Principles of Algorithm Analysis

- Key to good understanding of algorithms for practical applications
  - We do not analyze every program we write
  - Enough to understand basic [standard] algorithms and their performance so that we can select the best algorithm for the job at hand
- Important for the study of algorithm properties so that we can save time and resources, with reasonable sacrifice in terms of complexity of coding
- Consider the following three codes

```
sum ← 0                        sum ← 0                  sum ← n^2
for i ← 1 to n                 for i ← 1 to n
    for j ← 1 to n                 sum ← sum + n
        sum ← sum + 1
```

  - What can you say about their performance? Do they achieve the same goal?

Implementation and Empirical Analysis

- Design, develop, and express algorithms in terms of layers of abstract operations
- Empirical analysis
  - Compare the performance of two algorithms by actually running them
  - Requires a correct and complete implementation
  - Look for resource usage and time required, with the same input data and running on the same machine, with the same type of environment
    - Selection of input data is extremely important
    - You can select random data, actual data, or perverse data
  - Code may execute at different speed depending on load on the system (overall resource usage)
  - Useful to validate the mathematical analysis
- Pitfalls in algorithm selection
  - Ignoring performance characteristics
    - Addition of a few lines of code (increase in complexity) can endow the code with more intelligence to make it run faster
  - Paying too much attention to performance characteristics
    - Is it worth spending 10 hours of your time to save 10 milliseconds of run time?

Analysis of algorithms

- It may not be always possible to perform empirical analysis
- Mathematical analysis is more informative and less expensive but can be difficult if we do not know all the mathematical formulas
- The high-level program code may not correctly reflect the performance in terms of machine language

  - The code may compile differently depending on the level of optimization turned on in the compiler

- Identify the abstract operations on which the algorithm is based, and separate analysis from implementation (think of the abstract operations outlined in selection sort analysis)
- Identify the data for best case comparison, average case comparison, and worst case comparison
  - It is possible that the best case data for an algorithm turns out to be the worst case data for a different algorithm

Growth of Functions

- Simple characterization of algorithm efficiency
- Allows to compare relative performance of alternative algorithms
- Depends on input data size N
  - If there are multiple input parameters, we will try to reduce them to a single parameter, expressing some parameters in terms of the selected parameter
- The performance of algorithm on an input of size N is generally represented in terms of 1, $\lg N$, N, $N \lg N$, N^2, N^3, and 2^N
  - The performance depends heavily on loops, and can be increased by minimizing the inner loops (or work done in inner loops)
- Asymptotic efficiency of algorithms
  - Effect of input size increase without bound on running time of algorithm

Standard Notation and Common Functions

- Monotonicity
  - Monotonically increasing – $m \leq n \Rightarrow f(m) \leq f(n)$
  - Monotonically decreasing – $m \leq n \Rightarrow f(m) \geq f(n)$
  - Strictly increasing – $m < n \Rightarrow f(m) < f(n)$
  - Strictly decreasing – $m < n \Rightarrow f(m) > f(n)$
- Floors and ceilings
  - floor(x) – greatest integer $\leq$ x
  - ceiling(x) – smallest integer $\geq$ x
  - $\forall$ real x

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$$

  - For any integer n

$$\lceil \frac{n}{2} \rceil + \lfloor \frac{n}{2} \rfloor = n$$

  - For any integer n, and integers $a \neq 0$ and $b \neq 0$

$$\lceil \lceil n/a \rceil / b \rceil = \lceil n/ab \rceil$$

$$\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor$$

- Floor and ceiling functions are monotonically increasing

- Polynomials
  - Polynomial in n of degree d

    $$p(n) = \sum_{i=0}^{d} a_i n^i$$

    $a_0, a_1, \ldots, a_d$ are *coefficients* of polynomial, and $a_d \neq 0$
  - Polynomial is *asymptotically positive* iff $a_d > 0$
  - For an asymptotically positive polynomial $p(n)$ of degree d, $p(n) = \Theta(n^d)$

