CS 214
Introduction to Discrete Structures

Chapter 3
Recursion, Recurrence Relations, and Analysis of Algorithms

Mikel D. Petty, Ph.D.

Chapter sections and objectives

- 3.1 Recursive Definitions
  - Understand recursive definitions of sequences, sets, and operations
  - Write recursive definitions of sequences, sets, and operations
  - Understand how recursive algorithms execute
  - Write recursive algorithms to generate sequences
- 3.2 Recurrence Relations
  - Find closed-form solutions for recurrence relations
- 3.3 Analysis of Algorithms
  - Analyze algorithms by counting number of executions of basic operations, possibly with recurrence relations

Sample problem

You are serving on the city council's Board of Land Management, which is considering a proposal by a private contractor to manage a chemical disposal site. The material to be stored at the site degrades to inert matter at the rate of 5% per year. The contractor claims that, at this rate of stabilization, only about one-third of the original active material will remain at the end of 20 years.

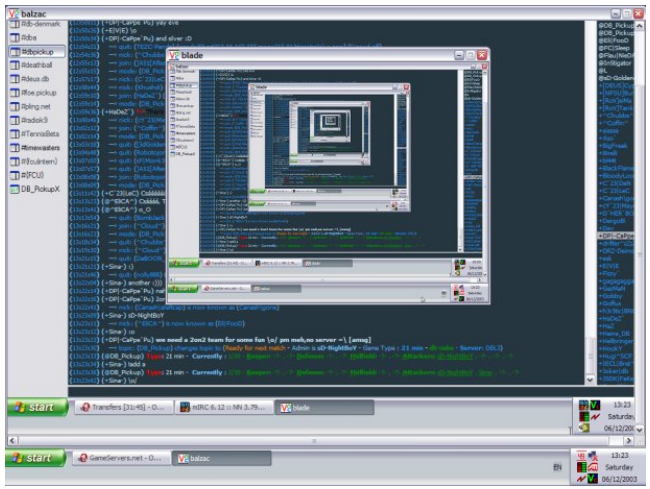
Is the contractor's estimate correct?

3.1 Recursive Definitions

Recursive definitions

- Recursive definition; definition where the object being defined appears as part of the definition
  - Possible because recursive reference is to smaller or simpler case of object being defined
  - AKA inductive definition

//www.math.ubc.ca/~jbryan/



www.ecs.soton.ac.uk

- Parts of a recursive definition
  - Basis; simple case(s) of the object defined explicitly
  - Recursive step; new case of object defined in terms of previous case(s)
- Categories of recursively defined objects
  - Sequences
  - Sets
  - Operations
  - Algorithms

Recursively defined sequences

- Sequence
  - List of objects, enumerated in order
  - Sequence S; $S(k)$ denotes object k in sequence
  - e.g., $S = 2, 4, 6, 8, \ldots, S(4) = 8$
  - Sequence may be infinite: $S(1), S(2), \ldots, S(k), \ldots$
  - Alternative notation: $S_1, S_2, \ldots, S_k, \ldots$
- Sequence S defined recursively by
  - Defining first (or first few) objects explicitly
  - Defining later objects in terms of earlier ones

Example recursively defined sequence

Sequence S is defined recursively by

1. $S(1) = 2$ basis
2. $S(n) = 2 \cdot S(n-1)$ for $n \geq 2$ recursive step

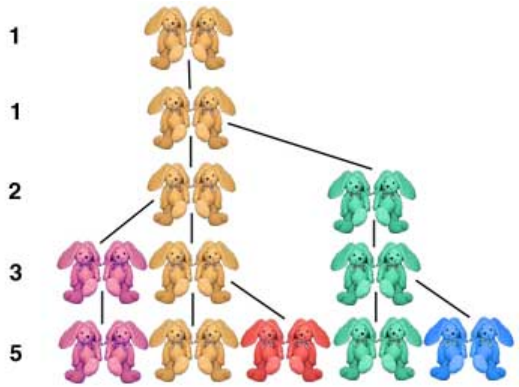
$n = 1 \quad S(1) = 2$
$n = 2 \quad S(2) = 2 \cdot S(2-1) = 2 \cdot S(1) = 2 \cdot 2 = 4$
$n = 3 \quad S(3) = 2 \cdot S(3-1) = 2 \cdot S(2) = 2 \cdot 4 = 8$
$n = 4 \quad S(4) = 2 \cdot S(4-1) = 2 \cdot S(3) = 2 \cdot 8 = 16$
$n = 5 \quad S(5) = 2 \cdot S(5-1) = 2 \cdot S(4) = 2 \cdot 16 = 32$
…

Recurrence relation; rule which defines a sequence object in terms of one (or more) earlier ones, e.g., statement 2.

Example 1

Example recursively defined sequence

- Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13, ...
- Definition
  - $F(1) = 1$, $F(2) = 1$
  - $F(n) = F(n-2) + F(n-1)$, for integer $n > 2$
- Fibonacci numbers found in nature
  - e.g., idealized rabbit breeding, actual plant structures



www.bbc.co.uk/radio4/science/media/

Alvesgaspar, RDBury

Example 2

Sequence F is defined recursively by

1. $F(1) = 1, F(2) = 1$
2. $F(n) = F(n-2) + F(n-1)$, for integer $n > 2$

$n = 1 \quad F(1) = 1$
$n = 2 \quad F(2) = 1$
$n = 3 \quad F(3) = F(1) + F(2) = 1 + 1 = 2$
$n = 4 \quad F(4) = F(2) + F(3) = 1 + 2 = 3$
$n = 5 \quad F(5) = F(3) + F(4) = 2 + 3 = 5$
$n = 6 \quad F(6) = F(4) + F(5) = 3 + 5 = 8$
$n = 7 \quad F(7) = F(5) + F(6) = 5 + 8 = 13$
$n = 8 \quad F(8) = F(6) + F(7) = 8 + 13 = 21$
…

Practice 2

Example proof of recursive sequence

Theorem

In the Fibonacci sequence

$F(n + 4) = 3 \cdot F(n + 2) - F(n)$ for all $n \geq 1$

Proof

Basis.

$n = 1 \quad F(5) = 3 \cdot F(3) - F(1) = 3 \cdot 2 - 1 = 5$

$n = 2 \quad F(6) = 3 \cdot F(4) - F(2) = 3 \cdot 3 - 1 = 8$

Inductive hypothesis.

Assume for all r, $1 \leq r \leq k$, $F(r + 4) = 3 \cdot F(r + 2) - F(r)$.

Example 3

Inductive hypothesis.
Assume for all r, $1 \leq r \leq k$, $F(r + 4) = 3 \cdot F(r + 2) - F(r)$. 1

Inductive step.
Show $F(k + 1 + 4) = 3 \cdot F(k + 1 + 2) - F(k + 1)$, 2
or $F(k + 5) = 3 \cdot F(k + 3) - F(k + 1)$. 3

From the Fibonacci sequence recurrence relation,
$F(k + 5) = F(k + 3) + F(k + 4)$. 4

By the inductive hypothesis, with $r = k - 1$ 5
$F(k + 3) = 3 \cdot F(k + 1) - F(k - 1)$.

By the inductive hypothesis, with $r = k$ 6
$F(k + 4) = 3 \cdot F(k + 2) - F(k)$.

Therefore

$$
\begin{aligned}
& F(k+5) \\
= \; & F(k+3) + F(k+4) && \text{Def of } F \\
= \; & [3 \cdot F(k+1) - F(k-1)] + [3 \cdot F(k+2) - F(k)] && \text{By IH} \\
= \; & 3[F(k+1) + F(k+2)] - [F(k-1) + F(k)] && \text{Rearrange} \\
= \; & 3 \cdot F(k+3) - F(k+1). \; \blacksquare && \text{Def of } F
\end{aligned}
$$

Recursively defined sets

- Set
  - Collection of objects
  - Set S; $s \in S$ denotes object s in set S
  - Object in set called element
  - e.g., $S = \{k, s, p\}$, $k \in S$
  - Each element unique, i.e., no repetitions in set
  - No order to elements
- Set defined recursively by
  - Defining first (or first few) elements explicitly
  - Defining later elements in terms of elements already in set

Example recursively defined set

A finite set of distinct symbols is an alphabet.
String; sequence of symbols from some alphabet.

Example alphabet: statement letters, logical connective symbols, and parentheses.
Example set S: set of strings from the example alphabet that are syntactically correct propositional wffs.

Set of wffs S is defined recursively by

1. Any statement letter is a wff, i.e., an element of S
2. If P and Q are wffs, then so are $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, (P'), and $(P \leftrightarrow Q)$
3. If (P) is a wff, then so is P.

?

Example 5

Is $((A \wedge B) \rightarrow C')' \in S$, i.e., is it a wff?

$A \;\; B \;\; C$	Rule 1
$(A \wedge B) \;\; (C')$	Rule 2
$(A \wedge B) \;\; C'$	Rule 3
$((A \wedge B) \rightarrow C')$	Rule 2
$(((A \wedge B) \rightarrow C')')$	Rule 2
$((A \wedge B) \rightarrow C')'$	Rule 3

$?_1$

Building up $(((A \wedge B) \rightarrow (C'))')$,
then applying rule 3 to get $((A \wedge B) \rightarrow C')'$ is incorrect; $?_2$
rule 3 only removes outer parentheses.

Recursively defined set example

The set of all finite-length strings of symbols consisting of symbols from ("over") alphabet A is denoted A^*.

Set A^* is defined recursively by

1. Empty string λ (string with no symbols) $\in A^*$.
2. Any single symbol $\in A$ is also $\in A^*$.
3. If x and y are strings $\in A^*$, then xy, the concatenation of strings x and y, is $\in A^*$.

e.g., if $x = 1011$ and $y = 001$,
then $xy = 1011001$, $yx = 0011011$, $yx\lambda x = 00110111011$

Example 6 and Practice 6

Example recursively defined set

Set of programming language identifiers;
alphanumeric strings, arbitrary length, begin with letter.

Set of identifiers is defined recursively by

1. A single letter is an identifier.
2. If A is an identifier, then so is the concatenation of A and any letter or digit.

