

CSE 15

Discrete Mathematics

Lecture 20 – Mathematical Induction (4)



Announcement

- ▶ HW #9 (out on 11/26, Mon)
 - Due **5pm** 12/04 (Tues) with 1 extra day of re-submission.
- ▶ Reading assignment
 - Ch.6.1 – 6.3 of textbook

Structural Induction

Definition: To prove a property of the elements of a recursively defined set, we use *structural induction*.

BASIS STEP: Show that the result holds for all elements specified in the basis step of the recursive definition.

RECURSIVE STEP: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

Full Binary Trees

Definition: The *height* $h(T)$ of a full binary tree T is defined recursively as follows:

- BASIS STEP: The height of a full binary tree T consisting of only a root r is $h(T) = 0$.
- RECURSIVE STEP: If T_1 and T_2 are full binary trees, then the full binary tree $T = T_1 \cdot T_2$ has height $h(T) = 1 + \max(h(T_1), h(T_2))$.

Full Binary Trees

- ▶ The number of vertices $n(T)$ of a full binary tree T satisfies the following recursive formula:
 - **BASIS STEP:** The number of vertices of a full binary tree T consisting of only a root r is $n(T) = 1$.
 - **RECURSIVE STEP:** If T_1 and T_2 are full binary trees, then the full binary tree $T = T_1 \cdot T_2$ has the number of vertices $n(T) = 1 + n(T_1) + n(T_2)$.

Structural Induction and Binary Trees

Theorem: If T is a full binary tree, then $n(T) \leq 2^{h(T)+1} - 1$.

Proof: Use structural induction.

- **BASIS STEP:** The result holds for a full binary tree consisting only of a root, $n(T) = 1$ and $h(T) = 0$.

Hence, $n(T) = 1 \leq 2^{0+1} - 1 = 1$.

Structural Induction and Binary Trees

- RECURSIVE STEP: Assume $n(T_1) \leq 2^{h(T_1)+1} - 1$ and also $n(T_2) \leq 2^{h(T_2)+1} - 1$ whenever T_1 and T_2 are full binary trees.

$$\begin{aligned} n(T) &= 1 + n(T_1) + n(T_2) && \text{(by recursive formula of } n(T)) \\ &\leq 1 + (2^{h(T_1)+1} - 1) + (2^{h(T_2)+1} - 1) && \text{(by inductive hypothesis)} \\ &\leq 2 \cdot \max(2^{h(T_1)+1}, 2^{h(T_2)+1}) - 1 \\ &= 2 \cdot 2^{\max(h(T_1), h(T_2))+1} - 1 && (\max(2^x, 2^y) = 2^{\max(x,y)}) \\ &= 2 \cdot 2^{h(T)} - 1 && \text{(by recursive definition of } h(T)) \\ &= 2^{h(T)+1} - 1 \end{aligned}$$



Recursive Algorithms (Ch. 5.4)

- ▶ Recursive Algorithms
- ▶ Proving Recursive Algorithm Correctness
- ▶ Merge Sort

Recursive Algorithms

Definition: An algorithm is called *recursive* if it solves a problem by reducing it to an instance of the same problem with smaller input.

>Terminal condition/value need to be known.



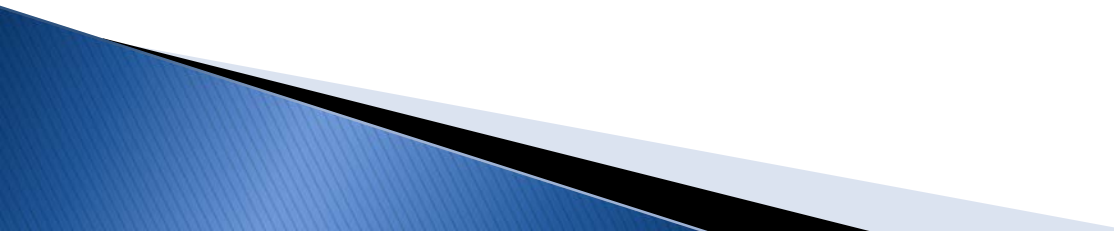
Recursive Factorial Algorithm

Example: Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

- ▶ **Solution:** Use the recursive definition of the factorial function.

```
procedure factorial( $n$ : nonnegative integer)
if  $n = 0$  then return 1
else return  $n \cdot \textit{factorial}(n - 1)$ 
{output is  $n!$ }
```

Merge Sort

- ▶ *Merge Sort* works by iteratively splitting a list (with an even number of elements) into two sublists of equal length until each sublist has one element.
 - ▶ Each sublist is represented by a balanced binary tree.
 - ▶ At each step a pair of sublists is successively merged into a list with the elements in increasing order. The process ends when all the sublists have been merged.
 - ▶ The succession of merged lists is represented by a binary tree.
- 

