Asymptotics, Recurrence relations

Analysis of an algorithm seeks to explore its *termination, correctness, time complexity, space complexity*.

These ensure the algorithm has the desired **functionality** (termination and correctness) as well as to **predict** the absolute and relative **performance** for solving a given problem.

Time complexity depends on **machine speed** and **input size and content**. Analysis of algorithm focuses only on the input size and content factor.

Worse-case time: Maximum time needed among all possible inputs of size n.

Asymptotic upper bounds

Let $f(n)$ and $g(n)$ be positive valued functions.

If there exists $N > 0$, $C > 0$ such that for all $n > N$,

$$f(n) \leq Cg(n)$$

We say,

$$f(n) \in O(g(n))$$

$O(g(n))$ is a **set of functions** which consists of all functions $f(n)$ satisfying the above. We often abuse notation and say.

$$f(n) = O(g(n))$$

Asymptotic lower bounds

Let $f(n)$ and $g(n)$ be positive valued functions.

If there exists $N > 0$, $C > 0$ such that for all $n > N$,

$$f(n) \geq Cg(n)$$

We say,

$$f(n) \in \Omega(g(n)) \qquad f(n) = \Omega(g(n))$$

Asymptotic tight bounds

If $f(n) = \Omega(g(n))$ and $f(n) \in O(g(n))$, we say

$$f(n) \in \Theta(g(n)) \quad or \quad f(n) = \Theta(g(n))$$

Asymptotics using limits

Theorem Let $f(n)$ and $g(n)$ be positive functions and $h(n) = f(n)/g(n)$

1. If $lim_{n \to \infty} h(n) = 0$, then $f(n) = O(g(n))$ and $f(n) \neq \Omega(g(n))$
2. If $lim_{n \to \infty} h(n) = b$, where b is some positive number, then $f(n) = \Theta(g(n))$

3. If $lim_{n \to \infty} h(n) = \infty$, then $f(n) = \Omega(g(n))$ and $f(n) \neq O(g(n))$

Common asymptotic growth functions

Function	Name
1	constant
$lg\,lg\,n$	log log
$lg\,n$	log
$n^c, 0 < c < 1$	sublinear
n	linear
$n\,lg\,n$	n log n
n^2	quadratic
n^3	cubic
$n^k, k \geq 1$	polynomial
c^n	exponential
$n!$	factorial

Note: For division, we always assume floor division for integers

Solving recurrence relations

Guess and prove by induction

- Guess a close form solution
  1. Compute value of function at a few points to form an equation
  2. Unfold recurrence by a few steps
- Try to prove by induction
- If failed, use the attempt to guide in refining the solution

Example

Given the following

$$T(0) = 0$$

$$T(n) = 2T(n-1) + 1$$

Compute a few values of $T(n)$,

$$T(1) = 1,\ T(2) = 3,\ T(3) = 7,\ T(4) = 15,\ T(5) = 31,\ T(6) = 63$$

We can guess that T(n) seems to be

$$T(n) = 2^n - 1$$

Proof:

$$T(0) = 2^0 - 1 = 0 \qquad \checkmark$$

$$T(n) = 2T(n-1) + 1$$

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2^{n-1} - 1) + 1 \\ &= 2^n - 2 + 1 \\ &= 2^n - 1 \qquad \checkmark \end{aligned}$$

We can attempt to guess by “unrolling” the recurrence.

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
&= 2(2T(n-2)+1)+1 = 4T(n-2)+3 \\
&= 4(2T(n-3)+1)+3 = 8T(n-3)+7 \\
&= 16T(n-4)+15
\end{aligned}
$$

We can guess that

$$T(n) = 2^kT(n-k) + 2^k - 1 \quad for \quad 1 \le k \le n$$

Proof:

For k = 1,

$$T(n) = 2T(n-1)+1 \qquad ✓$$

For k = i,

$$T(n) = 2^iT(n-i) + 2^i - 1$$

For k = i + 1,

$$T(n) = 2^{i+1}T(n-i-1) + 2^{i+1} - 1$$

Given that,

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
T(n-1) &= \frac{T(n)-1}{2} \\
T(n-i-1) &= \frac{T(n-i)-1}{2}
\end{aligned}
$$

Substituting,

$$
\begin{aligned}
T(n) &= 2^{i+1}T(n-i-1) + 2^{i+1} - 1 \\
2^{i+1}T(n-i-1) + 2^{i+1} - 1 &= 2^{i+1}\frac{T(n-i)-1}{2} + 2^{i+1} - 1 \\
&= 2^iT(n-i) - 2^i + 2^{i+1} - 1 \\
&= 2^iT(n-i) + 2^i - 1 \qquad ✓
\end{aligned}
$$

The above equation is correct, and when k = n,

$$
\begin{aligned}
T(n) &= 2^nT(0) + 2^n - 1 \\
&= 2^n - 1
\end{aligned}
$$

Example: Fibonacci

The recurrence is as follows

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n-1) + F(n-2) \quad for \quad n \ge 2$$

The sequence is increasing and thus,

$$2F(n-2) \le F(n) \le 2F(n-1)$$

F is implied to be exponential in n. Assume $F(n) < a\,b^n$ for some positive real numbers a and b. To prove by induction, we want:

$$
\begin{aligned}
ab^{n-1} + ab^{n-2} &\le ab^n \\
ab^n - ab^{n-1} - ab^{n-2} &\ge 0 \\
ab^{n-2}(b^2 - b - 1) &\ge 0 \\
b^2 - b - 1 &\ge 0
\end{aligned}
$$

Solving for the root, taking into account that b is positive, we get,

$$b \ge \frac{\sqrt{5}+1}{2}$$

Which is also the golden ratio ϕ. We take the lower bound of the ans $b = \frac{\sqrt{5}+1}{2}$

Show using induction on n that

$$\frac{\phi^n}{10} \leq F(n) \leq \phi^n \quad for \quad n \geq 1$$

$$\phi = \frac{\sqrt{5}+1}{2}$$

Taking the right side, for n = 1

$$\begin{aligned} F(1) &= 1 \\ &\leq \phi \quad ✓ \end{aligned}$$

Assume that $F(i) \leq \phi^i$ for all $i \leq k$ where $1 \leq k < n$

$$\begin{aligned} F(n) &= F(n-1) + F(n-2) \\ &\leq \phi^{n-1} + \phi^{n-2} \\ &= \phi^{n-2}(1+\phi) \\ &= \phi^{n-2}(\phi^2) = \phi^n \quad ✓ \end{aligned}$$

Remember from solving the root equation of $b^2 - b - 1 = 0$, we obtain the value of ϕ, thus,

$$\begin{aligned} \phi^2 - \phi - 1 = 0 \\ \phi^2 = \phi + 1 \end{aligned}$$

Taking the left side, for n = 1

$$\begin{aligned} F(1) &= 1 \\ &\geq \frac{\phi}{10} = 0.1618 \quad ✓ \end{aligned}$$

Assume that $F(i) \geq \dfrac{\phi^i}{10}$ for all $i \leq k$ where $1 \leq k < n$

$$\begin{aligned} F(n) &= F(n-1) + F(n-2) \\ &\geq \frac{\phi^{n-1}}{10} + \frac{\phi^{n-2}}{10} \\ &= \phi^{n-2}\frac{(1+\phi)}{10} \\ &= \phi^{n-2}\frac{(\phi^2)}{10} = \frac{\phi^n}{10} \quad ✓ \end{aligned}$$

Example: Merge Sort

$$T(n) = \begin{cases} \Theta(1) & if\ n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & otherwise \end{cases}$$

We shall assume n is a power of 2 for simplicity. Assuming constant $C > 0$,

$$T(n) \leq \begin{cases} C & if\ n = 1 \\ 2T(n/2) + Cn & otherwise \end{cases}$$

