Algorithm Analysis

V. Balasubramanian
SSN College of Engineering

Introduction

We should explain, before proceeding, that it is not our object to consider this programme with reference to the actual arrangement of the data on the Variables of the engine, but simply as an abstract question of the nature and number of the operations required to be perfomed during its complete solution.
Ada Augusta Byron King, Countess of Lovelace (1843)

Why Study Algorithms?

- An algorithm is a well-defined set of rules for solving a computational problem.
- For ex:
  - given a list of numbers, rearrange them into sorted order;
  - given a road network, an origin, and a destination, compute the shortest path from the origin to the destination;

Why Study Algorithms?

- Given a set of tasks with deadlines, determine whether or not it is possible to complete all the tasks by their deadlines.

Why Study Algorithms?

- important for all other branches of computer science.
  - routing in communication networks piggybacks on classical shortest-path algorithms;
  - the effectiveness of public-key cryptography rests on that of number-theoretic algorithms;

Why Study Algorithms?

- computer graphics needs the computational primitives supplied by geometric algorithms;
- database indices rely on balanced search tree data structures;
- computational biology uses dynamic programming algorithms to measure genome similarity

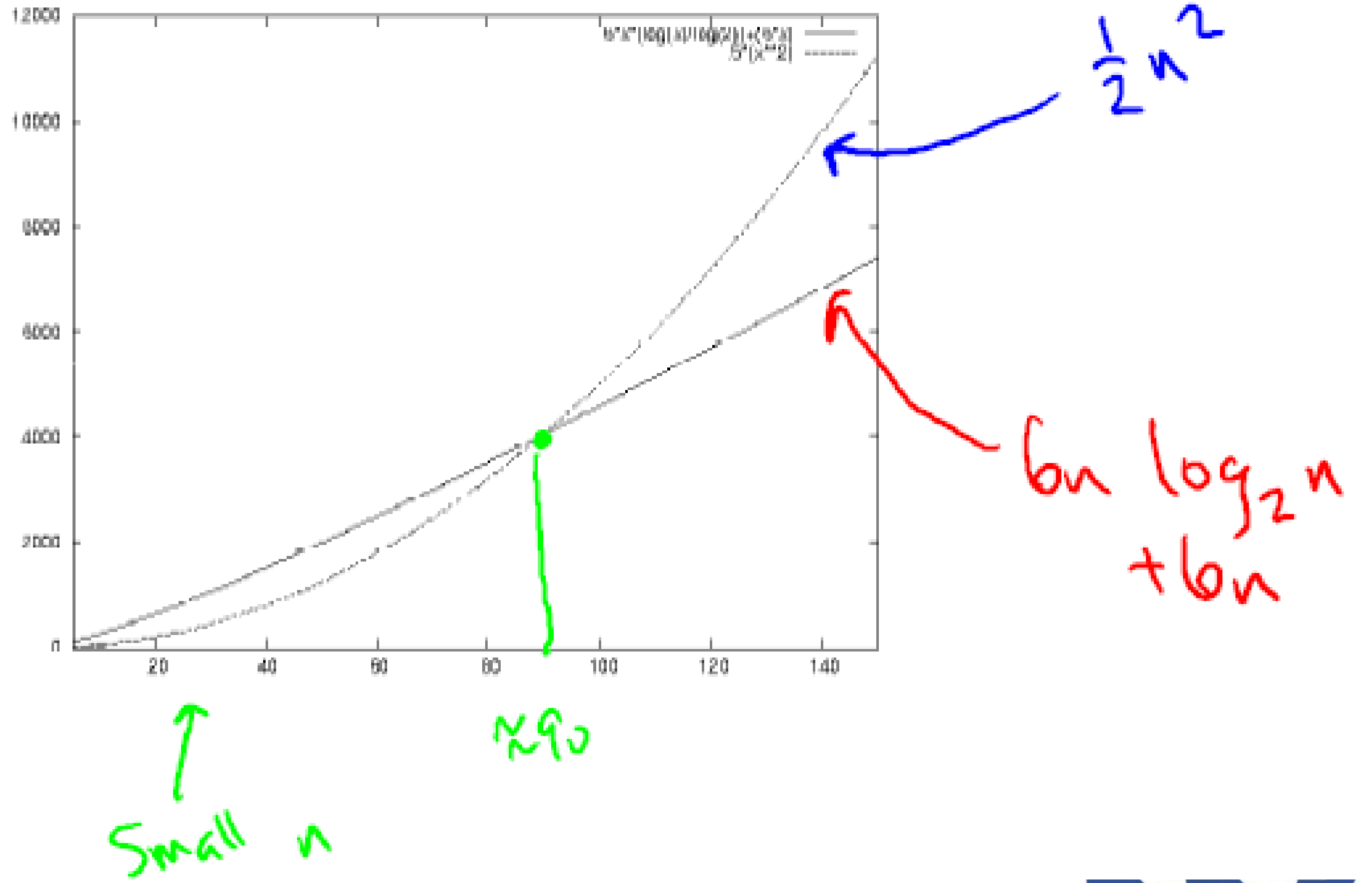
Why Study Algorithms?

