfloor n/4^i \rfloor < n/4^i$,

$$\begin{aligned} T(n) &\le n + 3n/4 + 9n/16 + \ldots + 3^{\log_4 n}\Theta(1) \\ &\le \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i + \Theta(n^{\log_4 3}) \\ &= 4n + o(n), \text{ as } \log_4 3 < 1, \Theta(n^{\log_4 3}) = o(n) \\ &= O(n) \end{aligned}$$

Recursion Tree

- It helps to visualize progress of iteration.
- Consider the figure below for

$$T(n) = 2 \cdot T(n/2) + n^2$$



Master Theorem

- In general a recurrence relation may be of the form
$$T(n) = aT(n/b) + f(n)$$
for constants $a \geq 1$, $b > 1$ and $f(n)$ is an aymptotically positive funtion.

- Master theorem states that

  1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$.
  2. If $f(n) = O(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$.
  3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and sufficiently large n then $T(n) = \Theta(f(n))$.

Master Theorem

- Intuitively $f(n)$ is compared with function $n^{\log_b a}$ and solution to recurrence is determined by larger of the two functions.
- In case 1 $n^{\log_b a}$ is larger and the solution is of $O(n^{\log_b a})$.
- In case 2 both are same. In this case factor $\log n$ is multiplied.
- In case 1 $n^{\log_b a}$ is smaller. The complexity is determined by $f(n)$.

Example: Using Master Theorem

- Consider $T(n) = 9T(n/3) + n$. Here $a = 9, b = 3$, and $f(n) = n$. Since $f(n) - O(n^{\log_3 9-\epsilon})$ where $\epsilon = 1$, case 1 applies.
- So time complexity will be $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$
- Consider $T(n) = T(2n/3) + 1$. Here $a = 1, b = 3/2$, and $f(n) = 1$. Since $n^{\log_{3/2} 1}) = n^0 = 1$. So, case 2 applies.
- Thus time complexity is $\Theta(n^{\log_b a} \log n) = \Theta(\log n)$

Example: Using Master Theorem

- Consider $T(n) = 3T(n/4) + n \log n$. Here $a = 3, b = 4$, and $f(n) = n \log n$. Since $n^{\log_4 3} = n^{0.793}$, and $f(n)/n^{0.793} = n^{\epsilon} \log n$, where $\epsilon = 0.207$. So, case 3 applies.
- Thus time complexity is $\Theta(f(n)) = \Theta(n \log n)$.
- Consider $T(n) = 2T(n/2) + n \log n$. Here $a = 2, b = 2$, and $f(n) = n \log n$.
- Since $n^{\log_2 2} = n$, and clearly $n \log n > n$. But only logarithmically which is asymtotically smaller than n^{ϵ} for any $\epsilon > 0$. So, case 3 does not apply. It falls in the gap between case 2 and case 3.

Recurrence and Annihilators

- Let $A = \{a_0, a_1, a_2, \ldots\}$ be a given sequence on which we use a left shift operator E. Then

$$E \cdot A = \{a_1, a_2, a_3, \ldots\}$$

- For concretness let $T = \{2^0, 2^1, 2^2, 2^3, \ldots\}$, then

$$2 \cdot T = \{2^1, 2^2, 2^3, 2^4, \ldots\}$$

- Similarly

$$E \cdot T = \{2^1, 2^2, 2^3, 2^4, \ldots\}$$

- Thus

$$(E - 2) \cdot T = \{0, 0, 0, 0, \ldots\}$$

- Then $(E - 2)$ is said to annihiliate A.
- In our first proof we use the annihiliator operation to solve Master theorem by applying technique of domain transformation.

