CSC3310 Algorithms

Divide and Conquer

William Retert

Milwaukee School of Engineering

Divide and Conquer

Problem-solving strategy

Divide Break problem into smaller subproblems
- Similar structure to original problem

Conquer Solve the subproblems
- Generally recursively

Combine Combine subproblem solutions into a solution for the original problem

Example: Find Minimum

Name	FINDMINIMUMVALUE
Description	Given a sequence of n values, return a minimal value from the sequence
Input	A sequence of $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$
Output	A value a_i such that a_i is in the sequence, and $a_i \leq a_j$ for all $0 \leq j < n$

Example: Find Minimum

Name FINDMINIMUMVALUE

Description Given a sequence of n values, return a minimal value from the sequence

Finding the minimum value with divide and conquer:

Divide split the array into 2 subarrays

Conquer Find the minimum value for each subarray

Combine The smallest value in the array will be the smaller of the two results

Example: Find Minimum

Name FINDMINIMUMVALUE

Description Given a sequence of n values, return a minimal value from the sequence

Finding the minimum value with divide and conquer:

Divide split the array into 2 subarrays

Conquer Find the minimum value for each subarray

Combine The smallest value in the array will be the smaller of the two results

Example: Find Minimum

Name FINDMINIMUMVALUE

Description Given a sequence of n values, return a minimal value from the sequence

Finding the minimum value with divide and conquer:

Divide split the array into 2 subarrays

Conquer Find the minimum value for each subarray

Combine The smallest value in the array will be the smaller of the two results

Example: Find Minimum

Name FINDMINIMUMVALUE

Description Given a sequence of n values, return a minimal value from the sequence

Finding the minimum value with divide and conquer:

Divide split the array into 2 subarrays

Conquer Find the minimum value for each subarray

Combine The smallest value in the array will be the smaller of the two results

Example: Find Minimum

Finding the minimum value with divide and conquer:

- **Divide** split the array into 2 subarrays
- **Conquer** Find the minimum value for each subarray
- **Combine** The smallest value in the array will be the smaller of the two results

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Correctness

$n = 1$ The only element is the smallest

$n > 1$ Either the smallest element will be one of the first $n - 1$ elements, and FindMin will return it, or it will be the last element.

Thus the smaller of *left* and *right* is the smallest element.

Note that we assume FindMin is correct in order to prove it is correct. More precisely, we use induction on n.

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Time Complexity

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ 5 + T(n-1) & \text{otherwise} \end{cases}$$

This is a *recurrence* relation.

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Time Complexity

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ 5 + T(n-1) & \text{otherwise} \end{cases}$$

This is a *recurrence* relation.

Substitution

$$T(n) = 5 + T(n-1)$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = 5 + T(n-1)$$
$$T(n-1) = 5 + T(n-2)$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = 5 + 5 + T(n-2)$$
$$T(n-1) = 5 + T(n-2)$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = 5 + 5 + T(n-2)$$
$$T(n-2) = 5 + T(n-3)$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = 5 + 5 + 5 + T(n-3)$$
$$T(n-2) = 5 + T(n-3)$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = 5 + 5 + 5 + \cdots + 5 + T(n - (n - 1))$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = \overbrace{5 + 5 + 5 + \cdots 5}^{n-1 \text{ times}} + T(n - (n - 1))$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = \overbrace{5 + 5 + 5 + \cdots 5}^{n-1 \text{ times}} + T(n - (n - 1))$$

$$T(n - (n - 1)) = T(1) = 3$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = \overbrace{5 + 5 + 5 + \cdots 5}^{n-1 \text{ times}} + T(n - (n - 1))$$

$$T(n - (n - 1)) = T(1) = 3$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$T(n) = \overbrace{5 + 5 + 5 + \cdots 5}^{n-1 \text{ times}} + 3$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n − 1])
        right ← a[n − 1]
        return left < right?left : right
    end if
end procedure
```

Substitution

$$
\begin{aligned}
T(n) &= \overbrace{5 + 5 + 5 + \cdots 5}^{n-1 \text{ times}} + 3 \\
&= (n-1)5 + 3 \\
T(n) &= 5n - 2
\end{aligned}
$$

Example: Find Minimum

procedure FINDMIN(a)
    $n = a$.length
    if $n = 1$ **then**
        return $a[0]$
    else
        $left \leftarrow$ FINDMIN($a[0 : \frac{n}{2}]$)
        $right \leftarrow$ FINDMIN($a[\frac{n}{2} : n]$)
        return $left < right ? left : right$
    end if
end procedure

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n/2])
        right ← FindMin(a[n/2 : n])
        return left < right?left : right
    end if
end procedure
```

Correctness

The smallest element must either be in the first half of the array or the second half. As *left* is the smallest element in the first half, and *right* is the smallest in the second, the smaller of the two must be the smallest overall

(We can use *strong* induction to show this formally.)