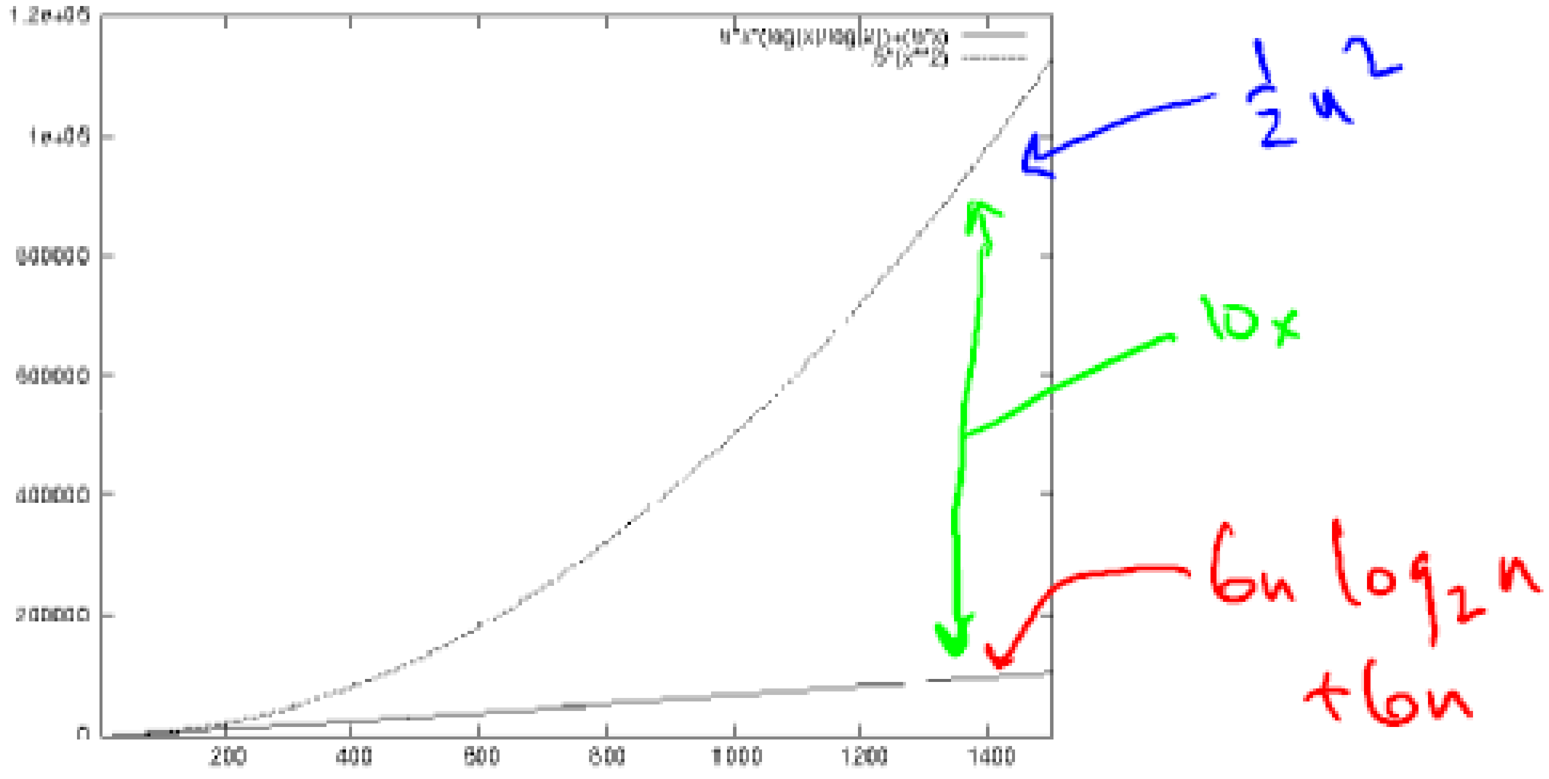
- plays a key role in modern technological innovation
  - "Everyone knows Moore's Law – a prediction made in 1965 by Intel co-founder Gordon Moore that the density of transistors in integrated circuits would continue to double every 1 to 2 years….in many areas, performance gains due to improvements in algorithms have vastly exceeded even the dramatic performance gains due to increased processor speed."