If we unfold this recurrence, we will obtain

$$\begin{aligned} T(n) &\leq 2(2T(n/4) + Cn/2) + Cn &= 4T(n/4) + 2Cn \\ &\leq 4(2T(n/8) + Cn/4) + 2Cn &= 8T(n/8) + 3Cn \end{aligned}$$

We can guess that
$T(n) \leq 2^k T(n/2^k) + kCn \quad for \quad 1 \leq k \leq log_2 n$

Proof:

For k = 1,

$$T(n) = 2T(n/2) + Cn \qquad ✓$$

For $k = i$ where $i < k$

$$T(n) \leq 2^i T(n/2^i) + iCn$$

For $k = i + 1$,

$$T(n) \leq 2^{i+1} T(n/2^{i+1}) + (i+1)Cn$$

$$\begin{aligned} 2^{i+1} T(n/2^{i+1}) + (i+1)Cn &= 2^{i+1}\left(\frac{T(n/2^i) - Cn/2^i}{2}\right) + (i+1)Cn \\ &= 2^i T(n/2^i) - Cn + iCn + Cn \\ &= 2^i T(n/2^i) + iCn \quad ✓ \end{aligned}$$

For maximum value $k = log_2 n$

$$\begin{aligned} T(n) &\leq 2^{log_2 n} T(n/2^{log_2 n}) + (log_2 n)Cn \\ &= nT(n/n) + Cnlog_2 n \qquad\qquad T(1) = C \\ &= Cn + Cnlog_2 n \end{aligned}$$

Thus, we can see that $T(n) = O(n \; log \; n)$

To prove that $T(n) = \Omega(n \; log \; n)$, we take a constant D for $0 < D < C$

$$T(n) \geq \begin{cases} D & if \; n = 1 \\ 2T(n/2) + Dn & otherwise \end{cases}$$

The steps for proving will be the same as above.

Example: Harder example

Assume $n = 2^{2^k}$ for some integer k. Consider the following recurrence

$$T(n) = \begin{cases} 1 & if \; n = 2 \\ \sqrt{n} T(\sqrt{n}) + n & if \; n > 2 \end{cases}$$

Assuming for a constant $C > 0$,

$$T(n) \leq \begin{cases} C & if \; n = 2 \\ \sqrt{n} T(\sqrt{n}) + Cn & if \; n > 2 \end{cases}$$

Unfolding this we get,

$$\begin{aligned} T(n) &\leq \sqrt{n}(n^{1/4} T(n^{1/4}) + Cn^{1/2}) + Cn &&= n^{3/4} T(n^{1/4}) + 2Cn \\ &\leq n^{3/4}(n^{1/8} T(n^{1/8}) + Cn^{1/4}) + 2Cn &&= n^{7/8} T(n^{1/8}) + 3Cn \end{aligned}$$

We can guess:

$$T(n) \leq n^{1 - 1/2^i} T(n^{1/2^i}) + i \cdot Cn \quad for \quad 1 \leq i \leq k$$

For $i = 1$,

$$T(n) \leq n^{1/2} T(n^{1/2}) + Cn \qquad ✓$$

For $i = j$,

$$T(n) \leq n^{1 - 1/2^j} T(n^{1/2^j}) + jCn$$

For $i = j + 1$,

$$T(n) \leq n^{1 - 1/2^{j+1}} T(n^{1/2^{j+1}}) + (j+1)Cn$$

$$\begin{aligned} n^{1 - 1/2^{j+1}} T(n^{1/2^{j+1}}) + (j+1)Cn &= n^{1 - 1/2^{j+1}} \frac{T(n^{1/2^j}) - Cn^{1/2^j}}{n^{1/2^{j+1}}} + (j+1)Cn \\ &= n^{1 - 1/2^j} T(n^{1/2^j}) - Cn + jCn + Cn \\ &= n^{1 - 1/2^j} T(n^{1/2^j}) + jCn \quad ✓ \end{aligned}$$