Examples: `quarter3`, `customerlist`, `n1248`, `me2`

- Backus-Naur Form (BNF)
  - Formal syntax notation
  - Used to define programming language syntax, e.g., originally ALGOL
  - Allows recursive definitions
- BNF rules
  - Items in angle brackets < > defined in terms of other items, may be recursive
  - Items not in angle brackets not further defined
  - Vertical bar | denotes choice
  - Other BNF rules not needed for example
  - For more detailed description, see https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form

BNF definition for identifiers

<letter> ::= a | b | c | … | z
<digit> ::= 0 | 1 | 2 | … | 9 **?**$_1$
<identifier> ::= <letter> | <identifier> <letter> | <identifier> <digit> **?**$_2$

Assembling identifier me2 using this definition:

<identifier>
<identifier><digit>
<identifier>2
<identifier><letter>2
<identifier>e2
<letter>e2
me2

Example 7

Structural induction

- Background
  - Connection between recursion and induction
  - First and second induction apply to integers
  - Structural induction applies to recursively defined sets
- Proof method

Given recursively defined set S and property $P(x)$ for $x \in S$, if these two conditions are true

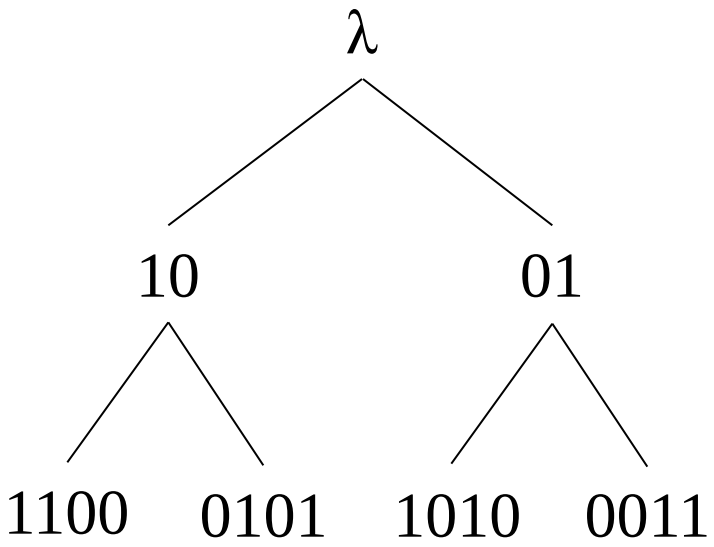
1. $P(x)$ is true for all elements $x \in S$ defined in the basis
2. $P(x)$ for $x \in S \rightarrow P(y)$ for $y \in S$ when y is constructed from x using the recursive step

then $P(x)$ is true for every $x \in S$.

Structural induction example

Set S is defined recursively by

1. Empty string λ (string with no symbols) $\in S$.
2. If string $x \in S$, then $1x0 \in S$ and $0x1 \in S$.



Note that not all bit strings are possible in S, e.g., 1001.

Example 8

Theorem
If string $s \in S$,
then s contains an equal number of 0s and 1s.

Proof (by structural induction)
Basis. λ has 0 0s and 0 1s.

Inductive hypothesis. Assume string $x \in S$
and has k 0s and k 1s.

Inductive step. Show that any string y constructed from x
has an equal number of 0s and 1s.
Case 1. $y = 1x0$. y has $k + 1$ 0s and $k + 1$ 1s.
Case 2. $y = 0x1$. y has $k + 1$ 0s and $k + 1$ 1s.

Therefore, every string $s \in S$ has an equal number
of 0s and 1s. ■

Recursively defined operations

- Operation
  - Action or procedure applied to objects
  - e.g., addition $1 + 2$; exponentiation a^n
- Operation defined recursively by
  - Explicitly defining action for first (or first few) cases
  - Defining larger cases in terms of smaller ones

Recursively defined operation example

Exponentiation, a^n where a nonzero real, n nonnegative integer, defined recursively as:

1. $a^0 = 1$
2. $a^n = (a^{n-1}) \cdot a$ for $n \geq 1$

$$
\begin{aligned}
2^4 &= (2^3) \cdot 2 \\
&= ((2^2) \cdot 2) \cdot 2 \\
&= (((2^1) \cdot 2) \cdot 2) \cdot 2 \\
&= ((((2^0) \cdot 2) \cdot 2) \cdot 2) \cdot 2 \\
&= 1 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \\
&= 16
\end{aligned}
$$

Example 9

Recursively defined operation example

Factorial, non-recursive definition

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1 & \text{if } n > 0 \end{cases}$$

Factorial, recursive definition

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$$

Examples

$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$

$4! = 4 \cdot 3!$

$3! = 3 \cdot 2 \cdot 1 = 6$

$0! = 1$

Recursively defined operation example

Disjunction of n statement letters
can be defined recursively as:

1. $A_1 \vee A_2$ defined as in §1.1 (truth table)
2. $A_1 \vee \ldots \vee A_n = (A_1 \vee \ldots \vee A_{n-1}) \vee A_n$ for $n > 2$

$A_1 \vee A_2 \vee A_3 \vee A_4$ undefined in §1.1

$= (A_1 \vee A_2 \vee A_3) \vee A_4$

$= ((A_1 \vee A_2) \vee A_3) \vee A_4$ defined in §1.1

Recursively defined algorithms

- Algorithm
  - Ordered sequence of unambiguous, executable, and terminating steps that solve a problem
- Algorithm defined recursively by
  - Handling smallest parameter value(s) directly; base case
  - Handling larger values by “calling” same algorithm on smaller values; recursive case

Recursively defined algorithm example

Sequence S is defined recursively by

1. $S(1) = 2$ basis
2. $S(n) = 2 \cdot S(n-1)$ for $n \geq 2$ recursive step

$n = 1 \quad S(1) = 2$

$n = 2 \quad S(2) = 2 \cdot S(2-1) = 2 \cdot S(1) = 2 \cdot 2 = 4$

$n = 3 \quad S(3) = 2 \cdot S(3-1) = 2 \cdot S(2) = 2 \cdot 4 = 8$

$n = 4 \quad S(4) = 2 \cdot S(4-1) = 2 \cdot S(3) = 2 \cdot 8 = 16$

$n = 5 \quad S(5) = 2 \cdot S(5-1) = 2 \cdot S(4) = 2 \cdot 16 = 32$

…

Sequence S can be generated by both iterative and recursive algorithms.

Example 1

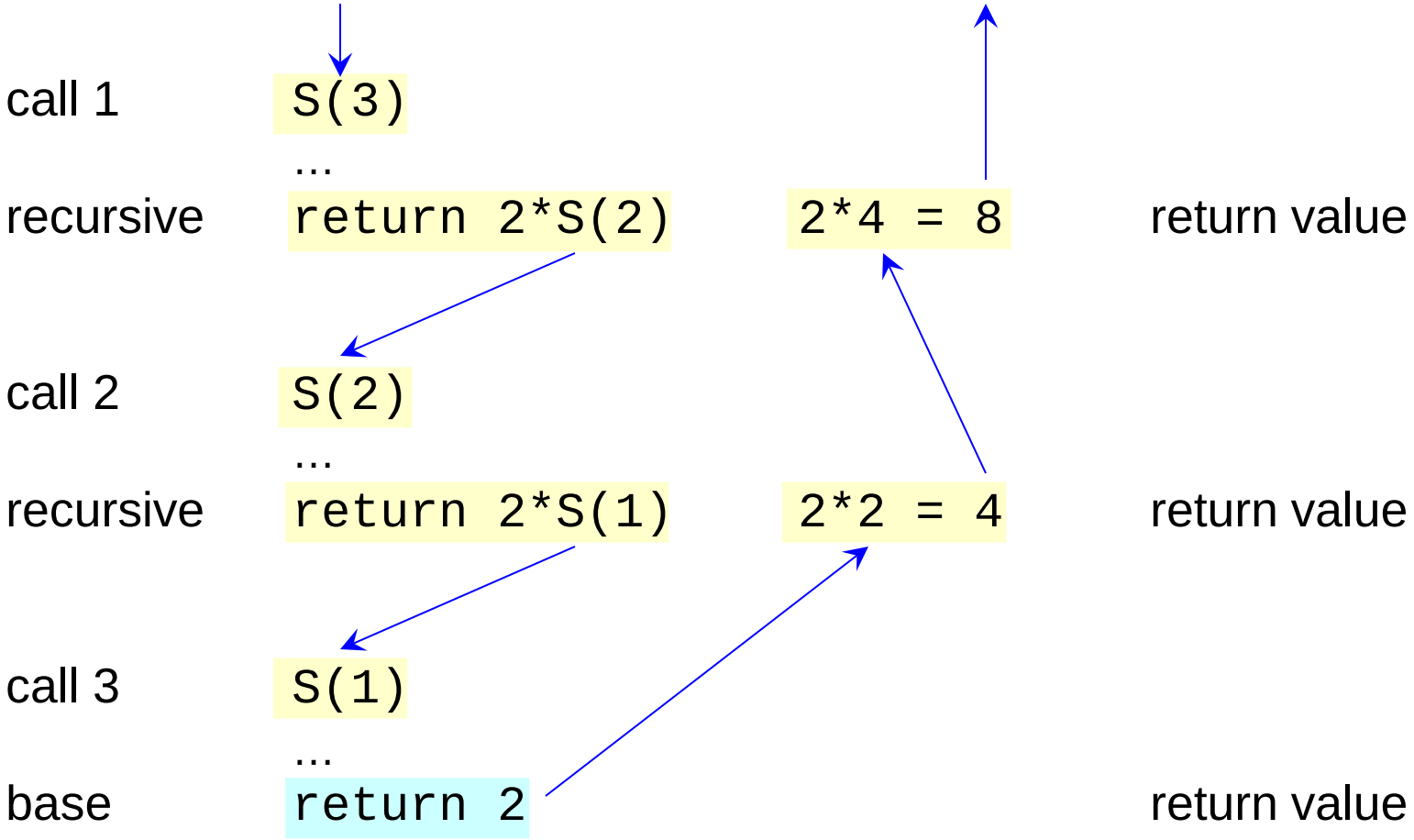
```
Algorithm S(integer n)
// Iteratively computes the value S(n) for sequence S of Example 1.
Local variables:
integer i                    // loop index
integer CurrentValue         // current value of function S
    if n = 1 then
        return 2
    else
        i = 2
        CurrentValue = 2
        while i <= n do
            CurrentValue = 2*CurrentValue
            i = i + 1
        end while
        // CurrentValue now has the value S(n)
        return CurrentValue
    end if
end function S
```

?

Algorithm *S*(integer *n*)
// Recursively computes the value *S*(*n*) for sequence *S* of Example 1.

```
    if   n = 1 then
         return 2
    else
         return 2*S(n – 1)
    end if

end function S
```



Recursively defined algorithm example

Sequence T is defined recursively by

1. $T(1) = 1$
2. $T(n) = T(n - 1) + 3$ for $n \geq 2$

```
Algorithm T(integer n)
// Recursively computes the value T(n) for sequence S of Practice 11.

    if n = 1 then
        return 1
    else
        return T(n – 1) + 3
    end if

end function T
```

Practice 9

Recursively defined algorithm example

Multiplication, $m \cdot n$ where m, n are positive integers:

1. $m \cdot 1 = m$
2. $m \cdot n = m \cdot (n - 1) + m$ for $n \geq 2$

```
Algorithm Product(integer m; integer n)
// Recursively computes the product of m and n.

    if   n = 1 then
         return m
    else
         return Product(m, n – 1) + m
    end if

end function Product
```

Example 11

Example recursively defined algorithm

Sorting; put an unordered list of items into order.
Many sorting algorithms exist.

