

COMP9020 Week 5

Term 3, 2020

Recursion

- [LLM] - Ch. 6, 21.1
- [RW] - Ch. 4

Outline

- Recursion
- Recursive Data Types
- Recursive programming
- Solving recurrences

Outline

- Recursion
- Recursive Data Types
- Recursive programming
- Solving recurrences

Recursion

Fundamental concept in Computer Science

- Recursion in algorithms: Solving problems by reducing to smaller cases
 - Factorial
 - Towers of Hanoi
 - Mergesort, Quicksort

Recursion

Fundamental concept in Computer Science

- Recursion in algorithms: Solving problems by reducing to smaller cases
 - Factorial
 - Towers of Hanoi
 - Mergesort, Quicksort
- Recursion in data structures: Finite definitions of **arbitrarily large** objects
 - Natural numbers
 - Words
 - Linked lists
 - Formulas
 - Binary trees

Recursion

Fundamental concept in Computer Science

- Recursion in algorithms: Solving problems by reducing to smaller cases
 - Factorial
 - Towers of Hanoi
 - Mergesort, Quicksort
- Recursion in data structures: Finite definitions of **arbitrarily large** objects
 - Natural numbers
 - Words
 - Linked lists
 - Formulas
 - Binary trees
- Analysis of recursion: Proving properties
 - Recursive sequences (e.g. Fibonacci sequence)
 - Structural induction

Recursion

Consists of a basis (B) and recursive process (R).

A sequence/object/algorithm is recursively defined when (typically)
(B) some initial terms are specified, perhaps only the first one;
(R) later terms stated as functional expressions of the earlier terms.

NB

(R) also called **recurrence formula (especially when dealing with sequences)**

Example: Factorial

Example

Factorial:

$$(B) \quad 0! = 1$$

$$(R) \quad (n + 1)! = (n + 1) \cdot n!$$

fact(n):

$$(B) \quad \text{if}(n = 0): 1$$

$$(R) \quad \text{else: } n * \text{fact}(n - 1)$$

Example: Euclid's gcd algorithm

Example

$$\gcd(m, n) = \begin{cases} m & \text{if } m = n \\ \gcd(m - n, n) & \text{if } m > n \\ \gcd(m, n - m) & \text{if } m < n \end{cases}$$

Example: Towers of Hanoi

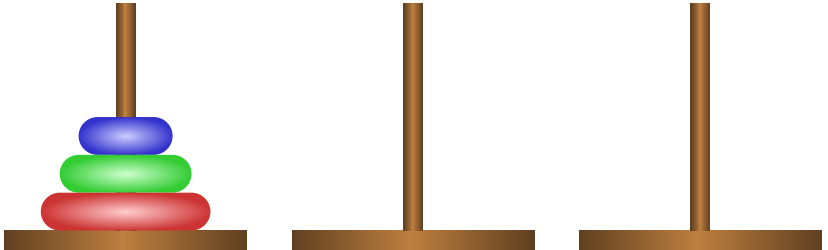
- There are 3 towers (pegs)
- n disks of decreasing size placed on the first tower
- You need to move all disks from the first tower to the last tower
- Larger disks cannot be placed on top of smaller disks
- The third tower can be used to temporarily hold disks

Example: Towers of Hanoi

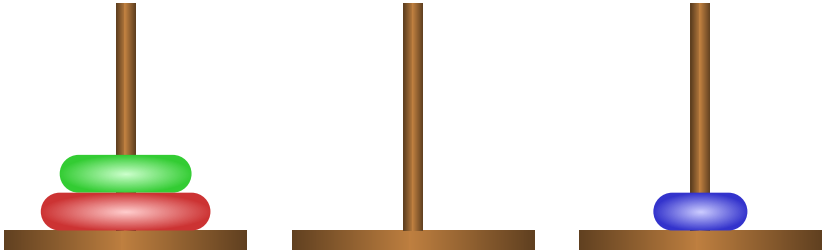
Questions

- Describe a general solution for n disks
- How many moves does it take?