Proof of Master Theorem

- Let $n = b^k$.
- Applying domain transformation generalized recurrence can be expressed as

$$\begin{aligned} T(n) &= a \cdot T(n/b) + \Theta(n^k) \\ &= a \cdot T(b^m/b) + \Theta(b^{mk}) \\ &= a \cdot T(b^m/b) + \Theta(b^{mk}) \\ &= a \cdot T(b^{m-1}) + \Theta(b^{mk}) \end{aligned}$$

- Setting $S(m) = T(b^m)$,

$$S(m) = a \cdot S(m-1) + \Theta(b^{mk})$$

$$S(m) - a \cdot S(m-1) = \Theta(b^{mk})$$

- Substituting m by $m+1$,

$$S(m+1) - a \cdot S(m) = \Theta(b^{(m+1)k})$$

- As b and k are constants b^k is constant, therefore, $b^{(m+1)k} = b^k \cdot b^{mk} = \Theta(b^{mk})$.
- Thus $S(m+1) - a \cdot S(m) = \Theta(b^{mk})$

Proof of Master Theorem

- Now we apply annihilator $(E-a)$ on the equation

$$S(m+1) - a \cdot S(m) = \Theta(b^{mk})$$

to obtain

$$(E - a) \cdot S(m) = \Theta(b^{mk})$$

- To eliminate $\Theta(b^{mk})$ apply annihilator $(E - b^k)$ to the last equation

$$(E - b^k)(E - a) \cdot S(m) = 0$$

Proof of Master Theorem

- If $a = b^k$ then we have a double root for characteristic equation. So general solution should be

$$\begin{aligned} S(m) &= (c_1 + c_2 m)a^m \\ T(2^m) &= (c_1 + c_2 m)a^m \\ T(n) &= (c_1 + c_2 \log_b n)a^{\log_b n} \\ &= (c_1 + c_2 \log_b n)n^{\log_b a} \end{aligned}$$

- So this proves the case 2 of master theorem $T(n) = \Theta(\log n \cdot n^{\log_b a})$.

Proof of Master Theorem

- If $a \neq b^k$ then

$$
\begin{aligned}
S(m) &= c_1 a^m + c_2 b^{km} \\
T(2^m) &= c_1 a^{\log_b n} + c_2 b^{k \log_b n} \\
T(n) &= c_1 n^{\log_b a} + c_2 b^{\log_b n^k} \\
&= c_1 n^{\log_b a} + c_2 n^k \\
&= \Theta\left(n^{\log_b a} + n^k\right)
\end{aligned}
$$

- So depending on whether n^k dominates $n^{\log_b a}$ we have the corresponding result of master theorem covering cases 1 and 3 respectively.

Direct Proof

- Let $b > 1$ and n be an exact power of b, i.e., $n = b^k$ for $k > 0$. So
$$T(n) = \begin{cases} \Theta(1) & \text{, if } n = 1 \\ aT(n/b) + f(n) & \text{, if } n = b^i \end{cases}$$
where i is a positive integer.
- Apply iteration method
$$T(n) = f(n) + af(n/b) + \ldots + a^{\log_b n}T(1)$$
- Since $a^{\log_b n} = n^{\log_b a}$ $a^{\log_b n}T(1) = n^{\log_b a}$.
- Thus
$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

Proof: Case 1

- For proving case 1, $f(n) = n^{\log_b a - \epsilon}$

$$\sum_{0}^{\log_b n - 1} a^j \cdot f(n/b^j) = \sum_{0}^{\log_b n - 1} a^j \cdot \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}$$
$$= n^{\log_b a - \epsilon} \sum_{0}^{\log_b n - 1} \left(\frac{a}{b}\right)^j \cdot (b^\epsilon)^j$$

since $b > a$

$$\sum_{0}^{\log_b n - 1} a^j \cdot f(n/b^j) \le n^{\log_b a - \epsilon} \sum_{0}^{\log_b n - 1} (b^\epsilon)^j$$

Proof: Case 1 (contd.)

- Since the sum on RHS of above inequality is now in form of GP

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

- Thus

$$\sum_{0}^{\log_b n-1} a^j \cdot f(n/b^j) = O(n^{\log_b a-\epsilon})$$

Proof: Case 2

- In case 2,

$$f(n) = \Theta(n^{\log_b a}) \Rightarrow f(n/b^j) = \Theta((n/b)^{\log_b a})$$

- Thus

$$\begin{aligned}\sum_{0}^{\log_b n-1} a^j \cdot f(n/b^j) &= \sum_{0}^{\log_b n-1} a^j \cdot (n/b^j)^{\log_b a} \\ &= n^{\log_b a} \sum_{0}^{\log_b n-1} (a/(b^{\log_b a}))^j \\ &\leq n^{\log_b a} \log_b n\end{aligned}$$