Data Structures

Analysis of divide-and-conquer algorithms

- If an algorithm contains recursive calls to itself, then its running time can be expressed by a recurrence equation or recurrence
- Running time of algorithm on input size n is expressed in terms of running time on smaller input sizes
- Assume that, if $n \leq c$, then the problem can be solved directly
- If $n > c$, then the problem is divided into "a" subproblems of size "n/b"
- $D(n)$ is the time to divide and $C(n)$ is the time to combine

$$T(n) = \begin{cases} \text{time to solve the trivial problem} & \text{if } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Analysis of divide-and-conquer algorithms

- Assume that n is a power of 2
- Recurrence for merge sort:

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

Solving recurrences

- Four methods to solve recurrences
- Iterative substitution method
  - Expand the recurrence by substitution and observe patterns
- The recursion tree method
  - Similar to iterative substitution method, visual approach
- Substitution method (guess-and-test method)
  - Make an educated guess of the closed form solution and then justify the solution usually by induction
- The master method
  - A general and cook-book method to determine asymptotic characterization of a wide variety of recurrences

Iterative substitution method

- Substitute the general form of the recurrence for each occurrence of function T on the right hand-side
- Substitutions are done with the hope that at some point we see a pattern that can converted into a general closed form equation (with T appearing only on the left hand-side

Iterative substitution method

- Ex: Consider the recurrence equation of merge-sort

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

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

$$4T(n/4) = 4(2(T(n/8)) + n/4)$$
$$= 8T(n/8) + n$$
$$T(n) = 8T(n/8) + 3n$$

Iterative substitution method

Continuing in this manner, we obtain

$$T(n) = 2^k T(n/2^k) + kn$$

Using $k = \log n$ ($\log n$ is $\log_2 n$), we obtain

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

$$= n + n \log n$$

The recursive tree method

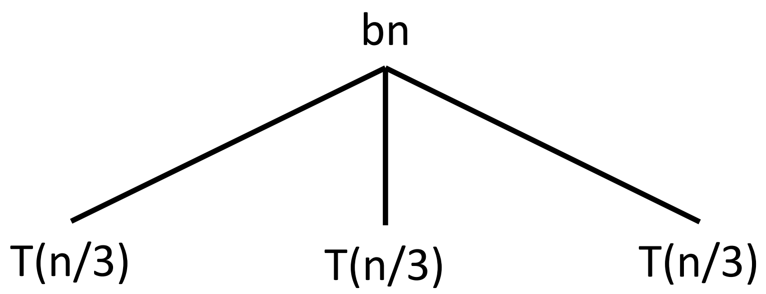
- This technique also uses repeated substitutions, not in an algebraic manner, but in visual manner
- Draw a tree R where each node represents a different substitution of the recurrence equation
- Each node v of R has a value of the argument n of T(n)
- An overhead is associated with each node v in R
- Solution: the summation of overheads associated with all nodes in R

The recursive tree method

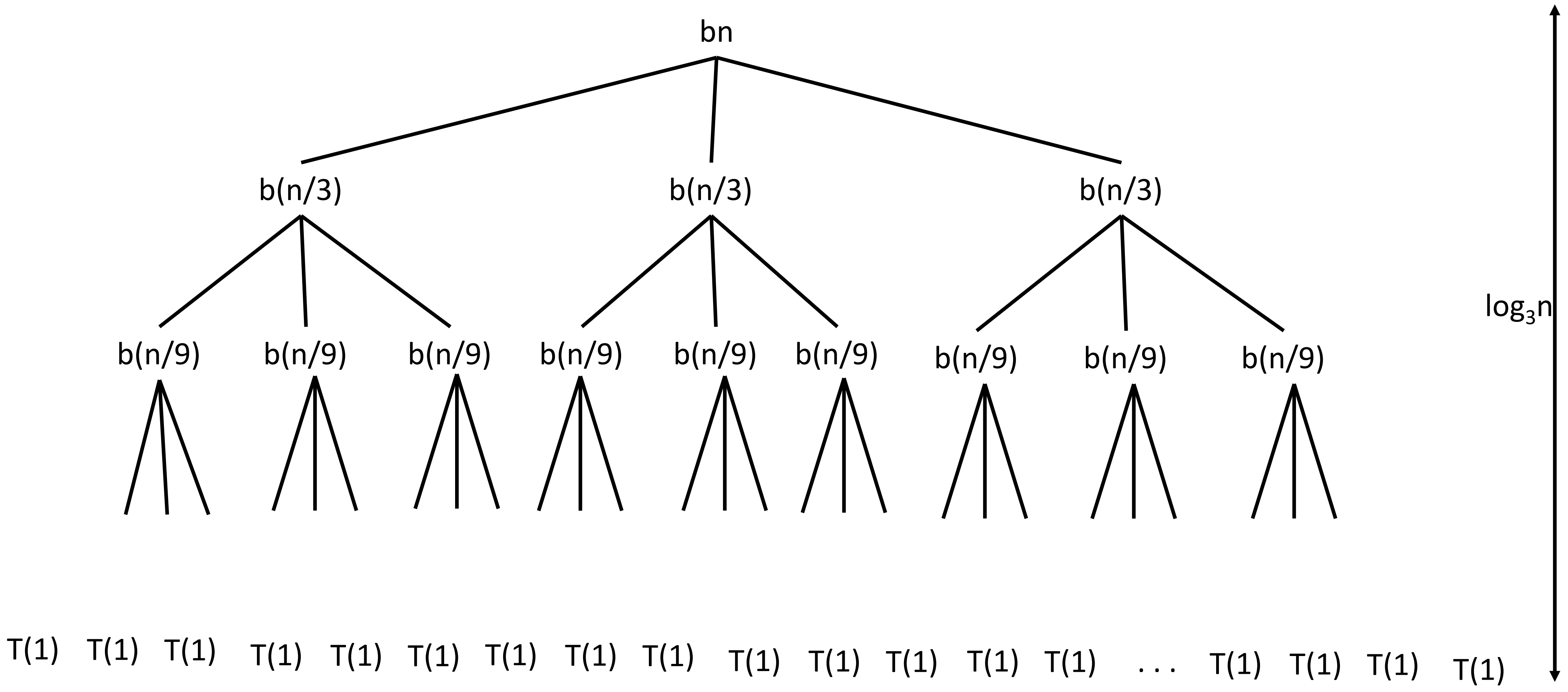
- Ex: Consider the below given recurrence (n is an exact power of 3)