Unsorted
Bravo
Foxtrot
Delta
Alpha
Echo
Charlie

Sorted
Alpha
Bravo
Charlie
Delta
Echo
Foxtrot

Selection sort:
1. Find largest value in list
2. Move to last position of list
3. Repeat on list minus last position

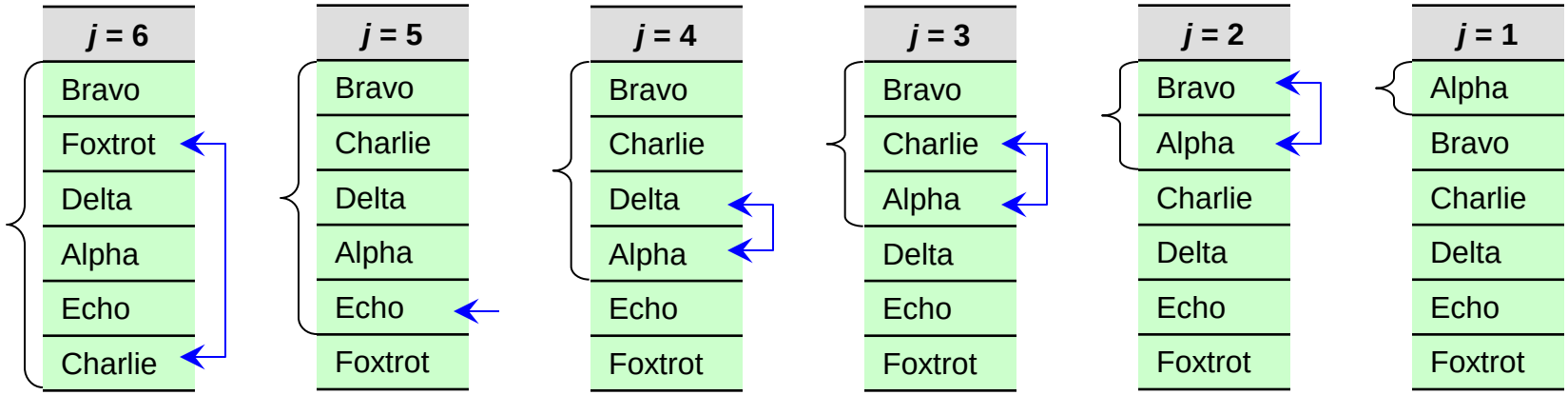
Algorithm *SelectionSort*(list *L*; integer *j*) ?

// Recursively sort items from 1 to *j* in list *L* into ascending order.

```
    if   j = 1 then                                                    5
        sort is complete, write out sorted list L
    else
        find the index of the maximum item i in L between 1 and j inclusive   2
        exchange L[i] and L[j]                                         3
        SelectionSort(L, j – 1)                                        4
    end if

end function SelectionSort
```

1



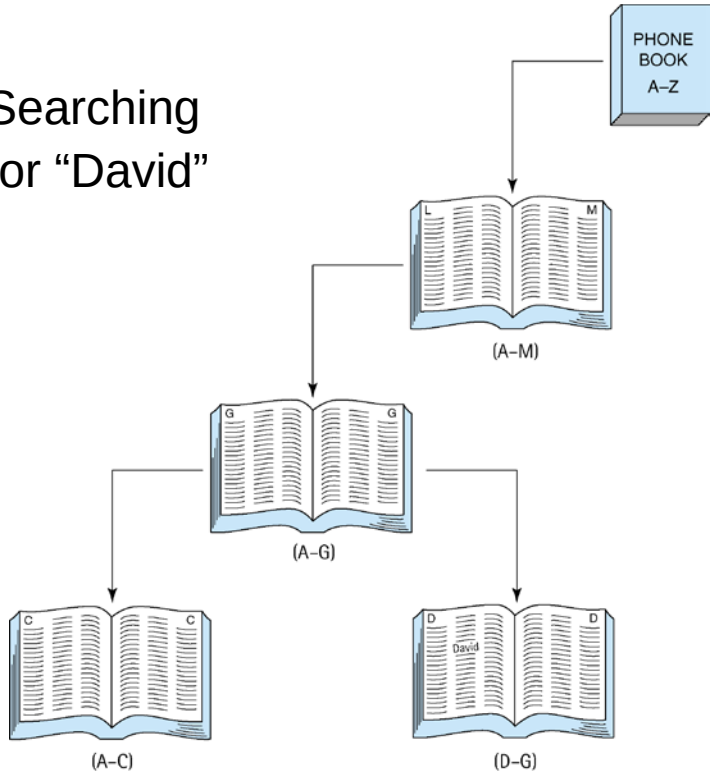
Example selection sort implementation

```
> SelectionSort <- function(L, j) {
+   if (j == 1) {
+     return(L)
+   } else {
+     i <- which(L[1:j] == max(L[1:j]))
+     L <- replace(L, c(i, j), L[c(j, i)])
+     SelectionSort(L, j - 1)
+   }
+ }
>
> (unsorted <- c("Bravo", "Foxtrot", "Delta", "Alpha", "Echo", "Charlie"))
[1] "Bravo"   "Foxtrot" "Delta"   "Alpha"   "Echo"    "Charlie"
>
> (sorted   <- SelectionSort(unsorted, length(unsorted)))
[1] "Alpha"   "Bravo"   "Charlie" "Delta"   "Echo"    "Foxtrot"
```

Example recursively defined algorithm

Binary search; search an ordered list of items for a particular "target" item.

Searching for "David"

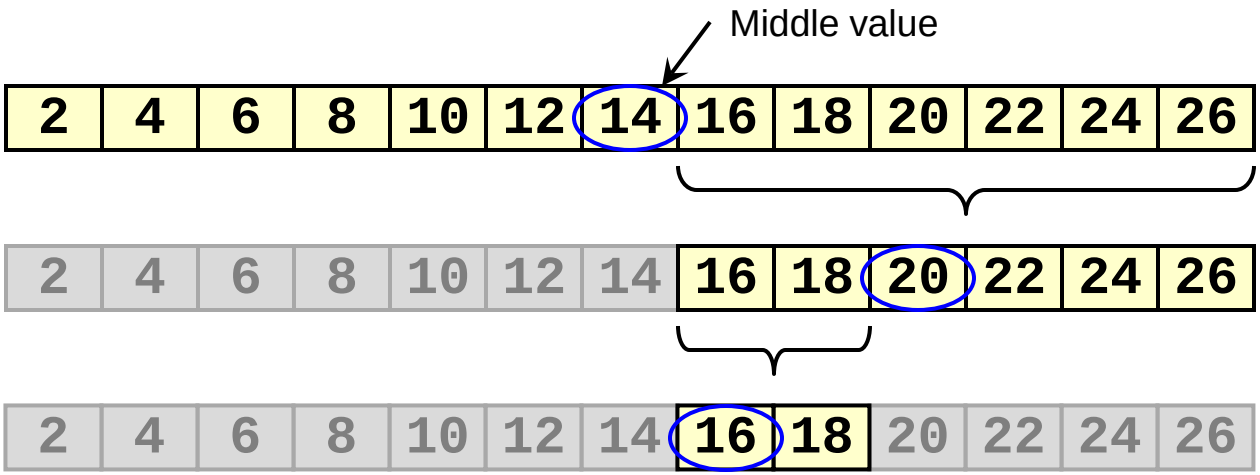


Binary search:

1. Divide list in half, compare target to list middle value
2. If target less than middle value, recursively search first half of list
3. If target greater than middle value, recursively search second half of list

Example 13

Binary search example



target 16

16 > 14, search second half of list

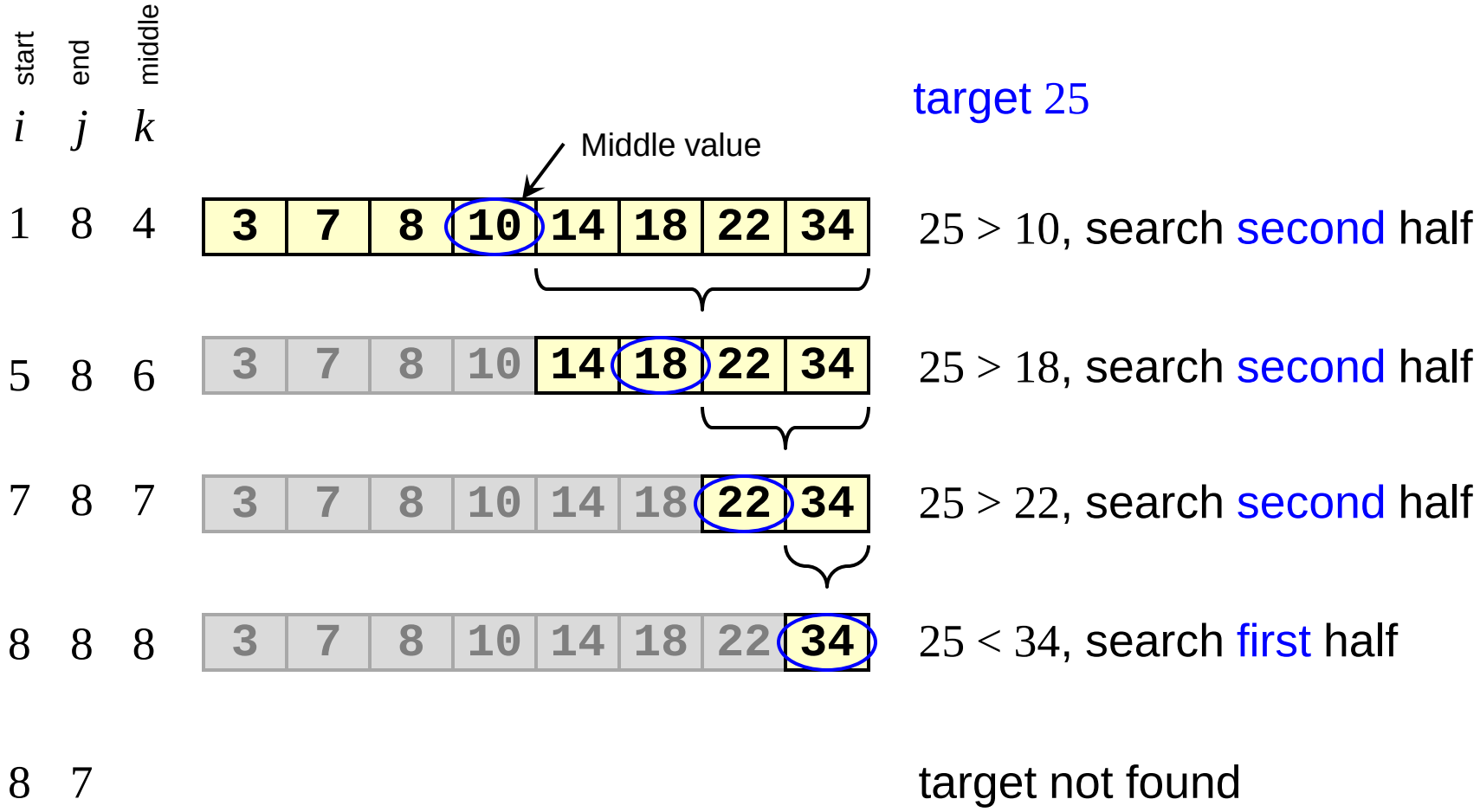
16 < 20, search first half of list

16 = 16, target found

```
Algorithm BinarySearch(list L; integer i; integer j; itemtype x))                1
// Searches sorted list L from L[i] to L[j] for item x.

    if i > j then
        write("not found")
    else
        find the index k of the middle item of the list L[i]–L[j]                2
        if x = L[k] then          // target = middle value
            write("found")
        else
            if x < L[k] then      // target < middle value
                BinarySearch(L, i, k – 1, x)                                     3
            else                  // target > middle value
                BinarySearch(L, k + 1, j, x)                                     4
            end if
        end if
    end if
end function BinarySearch
```