Our guess is correct so to get the equation, we need to form an equation with $T(2)$ where $n^{1/2^i} = 2$

$$
\begin{aligned}
n^{1/2^i} &= 2 \\
\frac{1}{2^i} log_2 n &= 1 \\
log_2 n &= 2^i \\
log_2 log_2 n &= i
\end{aligned}
$$

so substituting for $i = log\, log\, n$

$$
\begin{aligned}
T(n) &\leq n^{1-1/2^{log\, log\, n}} T(n^{1/2^{log\, log\, n}}) + (log\, log\, n)Cn \\
&= \frac{n}{2} T(2) + (log\, log\, n)Cn \\
&= \frac{Cn}{2} + Cnlog\, log\, n
\end{aligned}
$$

Thus, we can see that $T(n) = O(n\, log\, log\, n)$.

Repeating this with the following where D is a constant such that $0 < D < C$

$$
T(n) \geq \begin{cases} D & if\; n = 2 \\ \sqrt{n} T(\sqrt{n}) + Dn & if\; n > 2 \end{cases}
$$

will give you the proof for $T(n) = \Omega(n\, log\, log\, n)$ and thus,

$$
T(n) = \Theta(n\, log\, log\, n)
$$

Divide and conquer recurrences: Recursion Trees

Many divide and conquer algorithms give a running-time recurrence of the form:

$$
T(n) = a\, T(n/b) + f(n)
$$

In merge sort, $a = 2,\; b = 2,\; f(n) = n$

We will again assume a and b are integers, and n is a power of b, with $n = b^{L+1}$. It will later be shown the assumption is not necessary. We also assume

$$
T(1) = \Theta(1) \qquad f(1) = \Theta(1)
$$

Unfolding this recurrence,

$$
\begin{aligned}
T(n) &= a \cdot T(n/b) + f(n) \\
&= a(a \cdot T(n/b^2) + f(n/b)) + f(n) \\
&= a^2 T(n/b^2) + af(n/b) + f(n) \\
&= a^2(aT(n/b^3) + f(n/b^2)) + af(n/b) + f(n) \\
&= a^3 T(n/b^3) + a^2 f(n/b^2) + af(n/b) + f(n) \\
&= \ldots
\end{aligned}
$$

By observing the pattern, we can come up with the following equation

$$
\begin{aligned}
T(n) &= a^{L+1} T(n/b^{L+1}) + \ldots + af(n/b) + f(n) \\
&= a^{L+1} T(1) + \ldots + af(n/b) + f(n) && n = b^{L+1} \\
&= a^{L+1} T(1) + \sum_{i=0}^{L} a^i f(n/b^i) \\
&= \Theta\left(\sum_{i=0}^{L} a^i f(n/b^i) \right) && T(1) = \Theta(1)
\end{aligned}
$$

Exercise: Prove the above solution

Our guess:

$$T(n) = a^k T(n/b^k) + \sum_{i=0}^{k-1} a^i f(n/b^i) \quad for \quad 1 \le k \le L+1$$

When $k = 1$,

$$\begin{aligned} T(n) &= aT(n/b) + \sum_{i=0}^{0} a^i f(n/b^i) \\ &= aT(n/b) + f(n) \quad ✓ \end{aligned}$$

Take $k = j$ for $1 \le j < k$

$$T(n) = a^j T(n/b^j) + \sum_{i=0}^{j-1} a^i f(n/b^i)$$

For $k = j + 1$

$$T(n) = a^{j+1} T(n/b^{j+1}) + \sum_{i=0}^{j} a^i f(n/b^i)$$

With the following equation

$$\begin{aligned} T(n) &= aT(n/b) + f(n) \\ T(n/b^j) &= aT(n/b^{j+1}) + f(n/b^j) \\ T(n/b^{j+1}) &= \frac{T(n/b^j) - f(n/b^j)}{a} \end{aligned}$$