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : ⌊n/2⌋])
        right ← FindMin(a[⌊n/2⌋ : n])
        return left < right?left : right
    end if
end procedure
```

Correctness

(We can use *strong* induction to show this formally.)

Hypothesis: FindMin returns a minimal element in an an array of n elements

Base Case: If $n = 1$, the array contains a single element, which is the smallest.

Inductive Step: Assume FindMin returns a minimal element of any array whose length is less than n

By that assumption

$$\begin{aligned} left &\leq a[i] && (\forall 0 \leq i < \lfloor \tfrac{n}{2} \rfloor) \\ right &\leq a[j] && (\forall \lfloor \tfrac{n}{2} \rfloor \leq j < n) \\ \min(left, right) &\leq left \leq a[i] && (\forall 0 \leq i < \lfloor \tfrac{n}{2} \rfloor) \\ \min(left, right) &\leq right \leq a[j] && (\forall \lfloor \tfrac{n}{2} \rfloor \leq j < n) \\ \min(left, right) &\leq a[k] && (\forall 0 \leq k < n) \end{aligned}$$

$\therefore$ The hypothesis holds for all positive integers n.

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n/2])
        right ← FindMin(a[n/2 : n])
        return left < right?left : right
    end if
end procedure
```

Time Complexity

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ 5 + 2T(\frac{n}{2}) & \text{otherwise} \end{cases}$$

This is a *recurrence* relation.

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n/2])
        right ← FindMin(a[n/2 : n])
        return left < right?left : right
    end if
end procedure
```

Time Complexity

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ 5 + T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) & \text{otherwise} \end{cases}$$

This is a *recurrence* relation.

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n/2])
        right ← FindMin(a[n/2 : n])
        return left < right?left : right
    end if
end procedure
```

Time Complexity

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ 5 + 2T(\frac{n}{2}) & \text{otherwise} \end{cases}$$

This is a *recurrence* relation.

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n/2])
        right ← FindMin(a[n/2 : n])
        return left < right?left : right
    end if
end procedure
```

Time Complexity

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ 5 + 2T(\frac{n}{2}) & \text{otherwise} \end{cases}$$

This is a *recurrence* relation.

Substitution

$$T(n) = 5 + 2T(\frac{n}{2})$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n/2])
        right ← FindMin(a[n/2 : n])
        return left < right?left : right
    end if
end procedure
```

Time Complexity

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ 5 + 2T(\frac{n}{2}) & \text{otherwise} \end{cases}$$

This is a *recurrence* relation.

Substitution

$$\begin{aligned} T(n) &= 5 + 2T(\frac{n}{2}) \\ &= 5 + 2(5 + 2T(\frac{n}{4})) \\ &= 5 + 2(5 + 2(5 + 2T(\frac{n}{8}))) \end{aligned}$$

Example: Find Minimum

```
procedure FindMin(a)
    n = a.length
    if n = 1 then
        return a[0]
    else
        left ← FindMin(a[0 : n/2])
        right ← FindMin(a[n/2 : n])
        return left < right?left : right
    end if
end procedure
```

Time Complexity

$$T(n) = \begin{cases} 3 & \text{if } n = 1 \\ 5 + 2T(\frac{n}{2}) & \text{otherwise} \end{cases}$$

This is a *recurrence* relation.