Binary search example



Example 14

Example binary search implementation (1 of 2)

```
> BinarySearch <- function(L, i, j, x) {
+   cat()
+   if (i > j) {
+     return("not found")
+   } else {
+     k <- trunc((i + j) / 2)
+     cat("i=", i, "j=", j, "k=", k, x, "compared to", L[k], "\n")
+     if (x == L[k]) {
+       return("found")
+     } else {
+       if (x < L[k]) {
+         return(BinarySearch(L, i, k - 1, x))
+       } else {
+         return(BinarySearch(L, k + 1, j, x))
+       }
+     }
+   }
+ }
```

Example binary search implementation (2 of 2)

```
> L <- c("Boston", "Charlotte", "Indianapolis", "Norfolk", "Oakland",
"Seattle", "Washington")
>
> (BinarySearch(L, 1, 7, "Oakland"))
i= 1 j= 7 k= 4 Oakland compared to Norfolk
i= 5 j= 7 k= 6 Oakland compared to Seattle
i= 5 j= 5 k= 5 Oakland compared to Oakland
[1] "found"
>
> (BinarySearch(L, 1, 7, "Chicago"))
i= 1 j= 7 k= 4 Chicago compared to Norfolk
i= 1 j= 3 k= 2 Chicago compared to Charlotte
i= 3 j= 3 k= 3 Chicago compared to Indianapolis
[1] "not found"
```

Comparing recursion and iteration

- Recursion
  - Very simple algorithms for certain problems
  - Arguably, more elegant than iteration
- Iteration
  - More efficient computationally
  - Arguably, more obvious than recursion
- Deciding between the two
  - Which suits problem best?
  - Weigh clarity vs. efficiency
- Theoretically equivalent in computational power

Recursive definitions summary

Object	Characteristics
Recursive sequence	First objects in sequence defined explicitly. Later objects defined in terms of earlier objects.
Recursive set	First (or first few) elements of set defined explicitly. Later elements defined in terms of elements already in set.
Recursive operation	First (or first few) cases of operation defined explicitly. Later (larger) cases defined in terms of earlier (smaller) cases.
Recursive algorithm	Smallest argument value(s) handled directly. Larger argument values handled via recursive calls to algorithm.

Table 3.1

Section 3.1 homework assignment

See homework list for specific exercises.

3.2 Recurrence Relations

Solving recurrence relations

Recall sequence S (Example 1):

$S(1) = 2$
$S(n) = 2 \cdot S(n-1)$ for $n \geq 2$

$n = 1 \quad S(1) = 2 \quad = 2^1$
$n = 2 \quad S(2) = 4 \quad = 2^2$
$n = 3 \quad S(3) = 8 \quad = 2^3$
$n = 4 \quad S(4) = 16 \quad = 2^4$
…
$n \quad S(n) \quad = 2^n$

Pattern of values for S suggests formula for $S(n)$.
How can such formulas be verified?
Can formulas be found for all recurrence relations?

- Applications of recurrence relations
  - Chemical degradation
  - Interest-bearing accounts
  - Ecological modeling
  - Computer virus spread
  - Computation steps required by an algorithm
- Solving recurrence relations
  - Closed form solution; formula to compute recurrence relation value without computing intermediate values
  - "Subject to", i.e., depends on, basis step
  - Solving recurrence relation; finding a closed form solution

- Types of recurrence relations
  - Linear first order with constant coefficients
  - Linear first order without constant coefficients
  - Homogeneous linear second order with constant coefficients
  - Divide-and-conquer
  - Others, not covered in CS 214
- Methods to solve recurrence relations
  - Expand, guess, and verify
  - Direct solution using known formulas

Example "expand, guess, and verify"

Sequence S:

$S(1) = 2$

$S(n) = 2 \cdot S(n-1)$ for $n \geq 2$ recurrence relation

$S(n)$	$= 2 \cdot S(n-1)$		expansion 1
	$= 2 \cdot (2 \cdot S(n-2))$	$= 2^2 \cdot S(n-2)$	expansion 2
	$= 2^2 \cdot (2 \cdot S(n-3))$	$= 2^3 \cdot S(n-3)$	expansion 3
	…		
	$= 2^k \cdot S(n-k)$		expansion k
	…		
	$= 2^{n-1} \cdot S(n-(n-1))$		expansion $n-1$
	$= 2^{n-1} \cdot S(1)$		rewrite expansion $n-1$
	$= 2^{n-1} \cdot 2$		by basis
	$= 2^n$		closed form solution guess

expand

Verify the closed form solution for sequence S by induction.

Theorem
For sequence S, $S(n) = 2^n$ for $n \geq 1$. 1

Proof
Basis. $S(1) = 2^1 = 2$. 2

Inductive hypothesis. Assume $S(k) = 2^k$.

Inductive step. Show $S(k + 1) = 2^{k+1}$.

	$S(k + 1)$	left side	3
=	$2 \cdot S(k)$	by recurrence relation for S	
=	$2 \cdot 2^k$	by inductive hypothesis	
=	2^{k+1}	right side	

Thus $S(n) = 2^n$ for $n \geq 1$. ■

Example "expand, guess, and verify"

Sequence T:

1. $T(1) = 1$
2. $T(n) = T(n-1) + 3$ for $n \geq 2$

$$
\begin{aligned}
T(n) &= T(n-1) + 3 && && \text{expansion 1}\\
&= [T(n-2) + 3] + 3 && = T(n-2) + 2\cdot 3 && \text{expansion 2}\\
&= [T(n-3) + 3] + 2\cdot 3 && = T(n-3) + 3\cdot 3 && \text{expansion 3}\\
&\ldots\\
&= T(n-k) + k\cdot 3 && && \text{expansion } k\\
&\ldots\\
&= T(n-(n-1)) + (n-1)\cdot 3 && && \text{expansion } n-1\\
&= T(1) + (n-1)\cdot 3 && && \text{by basis}\\
&= 1 + (n-1)\cdot 3 && && \text{closed form solution}
\end{aligned}
$$

Practice 11

Verify the closed form solution for sequence T by induction.

Theorem
For sequence T, $T(n) = 1 + (n - 1)\cdot 3$ for $n \geq 1$.

Proof
Basis. $T(1) = 1 + (1 - 1)\cdot 3 = 1$. 1

Inductive hypothesis. Assume $T(k) = 1 + (k - 1)\cdot 3$.

Inductive step. Show
$T(k + 1) = 1 + ((k + 1) - 1)\cdot 3 = 1 + k\cdot 3$.

	$T(k + 1)$	left side
=	$T(k) + 3$	by recurrence relation for T 2
=	$1 + (k - 1)\cdot 3 + 3$	by inductive hypothesis
=	$1 + k \cdot 3$	right side

Thus $T(n) = 1 + (n - 1)\cdot 3$ for $n \geq 1$. ■

Linear recurrence relations

- Some known solution formulas exist for certain types of recurrence relations
- Linear recurrence relations
  - In recurrence relation for value $S(n)$, earlier values (e.g., $S(n-1)$) occur only to first power
  - e.g., sequences S and T of previous examples

e.g., $S(n) = 2 \cdot S(n-1)$
e.g., $T(n) = T(n-1) + 3$

General form for linear recurrence relation

$$S(n) = f_1(n)S(n-1) + f_2(n)S(n-2) + \ldots + f_k(n)S(n-k) + g(n)$$

- Special types of linear recurrence relations
  - Constant coefficients; all the $f_i(n)$ are constants
  - First-order; term n depends only on term $n - 1$
  - Homogeneous; $g(n) = 0$ for all n

Linear recurrence relation general form

$$S(n) = f_1(n)S(n-1) + f_2(n)S(n-2) + \ldots + f_k(n)S(n-k) + g(n)$$

Linear first-order recurrence relation with constant coefficients general form

$$S(n) = cS(n-1) + g(n)$$

Solution formula for linear first order recurrence relations with constant coefficients

Linear first-order recurrence relation with constant coefficients

$S(n) = cS(n-1) + g(n)$ general form

Solution formula for
linear first-order recurrence relation with constant coefficients

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^{n} c^{n-i} g(i)$$ solution formula

where

$$\sum_{i=2}^{n} c^{n-i} g(i) = c^{n-2}g(2) + c^{n-3}g(3) + \ldots + c^{1}g(n-1) + c^{0}g(n)$$

Example closed form solution using formula

Solution formula for
linear first-order recurrence relation with constant coefficients

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^{n} c^{n-i} g(i)$$

Sequence S is a linear first-order homogeneous recurrence relation with constant coefficients.

1. $S(1) = 2$
2. $S(n) = 2 \cdot S(n-1)$ for $n \geq 2$

Comparing to general form, $c = 2$ and $g(n) = 0$.
Closed form solution:

$$S(n) = 2^{n-1} \cdot 2 + \sum_{i=2}^{n} 2^{n-i} \cdot 0 = 2^n$$

Example 16

Example closed form solution using formula

Find a closed form solution to the recurrence relation $S(n) = 2 \cdot S(n-1) + 3$ for $n \geq 2$, subject to basis $S(1) = 4$.

S is a linear first-order recurrence relation with constant coefficients, so the formula may be used. 1
Comparing to the general form, $c = 2$ and $g(n) = 3$.

$$S(n) = 2^{n-1} \cdot 4 + \sum_{i=2}^{n} 2^{n-i} \cdot 3$$

$$= 2^{n-1} \cdot 2^2 + 3\sum_{i=2}^{n} 2^{n-i}$$

$$= 2^{n+1} + 3 \cdot [2^{n-2} + 2^{n-3} + \ldots + 2^1 + 2^0] \quad 2$$

$$= 2^{n+1} + 3 \cdot [2^{n-1} - 1]$$

e.g., $S(5) = 2^6 + 3 \cdot (2^4 - 1) = 64 + 3 \cdot 15 = 109$ Example 17

Example closed form solution using formula

Find a closed form solution to the recurrence relation $T(n) = T(n-1) + (n+1)$ for $n \geq 2$, subject to basis $T(1) = 2$.