Subbing the above into our equation for $k = j + 1$

$$\begin{aligned} a^{j+1} T(n/b^{j+1}) + \sum_{i=0}^{j} a^i f(n/b^i) &= a^{j+1} (\frac{T(n/b^j) - f(n/b^j)}{a}) + \sum_{i=0}^{j} a^i f(n/b^i) \\ &= a^j T(n/b^j) - a^j f(n/b^j) + \sum_{i=0}^{j} a^i f(n/b^i) \\ &= a^j T(n/b^j) + \sum_{i=0}^{j-1} a^i f(n/b^i) \quad ✓ \end{aligned}$$

In the last step, $a^j f(n/b^j)$ is the last term in $\sum_{i=0}^{j} a^i f(n/b^i)$

Thus, the equation we have guessed is correct.

To prove that $T(n) = \Theta\left(\sum_{i=0}^{L} a^i f(n/b^i)\right)$, we assume for a positive constant C,

$$T(n) \le C\left(\sum_{i=0}^{L} a^i f(n/b^i)\right)$$

$$T(n) \le aT(n/b) + Cf(n)$$

$$T(n) \le a^k T(n/b^k) + C\sum_{i=0}^{k-1} a^i f(n/b^i) \quad for \quad 1 \le k \le L+1$$

The base case $T(1)$ is proven as it is $\Theta(1)$ and we only have to find C greater than the constant.

Taking $k = 1$,

$$T(n) \le aT(n/b) + Cf(n) \quad ✓$$

Taking $k = j$ where $1 \le j < k$

$$T(n) \le a^j T(n/b^j) + C\sum_{i=0}^{j-1} a^i f(n/b^i)$$

Taking $k = j + 1$,

$$T(n) \le a^{j+1}T(n/b^{j+1}) + C\sum_{i=0}^{j} a^i f(n/b^i)$$
$$= a^{j+1}(\frac{T(n/b^j) - Cf(n/b^j)}{a}) + C\sum_{i=0}^{j} a^i f(n/b^i)$$
$$= a^j T(n/b^j) + C\sum_{i=0}^{j-1} a^i f(n/b^i)$$

Thus, the equation holds true. Finally, subbing k = L + 1

$$T(n) \le a^{L+1}T(1) + C\sum_{i=0}^{L} a^i f(n/b^i)$$

Looking at the term $a^{L+1}T(1)$ where $T(1) = \Theta(1)$.

$$a^{L+1} = a \cdot a^L = \Theta(a^L)$$

Since the last term of the summation is $a^L f(b) = \Theta(a^L)$, it dominates the other term.

$$T(n) \le C\sum_{i=0}^{L} a^i f(n/b^i)$$
$$T(n) = O\left(\sum_{i=0}^{L} a^i f(n/b^i)\right)$$

The same can be shown for lower bound as well with the initial assumption of

$$T(n) \ge aT(n/b) + Df(n)$$

where $0 \le D < C$

Masters Theorem

The recurrence $T(n) = a\,T(n/b) + f(n)$ can be solved as follows for all (large enough) n,

If $a \cdot f(n/b) \le \alpha f(n)$ for some $\alpha < 1$, then $T(n) = O(f(n))$

If $a \cdot f(n/b) \ge \beta f(n)$ for some $\beta > 1$, then $T(n) = O(a^{log_b n})$

If $a \cdot f(n/b) = f(n)$ then $T(n) = \Theta(f(n) log_b n)$

Exercise: First Theorem

If $a \cdot f(n/b) \le \alpha f(n)$ for some $\alpha < 1$, then $T(n) = O(f(n))$

For all n, we have $a \cdot f(n/b) \le \alpha f(n)$. This means that in the expansion of the formula $\sum_{i=0}^{L} a^i f(n/b^i)$, the term with $i = 0$ dominates the other terms.

$$a^i \cdot f(n/b^i) \le \alpha \cdot a^{i-1} \cdot f(n/b^{i-1}) \le \ldots \le \alpha^i f(n)$$

