3. Recurrences

- A recurrence is an equation defining a function $f(n)$ recursively in terms of smaller values of $n$.

- E.g., the running time of Merge-Sort, if $n$ is a power of 2, is:
  $$T(n) = \Theta(1) \quad \text{if } n = 1$$
  $$T(n) = 2 \, T(n/2) + \Theta(n) \quad \text{if } n > 1$$

  For arbitrary $n > 0$, the running time is
  $$T(n) = \Theta(1) \quad \text{if } n = 1$$
  $$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{if } n > 1$$

- We use 3 methods for solving recurrences
  - Substitution Method
  - Iteration Method
  - Master Method

Floors and Ceilings

- For any real number $x$,
  $$\lfloor x \rfloor = \text{greatest integer less than or equal to } x$$
  $$\lceil x \rceil = \text{least integer greater than or equal to } x$$

- For any integer $n$,
  $$\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$$

- For integers $a \neq 0$ and $b \neq 0$,
  $$\lfloor n/a \rfloor / b = \lceil n/(a \cdot b) \rceil$$
  $$\lfloor n/a \rfloor / b = \lceil n/(a \cdot b) \rceil$$
Logarithms

- Definition: For any a, b, c:
  \[ \log_b a = c \iff b^c = a \]
- We use:
  \[ \lg n = \log_2 a \] (binary logarithm)
  \[ \ln n = \log_e a \] (natural logarithm)
- Properties (writing log for a logarithm with arbitrary base):
  \[ a = b^{\log_b a} \]
  \[ \log (a \cdot b) = \log a + \log b \]
  \[ \log a^n = n \log a \]
  \[ \log_b a = \frac{\log_c a}{\log_c b} \]
  \[ \log (1/a) = -\log a \]
  \[ a^{\log_b n} = n^{\log_b a} \]
- (*) implies that e.g. \( \Theta (\lg n) = \Theta (\log_c n) \) for any c.
  The base of the logarithm is irrelevant for asymptotic analysis!

Forward Substitution Method ...

- Guess a solution.
- Verify by induction.
- For example, for
  \[ T(n) = 2 \cdot T(\lfloor n / 2 \rfloor) + n \] and \( T(1) = 1 \)
  we guess \( T(n) = O(n \lg n) \)
- Induction Goal:
  \[ T(n) \leq c \cdot n \lg n, \text{ for some } c \text{ and all } n > n_0 \]
- Induction Hypothesis:
  \[ T(\lfloor n / 2 \rfloor) \leq c \cdot \lfloor n / 2 \rfloor \lg \lfloor n / 2 \rfloor \]
- Proof of Induction Goal:
  \[ T(n) = 2 \cdot T(\lfloor n / 2 \rfloor) + n \]
  \[ \leq 2 \cdot (c \cdot \lfloor n / 2 \rfloor \lg \lfloor n / 2 \rfloor) + n \]
  \[ \leq c \cdot n \lg (n / 2) + n \]
  \[ = c \cdot n \lg n - c \cdot n \lg 2 + n \]
  \[ = c \cdot n \lg n - c \cdot n + n \]
  \[ \leq c \cdot n \lg n \quad \text{provided } c \geq 1 \]
... Forward Substitution Method

- So far the restrictions on $c, n_0$ are only $c \geq 1$
- Base Case:
  
  $$T(n_0) \leq c \cdot n \cdot \lg n$$

  Here, $n_0 = 1$ does not work, since $T(1) = 1$ but $c \cdot 1 \cdot \lg 1 = 0$.

  However, taking $n_0 = 2$ we have:
  
  $$T(2) = 4 \cdot 2 \cdot \lg 2 = 4$$

  so
  
  $$T(2) \leq c \cdot 2$$

  holds provided $c \geq 2$.

---

Summations...

- Linearity:
  
  $$(\sum k \mid 1 \leq k \leq n \cdot c a_k + b_k)$$

  $$= c \cdot (\sum k \mid 1 \leq k \leq n \cdot a_k) + (\sum k \mid 1 \leq k \leq n \cdot b_k)$$

  Use for asymptotic notation:
  
  $$(\sum k \mid 1 \leq k \leq n \cdot \Theta(f(k))) = \Theta(\sum k \mid 1 \leq k \leq n \cdot f(k))$$

  In this equation, the $\Theta$-notation on the left hand side applies to variable $k$, but on the right-hand side, it applies to $n$.

- Arithmetic Series:
  
  $$(\sum k \mid 1 \leq k \leq n \cdot k) = \frac{n \cdot (n + 1)}{2}$$

  $$= \Theta(n^2)$$

- Geometric (or Exponential) Series: If $x \neq 1$ then
  
  $$(\sum k \mid 0 \leq k \leq n \cdot x^k) = \frac{x^{n+1} - 1}{x - 1}$$

  $= \Theta(n^2)$$
... Summations

- Infinite Decreasing Geometric Series: If $|x| < 1$ then
  \[\sum_{k = 0}^{\infty} x^k = \frac{1}{1 - x}\]

- Harmonic Series:
  \[H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\]
  \[= \sum_{k = 1}^{n} \frac{1}{k}\]
  \[= \ln n + O(1)\]

- Further series obtained by integrating or differentiating the formulas above.
  For example, by differentiating the infinite decreasing geometric series and multiplying with $x$ we get:
  \[\sum_{k = 0}^{\infty} k x^k = \frac{x}{(1 - x)^2}\]

Iteration (Backward Substitution) Method ...

