



# Discrete Structures

(CKC111)

*Week 11 & Week 12*



# Induction and Recursion

## Recursive Algorithms

# Section Summary



- ✓ Recursive Algorithms
- ✓ Proving Recursive Algorithms Correct
- ✓ Merge Sort

# Recursive Algorithms

# 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.

For the algorithm to terminate, the instance of the problem must eventually be reduced to some initial case for which the solution is known.

# Algorithm 1: 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  $n \cdot \text{factorial}(n - 1)$ 
  {output is  $n!$ }
```

# Algorithm 1: Recursive Factorial Algorithm

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

To help understand – trace the steps, e.g., compute  $4!$ .

$$4! = 4 \cdot 3!$$

$$3! = 3 \cdot 2!$$

$$2! = 2 \cdot 1!$$

$$1! = 1 \cdot 0!.$$

Inserting the value of  $0! = 1$

Working back through the steps,

$$1! = 1 \cdot 1 = 1$$

$$2! = 2 \cdot 1! = 2 \cdot 1 = 2$$

$$3! = 3 \cdot 2! = 3 \cdot 2 = 6$$

$$4! = 4 \cdot 3! = 4 \cdot 6 = 24.$$

## Algorithm 2: Recursive Exponentiation Algorithm



**Example:** Give a recursive algorithm for computing  $a^n$ , where  $a$  is a nonzero real number and  $n$  is a nonnegative integer.

**Solution:** Use the recursive definition of  $a^n$ .

```
procedure power(a: nonzero real number, n: nonnegative integer)
if  $n = 0$  then return 1
else return  $a \cdot \text{power}(a, n - 1)$ 
{output is  $a^n$ }
```

## A Recursive Algorithm for Computing $a^n$ .

Student  
Work

Trace the steps when the input is  $a = 2, n = 5$

$$2^5 = 2 \cdot 2^4$$

$$2^4 = 2 \cdot 2^3$$

$$2^3 = 2 \cdot 2^2$$

$$2^2 = 2 \cdot 2^1$$

$$2^1 = 2 \cdot 2^0$$

Inserting the value of  $2^0 = 1$

Working back through the steps,

$$2^1 = 2 \cdot 1 = 2$$

$$2^2 = 2 \cdot 2 = 4$$

$$2^3 = 2 \cdot 4 = 8$$

$$2^4 = 2 \cdot 8 = 16$$

$$2^5 = 2 \cdot 16 = 32$$

## Algorithm 3: Recursive GCD Algorithm



**Example:** Give a recursive algorithm for computing the greatest common divisor (GCD) of two nonnegative integers  $a$  and  $b$  with  $a < b$ .

**Solution:** Use the reduction  $\text{gcd}(a, b) = \text{gcd}(b \bmod a, a)$  and the condition  $\text{gcd}(0, b) = b$  when  $b > 0$ .

```
procedure gcd( $a, b$ : nonnegative integers with  $a < b$ )  
if  $a = 0$  then return  $b$   
else return gcd( $b \bmod a, a$ )  
{output is gcd( $a, b$ )}
```

## A Recursive Algorithm for Computing $\gcd(a, b)$ .

Student  
Work

Trace the steps when the input is  $a = 5, b = 8$ .

$$\gcd(5, 8) = \gcd(8 \bmod 5, 5) = \gcd(3, 5).$$

$$\gcd(3, 5) = \gcd(5 \bmod 3, 3) = \gcd(2, 3),$$

$$\gcd(2, 3) = \gcd(3 \bmod 2, 2) = \gcd(1, 2),$$

$$\gcd(1, 2) = \gcd(2 \bmod 1, 1) = \gcd(0, 1).$$

$\gcd(0, 1)$  it uses the first step with  $a = 0$  to find that  $\gcd(0, 1) = 1$ .

Consequently, the algorithm finds that  $\gcd(5, 8) = 1$ .

## Algorithm 4: Recursive Linear Search Algorithm



**Example:** Express the linear search algorithm as a recursive procedure.

**Solution:** To search for the first occurrence, of  $x$  in the sequence  $a_1, a_2, \dots, a_n$ , at the  $i^{th}$  step of the algorithm,  $x$  and  $a_i$  are compared. If equals, returns  $i$ . Otherwise, the search for the first occurrence of  $x$  is reduced to a search in a sequence with one fewer element, namely, the sequence  $a_{i+1}, \dots, a_n$ . When  $x$  never found, returns 0.

```
procedure search( $i, j, x$ :  $i, j, x$  integers,  $1 \leq i \leq j \leq n$ )  
  if  $a_i = x$  then  
    return  $i$   
  else if  $i = j$  then  
    return 0  
  else  
    return search( $i + 1, j, x$ )  
{output is the location of  $x$  in  $a_1, a_2, \dots, a_n$  if it appears; otherwise,  
it is 0}
```

## Algorithm 5: Recursive Binary Search Algorithm



**Example:** Construct a recursive version of a binary search algorithm.

**Solution:** To locate  $x$  in the sequence  $a_1, a_2, \dots, a_n$ , in increasing order. Begin by comparing  $x$  with the middle term,  $a_{\lfloor (n+1)/2 \rfloor}$ . If equals, algorithm will terminate and return the location of this term. Otherwise, it reduce to a search the search to a smaller search sequence.