T is a linear first-order recurrence relation with constant coefficients, so the formula may be used.
Comparing to the general form, $c = 1$ and $g(n) = n + 1$.

$$T(n) = 1^{n-1} \cdot 2 + \sum_{i=2}^{n} 1^{n-i} \cdot (i+1)$$

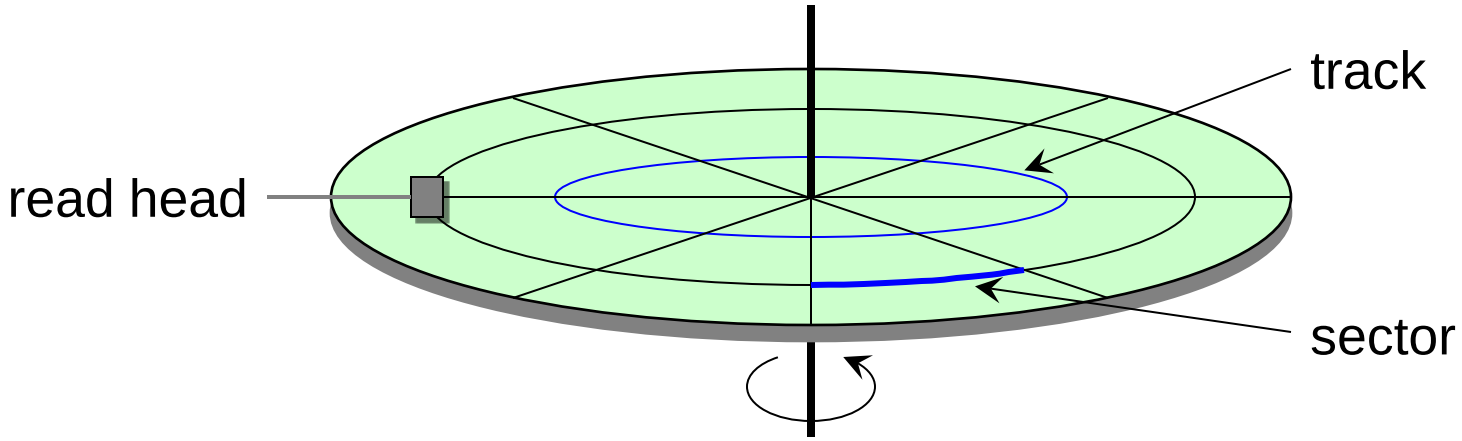
$$= 2 + \sum_{i=2}^{n} (i+1)$$

$$= 2 + (3 + 4 + \ldots + (n+1))$$

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

Example 18

Example application of recurrence relation



Disk read time

1. Seek time: move read head laterally to proper track; time function of number of tracks to move.
2. Latency time: wait for desired sector to rotate to head; time function of rotational speed, number sectors to wait.
3. Transfer time: read block and send to processor; time function of block size, transfer rate.

Example 19

Problem: Calculate average seek time $A(n)$, in terms of numbers of tracks to move read head, as function of number of tracks on disk n.

Table shows number of tracks to move for all possible combinations of start and destination track.

Destination track / Start track	1	2	3	. . .	$n-1$	n
1	0	1	2	. . .	$n-2$	$n-1$
2	1	0	1	. . .	$n-3$	$n-2$
3	2	1	0	. . .	$n-4$	$n-3$
. . .	. . .	. . .	. . .	. . .	. . .	. . .
$n-1$	$n-2$	$n-3$	$n-4$	. . .	0	1
n	$n-1$	$n-2$	$n-3$	. . .	1	0

Table 3.3

Find average $A(n)$ by computing total of all entries in table, denoted $T(n)$, and dividing by number of entries n^2.

$T(n)$ = total of all entries in table of size $n \times n$
= $T(n-1)$ + total of last row plus last column

Last row and last column contribute
$2[(n-1) + \ldots + 3 + 2 + 1 + 0] = 2[(n-1)n/2] = (n-1)n$

$T(1) = 0$ basis; no tracks to move
$T(n) = T(n-1) + (n-1)n$ recurrence relation

Solve recurrence relation using formula with
$c = 1$ and $g(n) = (n-1)n$.

Total of table entries

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

from solution formula

$$= 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \ldots + (n-1)n$$

$$= \frac{(n-1)n(n+1)}{3}$$

Section 2.2, Exercise 19

Average tracks to move

$$A(n) = \frac{(n-1)n(n+1)}{3} / n^2$$

$$= \frac{n^3 - n^2 + n^2 - n}{3n^2}$$

multiply through

$$= \frac{n^3 - n}{3n^2} = \frac{n^2 - 1}{3n}$$

simplify

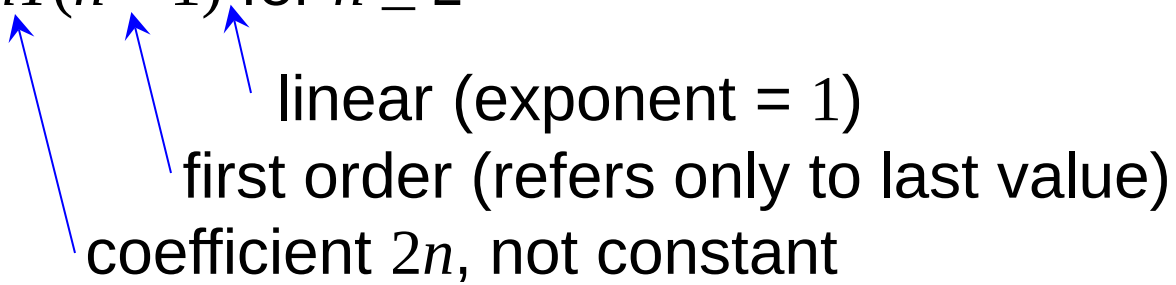
$$= \frac{n}{3} - \frac{1}{3n}$$

Linear first order recurrence relation without constant coefficients

Example

$T(1) = 1$

$T(n) = 2nT(n-1)$ for $n \geq 2$



Earlier solution formula requires constant coefficients, does not apply. Solve using expand, guess, verify.

Example 20

$T(1) = 1$

$T(n) = 2nT(n-1)$ for $n \geq 2$

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

$= 2n[2(n-1)T(n-2)]$ $= 2^2n(n-1)T(n-2)$

$= 2^2n(n-1)[2(n-2)T(n-3)]$ $= 2^3n(n-1)(n-2)T(n-3)$

…

$= 2^kn(n-1)(n-2)\ \ldots\ (n-(k-1))T(n-k)$ expansion k

…

$= 2^{n-1}n(n-1)(n-2)\ \ldots\ (2)T(n-(n-1))$ expansion $k = n-1$

$= 2^{n-1}n(n-1)(n-2)\ \ldots\ (2)T(1)$ by basis

$= 2^{n-1}n(n-1)(n-2)\ \ldots\ (2)(1)$

$= 2^{n-1}n!$ guess

Verify the closed form solution for sequence T by induction.

Theorem
For sequence T, $T(n) = 2^{n-1}n!$ for $n \geq 1$. 1

Proof
Basis. $T(1) = 2^{1-1}1! = 2^0(1) = 1$. 2

Inductive hypothesis. Assume $T(k) = 2^{k-1}k!$.

Inductive step. Show $T(k + 1) = 2^k(k + 1)!$.

	$T(k + 1)$	left side
=	$2(k + 1)T(k)$	by recurrence relation for T 3
=	$2(k + 1)2^{k-1}k!$	by inductive hypothesis
=	$2^k(k + 1)!$	algebra and def of !

Thus $T(n) = 2^{n-1}n!$ for $n \geq 1$. ■

Homogeneous linear second-order recurrence relation with constant coefficients

General form

$$S(n) = c_1 S(n-1) + c_2 S(n-2)$$

Example: Fibonacci sequence

$$F(1) = 1$$
$$F(2) = 1$$
$$F(n) = F(n-1) + F(n-2) \text{ for } n > 2$$

Two basis values needed because recurrence relation $F(n)$ refers back two values.

Refresher: quadratic formula

Find roots of a quadratic equation $ax^2 + bx + c = 0$

with quadratic formula $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Example $x^2 - 2x - 3 = 0$ two distinct roots

$a = 1 \quad b = -2 \quad c = -3$

$$x = \frac{-(-2) \pm \sqrt{(-2)^2 - 4 \cdot 1 \cdot -3}}{2 \cdot 1} = \frac{2 \pm \sqrt{16}}{2} = 3, -1$$

Example $x^2 - 8x + 16 = 0$ one repeated root

$a = 1 \quad b = -8 \quad c = 16$

$$x = \frac{-(-8) \pm \sqrt{(-8)^2 - 4 \cdot 1 \cdot 16}}{2 \cdot 1} = \frac{8 \pm \sqrt{0}}{2} = 4$$

Solution formula (distinct roots)

Homogeneous linear second-order recurrence relation with constant coefficients

1. Set up characteristic equation for recurrence relation
   $t^2 - c_1 t - c_2 = 0$
2. Find roots r_1 and r_2 of characteristic equation.
3. Set up linear equations
   $p + q = S(1)$
   $pr_1 + qr_2 = S(2)$
4. Solve linear equations for p and q.
5. Apply closed form solution formula using r_1, r_2, p, q.
   $S(n) = pr_1^{n-1} + qr_2^{n-1}$ for $n \geq 1$

Example closed form solution using formula

Homogeneous linear second-order recurrence relation with constant coefficients

$S(1) = 3$

$S(2) = 1$

$S(n) = 2S(n-1) + 3S(n-2)$ for $n \geq 3$

Characteristic equation

$t^2 - 2t - 3 = 0$

Roots $r_1 = 3$, $r_2 = -1$

Example 21

Linear equations

$p + q = 3$ $\quad\quad$ $S(1) = 3$

$3p - q = 1$ $\quad\quad$ $r_1 = 3, r_2 = -1, S(2) = 1$

Solutions to linear equations $p = 1, q = 2$

Put roots r_1, r_2 and solutions p, q into solution formula

$$S(n) = pr_1^{n-1} + qr_2^{n-1}$$
$$= 3^{n-1} + 2 \cdot -1^{n-1}$$

Closed form solution for $S(n)$

Solution formula (repeated roots)

Homogeneous linear second-order recurrence relation with constant coefficients

1. Set up characteristic equation for recurrence relation
   $t^2 - c_1 t - c_2 = 0$
2. Find repeated root r of characteristic equation.
3. Set up linear equations
   $p = S(1)$
   $pr + qr = S(2)$
4. Solve linear equations for p and q.
5. Apply closed form solution formula using r, p, q.
   $S(n) = pr^{n-1} + q(n-1)r^{n-1}$ for $n \geq 1$

Example closed form solution using formula

Homogeneous linear second-order recurrence relation with constant coefficients

$S(1) = 1$

$S(2) = 12$

$S(n) = 8S(n-1) - 16S(n-2)$ for $n \geq 3$

Characteristic equation

$t^2 - 8t + 16 = 0$

Root $r = 4$

Example 22

Linear equations

$p = 1$ $S(1) = 1$

$4p + 4q = 12$ $r = 4, S(2) = 12$

Solutions to linear equations $p = 1, q = 2$

Put root r and solutions p, q into solution formula

$$\begin{aligned} S(n) &= pr^{n-1} + q(n-1)r^{n-1} \\ &= 1 \cdot 4^{n-1} + 2 \cdot (n-1) \cdot 4^{n-1} \\ &= 4^{n-1} + (2n-2) \cdot 4^{n-1} \\ &= (2n-1) \cdot 4^{n-1} \end{aligned}$$

Closed form solution for $T(n)$

Divide-and-conquer recurrence relation

General form

$S(n) = cS(n/2) + g(n)$ for $n \geq 2$, $n = 2^m$

Example: Binary search

$C(1) = 1$
$C(n) = 1 + C(n/2)$ for $n \geq 2$

Neither first-order nor second-order because value n depends on value "halfway back".