- Exponentials
  - $\forall$ real $a \neq 0$, m and n, we have following identities
    * $a^0 = 1$
    * $a^1 = a$
    * $a^{-1} = \frac{1}{a}$
    * $(a^m)^n = a^{mn}$
    * $(a^m)^n = (a^n)^m$
    * $a^m a^n = a^{m+n}$
  - $\forall$ n and $a \geq 1$, a^n is monotonically increasing in n
  - Assume $0^0 = 1$
  - $\forall$ real constants a and b such that $a > 1$

    $$\lim_{n\to\infty} \frac{n^b}{a^n} = 0$$

    $$n^b = o(a^n)$$

    Any positive exponential function grows faster than any polynomial
  - Base of natural logarithm function $e = 2.71828\ldots$
  - $\forall$ real x

    $$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

  - $\forall$ real x, $e^x \geq 1 + x$
  - When $|x| \leq 1$, we have $1 + x \leq e^x \leq 1 + x + x^2$
  - When $x \to 0$, e^x can be approximated by

    $$e^x = 1 + x + \Theta(x^2)$$

- Logarithms
  - Notation

    $$\begin{array}{rcll} \lg n & = & \log_2 n & \text{(binary logarithm)} \\ \ln n & = & \log_e n & \text{(naturl logarithm)} \\ \lg^k n & = & (\lg n)^k & \text{(exponentiation)} \\ \lg\lg n & = & \lg(\lg n) & \text{(composition)} \end{array}$$

- For all real $a > 0, b > 0, c > 0$, and n

$$\begin{aligned} a &= b^{\log_b a} \\ \log_c(ab) &= \log_c a + \log_c b \\ \log_b a^n &= n \log_b a \\ \log_b a &= \frac{\log_c a}{\log_c b} \\ \log_b \frac{1}{a} &= -\log_b a \\ \log_b a &= \frac{1}{\log_a b} \\ a^{\log_b n} &= n^{\log_b a} \end{aligned}$$

- When $|x| < 1$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots$$

- For $x > -1$

$$\frac{x}{1+x} \leq \ln(1+x) \leq x$$

- A function $f(n)$ is *polylogarithmically bounded* if $f(n) = \lg^{O(1)} n$
- $\lim_{n\to\infty} \frac{\lg^b n}{2^{a \lg n}} = \lim_{n\to\infty} \frac{\lg^b n}{n^a} = 0$
  $\lg^b n = o(n^a)$
  Any positive polynomial function grows faster than any polylogarithmic function

- Factorials
  - $n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$
- Fibonacci numbers
  - Definition

$$\begin{aligned} &F_0 = 0 \\ &F_1 = 1 \\ &F_i = F_{i-1} + F_{i-2}, \ \ i \geq 2 \end{aligned}$$

  - Golden ratio Φ and conjugate $\hat{\Phi}$
    * $\Phi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots$
    * $\hat{\Phi} = \frac{1-\sqrt{5}}{2} = -.61803\ldots$
  - $F_i = \frac{\Phi^i - \hat{\Phi}^i}{\sqrt{5}}$

Asymptotic Notation (including Big-Oh)

- Function with domain as the set of natural numbers
- Allows the suppression of detail when analyzing algorithms
- Allows the description to be accurate while losing little detail
- Convenient to describe the worst case running time function $T(n)$
- Θ-notation
  - Consider a given function $g(n)$
  - $\Theta(g(n))$ – Set of functions
  - $\Theta(g(n)) = \{f(n) : \exists$ positive constants $c_1, c_2,$ and $n_0 \mid 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_0\}$.

- $f(n)$ can be *sandwiched* between $c_1 g(n)$ and $c_2 g(n)$, for sufficiently large n
- $\Theta(g(n))$ is a set
- We write $f(n) = \Theta(g(n))$ to imply $f(n) \in \Theta(g(n))$
- For all values of $n \geq n_0$, $f(n)$ lies at or above $c_1 g(n)$ and at or below $c_2 g(n)$
- $\forall n \geq n_0$, $f(n)$ is equal to $g(n)$ within a constant factor
- $g(n)$ is an asymptotically tight bound for $f(n)$
- Every member of $\Theta(g(n))$ must be *asymptotically nonnegative*
- $f(n)$ must be nonnegative whenever n is sufficiently large
- Consequently, $g(n)$ itself must be asymptotically nonnegative, or else, the set $\Theta(g(n))$ is empty
- Therefore, it is reasonable to assume that every function used with Θ-notation is asymptotically nonnegative
- Prove $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
  * Determine positive constants c_1, c_2, and n_0 such that
  $$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \forall n \geq n_0$$
  * Dividing by n^2 we have
  $$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$
  * $c_1 \leq \frac{1}{14}$ for $n \geq 7$
  * $c_2 \geq \frac{1}{14}$ for $n \geq 7$, but preferably, $c_2 \geq \frac{1}{2}$ for arbitrarily large n
- Prove $6n^3 \neq \Theta(n^2)$
  Assume that c_2 and n_0 exist such that $6n^3 \leq c_2 n^2 \; \forall n \geq n_0$
  $n \leq \frac{c_2}{6}$, not possible for arbitrarily large n because c_2 is a constant
- Since any constant is a degree-0 polynomial, constant function can be expressed as $\Theta(n^0)$ or $\Theta(1)$