$$T(n) = \begin{cases} c & \text{if } n < 3, \\ 3T(n/3) + bn & \text{otherwise} \end{cases}$$

What does this recurrence equation represent?



The recursive tree method



The guess-and-test method

- First make an educated guess as to what a closed form solution of the recurrence equation might look like
- Then justify the guess usually by induction
- Powerful yet must be able to guess the form of the solution

The guess-and-test method

- Ex: Consider the following recurrence

$$T(n) = \begin{cases} 1 & \text{if } n < 2, \\ 2T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

- First guess: $T(n) \le cn \log n$, for some constant $c > 0$
- Assume that this first guess is inductive hypothesis for input sizes smaller than n
- This guess holds true for $m<n$, in particular $m = \lfloor n/2 \rfloor$
- $T(\lfloor n/2 \rfloor) \le c\lfloor n/2 \rfloor \log (\lfloor n/2 \rfloor)$

The guess-and-test method

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

The last step holds as long as $c \ge 1$

The guess-and-test method

- We have to show that our guess $T(n) \le cn \log n$ works for boundary conditions as well
- According to the guess, $T(1) \le c\,1 \log 1 = 0$
- According asymptotic notation, we have to show that $T(n) \le cn \log n$, for $n \ge n_0$
- Do not consider the problematic boundary condition in the induction proof

$$T(n) = \begin{cases} 1 & \text{if } n < 2, \\ 2T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

- T(4), T(5), . . . do not directly depend on T(1)
- Only T(2) and T(3) depend on T(1)

The guess-and-test method

- Replace T(1) with T(2) and T(3) (base cases of the induction proof)
- $n_0 = 2$
- T(2) = 4 and T(3) = 5
- Now, we can complete the induction proof by choosing c s.t. $T(2) \leq c\ 2 \log 2$ and $T(3) \leq c\ 3 \log 3$

The master method

- It is a cook-book based approach for determining asymptotic characterization
- Used for recurrences of the form:

$$T(n) = \begin{cases} c & \text{if } n < d, \\ aT(\lfloor n/b \rfloor) + f(n) & \text{otherwise} \end{cases}$$

where $d \geq 1$ is an integer constant and $a > 0$, $c > 0$, and $b > 1$ are real constants and $f(n)$ is a function that is positive for $n \geq d$

The master method

- Assume that T(n) and f(n) be as defined previously
- The master theorem
  - If there is a small constant ε > 0, s. t. f(n) is O($n^{log_b a \,-\varepsilon}$), then T(n) is θ($n^{log_b a}$)
  - If there is a small constant k ≥ 0, s. t. f(n) is θ($n^{log_b a}\ log^k n$), then T(n) is θ($n^{log_b a}\ \ log^{k+1} n$)
  - If there are small constants ε > 0 and δ < 1, s. t. f(n) is Ω($n^{log_b a+\varepsilon}$) and af(n/b) ≤ δf(n), for n ≥ d, then T(n) is θ($f(n)$)