Substitution

$$\begin{aligned} T(n) &= 5 + 2T(\frac{n}{2}) \\ &= 5 + 2(5 + 2T(\frac{n}{4})) \\ &= 5 + 2(5 + 2(5 + 2T(\frac{n}{8}))) \\ &= \dots \end{aligned}$$

We need a more systematic approach to solving recurrences

Divide-and-Conquer

Divide Break problem into smaller subproblems

Conquer Solve the subproblems

Combine Combine subproblem solutions into a solution for the original problem

If there are a subproblems of size $\frac{n}{b}$, we can represent the time complexity with the recurrence

$$T(n) = D(n) + aT(\frac{n}{b}) + C(n)$$

$D(n)$ is the time to divide the problem

$C(n)$ is the time to combine the solutions

Solving Recurrence Relations

Convert the recurrence into a “closed-form” solution

- Closed-form is *not* recursive

Example 1 Recursive $T(n) = \begin{cases} 0 & n = 1 \\ T(\frac{n}{2}) + 1 & \text{otherwise} \end{cases}$

Closed-form $T(n) \in \Theta(\log_2 n)$

Example 2 Recursive $T(n) = \begin{cases} 6 & n = 1 \\ 2T(n-1) & \text{otherwise} \end{cases}$

Closed-form $T(n) = 6 \times 2^n \in \Theta(2^n)$

Methods for Solving Recurrences

- Repeated substitution
- Guess and check
- Recurrence Trees
- Master method
- Generating Functions

Methods for Solving Recurrences

- Repeated substitution
- **Guess and check**
- Recurrence Trees
- **Master method**
- Generating Functions

In this class, we will only discuss the *guess-and-check* and *master* methods

Guess and Check

Used for upper/lower asymptotic bounds
Approach:

1. guess the closed form
2. substitute the guess into the right-hand side
3. simplify the result

If the bound still holds, we know the guess was accurate

Upper Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-O values to find an upper bound

$T(n) \in O\left(n^2\right)$ If so, then $T(n) \leq cn^2$, and $T(\frac{n}{2}) \leq c\left(\frac{n}{2}\right)^2$. Then

$$T(n) \leq 2(c\left(\frac{n}{2}\right)^2) + 5$$
$$= \frac{c}{2}n^2 + 5$$

We then show that this is bounded by cn^2 (for some c, n_0)

$$\frac{c}{2}n^2 + 5 \leq cn^2$$
$$5 \leq \frac{c}{2}n^2$$
$$\frac{10}{c} \leq n^2 \text{ which is true for } c = 10, n_0 = 1$$

Upper Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-O values to find an upper bound

$T(n) \in O(n^2)$

$T(n) \in O(n \log n)$ Show $T(n) \leq cn \log n$ assuming $T(\frac{n}{2}) \leq c\frac{n}{2} \log \frac{n}{2}$.

$$\begin{aligned} T(n) &\leq 2\left(c\frac{n}{2}\log\frac{n}{2}\right) + 5 \\ &= cn\log\frac{n}{2} + 5 \\ &= cn\log n - cn\log 2 + 5 \end{aligned}$$

$$cn\log n - cn\log 2 + 5 \stackrel{?}{\leq} cn\log n$$

$$5 \stackrel{?}{\leq} (c\log 2)n, \text{ true for } c = \frac{1}{\log 2}, n_0 = 5$$

Upper Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-O values to find an upper bound

$T(n) \in \mathrm{O}\left(n^2\right)$

$T(n) \in \mathrm{O}\left(n \log n\right)$

$T(n) \in \mathrm{O}\left(n\right)$ Show $T(n) \leq cn$ assuning $T(\frac{n}{2}) \leq c\frac{n}{2}$.

$$\begin{aligned} T(n) &\leq 2\left(c\frac{n}{2}\right) + 5 \\ &= cn + 5, \text{ but} \\ cn + 5 &\not\leq cn \end{aligned}$$

That didn't work!

Upper Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-O values to find an upper bound

$T(n) \in O\left(n^2\right)$

$T(n) \in O\left(n \log n\right)$

$T(n) \in O\left(n - 5\right)$ We show $T(n) \leq c(n-5)$ assuming $T(\frac{n}{2}) \leq c(\frac{n}{2} - 5)$

$$\begin{aligned} T(n) &\leq 2c\left(\frac{n}{2} - 5\right) + 5 \\ &= cn - 10c + 5 \end{aligned}$$

$$cn - 10c + 5 \overset{?}{\leq} c(n-5) = cn - 5c$$

$$5 \overset{?}{\leq} 5c \text{ true for } c = 1, n_0 = 2$$

Upper Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-O values to find an upper bound