From this, we can conclude that

$$\sum_{i=0}^{L} a^i f(n/b^i) \le \sum_{i=0}^{L} \alpha^i f(n)$$

The term on the right is a geometric progression, with the ratio r being α, recall that $\alpha < 1$ and the sum can then be obtained as follows

$$\sum_{i=0}^{L} a^i f(n/b^i) \le \sum_{i=0}^{L} \alpha^i f(n) \le \frac{1}{1-\alpha} f(n) = O(f(n))$$

Also we can prove the lower bound with

$$\sum_{i=0}^{L} a^i f(n/b^i) = f(n) + \sum_{i=1}^{L} a^i f(n/b^i)$$
$$\geq f(n) = \Omega(f(n))$$

Thus, $T(n) = \Theta(f(n))$

Exercise: Second Theorem

If $a \cdot f(n/b) \geq \beta f(n)$ **for some** $\beta > 1$**, then** $T(n) = O(a^{log_b n})$

This means that in the expansion of the formula $\sum_{i=0}^{L} a^i f(n/b^i)$, the term with $i = L$ dominates the other terms.

$$\beta^i \cdot f(n) \leq \beta^{i-1} \cdot a \cdot f(n/b) \leq \ldots \leq a^i f(n/b^i)$$

$$f(n) \leq \frac{1}{\beta} \cdot af(n/b) \leq \ldots \leq \frac{1}{\beta^i} \cdot a^i f(n/b^i)$$

From this, we can conclude that

$$\sum_{i=0}^{L} a^i f(n/b^i) \leq \sum_{i=0}^{L} a^L f(n/b^L) \cdot \frac{1}{\beta^i}$$

Again, this leaves us with a geometric progression where $r = \frac{1}{\beta}$ where $\frac{1}{\beta} < 1$

$$\sum_{i=0}^{L} a^i f(n/b^i) \leq \sum_{i=0}^{L} a^L f(n/b^L) \cdot \frac{1}{\beta^i}$$
$$\leq \frac{1}{1 - \frac{1}{\beta}} a^L f(b)$$
$$= O(a^L)$$
$$= O(a^{log_b n})$$

We can prove the lower bound with the following

$$\sum_{i=0}^{L} a^i f(n/b^i) = a^L f(b) + \sum_{i=0}^{L-1} a^i f(n/b^i)$$
$$\geq a^L = \Omega(a^{log_b n})$$

Thus, $T(n) = \Theta(a^{log_b n}) \quad or \quad \Theta(n^{log_b a})$

Exercise: Third Theorem

If $a \cdot f(n/b) = f(n)$ then $T(n) = \Theta(f(n) log_b n)$

This means that in the expansion of the formula $\sum_{i=0}^{L} a^i f(n/b^i)$, all terms are the same.

$$f(n) = a \cdot f(n/b) = \ldots = a^i f(n/b^i)$$

From this, we can conclude that

$$\sum_{i=0}^{L} a^i f(n/b^i) = \sum_{i=0}^{L} a^L f(n/b^L) = \sum_{i=0}^{L} f(n)$$

$$\sum_{i=0}^{L} f(n) = (L+1) f(n)$$
$$= log_b n \cdot f(n) \ = \Theta(f(n) log_b n)$$

Thus, $T(n) = \Theta(f(n) log_b n)$

Recursion Tree approach

Consider the following

$$T(n) \leq T(n/3) + T(2n/3) + O(n)$$

Drawing out the recursion tree, we can observe that the total contribution at each level is n. The branch that decays by $\frac{n}{3}$ each time decays the fastest while the branch that decays by $\frac{2n}{3}$ each time decays the slowest. Here, we get the number of levels as between $log_3\, n$ and $log_{3/2}\, n$. Since each level takes $O(n)$ work and the number of levels is bounded by $O(log\, n)$, the solution should be $T(n) = O(n\, log\, n)$

Exercise

Prove the above bound on time complexity

With our guess of $T(n) = O(n\, log\, n)$, we must prove that $T(n) \leq Cn\, log\, n$ for some positive constant C.

