Recurrences and Solving recurrence equations

Complexity Analysis of Recursive Functions

- Decide on a parameter indicating size of input
- Identify the basic operation
- Check whether the number of times the basic op. is executed may vary on different inputs of the same size.
    - If it may, the worst, average, and best cases must be investigated separately
- Set up a recurrence relation with an appropriate initial condition
- Solve the recurrence

Factorial Function – Recursive Definition

$n! = 1 * 2 * \ldots *(n-1) * n$ for $n \geq 1$

$0! = 1$

Recursive definition of n!

$F(n) = F(n-1) * n$ for $n \geq 1$

$F(0) = 1$

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Factorial Function – Recursive Definition

$T(n) = T(n-1) + 1 \qquad ; n > 0$

To calculate F(n-1) To multiply F(n-1) by n

$T(0) = 0$ No multiplication when n = 0

$$\boxed{\begin{array}{ll} T(n) = T(n-1) + 1 & ; n > 0 \\ T(0) = 0 & \end{array}}$$

ALGORITHM $F(n)$
//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Solving Recurrence Relation – Substitution method

$$T(n) = T(n-1) + 1 \quad ; n > 0$$
$$T(0) = 0$$

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

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

$\quad = T(n-3) + 1 + 2 = T(n-3) + 3$

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

When $n - i = 0$, $(n = i)$

$T(n) \quad = T(0) + n$

$T(n) \quad = n$

$T(n)$ is $O(n)$

Solving Recurrence Relation – Substitution method

$$T(n) = T(n/2) + 1 \quad ; n > 1$$
$$T(1) = 1$$

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

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

$\quad = T(n/8) + 1 + 2 = T(n\ /\ 8) + 3$

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

When $n\ /\ 2^i = 1$, $(n = 2^i\ ,\ i = \log n)$

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

$T(n) \quad = \log n$

$T(n)$ is $O(\log n)$

Solving Recurrence Relation – Substitution method

$$T(n) = 2T(n/2) + 1 \qquad ; n > 1$$

$$T(1) = 1$$

Solving Recurrence Relation – Substitution method

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

$\quad\quad = 2[2T(n/4) + 1\ ] + 1 \ = 4\ T(n/4) + 3$

$\quad\quad = 4[2\ T(n/8) + 1] + 3 = 8\ T(n/8) + 7$

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

When $n\ /\ i = 1$, $\quad (n = i\ )$

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

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

$T(n)$ is $O(n)$

$T(n\ ) = 2T(n/2) + 1 \qquad ; n > 1$

$T(1) = 1$

Solving Recurrence Relation – Substitution method

$$T(n) = T(n-1) + n \qquad ; n > 1$$

$$T(0) = 0$$

Solving Recurrence Relation – Substitution method

$T(n) = T(n-1) + n$; $n > 1$

$T(0) = 0$

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + (n-1) + n = T(n-2) + (n-1) + n \\
&= T(n-3) + (n-2) + (n-1) + n \\
&= T(n-i) + (n-i+1) + (n-i+2) + (n-i+i)
\end{aligned}
$$

When $n - i = 0$, $(n = i)$

$T(n) = T(0) + 1 + 2 + \dots + n$

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

$T(n)$ is $O(n^2)$

Solving Recurrence Relation – Substitution method

$T(n\ ) = 2T(n/2) + n \qquad ; n > 1$

$T(1) = 1$

Solving Recurrence Relation – Substitution method

$T(n) = 2T(n/2) + n \quad ; n > 1$

$T(1) = 1$

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

$= 2[2T(n/4) + n/2\ ] + n = 4\ T(n/4) + n + n$

$= 4[2\ T(n/8) + n/4] + 2n = 8\ T(n/8) + 3n$

$= 2^i\ T(n / 2^i) + i.n$

When $n / 2^i = 1$, $(n = 2^i , i = \log n)$

$T(n) = n\ T(1) + n \log n$

$T(n) = n \log n$

$T(n)$ is $O(n \log n)$

Mergesort

```
Merge_sort(A,le,r) //n = r-le+1
  if (le>=r) return
  else
      m = floor((le+r)/2)
      Merge_sort(A,le,m);
      Merge_sort(A,m+1,r);
      Merge(A,le,m,r);
```