$T(n) \in O\left(n^2\right)$

$T(n) \in O\left(n \log n\right)$

$T(n) \in O\left(n\right)$ We show $T(n) \leq c(n-5)$ assuming $T(\frac{n}{2}) \leq c(\frac{n}{2} - 5)$

$$\begin{aligned} T(n) &\leq 2c\left(\frac{n}{2} - 5\right) + 5 \\ &= cn - 10c + 5 \end{aligned}$$

$$cn - 10c + 5 \overset{?}{\leq} c(n-5) = cn - 5c$$

$$5 \overset{?}{\leq} 5c \text{ true for } c = 1, n_0 = 2$$

Upper Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-O values to find an upper bound

$T(n) \in O\left(n^2\right)$

$T(n) \in O\left(n \log n\right)$

$T(n) \in O\left(n\right)$

$T(n) \in O\left(\log n\right)$ If so, we can show $T(n) \leq c \log n$ assuming $T(\frac{n}{2}) \leq c \log \frac{n}{2}$

$$\begin{aligned} T(n) &\leq 2\left(c \log \frac{n}{2}\right) + 5 \\ &= 2c \log n - 2c \log 2 + 5 \end{aligned}$$

$$2c \log n - 2c \log 2 + 5 \overset{?}{\leq} c \log n$$

$$c \log n \overset{?}{\leq} 2c \log 2 - 5 \qquad \text{Cannot be true as } n \to \infty$$

Upper Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-O values to find an upper bound

$T(n) \in \mathrm{O}\left(n^2\right)$

$T(n) \in \mathrm{O}\left(n \log n\right)$

$T(n) \in \mathrm{O}\left(n\right)$

$T(n) \notin \mathrm{O}\left(\log n\right)$ If so, we can show $T(n) \leq c \log n$ assuming $T(\frac{n}{2}) \leq c \log \frac{n}{2}$

$$\begin{aligned} T(n) &\leq 2\left(c \log \frac{n}{2}\right) + 5 \\ &= 2c \log n - 2c \log 2 + 5 \end{aligned}$$

$$2c \log n - 2c \log 2 + 5 \overset{?}{\leq} c \log n$$

$$c \log n \overset{?}{\leq} 2c \log 2 - 5 \qquad \text{Cannot be true as } n \to \infty$$

Upper Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-O values to find an upper bound

$T(n) \in \mathrm{O}\left(n^2\right)$

$T(n) \in \mathrm{O}\left(n \log n\right)$

$T(n) \in \mathrm{O}\left(n\right)$

$T(n) \notin \mathrm{O}\left(\log n\right)$ If so, we can show $T(n) \leq c \log n$ assuming $T(\frac{n}{2}) \leq c \log \frac{n}{2}$

$$\begin{aligned} T(n) &\leq 2\left(c \log \frac{n}{2}\right) + 5 \\ &= 2c \log n - 2c \log 2 + 5 \end{aligned}$$

$$2c \log n - 2c \log 2 + 5 \overset{?}{\leq} c \log n$$

$$c \log n \overset{?}{\leq} 2c \log 2 - 5 \qquad \text{Cannot be true as } n \to \infty$$

$\mathrm{O}\left(n\right)$ is the tightest upper bound we found

Lower Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-Ω values to find an lower bound Since we know that $T(n) \in \mathrm{O}(n)$, we can start at

$T(n) \in \Omega(n)$ We show $cn \leq T(n)$ assuming $c\frac{n}{2} \leq T(\frac{n}{2})$.

$$\begin{aligned} 2(c\frac{n}{2}) + 5 &\leq T(n) \\ cn + 5 &\leq T(n) \\ \hline cn &\overset{?}{\leq} cn + 5 \qquad \text{Always true!} \end{aligned}$$

Lower Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-Ω values to find an lower bound

$T(n) \in \Omega(n)$

$T(n) \in \Omega(n \log n)$ True if we show $cn \log n \leq T(n)$ assuming $c\frac{n}{2} \log \frac{n}{2} \leq T(\frac{n}{2})$. Then

$$\begin{aligned}
2\left(c\frac{n}{2}\log\frac{n}{2}\right) + 5 &\leq T(n) \\
cn\log\frac{n}{2} + 5 &\leq T(n) \\
cn\log n - cn\log 2 + 5 &\leq T(n) \\
\hline
cn\log n &\stackrel{?}{\leq} cn\log n - cn\log 2 + 5 \\
cn\log 2 &\stackrel{?}{\leq} 5 \textbf{ false} \text{ for large } n
\end{aligned}$$