- Express the recurrence as a summation of terms.
- Use techniques for summations.

- For example, we iterate
  \[T(n) = 3 T\left\lfloor \frac{n}{4} \right\rfloor + n\]
  as follows:
  \[T(n) = n + 3 T\left\lfloor \frac{n}{4} \right\rfloor + 9 T\left\lfloor \frac{n}{16} \right\rfloor + 27 T\left\lfloor \frac{n}{64} \right\rfloor\]
  The $i$-th term in the series is $3^i \left\lfloor \frac{n}{4^i} \right\rfloor$.
  We have to iterate until $\left\lfloor \frac{n}{4^i} \right\rfloor = 1$, since $T(1) = \Theta(1)$,
  or equivalently until $i > \log_4 n$. 
... Iteration (Backward Substitution) Method

- We continue:
  \[
  T(n) = n + 3 \lfloor n / 4 \rfloor + 9 \lfloor n / 16 \rfloor + 27 \lfloor n / 64 \rfloor + \ldots + 3^{\log_4 n} \Theta(1)
  \]
  \(a \log_b n = n^{\log_b a}\)
  \[
  \leq n + 3 \frac{n}{4} + 9 \frac{n}{16} + 27 \frac{n}{64} + \ldots + 3^{\log_4 n} \Theta(1)
  \]
  \[
  \leq n \left( \sum_{i=0}^{\infty} \frac{3}{4}^i \right) + \Theta(n^{\log_4 3})
  \]
  \[
  \leq 4 n + \Theta(n^{\log_4 3})
  \]
  \[
  \leq 4 n + o(n)
  \]
  \[
  = O(n)
  \]

The Master Theorem

- Let \(a \geq 1\) and \(b > 1\) be constants and \(f(n)\) be a function. Assume
  \[
  T(n) = a T(n/b) + f(n)
  \]
  where \(n/b\) stands for \(\lfloor n/b \rfloor\) or \(\lceil n/b \rceil\). Then
  - \(T(n) = \Theta(n^{\log_b a})\) if \(f(n) = O(n^{\log_b a - \epsilon})\) for some \(\epsilon > 0\),
  - \(T(n) = \Theta(n^{\log_b a} \log n)\) if \(f(n) = \Theta(n^{\log_b a})\)
  - \(T(n) = \Theta(f(n))\) if \(f(n) = \Omega(n^{\log_b a + \epsilon})\) for some \(\epsilon > 0\) and if
    \[
    a f(n/b) \leq c f(n)
    \]
    for some \(c < 1\) and sufficiently large \(n\).

- Note 1: This theorem can be applied to divide-and-conquer algorithms, which are all of the form
  \[
  T(n) = a T(n/b) + D(n) + C(n)
  \]
  where \(D(n)\) is the cost of dividing and \(C(n)\) the cost of combining.

- Note 2: Not all possible cases are covered by the theorem.
**Merge Sort with the Master Theorem**

- For arbitrary \( n > 0 \), the running time of Merge-Sort is
  
  \[
  T(n) = \begin{cases} 
  \Theta(1) & \text{if } n = 1 \\
  T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + \Theta(n) & \text{if } n > 1 
  \end{cases}
  \]

  We can approximate this from below and above by
  
  \[
  T(n) = 2 T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + \Theta(n) \quad \text{if } n > 1
  \]

  respectively. According to the Master Theorem, both have the same solution which we get by taking

  \[ a = 2, \ b = 2, \ f(n) = \Theta(n) \].

  Since \( n = n^{\log_2 2} \), the second case applies and we get:

  \[ T(n) = \Theta(n \log n) \]

**Binary Search with the Master Theorem**

- The Master Theorem allows us to ignore the floor or ceiling function around \( n/b \) in \( T(n/b) \) in general.

- Binary Search has for any \( n > 0 \) a running time of
  
  \[ T(n) = T(n/2) + \Theta(1) \].

  Hence \( a = 1, b = 2, f(n) = \Theta(1) \). Since \( 1 = n^{\log_2 1} \) the second case applies and we get:

  \[ T(n) = \Theta(\log n) \]
Towers of Hanoi with the Master Theorem (a bit odd application)

- The Towers of Hanoi algorithm has for any \( n > 0 \) a running time of
  \[ T(n) = 2 \ T(n-1) + 1. \]
  In order to bring this into a form such that the Master Theorem is applicable, we rename \( n = \lg m \):
  \[
  \begin{align*}
  T(\lg m) &= 2 \ T(\lg m - 1) + 1 \\
           &= 2 \ T(\lg m - \lg 2) + 1 \\
           &= 2 \ T(\lg (m/2)) + 1
  \end{align*}
  \]
  Defining \( S(m) = T(\lg m) \) we get the new recurrence:
  \[
  S(m) = 2 \ S(m/2) + 1
  \]
  Hence \( a = 2, \ b = 2, \ f(m) = 1 \). Since \( 1 = m^{\log_2 2-1} \) the first case applies with \( \varepsilon = 1 \) and we get:
  \[
  S(m) = \Theta(m)
  \]
  With \( S(m) = T(\lg m) \) and \( n = \lg m \) we finally get:
  \[
  T(n) = \Theta(2^n)
  \]