Example divide-and-conquer solution

$C(1) = 1$

$C(n) = 1 + C(n/2)$ for $n \geq 2$

Solve with "expand, guess, and verify" method.

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

$$= 1 + \left(1 + C\left(\frac{n}{4}\right)\right)$$

$$= 1 + \left(1 + \left(1 + C\left(\frac{n}{8}\right)\right)\right)$$

K

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

$$= \log n + C(1) \quad 2$$

$$= \log n + 1 \quad 3$$

Expansion stops when $2^k = n$, because $n/n = 1$ is basis step. $2^k = n$ implies $k = \log_2 n$.

Example 24

Verify

Theorem
$C(n) = 1 + \log n$ for $n \geq 2$, $n = 2^m$

Proof
Basis. $C(1) = 1 + \log 1 = 1 + 0 = 1$. 1

Inductive hypothesis. Assume $C(k) = 1 + \log k$. 2

Inductive step. Induction will be on powers of 2,
thus show $C(2k) = 1 + \log 2k$. 3

$$
\begin{aligned}
C(2k) &= 1 + C(k) && \text{by recurrence relation} \\
&= 1 + 1 + \log k && \text{by inductive hypothesis} \\
&= 1 + \log 2 + \log k && \log 2 = 1 \\
&= 1 + \log 2k && \text{property of } \log
\end{aligned}
$$

Thus $C(n) = 1 + \log n$ for $n \geq 1$. ■

Divide-and-conquer closed form solution formula

Divide-and-conquer recurrence relation general form

$$S(n) = cS\left(\frac{n}{2}\right) + g(n) \quad \text{for } n \geq 2,\ n = 2^m$$

Divide-and-conquer recurrence relation solution formula

$$S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n)-i} g(2^i)$$

Example closed form solution using formula

Binary search recurrence relation

$$C(1) = 1$$

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

Apply solution formula with $c = 1$ and $g(n) = 1$.

$$C(n) = 1^{\log n} C(1) + \sum_{i=1}^{\log n} 1^{(\log n) - i}(1)$$

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

$$= 1 + \log n$$

Agrees with previous result for binary search.

Example 25

Example closed form solution using formula

Recurrence relation

$$T(1) = 3$$

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

Apply solution formula with $c = 2$ and $g(n) = 2n$.

$$T(n) = 2^{\log n} T(1) + \sum_{i=1}^{\log n} 2^{\log n - i} 2(2^i)$$

$$= 2^{\log n} 3 + \sum_{i=1}^{\log n} 2^{\log n + 1}$$

$$= n \cdot 3 + (2^{\log n + 1}) \log n$$

$$= 3n + (2^{\log n} \cdot 2) \log n$$

$$= 3n + 2n \log n$$

Example 26

Recurrence relation solution method summary

Type of recurrence relation	Solution method(s)
Linear first order with constant coefficients	▪ Expand, guess, verify ▪ Solution formula (summation) ▪ Summary: Table 3.2
Linear first order without constant coefficients	▪ Expand, guess, verify
Homogeneous linear second order with constant coefficients	▪ Expand, guess, verify ▪ Solution formula (characteristic equation) ▪ Summary: Table 3.4
Divide-and-conquer	▪ Expand, guess, verify ▪ Solution formula (summation) ▪ Summary: Table 3.5

Recurrence relation of unknown type?

Recurrence relation

$G(0) = 0$
$G(n) = G(n-1) + 2n - 1$ for $n \geq 1$

Linear?	Yes, no exponents on terms
First order?	Yes, refers only to term $n-1$
Constant coefficients?	Yes, coefficients don't use n
Homogeneous?	No, $g(n) = 2n - 1$

Solve using solution formula.

www.cs.ucr.edu/~jiang/cs141/recur-tut.txt

Solution formula for
linear first-order recurrence relation with constant coefficients

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^{n} c^{n-i} g(i)$$

Sequence G is a linear first-order homogeneous
recurrence relation with constant coefficients.

1. $G(0) = 0$
2. $G(n) = G(n-1) + 2n - 1$ for $n \geq 1$

Comparing to general form, $c = 1$ and $g(n) = 2n - 1$.

Apply solution formula now?
No; solution formula has $S(1)$, recurrence relation has $G(0)$.

$$G(1) = G(1-1) + 2 \cdot 1 - 1 = 0 + 2 - 1 = 1$$

Apply formula to get solution

$$G(n) = 1^{n-1}G(1) + \sum_{i=2}^{n} 1^{n-1} \cdot (2i-1)$$ Plug G into solution formula

$$= 1 + \left(3 + 5 + 7 + \cdots + (2n-1)\right)$$ Expand summation

$$= 1 + (4 + 6 + 8 + \cdots + 2n) - (n-1)$$ Add 1 to each of $n-1$ terms, subtract $n-1$

$$= 1 + 2(2 + 3 + 4 + \cdots + n) - (n-1)$$ Factor 2 out of sum in parens

$$= 1 + 2(1 + 2 + 3 + \cdots + n) - (n-1) - 2$$ Add 1 inside parens, subtract 2 outside

$$= 2\left(\frac{n(n+1)}{2}\right) - (n-1) - 2 + 1$$ Use summation formula

$$= n(n+1) - n$$
$$= n^2 + n - n$$
$$= n^2$$

} Algebra

Sample problem solution

You are serving on the city council's Board of Land Management, which is considering a proposal by a private contractor to manage a chemical disposal site. The material to be stored at the site degrades to inert matter at the rate of 5% per year. The contractor claims that, at this rate of stabilization, only about one-third of the original active material will remain at the end of 20 years.

Sequence T

1. $T(1) = X$ X is original quantity
2. $T(n) = 0.95 \cdot T(n-1)$ for $n \geq 2$

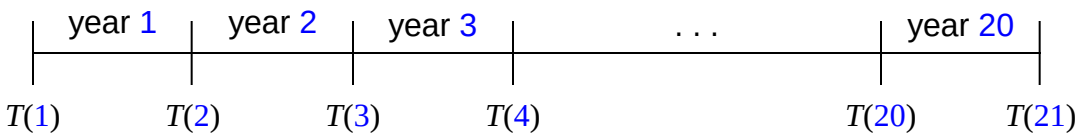
Exercise 13

$T(n)$ is a linear first order recurrence relation with constant coefficients; apply formula with $c = 0.95$ and $g(n) = 0$.

Closed form solution
$(0.95)^{n-1} \cdot X$

$T(21) = (0.95)^{20} \cdot X = 0.358 \cdot X$,
slightly more than 1/3 original amount.

The contractor was slightly optimistic.



Section 3.2 homework assignment

See homework list for specific exercises.

3.3 Analysis of Algorithms

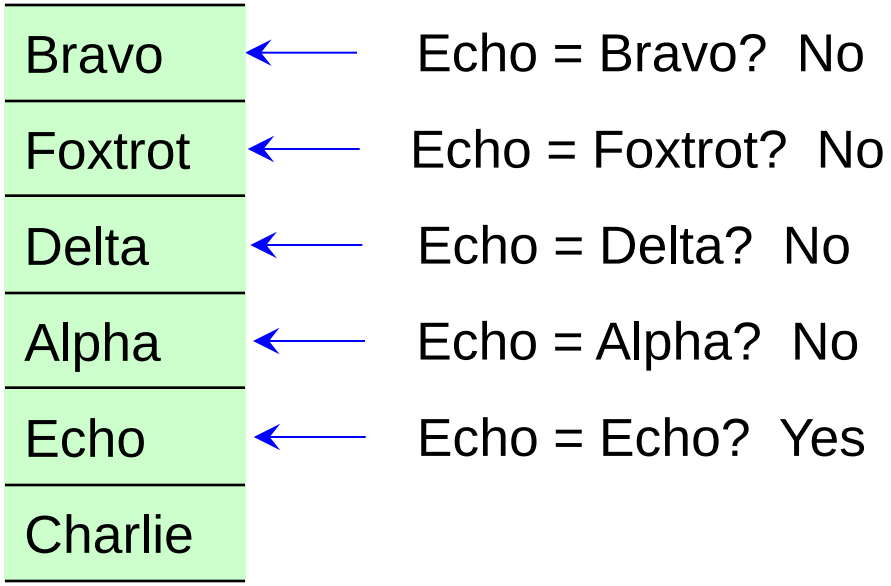
Analysis of algorithms

- There can be > 1 algorithm for a given task
- Comparing algorithms
  - Implementation effort
  - Maintenance effort (ease of understanding)
  - Time efficiency
  - Space (memory) efficiency
- Time efficiency
  - Seek to evaluate algorithm, not implementation
  - Count basic operations, not measure execution time

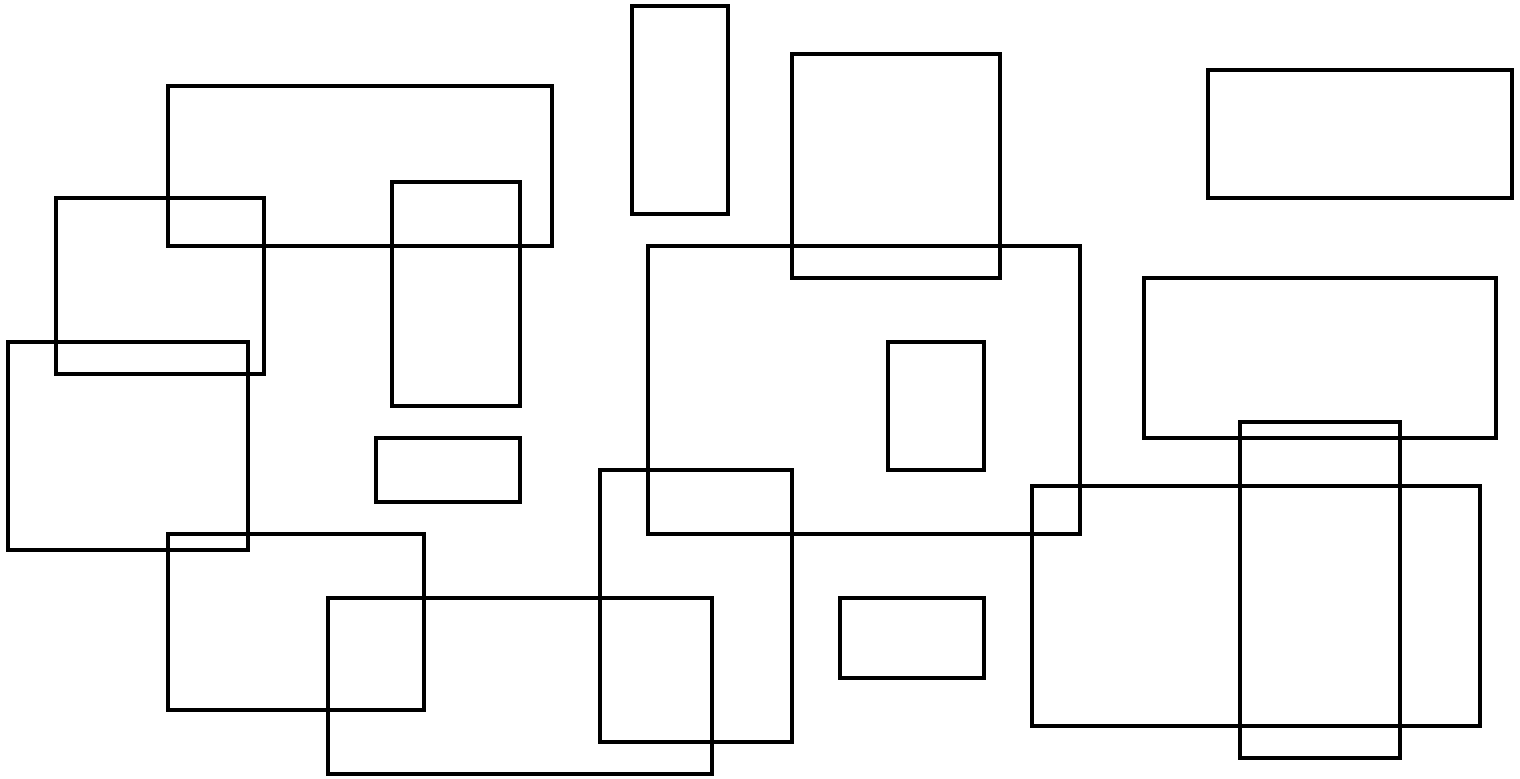
- What are basic operations?
  - Depends on application and algorithm
  - e.g., searching: compare target to data
  - e.g., sorting: compare two values in data
  - e.g., averaging: add two values
  - Each basic operation takes constant time
  - Number of operations is a function of input size

