Analysis of Algorithms & Recurrence Relations

Revised for use by Midwestern State University

Recursive Algorithms

Definition: An algorithm that calls itself

Components of a recursive algorithm

1. Base case(s)
    - Computation with no recursion
2. Recursive cases
    - Recursive calls
    - Combining recursive results

Recursion Example

Code (for input size n)

```
DoWork (int n)
  if (n == 1)
    A
  else
    DoWork(n/2)
    DoWork(n/2)
```

critical sections

Code execution

- A $\Rightarrow$ 1 times
- DoWork(n/2) $\Rightarrow$ 2 times

$\text{Time}(1) = C$ $\quad\quad$ $\text{Time}(n) = 2 \times \text{Time}(n/2) + C$

Solving Recurrence Relations

A **recurrence relation** is an equation that describes a function in terms of itself by using smaller inputs

The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

describes the **running time** for a function that contains recursion.

Solving Recurrence Relations

Three general methods for solving recurrences

- **Substitution method (not covered in this class)**
- **Iteration method**
- **Master method**

Iteration Method

Iteration method:

1. **Expand the recurrence *k* times**
2. **Work some algebra to express as a summation**
3. **Evaluate the summation**

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + c & n > 0 \end{cases}$$

T(n) = T(n-1) + c

Find the value of T(n-1) by replacing **n** with **n-1**:

T(n-1) = T((n-1 – 1) + c = T(n -2) + c

Now, we know T(n-1) = T(n -2) + c

so we can substitute into the original equation:

Original Eq:	T(n)	=	T(n-1) + c
Sub for T(n-1)	T(n)	=	T(n-2) + c + c
		=	T(n-2) + 2c

$$T(n)=\begin{cases} 0 & n=0 \\ T(n-1)+c & n>0 \end{cases}$$

$$\begin{aligned} T(n) &= T(n-1) + c = T(n-2) + c + c \\ &= T(n-2) + 2c = T(n-3) + 2c + c \\ &= T(n-3) + 3c \\ &= \ldots \\ &= T(n-k) + kc \end{aligned}$$

To stop the recursion, we should have

$n - k = 0 \Rightarrow k = n$

$\boldsymbol{T(n) = T(0) + cn = cn}$

Thus in general $\boldsymbol{T(n) = O(n)}$

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

$T(n) \quad = T(n-1) + n$

Find the value of $T(n-1)$ by replacing **n** with **n-1**:

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

Now, we know $T(n-1) = T(n-2) + n-1$

so we can substitute into the original equation:

Original Eq: $\quad T(n) \quad = \quad T(n-1) + n$

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

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

$T(n)$ = $T(n\text{-}1)$ + n

= $T(n\text{-}2) + n\text{-}1 + n$

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

= $T(n\text{-}4) + n\text{-}3 + n - 2 + n - 1 + n$

= …

= $T(n\text{-}k) + (n\text{-}k+1) + \ldots + n\text{-}3 + n - 2 + n - 1 + n$

To stop the recursion, we should have $n - k = 0$ ➔ $k = n$

= $T(0) + 1 + 2 + 3 + \ldots + n\text{-}3 + n - 2 + n - 1 + n$

$$T(0) + \sum_{i=1}^{n} i \quad = \quad 0 + \sum_{i=1}^{n} i \quad = \quad n\frac{n+1}{2}$$

$$T(n) \quad = \quad \frac{n(n+1)}{2} = O(n^2)$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

$T(n) \quad = \quad 2\,T(n/2) + c$

Find the value of $T(n/2)$ by replacing **n** with **(n/2)**:

$T(n/2) \quad = \quad 2T(n/2/2) + c = 2T(n/2^2) + c$

Now, we know $T(n/2) = 2T(n/2^2) + c$

so we can substitute into the original equation:

Original Eq: $\quad T(n) \quad = \quad 2\,T(n/2) + c$

$= \quad 2(2T(n/2^2) + c) + c$

$= \quad 2^2T(n/2^2) + 3c$ *//distribute 2 and combine*

$$T(n)=\begin{cases} c & n=1 \\ 2T\left(\dfrac{n}{2}\right)+c & n>1 \end{cases}$$

$T(n)$	$=$	$2\,T(n/2) + c$	1
	$=$	$2(2\,T(n/2/2) + c) + c$	2
	$=$	$2^2\,T(n/2^2) + 2c + c$	
	$=$	$2^2(2\,T(n/2^2/2) + c) + (2^2-1)c$	3
	$=$	$2^3\,T(n/2^3) + 4c + 3c$	
	$=$	$2^3\,T(n/2^3) + (2^3-1)c$	
	$=$	$2^3(2\,T(n/2^3/2) + c) + 7c$	4
	$=$	$2^4\,T(n/2^4) + (2^4-1)c$	
	$=$	…	
	$=$	$2^k\,T(n/2^k) + (2^k - 1)c$	k

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

So far, we have: $\boldsymbol{T(n) = 2^k T(n/2^k) + (2^k - 1)c}$

To stop the recursion, we should have

$\boldsymbol{n/2^k = 1 \Rightarrow n = 2^k \Rightarrow k = \lg n}$

$$\begin{aligned} \mathbf{T(n)} &= \mathbf{n\,T(n/n) + (n - 1)c} \\ &= \mathbf{n\,T(1) + (n-1)c} \\ &= \mathbf{nc + (n-1)c} \\ &= \mathbf{nc + nc - c = 2cn - c} \end{aligned}$$

$\mathbf{T(n) = 2cn - c = O(n)}$

Example Recurrence Solutions

Examples

- $T(n) = T(n-1) + c \Rightarrow O(n)$ Ex. Factorial
- $T(n) = T(n-1) + n \Rightarrow O(n^2)$ Ex. Worst case Quicksort
- $T(n) = T(n/2) + c \Rightarrow O(\log_2 n)$ Ex. Binary search
- $T(n) = 2\,T(n/2) + c \Rightarrow O(n)$ Ex. Max
- $T(n) = 2\,T(n/2) + n \Rightarrow O(n \log_2 n)$ Ex. Merge sort
- $T(n) = 2\,T(n-1) + c \Rightarrow O(2^n)$

Fibonacci: $T(n) = \Theta(\phi^n)$, where ϕ is the golden ratio ($\phi = \frac{(1+\sqrt{5})}{2}$).

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn^d & n > 1 \end{cases}$$

Recurrences in the form shown above can be evaluated using a <u>simplified version</u> of the Master Theorem:

$$T(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^d \log_b n) & a = b^d \\ O(n^{\log_b a}) & a > b^d \end{cases}$$

Prove by solving in general form.

Using The Simplified Master's Method

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

- *$a=1$, $b=2$, $d = 1$*
- *$1 < 2^1$*
- *Thus the solution is $T(n) = O(n)$*

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

- *$a=2$, $b=2$, $d = 1$*
- *$2 = 2^1$*
- *Thus the solution is $T(n) = O(n \log n)$*

$T(n) = 9T(n/3) + n$

- *$a=9$, $b=3$, $d = 1$*
- *$9 > 3^1$*
- *Thus the solution is $T(n) = O(n^{\log_3 9}) = O(n^2)$*