• O-notation

- Asymptotic upper bound
- Upper bound on a function within a constant factor
- Not as strong as Θ-notation
- $O(g(n)) = \{f(n) : \exists$ positive constants c and $n_0 \mid 0 \leq f(n) \leq cg(n) \; \forall n \geq n_0\}$
- $f(n) = \Theta(g(n)) \;\Rightarrow\; f(n) = O(g(n))$
- $\Theta(g(n)) \;\supseteq\; O(g(n))$
- O-notation used to describe the running time of algorithm by inspection of algorithm structure
  * Doubly nested loop structure $\Rightarrow O(n^2)$
  * Biggest concern is the terms with the larger exponent, or the leading terms in a polynomial
- Three purposes of O-notation:
  1. Bound the error when small terms in mathematical formulas are ignored
  2. Bound the error when we ignore parts of a program that contribute a small amount to the total being analyzed
     * Such items will include initialization code and/or heuristics which may have a small but significant effect on the actual run-time
  3. Classify algorithms according to upper bounds on their total running times
- Above reasoning allows us to focus on the leading term when comparing running times for algorithms (with the assumption that precise analysis can be performed, if necessary)
- $f(n) \in O(g(n)) \equiv f(n) = O(g(n))$

        * When $f(n)$ is asymptotically large compared to another function $g(n)$, i.e., $\lim_{N\to\infty} \frac{g(n)}{f(n)} = 0$, $f(n)$ is taken to mean $f(n) + O(g(n))$
        * We sacrifice mathematical precision in favor of clarity, with a guarantee that for large N, the effect of quantity given by $O(g(n))$ actually is negligible
            · As an example, we take the summation of the series $\sum_{i=1}^{N} i$ to be $\frac{N^2}{2}$ rather than $\frac{N(N+1)}{2}$
        * Such notation allows us to be both *precise* and *concise* when describing the performance of algorithms

- Ω-notation
    - Asymptotic lower bound
    - Best-case running time
    - $\Omega(g(n)) = \{f(n) \,:\, \exists$ positive constants c and $n_0 \mid 0 \leq cg(n) \leq f(n)\; \forall n > n_0\}$
    - Best case running time of insertion sort $\Omega(n)$
- **Theorem 1** *For any two functions $f(n)$ and $g(n)$, $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$*
    - Useful to prove asymptotically tight bounds from upper and lower bounds
    - Running time of insertion sort falls between $O(n^2)$ and $\Omega(n)$
- o-notation
    - Asymptotic upper bound provided by O-notation may or may not be asymptotically tight
    - o-notation denotes an upper bound that is not asymptotically tight
    - $o(g(n)) = \{f(n) \,:\,$ For any constant $c > 0$, $\exists$ a constant $n_0 > 0 \mid 0 \leq f(n) < cg(n)\; \forall n \geq n_0\}$
    - For example, $2n = o(n^2)$, but $2n^2 \neq o(n^2)$
    - $f(n)$ becomes insignificant compared to $g(n)$ as n approaches infinity, or

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

- ω-notation
    - ω-notation denotes the asymptotic lower bound that is not tight
    - $\omega(g(n)) = \{f(n) \,:\,$ For any constant $c > 0$, $\exists$ a constant $n_0 > 0 \mid 0 \leq cg(n) < f(n)\; \forall n \geq n_0\}$
    - For example, $\frac{n^2}{2} = \omega(n)$, but $\frac{n^2}{2} \neq \omega(n^2)$
    - $f(n) = \omega(g(n))$ implies