Lower Bound

$$T(n) = 2T(n/2) + 5$$

We will guess big-Ω values to find an lower bound

$T(n) \in \Omega(n)$

$T(n) \notin \Omega(n \log n)$

Guess and-Check Method

- Find upper and/or lower bounds
  - not Θ directly.
  - but if it is $O(n)$ and $\Omega(n)\ldots$
- Guessing Strategy
  - Start with a high guess (e.g. $O\left(n^3\right)$) and tighten the bound until you cannot.
    - Be careful! A guess failing does not mean a *proportional* guess cannot succeed.

Master Theorem

Suppose $T(n)$ is a recurrence of the form

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Master Theorem

Suppose $T(n)$ is a recurrence of the form

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Intuition: $aT(\frac{n}{b}) \approx n^{\log_b a}$. The first case is when this dominates $f(n)$; the third is when $f(n)$ dominates it. When they are proportional, we get a logrithmic factor.

Master Theorem

Suppose $T(n)$ is a recurrence of the form

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$

  N.B. $k = 0$ is an important special case. If $f(n) \in \Theta\left(n^{\log_b a}\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Intuition: $aT(\frac{n}{b}) \approx n^{\log_b a}$. The first case is when this dominates $f(n)$; the third is when $f(n)$ dominates it. When they are proportional, we get a logrithmic factor.

Master Theorem

Suppose $T(n)$ is a recurrence of the form

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$

  N.B. $k = 0$ is an important special case. If $f(n) \in \Theta\left(n^{\log_b a}\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Recall that divide-and-conquer algorithms tend to have the form

$$T(n) = D(n) + aT(\frac{n}{b}) + C(n)$$

Case 1

Let's solve

$$T(n) = 9T(\frac{n}{3}) + n$$

Case 1

Let's solve

$$T(n) = 9T(\frac{n}{3}) + n$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Case 1

Let's solve

$$T(n) = 9T(\frac{n}{3}) + n$$

Then,

$$a = 9$$
$$b = 3$$
$$f(n) = n$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Case 1

Let's solve

$$T(n) = 9T(\frac{n}{3}) + n$$

Then,

$$a = 9$$
$$b = 3$$
$$f(n) = n$$
$$n^{\log_3 9} = n^2$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta(f(n))$

Case 1

Let's solve

$$T(n) = 9T(\frac{n}{3}) + n$$

Then,

$$\begin{aligned} a &= 9 \\ b &= 3 \\ f(n) &= n \\ n^{\log_3 9} &= n^2 \\ n &\in O\left(n^{2-\frac{1}{2}}\right) \end{aligned}$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta(f(n))$

Case 1

Let's solve

$$T(n) = 9T(\frac{n}{3}) + n$$

Then,

$$\begin{aligned} a &= 9 \\ b &= 3 \\ f(n) &= n \\ n^{\log_3 9} &= n^2 \\ n &\in \mathrm{O}\left(n^{2-\frac{1}{2}}\right) \end{aligned}$$

Therefore, $T(n) \in \Theta\left(n^2\right)$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in \mathrm{O}\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$

Case 2

Let's solve

$$T(n) = T(\frac{2n}{3}) + 1$$

Case 2

Let's solve

$$T(n) = T(\frac{2n}{3}) + 1$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Case 2

Let's solve

$$T(n) = T(\frac{2n}{3}) + 1$$

Then,

$$a = 1$$
$$b = \frac{3}{2}$$
$$f(n) = 1$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Case 2

Let's solve

$$T(n) = T(\frac{2n}{3}) + 1$$

Then,

$$a = 1$$
$$b = \frac{3}{2}$$
$$f(n) = 1$$
$$n^{\log_{\frac{3}{2}} 1} = n^0 = 1$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Case 2

Let's solve

$$T(n) = T(\frac{2n}{3}) + 1$$

Then,

$$\begin{aligned} a &= 1 \\ b &= \frac{3}{2} \\ f(n) &= 1 \\ n^{\log_{\frac{3}{2}} 1} &= n^0 = 1 \\ 1 &\in \Theta(1) \end{aligned}$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta(f(n))$