Example basic operation

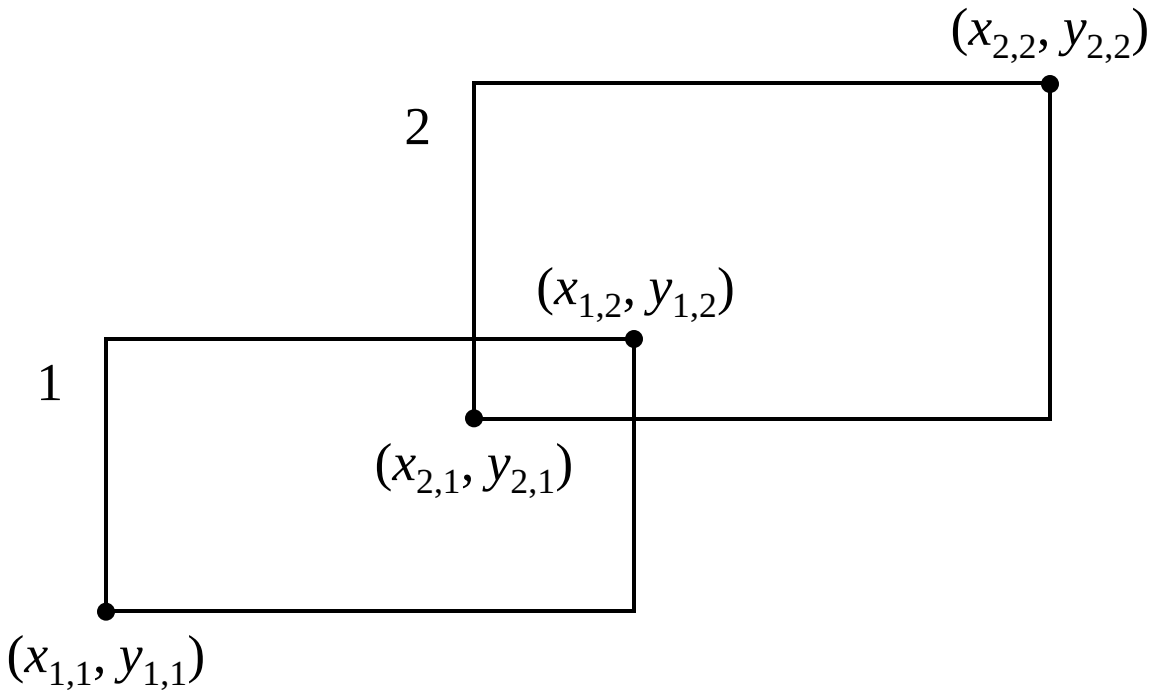
Algorithm: Sequential search
Basic operation: Comparison of target item and list item



Example basic operation



Problem: rectangle intersection; find all pairs of intersecting rectangles. Arises in (1) VLSI design, (2) Simulation (HLA DDM).



Basic operation

- Determine if two 2D axis-parallel rectangles intersect
- Rectangles 1 and 2 intersect iff
  $((x_{1,1} \leq x_{2,2}) \wedge (x_{2,1} \leq x_{1,2})) \wedge ((y_{1,1} \leq y_{2,2}) \wedge (y_{2,1} \leq y_{1,2}))$
- Constant-time operation; does not depend on number of rectangles

Essential concepts of algorithm analysis

- Analysis method depends on algorithm type
  - Iterative: Inspect algorithm, count operations
  - Recursive: Use recurrence relation, calculate operations
- Number of operations depends on input size
  - e.g., searching: length of list to search
  - e.g., sorting: number of entries to sort
- Analysis cases
  - Best case; fewest operations on ideal data
  - Average case; average operations on typical data
  - Worst case; most operations on worst data

Example iterative algorithm analysis

Sequential search; find target in list (array) by comparing it to every entry in list in turn until found or end reached.

```
Algorithm SequentialSearch(list L; integer n; itemtype x)
// Searches a list L of n items for item x.
Local variable:
integer i   // Marks a position in the list
    i = 1
    while L[i] ≠ x and i < n do
        i = i + 1
    end while
    if L[i] = x then
        write("Found")
    else
        write("Not found")
    end if
end function SequentialSearch
```

Analysis of sequential search algorithm

- Basic operation
  - Comparison of target item and list item
- Input size
  - n = items in list to search
- Worst case
  - Target not in list, or in last position
- Analysis
  - n comparisons

Example iterative algorithm analysis

Grade calculation; for each of n students,
sum m quiz grades, then subtract out lowest quiz grade.

```
for i = 1 to n do
    low = roster[i].quiz[1]
    sum = roster[i].quiz[1]
    for j = 2 to m do
        sum = sum + roster[i].quiz[j]        // Addition operation
        if roster[i].quiz[j] < low then
            low = roster[i].quiz[j]
        end if
    end for
    sum = sum – low;                         // Subtraction operation
    write("Total for student", i, " is ", sum)
end for
```

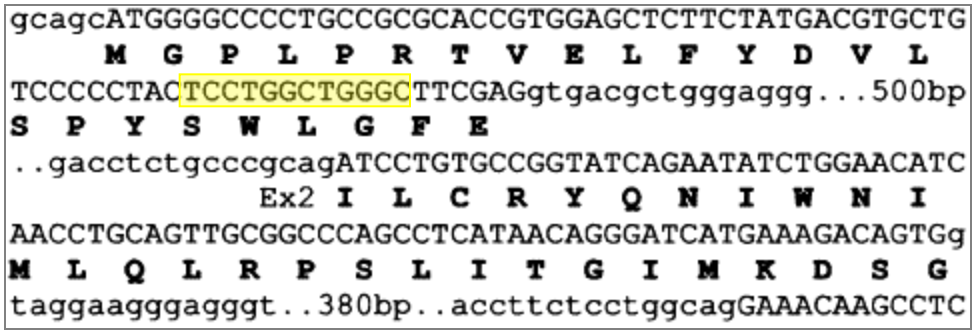
Example 27

Analysis of grade calculation algorithm

- Basic operation
  - Arithmetic operation (addition or subtraction)
- Input size
  - n = number of students
  - m = number of quiz grades per student
- Worst case
  - No difference between cases
- Analysis
  - $n(m-1)$ additions + n subtractions
    $= n + n(m-1)$ arithmetic operations
    $= n(1 + (m-1))$ arithmetic operations
    $= nm$ arithmetic operations

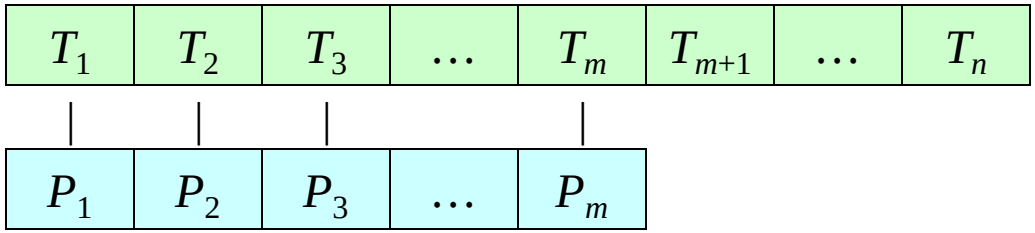
Example iterative algorithm analysis

- Text search
  - Given long string of text characters …
  - … find first occurrence of pattern string in text
- Applications
  - Find gene sequence within DNA strand
  - Find string in HTML document governed by style rule
  - Find string in file for UNIX “grep” or text editor “find”
- Algorithm: character-by-character comparison



Example 28

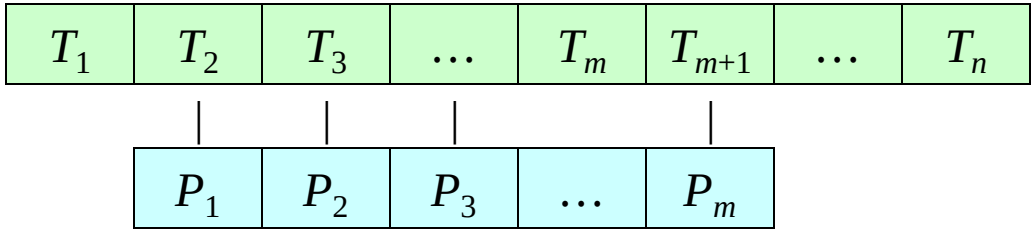
First pattern comparison



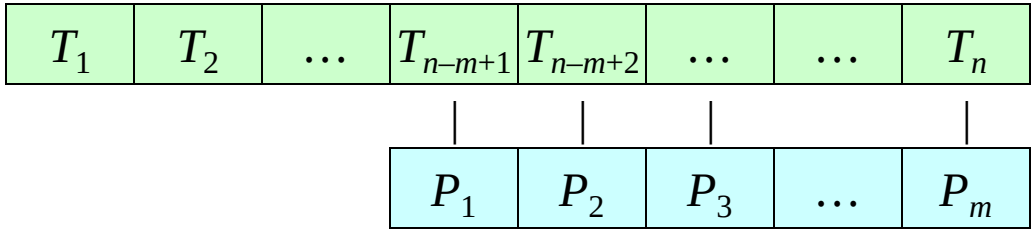
n = text length
m = pattern length

Up to m character comparisons per pattern comparison

Second pattern comparison



Last pattern comparison



Analysis of text search algorithm

- Basic operation
  - Comparison of text character with pattern character
- Input size
  - n = length of text to be searched (characters)
  - m = length of pattern (characters)
- Worst case
  - Pattern not found but text almost matches pattern at every pattern comparison, failing at last character
- Analysis
  - m character comparisons for each of $n - m + 1$ pattern comparisons
  - Total character comparisons $m(n - m + 1)$

