Analysis of Mergesort

Input size $n = \text{length of list to be sorted}$; $T_{ms}(n) = \text{running time of mergesort}$.

1. Call split on list of length $n$; takes $O(n)$ time (in book).
2. Then, mergesort calls itself on two lists of size $n/2$, taking $2T_{ms}(n/2)$.
3. Finally, call merge on two lists of total length $n$, taking $O(n)$ time (in book).
   - When $n = 1$ (basis), there are no calls; mergesort takes $O(1)$ time.

Recurrence

$$
T_{ms}(1) = O(1) \\
T_{ms}(n) = O(n) + 2T_{ms}(n/2)
$$

- Eliminate $O(1)$ and $O(n)$ in favor of concrete constants:
  $$
  T_{ms}(1) = a \\
  T_{ms}(n) = bn + 2T_{ms}(n/2)
  $$

Guess-And-Check Solutions

“Guess” the form of an upper bound on $T(n)$.

- Try to prove the bound inductively; in the process, we may get some constraints on parameters in the guessed form.

- Statement $S(n)$: (Not quite like pp. 148-9)
  $$
  T_{ms}(n) \leq cn \log_2 n + dn
  $$

- We prove $S(n)$ for $n$ a power of 2.

- $c$ and $d$ are parameters to be discovered.

Basis: If $n = 1$, we have $T_{ms}(1) = a$. If we want $a = T_{ms}(1) \leq (c)(1)(\log_2 1) + (d)(1)$ we must have $d \geq a$ because $\log_2 1 = 0$.

Induction: Assume

$$
T_{ms}(n/2) \leq (cn/2) \log_2 (n/2) + dn/2
$$
• Then $T_{ms}(n) = bn + 2T_{ms}(n/2) \leq bn + cn(\log_2 n - 1) + dn$.

• We want to show $T_{ms}(n) \leq cn\log_2 n + dn$.
  Only way: show
  
  $bn + cn\log_2 n - cn + dn \leq cn\log_2 n + dn$

  i.e., $bn \leq cn$.

• Conclusion: Proof goes through if $d \geq a$ and $c \geq b$. e.g., let $d = a$ and $c = b$:
  
  $T_{ms}(n) \leq bn\log_2 n + an$

  i.e., $T_{ms}(n)$ is $O(n\log n)$.

An Exponential Recurrence

How many strings of length $n$ over symbols 0, 1, 2 have no identical, consecutive symbols?

Basis: $T(1) = 3$; they are "0", "1", "2".

Induction: $T(n) = 2T(n-1)$ for $n > 1$. Expand:

$T(n) = 4T(n-2)$
$T(n) = 8T(n-3)$

$T(n) = 2^{n-1}T(1) = 3 \times 2^{n-1}$

Varieties of Recurrences

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

<table>
<thead>
<tr>
<th>$T(n-1)$</th>
<th>$T(n/2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$nf(n)$ if poly. $f(n)$ for larger</td>
<td>log $n$ if $f(n) = 1$ $f(n)$ for others</td>
</tr>
<tr>
<td>exponential</td>
<td>$n\log n$ if $f(n) = n$ $f(n)$ for larger</td>
</tr>
</tbody>
</table>

Linear Recursions

These are recursions in which $T(n)$ is defined in terms of $T(n-a)$ for various integers $a > 0$.

Example: How many strings of $a$’s, $b$’s, and $c$’s are there such that all $b$’s appear in consecutive pairs and all $c$’s appear in consecutive pairs. $a$’s may appear anywhere.
• We can define this set of strings recursively:

**Basis:** \( \epsilon \), the empty string, is acceptable.

**Induction:** If \( w \) is an acceptable string, then so are \( wa, wbb, \) and \( wcc \).

• Thus, acceptable strings include \( a, bb, bba, \)
  \( acc \), etc.

• Let \( T(n) \) be the number of acceptable strings of length \( n \).

**Basis:** \( T(0) = 1 \) and \( T(1) = 1 \) (the strings \( \epsilon \) and \( a \), are counted, respectively).

**Induction:** \( T(n) = T(n-1) + 2T(n-2) \). Every acceptable string of length \( n \) either is an acceptable string of length \( n-1 \) followed by \( a \), or an acceptable string of length \( n-2 \) followed by \( bb \) or \( cc \).

**Solving Linear Recursions**

Expansion doesn’t usually work, but guess-and-check works if we know the trick.

• Guess an exponential solution \( T(n) = \lambda^n \).

• Substitute this guess for \( T(n) \) and \( T(n-k) \) for all \( k \)’s that appear.

• Divide through by \( \lambda \) to the largest possible power.

• Result is a polynomial in \( \lambda \) that equals 0. Solve this equation for possible values of \( \lambda \), say \( \lambda_1, \lambda_2, \ldots, \lambda_r \).

• Assume \( T(n) = \sum_{i=1}^{r} c_i \lambda_i \).

• Use basis values to solve for the \( c_i \)’s.

**Example:** Consider \( T(n) = T(n-1) + 2T(n-2) \).

• Substitute \( T(n) = \lambda^n \): \( \lambda^n = \lambda^{n-1} + 2\lambda^{n-2} \).

• Divide by \( \lambda^{n-2} \): \( \lambda^2 = \lambda + 2 \), or \( \lambda^2 - \lambda - 2 = 0 \).

• Solve quadratic equation: \( \lambda = 2, \lambda = -1 \).

• Trial solution: \( T(n) = a2^n + \beta(-1)^n \).
• Use basis for $n = 0, 1$: $\alpha + \beta = 1; 2\alpha - \beta = 1$.
• Solve: $\alpha = 2/3; \beta = 1/3$.
• Thus, $T(n) = (2^{n+1} + (-1)^n)/3$.
  □ For $n = 2, 3, 4, 5, \ldots$, $T(n) = 3, 5, 11, 21, \ldots$. 

Glitch: Multiple Roots of Polynomial

If $\lambda_i$ appears $k \geq 1$ times as a root of the polynomial, then you need to use terms $\lambda_i^n$, $n \lambda_i^n$, $n^2 \lambda_i^n$, $\ldots$, $n^{k-1} \lambda_i^n$.

Example: $T(n) - 4T(n - 1) + 4T(n - 2) = 0$.
Assume basis: $T(0) = 2; T(1) = 6$.
• $\lambda = 2$ is a double root.
• Trial solution is $T(n) = \alpha 2^n + \beta n 2^n$.
• Basis gives $\alpha = 2; 2\alpha + 2\beta = 6$.
• Solution: $T(n) = 2^{n+1} + n 2^n$, or $T(n) = (n + 2) 2^n$