Writing Recurrence Relation : Example 1

Mergesort

```
Merge_sort(A,le,r) //n = r-le+1
  if (le>=r) return
  else
      m = floor((le+r)/2)
      Merge_sort(A,le,m);
      Merge_sort(A,m+1,r);
      Merge(A,le,m,r);
```

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

$T(1) = c$

Writing Recurrence Relation : Example 2

Binary Search - recursive

```
/* Adapted from Sedgewick, n = right-left+1 */
int search(int A[], int left, int right, int v)
  { int m = (left+right)/2;
    if (left > right) return -1;
    if (v == A[m]) return m;
    if (left == right) return -1;
    if (v < A[m])
         return search(A, left, m-1, v);
    else
          return search(A, m+1, right, v);
  }
```

Common recurrence relations

Recurrence	Algorithm	Big O Solution
$T(n) = T(n/2) + O(1)$	Binary Search	$O(\log n)$
$T(n) = T(n-1) + O(1)$	Linear Search	$O(n)$
$T(n) = 2T(n/2) + O(1)$	Tree traversal	$O(n)$
$T(n) = T(n-1) + O(n)$	Selection Sort	$O(n^2)$
$T(n) = 2T(n/2) + O(n)$	Merge Sort	$O(n \log n)$

Master Theorem

Theorem (Master Theorem)

Let $T(n)$ be a monotonically increasing function that satisfies

$$\begin{aligned} T(n) &= aT(\tfrac{n}{b}) + f(n) \\ T(1) &= c \end{aligned}$$

where $a \geq 1, b \geq 2, c > 0$. If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

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

Master Theorem - Pitfalls

You cannot use the Master Theorem if

- $T(n)$ is not monotone, ex: $T(n) = \sin(n)$
- $f(n)$ is not a polynomial, ex: $T(n) = 2T(n/2) + 2^n$
- b cannot be expressed as a constant, ex: $T(n) = T(\sqrt{n})$

- The Master Theorem does not solve a recurrence relation.
- It describes only the asymptotic behaviour.
  - Rather than solving exactly the recurrence relation

Master Theorem : Example 1

Let $T(n) = T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$. What are the parameters?

$$\begin{aligned} a &= 1 \\ b &= 2 \\ d &= 2 \end{aligned}$$

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

Therefore which condition?

Since $1 < 2^2$, case 1 applies.

Thus we conclude that

$$T(n) \in \Theta(n^d) = \Theta(n^2)$$

Theorem (Master Theorem)

Let $T(n)$ be a monotonically increasing function that satisfies

$$\begin{aligned} T(n) &= aT(\tfrac{n}{b}) + f(n) \\ T(1) &= c \end{aligned}$$

where $a \geq 1, b \geq 2, c > 0$. If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

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

Master Theorem : Example 2

Let $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 42$. What are the parameters?

$$\begin{aligned} a &= 2 \\ b &= 4 \\ d &= \tfrac{1}{2} \end{aligned}$$

Therefore which condition?

Since $2 = 4^{\frac{1}{2}}$, case 2 applies.

Thus we conclude that

$$T(n) \in \Theta(n^d \log n) = \Theta(\sqrt{n} \log n)$$

Theorem (Master Theorem)

Let $T(n)$ be a monotonically increasing function that satisfies

$$\begin{aligned} T(n) &= aT(\tfrac{n}{b}) + f(n) \\ T(1) &= c \end{aligned}$$

where $a \geq 1, b \geq 2, c > 0$. If $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

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

Exercise

1. $T(n) = 3T(n/2) + n^2$

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

3. $T(n) = T(n/2) + 2^n$

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

5. $T(n) = 16T(n/4) + n$

6. $T(n) = 2T(n/2) + n \log n$

Exercise - Solution

1. $T(n) = 3T(n/2) + n^2$
2. $T(n) = 4T(n/2) + n^2$
3. $T(n) = T(n/2) + 2^n$
4. $T(n) = 2^n T(n/2) + n^n$
5. $T(n) = 16T(n/4) + n$
6. $T(n) = 2T(n/2) + n \log n$

1. $\Theta\ (n^2)$ Case 3
2. $\Theta\ (n^2 \log n)$ Case 2
3. $\Theta\ (2^n)$ Case 3
4. Does not apply. a is not a constant
5. $\Theta\ (n^2)$ Case 1
6. $\Theta\ (n \log^{2+n})$ Case 3