```
procedure binary search( $i, j, x$ :  $i, j, x$  integers,  $1 \leq i \leq j \leq n$ )  
 $m := \lfloor (i + j)/2 \rfloor$   
if  $x = a_m$  then  
    return  $m$   
else if ( $x < a_m$  and  $i < m$ ) then  
    return binary search( $i, m - 1, x$ )  
else if ( $x > a_m$  and  $j > m$ ) then  
    return binary search( $m + 1, j, x$ )  
else return 0  
{output is the location of  $x$  in  $a_1, a_2, \dots, a_n$  if it appears; otherwise, it is 0}
```

# Proving Recursive Algorithms Correct

# Proving Recursive Algorithms Correct



Both mathematical and strong induction are useful techniques to show that recursive algorithms always produce the correct output.

**Example:** Prove that the algorithm for computing the powers of real numbers is correct.

```
procedure power( $a$ : nonzero real number,  $n$ : nonnegative integer)
if  $n = 0$  then return 1
else return  $a \cdot \text{power}(a, n - 1)$ 
{output is  $a^n$ }
```

**Solution:** Use mathematical induction on the exponent  $n$ .

**Basis step:** If  $n=0$ ,  $\text{power}(a, 0) = 1$ . This is correct because  $a^0 = 1$  for every nonzero real number  $a$ .

**Inductive step:** The inductive hypothesis is that  $\text{power}(a, k) = a^k$ , for all  $a \neq 0$ . the inductive hypothesis is the algorithm correctly computes  $a^k$ . Next, we prove the inductive hypothesis is true, then the algorithm correctly computes  $a^{k+1}$ , since  $\text{power}(a, k + 1) = a \cdot \text{power}(a, k) = a \cdot a^k = a^{k+1}$ .

# Proving Recursive Algorithms Correct



Both mathematical and strong induction are useful techniques to show that recursive algorithms always produce the correct output.

**Example:** Prove that the algorithm for computing the powers of real numbers is correct.

```
procedure power( $a$ : nonzero real number,  $n$ : nonnegative integer)
  if  $n = 0$  then return 1
  else return  $a \cdot \text{power}(a, n - 1)$ 
  {output is  $a^n$ }
```

**Solution:** Use mathematical induction on the exponent  $n$ .

**Basis step:** If  $n=0$ ,  $\text{power}(a, 0) = 1$ . This is correct because

**Inductive step:** The inductive hypothesis is that  $\text{power}(a, k)$  hypothesis is the algorithm correctly computes  $a^k$ . Next, then the algorithm correctly computes  $a^{k+1}$ , since  $\text{power}(a, k + 1) = a \cdot \text{power}(a, k) = a \cdot a^k = a^{k+1}$ .

We have completed the basis step and the inductive step, so we can conclude that this algorithm always computes  $a_n$  correctly when  $a \neq 0$  and  $n$  is a nonnegative integer.

# Merge Sort

## Merge Sort



*Merge Sort* works by iteratively splitting a list (with an even number of elements) into two sub-lists of equal length until each sublist has one element.

Each sublist is represented by a balanced binary tree.

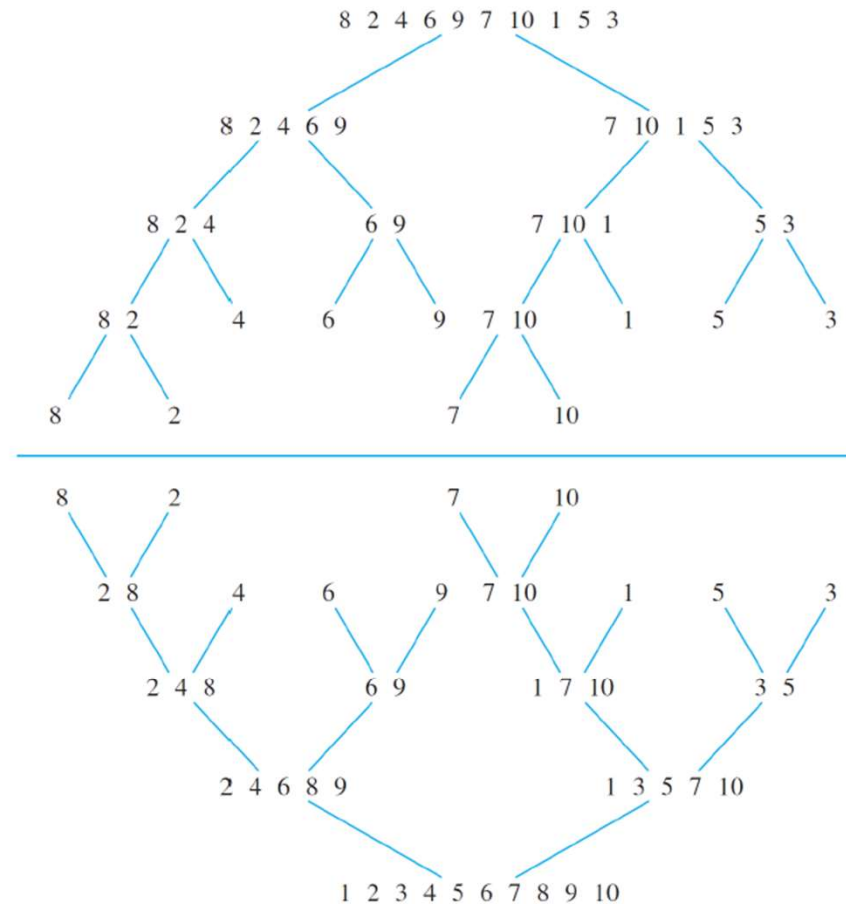
At each step a pair of sublists is successively merged into a list with the elements in increasing order. The process ends when all the sublists have been merged.

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

## Merging Two List

**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.

First List	Second List	Merged List	Comparison
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	

**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 with the elements in increasing order}

# Thank you