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

    - $f(n)$ becomes arbitrarily large relative to $g(n)$ as n approaches infinity.
- Comparison of functions
    - $f(n)$ and $g(n)$ are asymptotically positive
    - Transitivity

$$\begin{array}{lcl}
f(n) = \Theta(g(n)) \quad \wedge \quad g(n) = \Theta(h(n)) & \Rightarrow & f(n) = \Theta(h(n)) \\
f(n) = O(g(n)) \quad \wedge \quad g(n) = O(h(n)) & \Rightarrow & f(n) = O(h(n)) \\
f(n) = \Omega(g(n)) \quad \wedge \quad g(n) = \Omega(h(n)) & \Rightarrow & f(n) = \Omega(h(n)) \\
f(n) = o(g(n)) \quad \wedge \quad g(n) = o(h(n)) & \Rightarrow & f(n) = o(h(n)) \\
f(n) = \omega(g(n)) \quad \wedge \quad g(n) = \omega(h(n)) & \Rightarrow & f(n) = \omega(h(n))
\end{array}$$

    - Reflexivity

$$\begin{aligned} f(n) &= \Theta(f(n)) \\ f(n) &= O(f(n)) \\ f(n) &= \Omega(f(n)) \end{aligned}$$

- Symmetry

$$f(n) = \Theta(g(n)) \quad \text{if and only if} \quad g(n) = \Theta(f(n))$$

- Transpose symmetry

$$\begin{aligned} f(n) = O(g(n)) &\quad \text{if and only if} \quad g(n) = \Omega(f(n)) \\ f(n) = o(g(n)) &\quad \text{if and only if} \quad g(n) = \omega(f(n)) \end{aligned}$$

- Analogy with two real numbers a and b

$$\begin{aligned} f(n) = O(g(n)) &\approx a \leq b \\ f(n) = \Omega(g(n)) &\approx a \geq b \\ f(n) = \Theta(g(n)) &\approx a = b \\ f(n) = o(g(n)) &\approx a < b \\ f(n) = \omega(g(n)) &\approx a > b \end{aligned}$$

Summations – Formulas and Properties

- Infinite series

$$\sum_{i=1}^{\infty} a_i = a_1 + a_2 + \cdots = \lim_{n\to\infty} \sum_{i=1}^{n} a_i$$

- Divergent series – no limit
- Convergent series – some limit
- Linearity
  - For any real number c and any finite sequences $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$

$$\sum_{i=1}^{n}(ca_i + b_i) = c\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$$

  - Usage in growth estimation

$$\sum_{i=1}^{n} \Theta(f(i)) = \Theta\left(\sum_{i=1}^{n} f(i)\right)$$

- Arithmetic series

$$\begin{aligned} \sum_{i=1}^{n} i &= 1 + 2 + 3 + \cdots + n \\ &= \frac{1}{2}n(n+1) \\ &= \Theta(n^2) \end{aligned}$$

- Geometric series
  - For real $x \neq 1$

$$\begin{aligned} \sum_{i=0}^{n} x^i &= 1 + x + x^2 + x^3 + \cdots + x^n \\ &= \frac{x^{n+1} - 1}{x - 1} \end{aligned}$$

- For $|x| < 1$

$$\sum_{i=0}^{n} x^i = \frac{1}{1-x}$$

- Harmonic series
  - For $n > 0$, the nth harmonic number is

$$\begin{aligned} H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \\ &= \sum_{i=1}^{n} \frac{1}{i} \\ &= \ln n + O(1) \end{aligned}$$

- Telescoping series
  - For any sequence $a_0, a_1, \ldots, a_n$

$$\sum_{i=1}^{n} (a_i - a_{i-1}) = a_n - a_0$$

$$\sum_{i=0}^{n-1} (a_i - a_{i+1}) = a_0 - a_n$$

  - Example

$$\begin{aligned} \sum_{i=1}^{n-1} \frac{1}{i(i+1)} &= \sum_{i=0}^{n-1} \left(\frac{1}{i} - \frac{1}{i+1} \right) \\ &= 1 - \frac{1}{n} \end{aligned}$$

- Products
  - Finite product

$$\prod_{i=1}^{n} a_i$$

  - Convert a formula with a product to one with summation

$$\lg \left(\prod_{i=1}^{n} a_i \right) = \sum_{i=1}^{n} \lg a_i$$

Bounding Summations

- Mathematical induction
  - Prove that

$$\sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$$

    Base case: For $n = 1$, trivially proven
    Inductive assumption: True for all values of n such that $1 \leq n \leq k$.