Example worst case
Text: $TTTTTT \ldots T$
Pattern: $TTTS$

Example recursive algorithm analysis

Recursive sequential search; find target in list (array) by comparing it to first entry, then calling algorithm recursively on the rest of the list.

```
Algorithm SequentialSearchRecursive(list L; integer i, n; itemtype x)
// Searches a list L from L[i] to L[n] for item x.

    if i > n then
        write("not found")
    else
        if L[i] = x then
            write("found")
        else
            SequentialSearchRecursive(L, i + 1, n, x)
        end if
    endif
end function SequentialSearchRecursive
```

Example 29

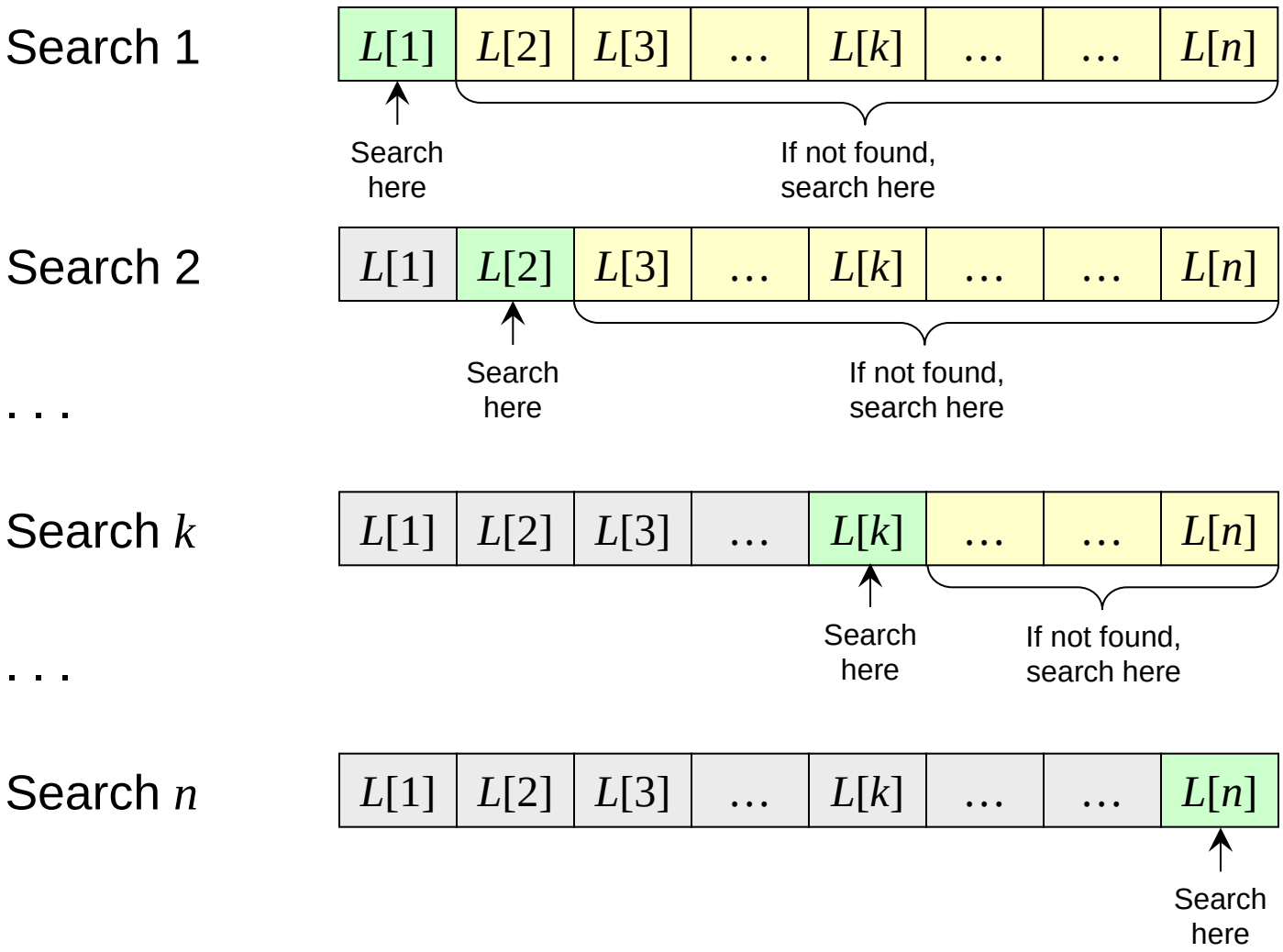


Figure 3.3

Analysis of recursive sequential search algorithm

- Basic operation
  - Comparison of target item and list item
- Input size
  - n = number of items in list
- Worst case
  - Target item not in list, or in last position
- Analysis
  - Recurrence relation for number of comparisons

    $C(1) = 1$ 1-item list

    $C(n) = 1 + C(n - 1)$ 1 comparison plus rest of list
  - Total number of comparisons n

$C(1) = 1$ 1-item list

$C(n) = 1 + C(n-1)$ for $n \geq 2$ 1 comparison plus rest of list

$C(n)$ is linear first-order recurrence relation with constant coefficients; apply solution formula.

$$C(n) = c^{n-1} \cdot C(1) + \sum_{i=2}^{n} c^{n-i} g(i)$$

$$= 1^{n-1} \cdot 1 + \sum_{i=2}^{n} 1^{n-i} \cdot 1$$

$$= 1 + (n-1)$$

$$= n$$

Example recursive algorithm analysis

Binary search; find target in list (array) by recursively dividing list in half and searching half where target must be.

From § 3.1

Algorithm *BinarySearch*(list *L*; integer *i*; integer *j*; itemtype *x*))
// Searches sorted list *L* from *L*[*i*] to *L*[*j*] for item *x*.

```
    if i > j then
        write("not found")
    else
        find the index k of the middle item of the list L[i]–L[j]
        if x = L[k] then   // target = middle value
            write("found")
        else
            if x < L[k] then         // target < middle value
                BinarySearch(L, i, k – 1, x)
            else // target > middle value
                BinarySearch(L, k + 1, j, x)
            end if
        end if
    end if
end function BinarySearch
```

Example 30

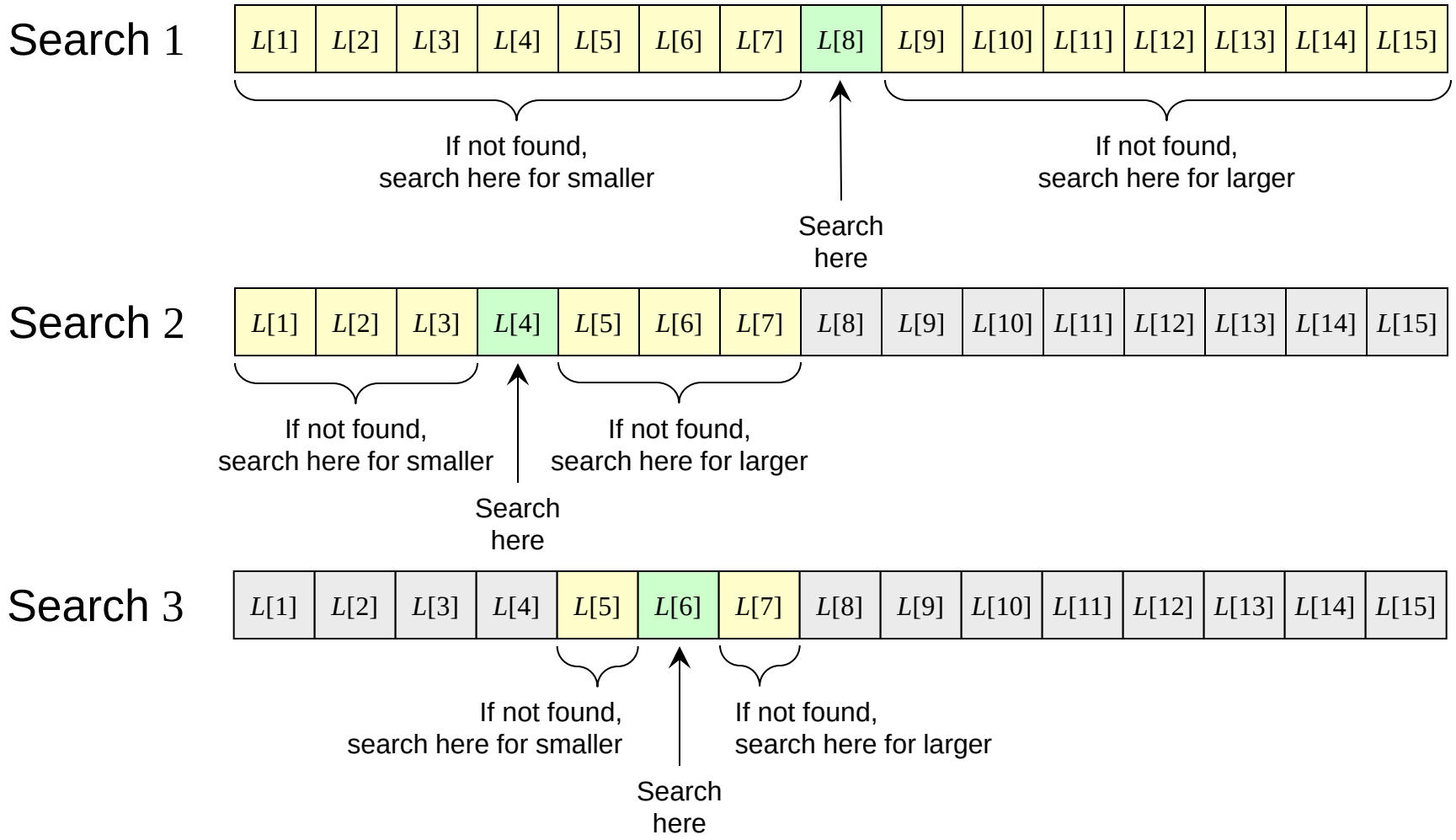


Figure 3.4

Analysis of binary search algorithm

- Basic operation
  - Comparison of target item and list item
- Input size
  - n = number of items in list
- Worst case
  - Target item not in list, or in first or last position in list
- Analysis
  - Recurrence relation for number of comparisons
    $C(1) = 1$ 1-item list
    $C(n) = 1 + C(n/2)$ for $n \geq 2$ 1 comparison plus half of list
  - Total number of comparisons $1 + \log n$
    by § 3.1 Example 24

Comparing binary and sequential search

- Binary search (worst case)
  - List with $n = 2^m$ items requires $1 + \log n$ comparisons
- Sequential search (worst case)
  - List with n items requires n comparisons
- Comparison
  - $1 + \log n < n$ for $n \geq 4$
  - Binary search better than sequential search
- Caveat
  - Binary search requires sorted list ($n \log n$ comparisons)
  - Sequential search works on sorted or unsorted
- Binary search example of “divide-and-conquer”

n	Sequential search	Binary search
64	64	7
1024	1024	11
4096	4096	13
32768	32768	16

Section 3.3 homework assignment

See homework list for specific exercises.

End