We have the equation

$$T(n) \leq T(n/3) + T(2n/3) + O(n)$$

which can be expressed with a constant D

$$T(n) \leq T(n/3) + T(2n/3) + Dn$$

Proof:

Assume that $T(k) \leq Ck\, log\, k$ is true for $2 \leq k \leq n-1$

$$\begin{aligned}
T(n) &= T(n/3) + T(2n/3) + Dn \\
&\leq C(\frac{n}{3} log(n/3)) + C(\frac{2n}{3} log(2n/3)) + Dn \\
&= Cn(\frac{1}{3} log(n/3) + \frac{2}{3} log(2n/3) + D/C) \\
&= \frac{Cn}{3}(log(n) - log(3) + 2log(n) + 2log(2) - 2log(3) + 3D/C) \\
&= \frac{Cn}{3}(3log(n) - 3log(3) + 2 + 3D/C) \\
&= Cnlog(n) - Cnlog(3) + \frac{2}{3}Cn + Dn
\end{aligned}$$

We can see that if the term $-Cnlog(3) + \frac{2}{3}Cn + Dn \leq 0$, then $T(n) \leq Cn\, log\, n$.

$$\begin{aligned}
-Cnlog(3) + \frac{2}{3}Cn + Dn \leq 0 \\
D \leq C(log(3) - \frac{2}{3}) \\
D \leq 0.918C
\end{aligned}$$

We can just take $C = 2D$ so now $T(n) \leq 2Dn\, log\, n$. Continuing from our previous equation

$$\begin{aligned}
&= Cnlog(n) - Cnlog(3) + \frac{2}{3}Cn + Dn \\
&= 2Dnlog(n) - 2Dnlog(3) + \frac{4}{3}Dn + Dn \\
&= 2Dnlog(n) - Dn(2log(3) - 7/3) \\
&\leq 2Dn\, log\, n \\
&= O(n\, log\, n)
\end{aligned}$$

Transformations

The methods above may not be able to solve recurrence relations directly, but we can perform a transformation that can change the relation to something we already know.

- **Domain transformation**: $S(n) = T(g(n))$ for some appropriately chosen function g.
  E.g. $S(n) = T(n + a)$ where $g(n) = n + a$.

- **Range transformation**: $S(n) = g(T(n))$ for some appropriately chosen function g.
  E.g. $S(n) = T(n) - 1$ where $g(n) = n - 1$.

Example: Unsimplified Mergesort

Consider the recurrence relation of merge-sort without any simplifying assumption.

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$$

We can overestimate the time bound a little as follows.

$$T(n) \leq 2T(\lceil n/2 \rceil) + n \leq 2T(n/2 + 1) + n$$

With transformation, we can obtain the recurrence in a more standard form which we have seen: $S(n) \leq 2S(n/2) + n$

We make the **domain** transformation $S(n) = T(n + a)$ where a is to be determined which gives us

$$S(n) \leq 2T(n/2 + a/2 + 1) + n + a$$

We want the term such that

$$\begin{aligned} n/2 + a/2 + 1 &= n/2 + a \\ a + 2 &= 2a \\ a &= 2 \end{aligned}$$

Choosing $a = 2$ such that $S(n) = T(n + 2)$ will give us the following

$$\begin{aligned} S(n) &\leq 2T(n/2 + 2/2 + 1) + n + 2 \\ S(n) &\leq 2T(n/2 + 2) + n + 2 \qquad & S(n/2) = T(n/2 + 2) \\ S(n) &\leq 2S(n/2) + n + 2 \\ &= O(n \, log \, n) \end{aligned}$$

Alternative method

Instead of **domain substitution**, we can reason with the following.

Let $2^k - 1 \leq n < 2^k$

$$T(n) < T(2^k) = O(2^k \cdot k)$$

We know the above is true as $T(m) = O(m \, log \, m)$ if m is a power of 2. In this case, $m = 2k$ which is a power of 2.