Induction:

$$\begin{aligned}\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + (k+1) \\ &= \frac{1}{2}k(k+1) + (k+1) \\ &= \frac{1}{2}(k+1)(k+2)\end{aligned}$$

- Use of induction to show a bound.
  Prove that $\sum_{i=0}^{n} 3^i$ is $O(3^n)$;
  Or, for any constant c

$$\sum_{i=0}^{n} 3^i \leq c \cdot 3^n$$

  Base case: $n = 0$

$$\sum_{i=0}^{0} 3^i = 1 \quad \leq \quad c, \text{ for} c \geq 1$$

  Inductive assumption: True for all values of n such that $1 \leq n \leq k$.
  Induction:

$$\begin{aligned}\sum_{i=0}^{k+1} 3^i &= \sum_{i=0}^{k} 3^i + 3^{k+1} \\ &\leq c3^k + 3^{k+1} \\ &= \left(\frac{1}{3} + \frac{1}{c}\right) c3^{k+1} \\ &\leq c3^{k+1} \qquad \forall c \leq \frac{3}{2}\end{aligned}$$

- Use of asymptotic notation to prove a bound
  Fallacious proof for

$$\sum_{i=1}^{n} i = O(n)$$

  Base case: $n = 1$. Trivial proof
  Inductive assumption: True for all values of n such that $1 \leq n \leq k$.
  Induction:

$$\begin{aligned}\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + (k+1) \\ &= O(k) + (k+1) \quad \Leftarrow \text{ error} \\ &= O(k+1)\end{aligned}$$

- Bounding the terms
  - Upper bound on arithmetic series

$$\begin{aligned}\sum_{i=1}^{n} i &\leq \sum_{i=1}^{n} n \\ &= n^2\end{aligned}$$

- For a series $\sum_{i=1}^{n} a_i$, let $a_{\max} = \max_{1 \le i \le n} a_i$. Then,

$$\sum_{i=1}^{n} a_i \le n a_{\max}$$

- Geometric series
  * For a series, $\sum_{i=0}^{n} a_i$, let $\frac{a_{i+1}}{a_i} \le r$ for all $i \ge 0$, where $r < 1$
    Sum can be bounded by an infinite decreasing geometric series, since $a_i \le a_0 r^i$

$$\begin{aligned} \sum_{i=0}^{n} a_i &\le \sum_{i=0}^{\infty} a_0 r^i \\ &= a_0 \sum_{i=0}^{\infty} r^i \\ &= a_0 \frac{1}{1-r} \end{aligned}$$

  * Bound the summation

$$\sum_{i=1}^{\infty} \frac{i}{3^i}$$

    First term $= \frac{1}{3}$
    Ratio of consecutive terms

$$\begin{aligned} \frac{(i+1)/3^{i+1}}{i/3^i} &= \frac{1}{3} \cdot \frac{i+1}{i} \\ &\le \frac{2}{3} \quad \forall i \ge 1 \end{aligned}$$

    Each term is bounded above by $\left(\frac{1}{3}\right)\left(\frac{2}{3}\right)^i$

$$\begin{aligned} \sum_{i=1}^{\infty} \frac{i}{3^i} &\le \sum_{i=1}^{\infty} \frac{1}{3}\left(\frac{2}{3}\right)^i \\ &= \frac{1}{3} \cdot \frac{1}{1-\frac{2}{3}} \\ &= 1 \end{aligned}$$

  * A common pitfall

$$\begin{aligned} \sum_{i=1}^{\infty} \frac{1}{i} &= \lim_{n \to \infty} \sum_{i=1}^{n} \frac{1}{i} \\ &= \lim_{n \to \infty} \Theta(\lg n) \\ &= \infty \end{aligned}$$

- Splitting summations
  - Express the series as the sum of two or more summations
  - Lower bound of the series $\sum_{i=1}^{n} i$
  - Assume that n is even

$$\sum_{i=1}^{n} i = \sum_{i=1}^{n/2} i + \sum_{i=n/2+1}^{n} i$$

$$
\begin{aligned}
&\geq \sum_{i=1}^{n/2} 0 + \sum_{i=n/2+1}^{n} \frac{n}{2} \\
&\geq \left(\frac{n}{2}\right)^2 \\
&= \Omega(n^2)
\end{aligned}
$$

– If each term a_i in a summation $\sum_{i=0}^{n} a_i$ is independent of n, then, for any constant $i_0 > 0$

$$
\begin{aligned}
\sum_{i=0}^{n} a_i &= \sum_{i=0}^{i_0-1} a_i + \sum_{i=i_0}^{n} a_i \\
&= \Theta(1) + \sum_{i=i_0}^{n} a_i
\end{aligned}
$$