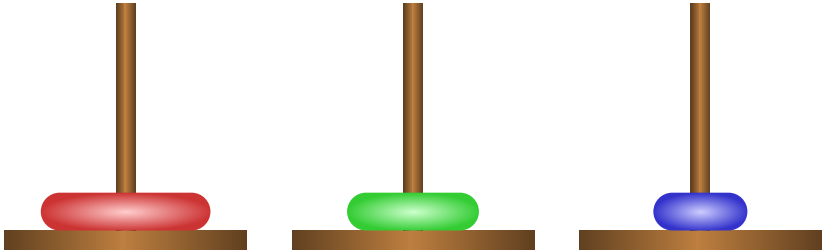
Example: Towers of Hanoi



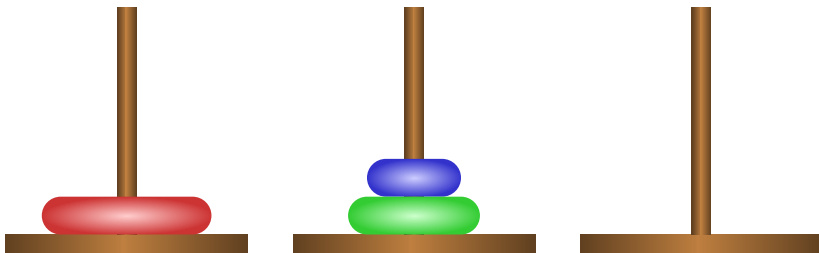
Example: Towers of Hanoi



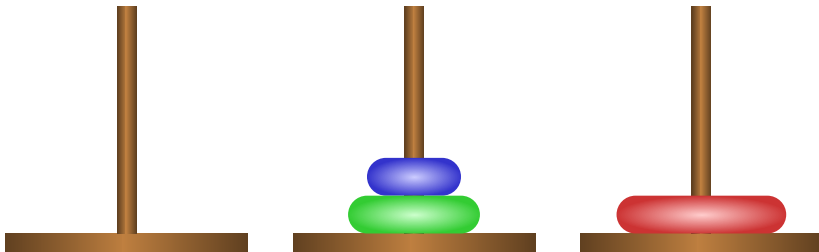
Example: Towers of Hanoi



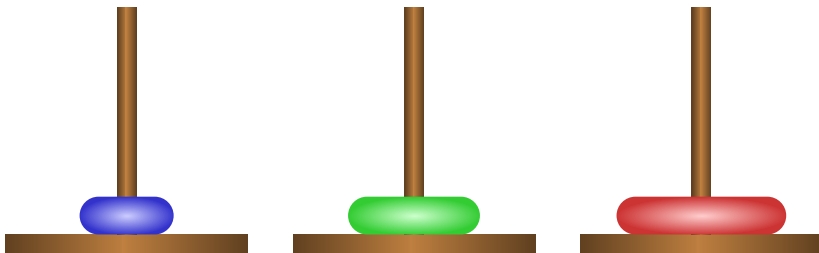
Example: Towers of Hanoi



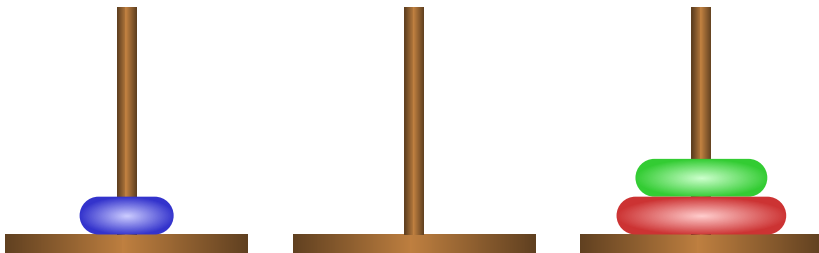
Example: Towers of Hanoi



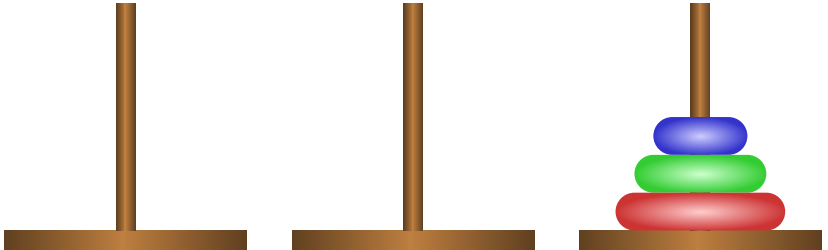
Example: Towers of Hanoi



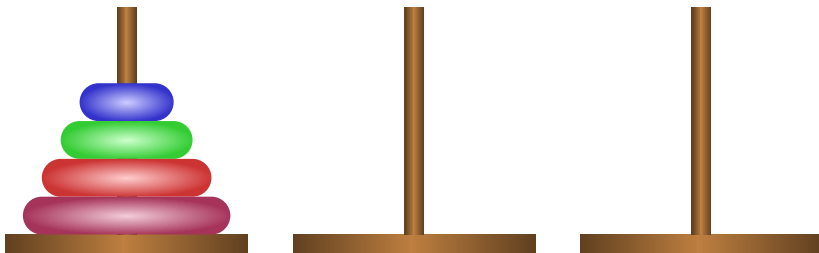
Example: Towers of Hanoi



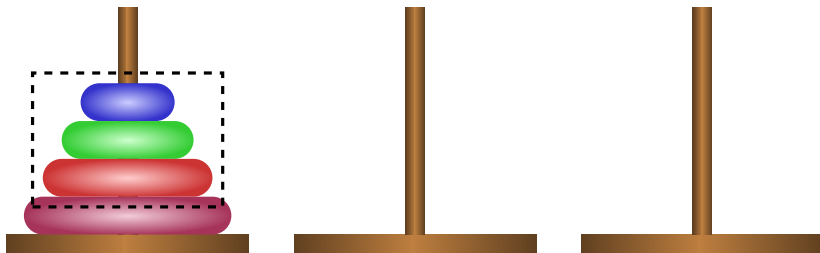
Example: Towers of Hanoi



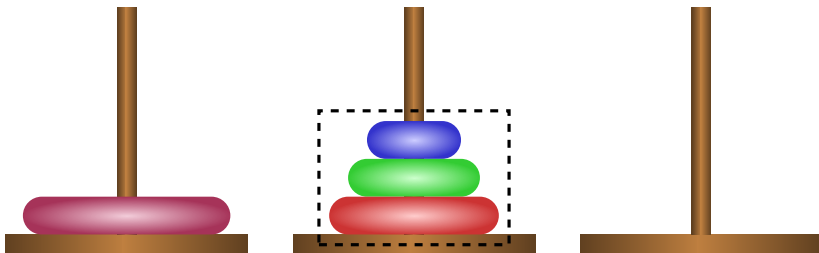
Example: Towers of Hanoi



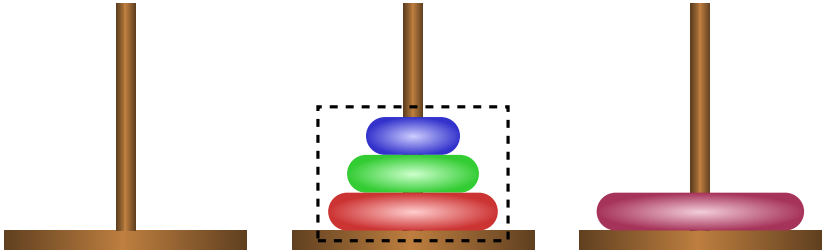
Example: Towers of Hanoi



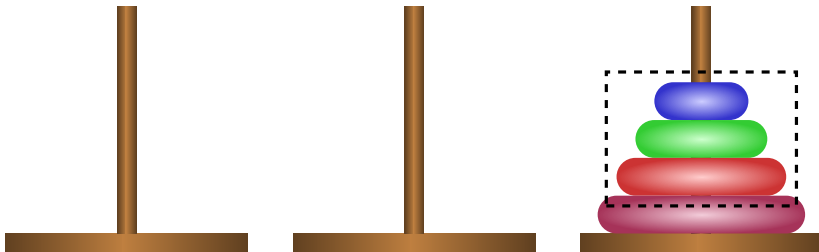
Example: Towers of Hanoi



Example: Towers of Hanoi



Example: Towers of Hanoi



Example: Towers of Hanoi

Questions

- Describe a general solution for n disks
- How many moves does it take? ?

Outline

- Recursion
- Recursive Data Types
- Recursive programming
- Solving recurrences

Example: Natural numbers

Example

