1 The Master Method (Theorem)

1.1 Recurrence Format

1. Base Case: $T(n) \leq$a constant for all sufficiently small n
2. For all larger n:

$$T(n) \leq aT(\frac{n}{b}) + O(n^d)$$

   Where:
   $a =$ number of recursive calls (≥ 1)
   $b =$ input size shrinkage factor (> 1)
   $d =$ exponent in running time of "combine step" (≥ 0) (outside of loop)
   [a, b, d independent of n]

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

In first case the base of log doesn't matter (differs by constant factor). In third case, log is in the exponent, so it matters.

Ex. 1: Merge Sort

$a = 2$ number of recursive calls

$b = 2$ we recurse on $\frac{1}{2}$ of the array

$d = 1$ merge step

Case 1:

$$T(n) \leq O(n^d \log n) = O(n \log n)$$

Ex. 2: Binary Search

$a = 1$ compare element to the middle element and recurse on left or right

$b = 2$ half of the array

$d = 0$ outside of loop you just do one comparison, constant time

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

Ex. 3: Integer Multiplication (no Gauss trick)

$a = 4$

$b = 2$ $\frac{n}{2}$ digits

$d = 1$ linear time

Case 3:

$$T(n) = O(n^{\log_b a}) = O(n^{\log_2 4}) = O(n^2)$$

Ex. 4: With Gauss's trick:

$a = 3$

$b = 2$

$c = 1$

Case 3:

$$T(n) \quad = \quad O(n^{\log_b a}) = O(n^{\log_2 3}) = O(n^{1.59})$$

Ex. 5: Strassen's Matrix Multiplication

$a = 7$

$b = 2$

$d = 2$ linear in the matrix size (quadratic number of entries)

$$T(n) = O(n^{\log_2 7}) = O(n^{2.81})$$

Ex. 6: Fictitious recurrence

$T(n) \leq 2T(\frac{n}{2}) + O(n^2)$

$a = 2$

$b = 2$

$d = 2$

Case 2:

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

1.2 Proof I

Recursion tree approach, remember what tree types the three cases correspond to.

Assume: recurrence is

1. $T(1) \leq c$
2. $T(n) \leq aT(\frac{n}{b}) + cn^d$
3. n is a power of b

Idea: generalize MergeSort analysis

At each level $j = 0, 1, 2, \ldots, \log_b n$, there are $\mathbf{a^j}$ subproblems each of size $\mathbf{\frac{n}{b^j}}$.

$\log_b n$ - number of times you can divide n by b before reaching 1.

[5:40] - The Recursion Tree

1.2.1 Work at a Single Level

Total work at level j [ignoring work in recursive calls]

$\leq a^j \times c \times \left[\frac{n}{b^j}\right]^d = cn^d \times \left[\frac{a}{b^d}\right]^j$

Total Work

Summing ? all levels $j = 0, 1, 2, \ldots, \log_b n$:

$$^*Total\,work \leq cn^d \times \sum_{j=0}^{\log_b n} \left[\frac{a}{b^d}\right]^j$$

1.3 Interpretation of the 3 Cases

How to think about (*)

Our upper bound on the work at level j:

$$cn^d \times \left[\frac{a}{b^d}\right]^j$$

*is a sum of the above expressions

1.3.1 Interpretation

$a =$rate of subproblem proliferation (RSP)

$b^d=$ rate of work shrinkage per subproblem (RWS)

If RSP $<$ RWS, then the amount of work is decreasing with the recursion level j. Most work at the root [might expect $O(n^d)$].

If RSP $>$ RWS, then the amount of work is increasing with the recursion level j. Most work at the leaves [might expect $O(\#leaves)$.

If RSP $=$ RWS, then the amount of work is the same at every recursion level j. [merge sort, expect $O(n^d \log n)$].

1.4 Proof II

$$Total\,work \leq cn^d \times \sum_{j=0}^{\log_b n} \left[\frac{a}{b^d}\right]^j \qquad (*)$$

If $a = b^d$, then

$$\begin{aligned} total\,work &= cn^d(\log_b n + 1) \\ &= O(n^d \log n) \end{aligned}$$

If $a < b^d$, then

call $\frac{a}{b^d} := r \leq$a constant (independent of n)

$$total\,work = O(n^d)$$

If $a > b^d$, then

call $\frac{a}{b^d} := r > 1$

$$total\,work = O(n^d \times (\frac{a}{b^d})^{\log_b n})$$

Note: $b^{-d\log_b n} = (b^{\log_b n})^{-d} = n^{-d}$, so

$$total\,work = O(a^{\log_b n}) = O(\#leaves)$$
$$a^{\log_b n} = n^{\log_b a}$$

Expression on left more intuitive, on right simpler to apply.