– Find an asymptotic upper bound on

$$\sum_{i=0}^{\infty} \frac{i^2}{2^i}$$

Observe that the ratio of consecutive terms, for $i \geq 3$, is

$$
\begin{aligned}
\frac{(i+1)^2/2^{i+1}}{i^2/2^i} &= \frac{(i+1)^2}{2i^2} \\
&\leq \frac{8}{9}
\end{aligned}
$$

The summation can be split into

$$
\begin{aligned}
\sum_{i=0}^{\infty} \frac{i^2}{2^i} &= \sum_{i=0}^{2} \frac{i^2}{2^i} + \sum_{i=3}^{\infty} \frac{i^2}{2^i} \\
&\leq O(1) + \frac{9}{8} \sum_{i=0}^{\infty} \left(\frac{8}{9}\right)^i \\
&= O(1)
\end{aligned}
$$

since the second summation is a decreasing geometric series.

– Find the asymptotic bound on the harmonic series

$$H_n = \sum_{i=1}^{n} \frac{1}{i}$$

Split the range 1 to n into $\lfloor \lg n \rfloor$ pieces and upper bound the contribution of each piece by 1.

$$
\begin{aligned}
\sum_{i=1}^{n} \frac{1}{i} &\leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i + j} \\
&\leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i} \\
&\leq \sum_{i=0}^{\lfloor \lg n \rfloor} 1 \\
&\leq \lg n + 1
\end{aligned}
$$

Recurrences

- Recursively decompose a large problem into a set of smaller problems
  - Decomposition is directly reflected in analysis
  - Run-time determined by the size and number of subproblems to be solved in addition to the time required for decomposition
- An equation or inequality that describes a function in terms of its value on smaller inputs
  - Also known as *recurrence relation*
  - Recurrence can be solved to derive the running time
- Example, mergesort recurrence

$$T_n = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T_{\frac{n}{2}} + \Theta(n) & \text{if } n > 1 \end{cases}$$

  Solution for the mergesort recurrence: $\Theta(n \lg n)$
- You can ignore extreme details like floor, ceiling, and boundary in recurrence description.

Substitution Method

- Guess the form of solution and use induction to find constants
- Determine upper bound on the recurrence

$$T_n = 2T_{\lfloor \frac{n}{2} \rfloor} + n$$

  Guess the solution as: $T_n = O(n \lg n)$
  Now, prove that $T_n \leq cn \lg n$ for some $c > 0$
  Assume that the bound holds for $\left\lfloor \frac{n}{2} \right\rfloor$
  Substituting into the recurrence

$$\begin{aligned} T_n &\leq 2(c \left\lfloor \frac{n}{2} \right\rfloor \lg(\left\lfloor \frac{n}{2} \right\rfloor)) + n \\ &\leq cn \lg \left(\frac{n}{2}\right) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n \quad \forall c \geq 1 \end{aligned}$$

  Boundary condition: Let the only bound be $T_1 = 1$

$$\nexists c \mid T_1 \leq c1 \lg 1 = 0$$

  Problem overcome by the fact that asymptotic notation requires us to prove

$$T_n \leq cn \lg n \text{ for } n \geq n_0$$

  Include T_2 and T_3 as boundary conditions for the proof

$$T_2 = 4 \quad T_3 = 5$$

  Choose c such that $T_2 \leq c2 \lg 2$ and $T_3 \leq c3 \lg 3$
  True for any $c \geq 2$
- Making a good guess

- If a recurrence is similar to a known recurrence, it is reasonable to guess a similar solution

$$T_n = 2T_{\lfloor \frac{n}{2} \rfloor} + n$$

  If n is large, difference between $T_{\lfloor \frac{n}{2} \rfloor}$ and $T_{\lfloor \frac{n}{2} \rfloor + 17}$ is relatively small

- Prove upper and lower bounds on a recurrence and reduce the range of uncertainty.
  Start with a lower bound of $T_n = \Omega(n)$ and an initial upper bound of $T_n = O(n^2)$. Gradually lower the upper bound and raise the lower bound to get asymptotically tight solution of $T_n = \Theta(n \lg n)$

• Pitfall

- $T_n = 2T_{\lfloor \frac{n}{2} \rfloor} + n$
  Assume inductively that $T_n \leq cn$ implying that $T_n = O(n)$

$$\begin{aligned} T_n &\leq 2c\left\lfloor \frac{n}{2} \right\rfloor + n \\ &\leq cn + n \\ &= O(n) \quad \Leftarrow \text{ wrong} \end{aligned}$$