A natural number is either 0 (B) or one more than a natural number (R).

Formal definition of \mathbb{N} :

- (B) $0 \in \mathbb{N}$
- (R) If $n \in \mathbb{N}$ then $(n + 1) \in \mathbb{N}$

Example: Odd/Even numbers

Example

The set of even numbers can be defined as:

- (B) 0 is an even number
- (R) If n is an even number then $n + 2$ is an even number

Example: Odd/Even numbers

Example

The set of odd numbers can be defined as:

- (B) 1 is an odd number
- (R) If n is an odd number then $n + 2$ is an odd number

Example: Fibonacci numbers

Example

The Fibonacci sequence starts $0, 1, 1, 2, 3, \dots$ where, after $0, 1$, each term is the sum of the previous two terms.

Formally, the sequence of Fibonacci numbers: F_0, F_1, F_2, \dots where the n -th Fibonacci number F_n is defined as:

- (B) $F_0 = 0$,
- (B) $F_1 = 1$,
- (R) $F_n = F_{n-1} + F_{n-2}$

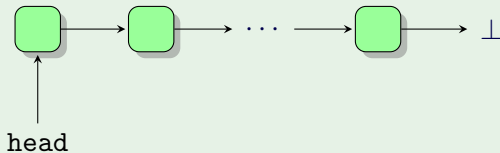
NB

Could also define the Fibonacci sequence as a function
 $\text{FIB} : \mathbb{N} \rightarrow \mathbb{F}$.

Example: Linked lists

Example

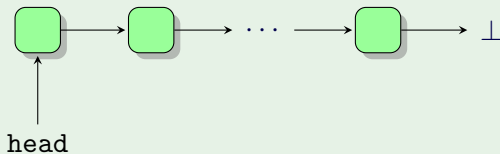
A linked list is zero or more linked list nodes:



Example: Linked lists

Example

A linked list is zero or more linked list nodes:



In C:

```
struct node{  
    int data;  
    struct node *next;  
}
```


Example: Linked lists

Example

We can view the linked list **structure** abstractly. A linked list is either:

- (B) an empty list, or
- (R) an ordered pair (Data, List).

Example: Words over Σ

Example

A word over an alphabet Σ is either λ (B) or a symbol from Σ followed by a word (R).

Formal definition of Σ^* :

- (B) $\lambda \in \Sigma^*$
- (R) If $w \in \Sigma^*$ then $aw \in \Sigma^*$ for all $a \in \Sigma$

NB

*This matches the recursive definition of a **Linked List** data type.*

Example: Expressions in the Proof Assistant

Example

- (B) $A, B, \dots, Z, a, b, \dots, z$ are expressions
- (B) \emptyset and \mathcal{U} are expressions
- (R) If E is an expression then so is (E) and E^c
- (R) If E_1 and E_2 are expressions then:
 - $(E_1 \cup E_2)$,
 - $(E_1 \cap E_2)$,
 - $(E_1 \setminus E_2)$, and
 - $(E_1 \oplus E_2)$ are expressions.