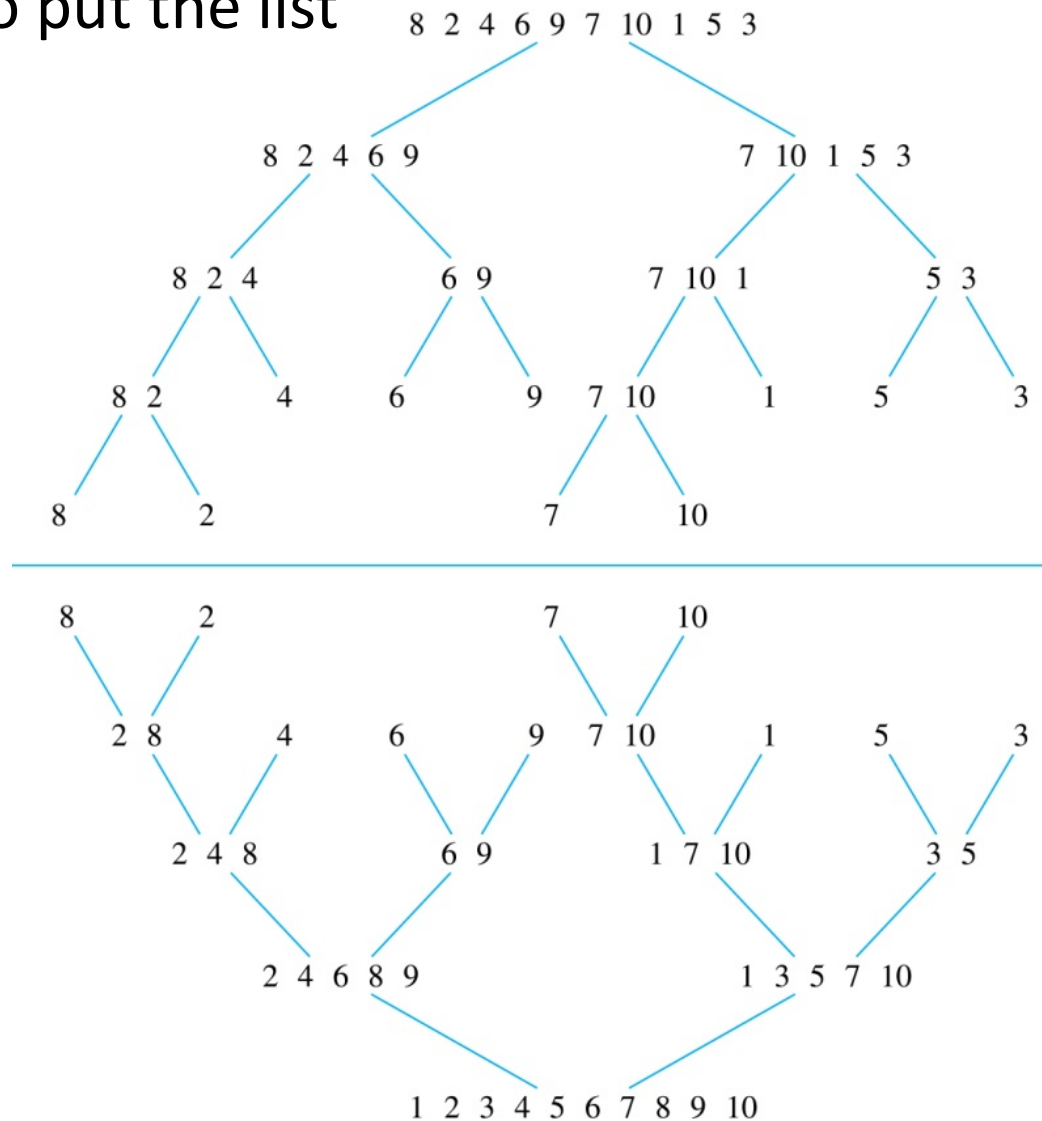
Merge Sort

Example: Use merge sort to put the list

8,2,4,6,9,7,10,1,5,3

into increasing order.

Solution:



Recursive Merge Sort

Example: Construct a recursive merge sort algorithm.

Solution: Begin with the list of n elements L .

```
procedure mergesort( $L = a_1, a_2, \dots, a_n$ )  
if  $n > 1$  then  
     $m := \lfloor n/2 \rfloor$   
     $L_1 := a_1, a_2, \dots, a_m$   
     $L_2 := a_{m+1}, a_{m+2}, \dots, a_n$   
     $L := \textit{merge}(\textit{mergesort}(L_1), \textit{mergesort}(L_2))$   
{ $L$  is now sorted in increasing order}
```

continued →

Recursive Merge Sort

- ▶ Subroutine *merge*, which merges two sorted lists.

procedure *merge*(L_1, L_2 :sorted lists)

L := empty list

while L_1 and L_2 are both nonempty

 remove smaller of first elements of L_1 and L_2 from its list;

 put at the right end of L

if this removal makes one list empty

then remove all elements from the other list and append
 them to L

return L { L is the merged list in increasing order}

Complexity of Merge: Two sorted lists with m elements and n elements can be merged into a sorted list using no more than $m + n - 1$ comparisons.

Merging Two Lists

Example: Merge the two lists 2,3,5,6 and 1,4.

Solution:

TABLE 1 Merging the Two Sorted Lists 2, 3, 5, 6 and 1, 4.

<i>First List</i>	<i>Second List</i>	<i>Merged List</i>	<i>Comparison</i>
2 3 5 6	1 4		$1 < 2$
2 3 5 6	4	1	$2 < 4$
3 5 6	4	1 2	$3 < 4$
5 6	4	1 2 3	$4 < 5$
5 6		1 2 3 4	
		1 2 3 4 5 6	