  We haven't proved the exact form of inductive hypothesis $T_n \leq cn$

• Changing variables

- Consider the recurrence

$$T_n = 2T_{\lfloor \sqrt{n} \rfloor} + \lg n$$

  Let $m = \lg n$.

$$T_{2^m} = 2T_{2^{\frac{m}{2}}} + m$$

  Rename $S_m = T_{2^m}$

$$S_m = 2S_{\frac{m}{2}} + m$$

  Solution for the recurrence: $S_m = m \lg m$
  Change back from S_m to T_n

$$T_n = T_{2^m} = S_m = O(m \lg m) = O(\lg n \lg \lg n)$$

The iteration method

• Also known as *telescoping method*

• No guessing but more algebra, by applying the recurrence to itself (on the right hand side of the equation)

• Expand the recurrence and express it as summation dependent on only n and initial conditions

• Recurrence

$$T_n = 3T_{\lfloor \frac{n}{4} \rfloor} + n$$

$$\begin{aligned} T_n &= n + 3T_{\lfloor \frac{n}{4} \rfloor} \\ &= n + 3(\left\lfloor \frac{n}{4} \right\rfloor + 3T_{\lfloor \frac{n}{16} \rfloor}) \\ &= n + 3(\lfloor \frac{n}{4} \rfloor + 3(\lfloor \frac{n}{16} \rfloor + 3T_{\lfloor \frac{n}{64} \rfloor})) \\ &= n + 3\lfloor \frac{n}{4} \rfloor + 9\lfloor \frac{n}{16} \rfloor + 27T_{\lfloor \frac{n}{64} \rfloor} \end{aligned}$$

  ith term is given by $3^i \lfloor \frac{n}{4^i} \rfloor$
  Bound $n = 1$ when $\lfloor \frac{n}{4^i} \rfloor = 1$ or $i > \log_4 n$

Bound $\lfloor \frac{n}{4^i} \rfloor \leq \frac{n}{4^i}$
Decreasing geometric series

$$\begin{aligned} T_n &\leq n + \frac{3}{4}n + \frac{9}{16}n + \frac{27}{64}n + \cdots + 3^{\log_4 n}\Theta(1) \\ &\leq n\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i + \Theta(n^{log_4 3}) & 3^{\log_4 n} = n^{log_4 3} \\ &= 4n + o(n) & \log_4 3 < 1 \Rightarrow \Theta(n^{log_4 3}) = o(n) \\ &= O(n) \end{aligned}$$

Focus on

- Number of iterations to reach boundary condition
- Sum of terms arising from each level of iteration

- Recursion trees
  - Recurrence
    $$T_n = 2T_{\frac{n}{2}} + n^2$$
    Assume n to be an exact power of 2.
    $$\begin{aligned} T_n &= n^2 + 2T_{\frac{n}{2}} \\ &= n^2 + 2\left(\left(\frac{n}{2}\right)^2 + 2T_{\frac{n}{4}}\right) \\ &= n^2 + \frac{n^2}{2} + 4\left(\left(\frac{n}{4}\right)^2 + 2T_{\frac{n}{8}}\right) \\ &= n^2 + \frac{n^2}{2} + \frac{n^2}{4} + 8\left(\left(\frac{n}{8}\right)^2 + 2T_{\frac{n}{16}}\right) \\ &= n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \frac{n^2}{8} + \cdots \\ &= n^2(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots) \\ &= \Theta(n^2) \end{aligned}$$
    The values above decrease geometrically by a constant factor.
  - Recurrence
    $$T_n = T_{\frac{n}{3}} + T_{\frac{2n}{3}} + n$$
    Longest path from root to a leaf
    $$n \rightarrow \left(\frac{2}{3}\right)n \rightarrow \left(\frac{2}{3}\right)^2 n \rightarrow \cdots 1$$
    $\left(\frac{2}{3}\right)^k n = 1$ when $k = \log_{\frac{3}{2}} n$, k being the height of the tree
    Upper bound to the solution to the recurrence – $n \log_{\frac{3}{2}} n$, or $O(n \log n)$

The Master Method

- Suitable for recurrences of the form
  $$T_n = aT_{\frac{n}{b}} + f(n)$$
  where $a \geq 1$ and $b > 1$ are constants, and
  $f(n)$ is an asymptotically positive function