Example: Propositional formulas

Example

A well-formed formula (wff) over a set of propositional variables, PROP is defined as:

- (B) \top is a wff
- (B) \perp is a wff
- (B) p is a wff for all $p \in \text{PROP}$
- (R) If φ is a wff then $\neg\varphi$ is a wff
- (R) If φ and ψ are wffs then:
 - $(\varphi \wedge \psi)$,
 - $(\varphi \vee \psi)$,
 - $(\varphi \rightarrow \psi)$, and
 - $(\varphi \leftrightarrow \psi)$ are wffs.

Exercises

Exercises

RW: 4.4.4 (a) Give a recursive definition for the sequence

$(2, 4, 16, 256, \dots)$

(b) Give a recursive definition for the sequence

$(2, 4, 16, 65536, \dots)$

Exercises

Exercises

RW: 4.4.4 (a) Give a recursive definition for the sequence

$(2, 4, 16, 256, \dots)$

?

(b) Give a recursive definition for the sequence

$(2, 4, 16, 65536, \dots)$

?

Outline

- Recursion
- Recursive Data Types
- Recursive programming
- Solving recurrences

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

The factorial function:

```
fact( $n$ ):  
( $B$ )    if( $n = 0$ ): 1  
( $R$ )    else:  $n * \text{fact}(n - 1)$ 
```


Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Summing the first n natural numbers:

```
sum( $n$ ):  
( $B$ )    if( $n = 0$ ): 0  
( $R$ )    else:  $n + \text{sum}(n - 1)$ 
```

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Summing elements of a linked list:

```
sum(L):  
(B)   if(L.isEmpty()):  
       return 0  
(R)   else:  
       return L.data + sum(L.next)
```

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Sorting elements of a linked list (insertion sort):

```
sort(L):  
  (B)    if(L.isEmpty()):  
          return L  
  else:  
    (R)    L2 = sort(L.next)  
            insert L.data into L2  
            return L2
```

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Concatenation of words (defining wv):

$$\begin{array}{ll} & \text{For all } w, v \in \Sigma^* \text{ and } a \in \Sigma : \\ (B) & \lambda v = v \\ (R) & (aw)v = a(wv) \end{array}$$

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

Length of words:

$$(B) \quad \text{length}(\lambda) = 0$$

$$(R) \quad \text{length}(aw) = 1 + \text{length}(w)$$

Programming over recursive datatypes

Recursive datatypes make recursive programming/functions easy.

Example

“Evaluation” of a propositional formula

Exercise

Exercise

Let Σ be a finite set.

Define $\text{append} : \Sigma^* \times \Sigma \rightarrow \Sigma^*$ by

$$\text{append}(w, a) = wa$$

Give a (direct) definition of append [i.e. only concatenates symbols on the left].

Exercise

Exercise

Let Σ be a finite set.

Define $\text{append} : \Sigma^* \times \Sigma \rightarrow \Sigma^*$ by

$$\text{append}(w, a) = wa$$

Give a (direct) definition of append [i.e. only concatenates symbols on the left].

?

Pitfall: Correctness of Recursive Definition

A recurrence formula is correct if the computation of any later term can be reduced to the initial values given in (B).

Example (Incorrect definition)

- Function $g(n)$ is defined recursively by

$$g(n) = g(g(n-1) - 1) + 1, \quad g(0) = 2.$$

The definition of $g(n)$ is incomplete — the recursion may not terminate:

Attempt to compute $g(1)$ gives

$$g(1) = g(g(0) - 1) + 1 = g(1) + 1 = \dots = g(1) + 1 + 1 + 1 \dots$$

When implemented, it leads to an overflow; most static analyses cannot detect this kind of ill-defined recursion.

Pitfall: Correctness of Recursive Definition

Example (continued)

However, the definition could be repaired. For example, we can add the specification specify $g(1) = 2$.

Then $g(2) = g(2 - 1) + 1 = 3$,
 $g(3) = g(g(2) - 1) + 1 = g(3 - 1) + 1 = 4$,
...