Algorithm Designer’s Mantra

- “Perhaps the most important principle for the good algorithm designer is to refuse to be content.” - Aho, Hopcroft, and Ullman

Example


6*x*(log(x)/log(2))+(6*x)
.5*(x**2)
12000
10000
8000
6000
4000
2000
0
20
40
60
80
100
120
140
$\frac{1}{2}n^2$
$6n \log_2 n + 6n$
≈ 90
Small n



6*x*(log(x)/log(2))+(6*x)
.5*(x**2)
1.2e+06
1e+06
800000
600000
400000
200000
0
200
400
600
800
1000
1200
1400
$\frac{1}{2}n^2$
10x
$6n\log_2 n$
$+6n$

Algorithms

- Fast Algorithm: worst case running time grows slowly with input size.

DAA Course

- OBJECTIVES:
- The student should be made to:
  - Learn the algorithm analysis techniques.
  - Become familiar with the different algorithm design techniques.
  - Understand the limitations of Algorithm power

OUTCOMES

- At the end of the course, the student should be able to:
- Design algorithms for various computing problems.
- Analyze the time and space complexity of algorithms.
- Critically analyze the different algorithm design techniques for a given problem.
- Modify existing algorithms to improve efficiency.

Analysis of algorithms

- Issues:
  - correctness
  - time efficiency
  - space efficiency
  - optimality
- Approaches:
  - theoretical analysis
  - empirical analysis

Theoretical analysis of time efficiency

Time efficiency is analyzed by determining the number of repetitions of the *basic operation* as a function of *input size*

- *Basic operation*: the operation that contributes most towards the running time of the algorithm

Contd…

input size

$$T(n) \approx c_{op}C(n)$$

running time

execution time
for basic operation

Number of times
basic operation is
executed

Input size and basic operation examples

Problem	*Input size measure*	*Basic operation*
Searching for key in a list of n items	*Number of list's items, i.e. n*	*Key comparison*
Multiplication of two matrices	*Matrix dimensions or total number of elements*	*Multiplication of two numbers*
Checking primality of a given integer n	*n'size = number of digits (in binary representation)*	*Division*
Typical graph problem	*#vertices and/or edges*	*Visiting a vertex or traversing an edge*

Empirical analysis of time efficiency

- Select a specific (typical) sample of inputs
- Use physical unit of time (e.g., milliseconds) or Count actual number of basic operation’s executions
- Analyze the empirical data

General Plan for Nonrecursive Algorithms

- Decide on parameter *n* indicating *input size*
- Identify algorithm’s *basic operation*
- Determine *worst*, *average*, and *best* cases for input of size *n*
- Set up a sum for the number of times the basic operation is executed

Contd…

- Check whether the number of times the basic operation is executed depends only on the size of an input. If it also depends on some additional property, the worst-case, average-case, and, if necessary, best-case efficiencies have to be investigated separately.