Case 2

Let's solve

$$T(n) = T(\frac{2n}{3}) + 1$$

Then,

$$\begin{aligned} a &= 1 \\ b &= \frac{3}{2} \\ f(n) &= 1 \\ n^{\log_{\frac{3}{2}} 1} &= n^0 = 1 \\ 1 &\in \Theta(1) \\ k &= 0 \end{aligned}$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$

Case 2

Let's solve

$$T(n) = T(\frac{2n}{3}) + 1$$

Then,

$$\begin{aligned} a &= 1 \\ b &= \frac{3}{2} \\ f(n) &= 1 \\ n^{\log_{\frac{3}{2}} 1} &= n^0 = 1 \\ 1 &\in \Theta(1) \\ k &= 0 \end{aligned}$$

Therefore, $T(n) \in \Theta(\log n)$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$

Case 3

Let's solve

$$T(n) = 3T(\frac{n}{4}) + n \log n$$

Case 3

Let's solve

$$T(n) = 3T(\frac{n}{4}) + n \log n$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Case 3

Let's solve

$$T(n) = 3T(\frac{n}{4}) + n \log n$$

Then,

$$a = 3$$
$$b = 4$$
$$f(n) = n \log n$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta(f(n))$

Case 3

Let's solve

$$T(n) = 3T(\frac{n}{4}) + n \log n$$

Then,

$$a = 3$$
$$b = 4$$
$$f(n) = n \log n$$
$$n^{\log_4 3} \leq n$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta(f(n))$

Case 3

Let's solve

$$T(n) = 3T(\frac{n}{4}) + n \log n$$

Then,

$$\begin{aligned} a &= 3 \\ b &= 4 \\ f(n) &= n \log n \\ n^{\log_4 3} &\leq n \\ n \log n \in \Omega(n) &= \Omega\left(n^{\log_4 3 + \epsilon}\right) (\epsilon = 1 - \log_4 3) \end{aligned}$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there is a constant $\epsilon > 0$ such that $f(n) \in O\left(n^{\log_b a - \epsilon}\right)$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
- If there is a constant $k \geq 0$ such that $f(n) \in \Theta\left(n^{\log_b a} \log^k n\right)$, then $T(n) \in \Theta\left(n^{\log_b a} \log^{k+1} n\right)$
- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta(f(n))$

Case 3

Let's solve

$$T(n) = 3T(\frac{n}{4}) + n \log n$$

Then,

$$\begin{aligned} a &= 3 \\ b &= 4 \\ f(n) &= n \log n \\ n^{\log_4 3} &\leq n \\ n \log n \in \Omega(n) &= \Omega\left(n^{\log_4 3 + \epsilon}\right) (\epsilon = 1 - \log_4 3) \\ 3\frac{n}{4} \log \frac{n}{4} &= \frac{3}{4} n \log n - \frac{3}{4} n \log 4 \leq \frac{3}{4} n \log n \qquad (\delta = \frac{3}{4}) \end{aligned}$$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta(f(n))$

Case 3

Let's solve

$$T(n) = 3T(\frac{n}{4}) + n \log n$$

Then,

$$\begin{aligned} a &= 3 \\ b &= 4 \\ f(n) &= n \log n \\ n^{\log_4 3} &\leq n \\ n \log n \in \Omega(n) &= \Omega\left(n^{\log_4 3 + \epsilon}\right) (\epsilon = 1 - \log_4 3) \\ 3\frac{n}{4} \log \frac{n}{4} &= \frac{3}{4} n \log n - \frac{3}{4} n \log 4 \leq \frac{3}{4} n \log n \qquad (\delta = \frac{3}{4}) \end{aligned}$$

Therefore, $T(n) \in \Theta(n \log n)$

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- If there are constants $\epsilon > 0$ and $\delta < 1$ such that $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right)$ and $af(n/b) \leq \delta f(n)$ for sufficiently large n, then $T(n) \in \Theta(f(n))$

Example

$$T(n) = 2T(\frac{n}{2}) + 5$$

Example

$$T(n) = 2T(\frac{n}{2}) + 5$$

$$a = 2$$
$$b = 2$$
$$f(n) = 5$$

Example

$$T(n) = 2T(\frac{n}{2}) + 5$$

$$a = 2$$
$$b = 2$$
$$f(n) = 5$$

Is this Case 1, Case 2, or Case 3?