In fact, by induction ... $g(n) = n + 1$

Pitfall: Correctness of Recursive Definition

Check your base cases!

Example

Function $f(n)$ is defined by

$$f(n) = f(\lceil n/2 \rceil), \quad f(0) = 1$$

When evaluated for $n = 1$ it leads to

$$f(1) = f(1) = f(1) = \dots$$

This one can also be repaired. For example, one could specify that $f(1) = 1$.

This would lead to a constant function $f(n) = 1$ for all $n \geq 0$.

Mutual Recursion

Sometimes recursive definitions use more than one function, with each calling each other.

Example (Fibonacci, again)

Recall:

- (B) $f(0) = 0$; $f(1) = 1$,
- (R) $f(n) = f(n-1) + f(n-2)$

Mutual Recursion

Sometimes recursive definitions use more than one function, with each calling each other.

Example (Fibonacci, again)

Recall:

- (B) $f(0) = 0$; $f(1) = 1$,
- (R) $f(n) = f(n-1) + f(n-2)$

Alternative, mutually recursive definition:

- (B) $f(1) = 1$; $g(1) = 0$
- (R) $f(n) = f(n-1) + g(n-1)$
- (R) $g(n) = f(n-1)$

Mutual Recursion

Sometimes recursive definitions use more than one function, with each calling each other.

Example (Fibonacci, again)

Recall:

- (B) $f(0) = 0$; $f(1) = 1$,
- (R) $f(n) = f(n-1) + f(n-2)$

Alternative, mutually recursive definition:

- (B) $f(1) = 1$; $g(1) = 0$
- (R) $f(n) = f(n-1) + g(n-1)$
- (R) $g(n) = f(n-1)$

$$\begin{pmatrix} f(n) \\ g(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f(n-1) \\ g(n-1) \end{pmatrix}$$

Outline

- Recursion
- Recursive Data Types
- Recursive programming
- Solving recurrences

Solving recurrences

Approaches:

- Unwinding the recurrence
- Approximating with big-O
- The Master Theorem

NB

Each approach gives an informal “solution”: ideally one should prove a solution is correct (using e.g. induction).

Examples

Example (Unwinding)

$$f(0) = 1 \quad f(n) = 2f(n-1)$$

Examples

Example (Unwinding)

$$f(0) = 1 \quad f(n) = 2f(n-1)$$

Unwinding:

$$\begin{aligned} f(n) &= 2f(n-1) \\ &= 2(2f(n-2)) = 4f(n-2) \\ &= 4(2f(n-3)) = 8f(n-3) \\ &\vdots \\ &= 2^i f(n-i) \\ &\vdots \\ &= 2^n f(0) = 2^n \end{aligned}$$

Examples

Example (Unwinding)

$$f(1) = 0 \quad f(n) = 1 + f\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

Examples

Example (Unwinding)

$$f(1) = 0 \quad f(n) = 1 + f\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

Unwinding:

$$\begin{aligned} f(n) &= 1 + f(n/2) \\ &= 1 + (1 + f(n/4)) = 2 + f(n/4) \\ &= 2 + (1 + f(n/8)) \\ &\quad \vdots \\ &= i + f(n/2^i) \\ &\quad \vdots \\ &= \log(n) + f(0) = \log(n) \end{aligned}$$

Examples

Example (Approximating with big-O)

$$f(0) = 1 \quad f(1) = 1 \quad f(n) = f(n-1) + f(n-2)$$

Examples

Example (Approximating with big-O)

$$f(0) = 1 \quad f(1) = 1 \quad f(n) = f(n-1) + f(n-2)$$

Assuming $f(n)$ is increasing:

$$f(n-2) \leq f(n-1)$$

Examples

Example (Approximating with big-O)

$$f(0) = 1 \quad f(1) = 1 \quad f(n) = f(n-1) + f(n-2)$$

Assuming $f(n)$ is increasing:

$$f(n-2) \leq f(n-1)$$

so:

$$f(n) \leq 2f(n-1)$$

Examples

Example (Approximating with big-O)