Important Summations

1. $\sum_{i=l}^{u} 1 = \underbrace{1 + 1 + \cdots + 1}_{u-l+1 \text{ times}} = u - l + 1$ (l, u are integer limits, $l \leq u$); $\sum_{i=1}^{n} 1 = n$

2. $\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$

3. $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$

$$\sum_{i=l}^{u} 1 = u - l + 1$$

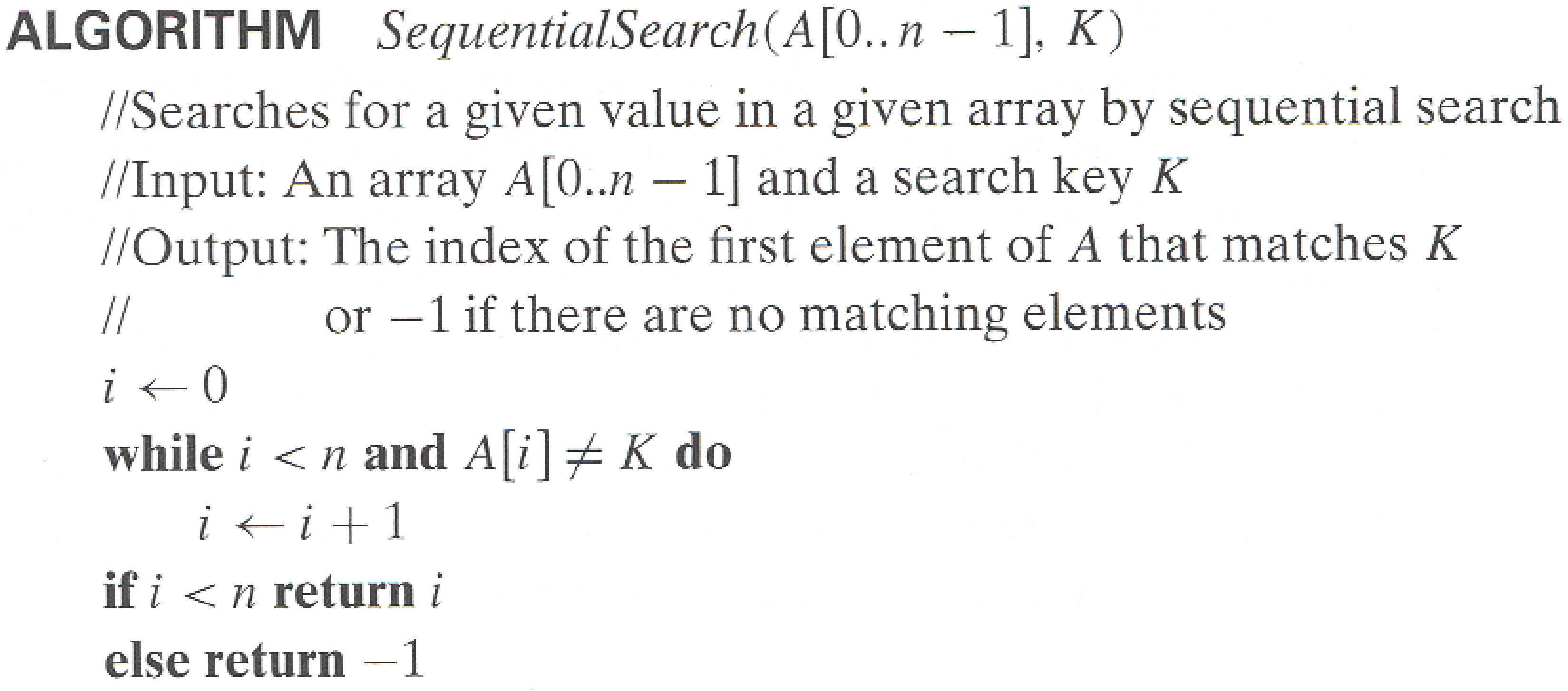
Important Summations

4. $\sum_{i=1}^{n} i^k = 1^k + 2^k + \cdots + n^k \approx \frac{1}{k+1} n^{k+1}$

5. $\sum_{i=0}^{n} a^i = 1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1} \ (a \neq 1); \quad \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

