Solving Recurrences & The Master Theorem

Solving Recurrences

- Substitution method
- Iteration method
- Master method

Solving Recurrences – contd.

- The substitution method
  - A.k.a. the "making a good guess method"
  - Guess the form of the answer, then use induction to find the constants and show that solution works
  - *Run an example*: merge sort
    - $T(n) = 2T(n/2) + cn$
    - We guess that the answer is $O(n \lg n)$
    - Prove it by induction
  - Can similarly show $T(n) = \Omega(n \lg n)$, thus $\Theta(n \lg n)$

Solving Recurrences – contd.

- The "iteration method"
  - Expand the recurrence
  - Work some algebra to express as a summation
  - Evaluate the summation

Example - Merge Sort

```
MergeSort(A, left, right) {
  if (left < right) {
     mid = floor((left + right) / 2);
     MergeSort(A, left, mid);
     MergeSort(A, mid+1, right);
     Merge(A, left, mid, right);
  }
}

// Merge() takes two sorted subarrays of A and
// merges them into a single sorted subarray of A
//     (how long should this take?)
```

Example Merge Sort - Analysis

Statement / Effort

```
MergeSort(A, left, right) {                   T(n)
   if (left < right) {                        Θ(1)
      mid = floor((left + right) / 2);           Θ(1)
      MergeSort(A, left, mid);                   T(n/2)
      MergeSort(A, mid+1, right);                T(n/2)
      Merge(A, left, mid, right);                Θ(n)
   }
}
```

- So T(n) = $\Theta(1)$ when $n = 1$, and
  $2T(n/2) + \Theta(n)$ when $n > 1$
- What is T(n)?

Example Merge Sort - Recurrences

- The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

  is a *recurrence*.
- T(n) = 2T(n/2) + cn → T(n) = Θ(n lg n)

Solving Recurrences – Example

- Example: For a > = 1, and b > 1, and n a positive integer, Find an asymptotic expression for T(n).

$$\boxed{T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}}$$

Solving Recurrences – Example - contd.

- Example: Continued

$$T(n) = \begin{cases} \Theta(n) & a < b \\ \Theta(n \log_b n) & a = b \\ \Theta\left(n^{\log_b a}\right) & a > b \end{cases}$$

The Master Theorem

- Given: a *divide and conquer* algorithm
  - An algorithm that divides the problem of size n into a subproblems, each of size n/b
  - Let the cost of each stage (i.e., the work to divide the problem + combine solved subproblems) be described by the function f(n)
- Then, the Master Theorem gives us a cookbook for the algorithm's running time:

The Master Theorem

- if T(n) = aT(n/b) + f(n) then

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a - \varepsilon}\right) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right) & f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \text{ AND} \\ & af(n/b) < cf(n) \text{ for large } n \end{cases} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \begin{matrix} \varepsilon > 0 \\ c < 1 \end{matrix}$$

The simple format of master theorem

- $T(n)=aT(n/b)+cn^k$, with a, b, c, k are positive constants, and $a \geq 1$ and $b \geq 2$,
- $$T(n) = \begin{cases} O(n^{\log_b a}), & \text{if } a>b^k. \\ O(n^k \log n), & \text{if } a=b^k. \\ O(n^k), & \text{if } a<b^k. \end{cases}$$

12

Using The Master Method

- $T(n) = 9T(n/3) + n$
  - $a = 9$, $b = 3$, $f(n) = n$
  - $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$
  - Since $f(n) = O(n^{\log_3 9 - \varepsilon})$, where ε=1, case 1 applies:

$$T(n) = \Theta\left(n^{\log_b a}\right) \text{when } f(n) = O\left(n^{\log_b a - \varepsilon}\right)$$

  - Thus the solution is $T(n) = \Theta(n^2)$