The term $O(2^k \cdot k) = O(2nlog(2n))$ as $2^k \leq 2n$ which simplifies to $O(n \, log \, n)$

Example

$$T(n) = T(n/2) + T(n/4) + 1$$

We can see that substituting $n = 2^k$ gives us

$$T(2^k) = T(2^{k-1}) + T(2^{k-2}) + 1$$

Which suggests that a **domain** transformation of $t(k) = T(2^k)$ would work, giving us

$$t(k) = t(k-1) + t(k-2) + 1$$

This almost resembles the Fibonacci recurrence and we can get the exact form with a **range** transformation $s(k) = t(k) + a$

$$s(k) - a = s(k-1) - a + s(k-2) - a + 1$$
$$s(k) = s(k-1) + s(k-2) - a + 1$$

We can see that we can remove the constant term if $a = 1$

$$s(k) = s(k-1) + s(k-2)$$

This fits our previously solved Fibonacci sequence and we get

$$s(k) = \Theta(\phi^k) \quad where\ \phi = \frac{1+\sqrt{5}}{2}$$

which implies (via $k = log_2 n$)

$$T(n) = \Theta(\phi^{log_2 n})$$

Exercise

Solve the following recurrence:

$$T(n) = \sum_{i=1}^{n-1} T(i) + n$$

HINT: What is $T(n) - T(n-1)$

$$\begin{aligned} T(n) - T(n-1) &= \sum_{i=1}^{n-1} T(i) + n - \sum_{i=1}^{n-2} T(i) - n + 1 \\ &= T(n-1) + 1 \end{aligned}$$

Unfolding the recurrence

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \ = 4T(n-2) + 3 \\ &= 4(2T(n-3) + 1) + 3 \ = 8T(n-3) + 7 \\ &= 8(2T(n-4) + 1) + 7 \ = 16T(n-4) + 15 \end{aligned}$$

We can guess that the equation is something like

$$T(n) = 2^k T(n-k) + 2^k - 1$$

for $0 \leq k < n$

Taking the case $k = 0$,

$$T(n) = 2^0 T(n-0) + 2^0 - 1 = T(n) \quad ✓$$

For the case $k = i$ for $0 \leq i < k$

$$T(n) = 2^i T(n-i) + 2^i - 1$$

Now for $k = i + 1$, the equation would be

$$T(n) = 2^{i+1} T(n-i-1) + 2^{i+1} - 1$$

To prove the above from $k = i$, we make use of the original equation $T(n) = 2T(n-1) + 1$

$$\begin{aligned} T(n) &= 2^i T(n-i) + 2^i - 1 \\ &= 2^i (2T(n-i-1) + 1) + 2^i - 1 \\ &= 2^{i+1} T(n-i-1) + 2^i + 2^i - 1 \\ &= 2^{i+1} T(n-i-1) + 2^{i+1} - 1 \quad ✓ \end{aligned}$$

Thus, the equation $T(n) = 2^k T(n-k) + 2^k - 1$ is true for $0 \leq k \leq n$

Taking $k = n - 1$

$$
\begin{aligned}
T(n) &= 2^{n-1}T(1) + 2^{n-1} - 1 \qquad T(1) = 1 \\
&= 2^n - 1 \\
&\leq 2^n \\
&= O(2^n)
\end{aligned}
$$

Claim $T(n) \leq C2^n$ for some postive constant C for all $n \geq 1$

Assuming that the above claim holds true for $1 \leq k \leq n-1$

$$
\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&\leq 2(C2^{n-1}) + 1 \\
&\leq C2^n + 1 \\
&= O(2^n)
\end{aligned}
$$

Claim $T(n) \geq D2^n$ for some postive constant $0 \leq D < C$ for all $n \geq 1$

Assuming that the above claim holds true for $1 \leq k \leq n-1$

$$
\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&\geq 2(D2^{n-1}) + 1 \\
&\geq D2^n + 1 \\
&= \Omega(2^n)
\end{aligned}
$$

Thus, we can show that $T(n) = \Theta(2^n)$