$$f(0) = 1 \quad f(1) = 1 \quad f(n) = f(n-1) + f(n-2)$$

Assuming $f(n)$ is increasing:

$$f(n-2) \leq f(n-1)$$

so:

$$f(n) \leq 2f(n-1)$$

so (by unwinding):

$$f(n) \leq 2^n$$

Examples

Example (Approximating with big-O)

$$f(0) = 1 \quad f(1) = 1 \quad f(n) = f(n-1) + f(n-2)$$

Assuming $f(n)$ is increasing:

$$f(n-2) \leq f(n-1)$$

so:

$$f(n) \leq 2f(n-1)$$

so (by unwinding):

$$f(n) \leq 2^n$$

so:

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

Master Theorem

The following result covers many recurrences that arise in practice (e.g. divide-and-conquer algorithms)

Theorem

Suppose

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

where $f(n) \in \Theta(n^c(\log n)^k)$.

Let $d = \log_b(a)$. Then:

Case 1: *If $c < d$ then $T(n) = O(n^d)$*

Case 2: *If $c = d$ then $T(n) = O(n^c(\log n)^{k+1})$*

Case 3: *If $c > d$ then $T(n) = O(f(n))$*

Master Theorem: Examples

Example (Master Theorem)

$$T(n) = T\left(\frac{n}{2}\right) + n^2, \quad T(1) = 1$$

Master Theorem: Examples

Example (Master Theorem)

$$T(n) = T\left(\frac{n}{2}\right) + n^2, \quad T(1) = 1$$

Here $a = 1$, $b = 2$, $c = 2$, $k = 0$ and $d = 0$. So we have Case 3 and the solution is

$$T(n) = O(n^c) = O(n^2)$$

Master Theorem: Examples

Example (Master Theorem)

Mergesort has

$$T(n) = 2T\left(\frac{n}{2}\right) + (n - 1)$$

for the number of comparisons.

Master Theorem: Examples

Example (Master Theorem)

Mergesort has

$$T(n) = 2T\left(\frac{n}{2}\right) + (n - 1)$$

for the number of comparisons.

Here $a = b = 2$, $c = 1$, $k = 0$ and $d = 1$. So we have Case 2, and the solution is

$$T(n) = O(n^c \log(n)) = O(n \log(n))$$

Master Theorem: Examples

Example (Master Theorem)

Unwinding example:

$$T(1) = 0 \quad T(n) = 1 + T(\lfloor \frac{n}{2} \rfloor)$$

Master Theorem: Examples

Example (Master Theorem)

Unwinding example:

$$T(1) = 0 \quad T(n) = 1 + T(\lfloor \frac{n}{2} \rfloor)$$

Here $a = 1$, $b = 2$, $c = 0$, $k = 0$, and $d = 0$. So we have Case 2, and the solution is

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

The Master Theorem: Pitfalls

NB

- *a, b, c, k have to be constants (not dependent on n).*
- *Only one recursive term.*
- *Recursive term is of the form $T(n/b)$, not $T(n - b)$.*
- *Solution is only an asymptotic bound.*

Examples

The Master theorem does not apply to any of these:

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

$$T(n) = T(n/5) + T(7n/10) + n$$

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

The Master Theorem: Linear differences

NB

The Master Theorem applies to recurrences where $T(n)$ is defined in terms of $T(n/b)$; not in terms of $T(n-1)$.

However, the following is a consequence of the Master Theorem:

Theorem

Suppose

$$T(n) = a \cdot T(n-1) + bn^k$$

Then

$$T(n) = \begin{cases} O(n^{k+1}) & \text{if } a = 1 \\ O(a^n) & \text{if } a > 1 \end{cases}$$

Exercise

Exercise

Solve $T(n) = 3^n T\left(\frac{n}{2}\right)$ with $T(1) = 1$

Exercise

Exercise

Solve $T(n) = 3^n T\left(\frac{n}{2}\right)$ with $T(1) = 1$

?