- For mergesort, $a = 2$, $b = 2$, and $f(n) = \Theta(n)$
- Master Theorem

  Theorem 2 *Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let T_n be defined on the nonnegative integers by the recurrence*

  $$T_n = aT_{\frac{n}{b}} + f(n)$$

  where we interpret $\frac{n}{b}$ to mean either $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$. Then T_n can be bounded asymptotically as follows

  1. *If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T_n = \Theta(n^{\log_b a})$*
  2. *If $f(n) = \Theta(n^{\log_b a})$, then $T_n = \Theta(n^{\log_b a} \lg n)$*
  3. *If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T_n = \Theta(f(n))$*

  - In all three cases, compare $f(n)$ with $n^{\log_b a}$
  - Solution determined by the larger of the two
    * Case 1: $n^{\log_b a} > f(n)$
      Solution $T_n = \Theta(n^{\log_b a})$
    * Case 2: $n^{\log_b a} \approx f(n)$
      Multiply by a logarithmic factor
      Solution $T_n = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n)$
    * Case 3: $f(n) > n^{\log_b a}$
      Solution $T_n = \Theta(f(n))$
  - In case 1, $f(n)$ must be asymptotically smaller than $n^{\log_b a}$ by a factor of n^ϵ for some constant $\epsilon > 0$
  - In case 3, f_n must be polynomially larger than $n^{\log_b a}$ and satisfy the "regularity" condition that $af(\frac{n}{b}) \leq cf(n)$

- Using the master method
  - Recurrence

    $$T_n = 9T_{\frac{n}{3}} + n$$

    $a = 9$, $b = 3$, $f(n) = n$
    $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$
    $f(n) = O(n^{\log_3 9 - \epsilon})$, where $\epsilon = 1$
    Apply case 1 of master theorem and conclude $T_n = \Theta(n^2)$
  - Recurrence

    $$T_n = T_{\frac{2n}{3}} + 1$$

    $a = 1$, $b = \frac{3}{2}$, $f(n) = 1$
    $n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0 = 1$
    $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$
    Apply case 2 of master theorem and conclude $T_n = \Theta(\lg n)$
  - Recurrence

    $$T_n = 3T_{\frac{n}{4}} + n \lg n$$

    $a = 3$, $b = 4$, $f(n) = n \lg n$
    $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
    $f(n) = \Omega(n^{\log_4 3 + \epsilon})$, where $\epsilon \approx 0.2$
    Apply case 3, if regularity condition holds for $f(n)$
    For large n, $af(\frac{n}{b}) = 3\frac{n}{4} \lg(\frac{n}{4}) \leq \frac{3}{4} n \lg n = cf(n)$ for $c = \frac{3}{4}$
    Therefore, $T_n = \Theta(n \lg n)$

- Recurrence

$$T_n = 2T_{\frac{n}{2}} + n \lg n$$

Recurrence has proper form – $a = 2$, $b = 2$, $f(n) = n \lg n$ and $n^{\log_b a} = n$
$f(n) = n \lg n$ is asymptotically larger than $n^{\log_b} = n$ but not *polynomially* larger
Ratio $\frac{f(n)}{n^{log_b a}} = \frac{n \lg n}{n} = \lg n$ is asymptotically less than n^{ϵ} for any positive constant ϵ
Recurrence falls between case 2 and case 3

Examples of algorithm analysis

- Sequential search, or linear search

  Property 1 *Sequential search examines N numbers for each unsuccessful search and about $N/2$ numbers for each successful search on the average.*

  Property 2 *Sequential search in an ordered table examines N numbers for each search in the worst case and about $N/2$ numbers for each search on the average.*

- Consider the effect of M transactions and N entries in the table; with a requirement of c μsec per comparison
- Binary search

  Property 3 *Binary search never examines more than $\lfloor \lg N \rfloor + 1$ numbers.*

  Easily showed by the recurrence for binary search:

$$T_N \leq T_{\lfloor N/2 \rfloor} + 1, \text{ for } N \geq 2 \text{ with } T_1 = 1$$

Guarantees, Predictions, and Limitations

- Run time depends on two things in data
  - Amount of data
  - Type of data (worst case/average case/best case)
- Worst case performance of algorithms
  - Allows to make guarantees about the run time of programs
  - Function provides the maximum number of times an abstract operation will be performed, *independent of data*
    * Property 3 for binary serach algorithms
  - Algorithms with lower worst case performance are preferable and are the goal of algorithm analysis