6. $\sum_{i=1}^{n} i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2$

Sequential Search



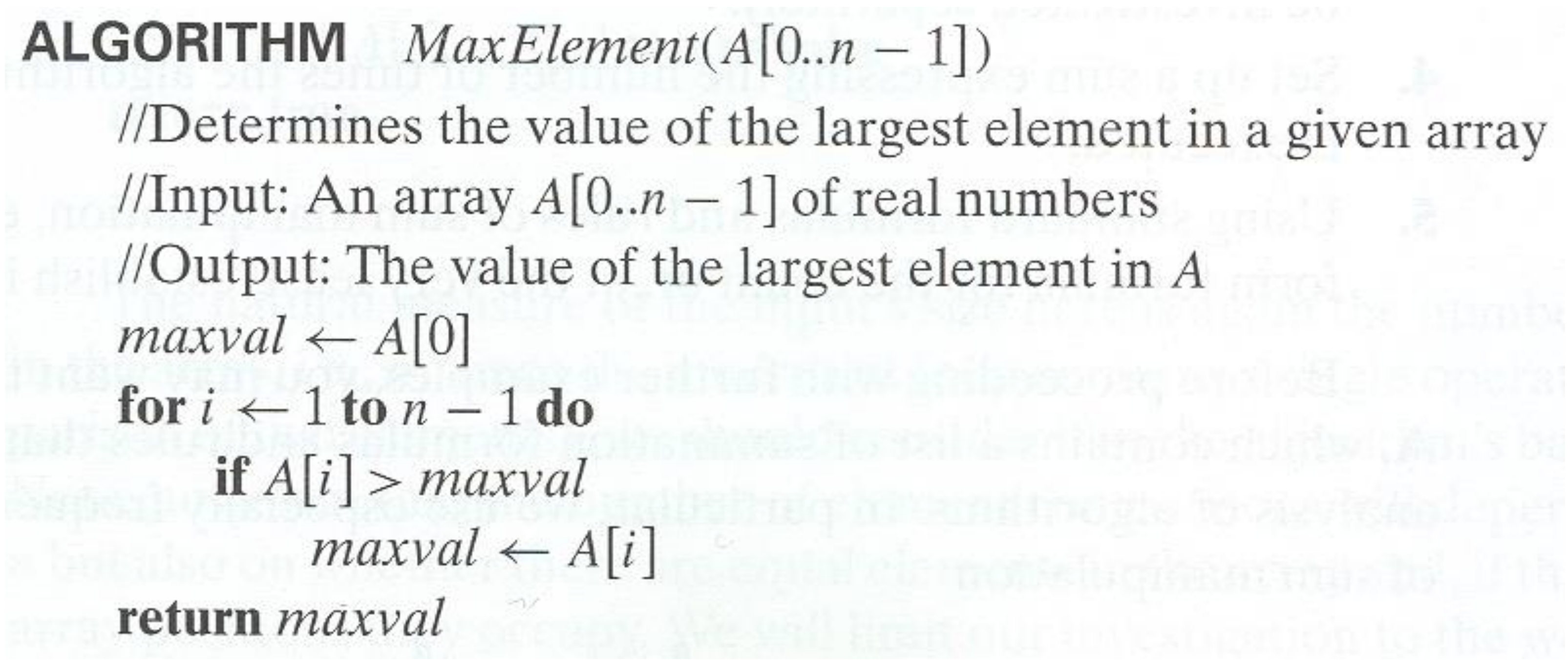
```
ALGORITHM SequentialSearch(A[0..n − 1], K)
    //Searches for a given value in a given array by sequential search
    //Input: An array A[0..n − 1] and a search key K
    //Output: The index of the first element of A that matches K
    //          or −1 if there are no matching elements
    $i \leftarrow 0$
    while $i < n$ and $A[i] \neq K$ do
        $i \leftarrow i + 1$
    if $i < n$ return $i$
    else return $-1$
```

Analysis

- Worst case : n comparisons
- Best case: 1
- Average case

$$\begin{aligned}C_{avg}(n) &= [1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}] + n \cdot (1-p) \\ &= \frac{p}{n}[1 + 2 + \cdots + i + \cdots + n] + n(1-p) \\ &= \frac{p}{n}\frac{n(n+1)}{2} + n(1-p) = \frac{p(n+1)}{2} + n(1-p).\end{aligned}$$

MaxElement

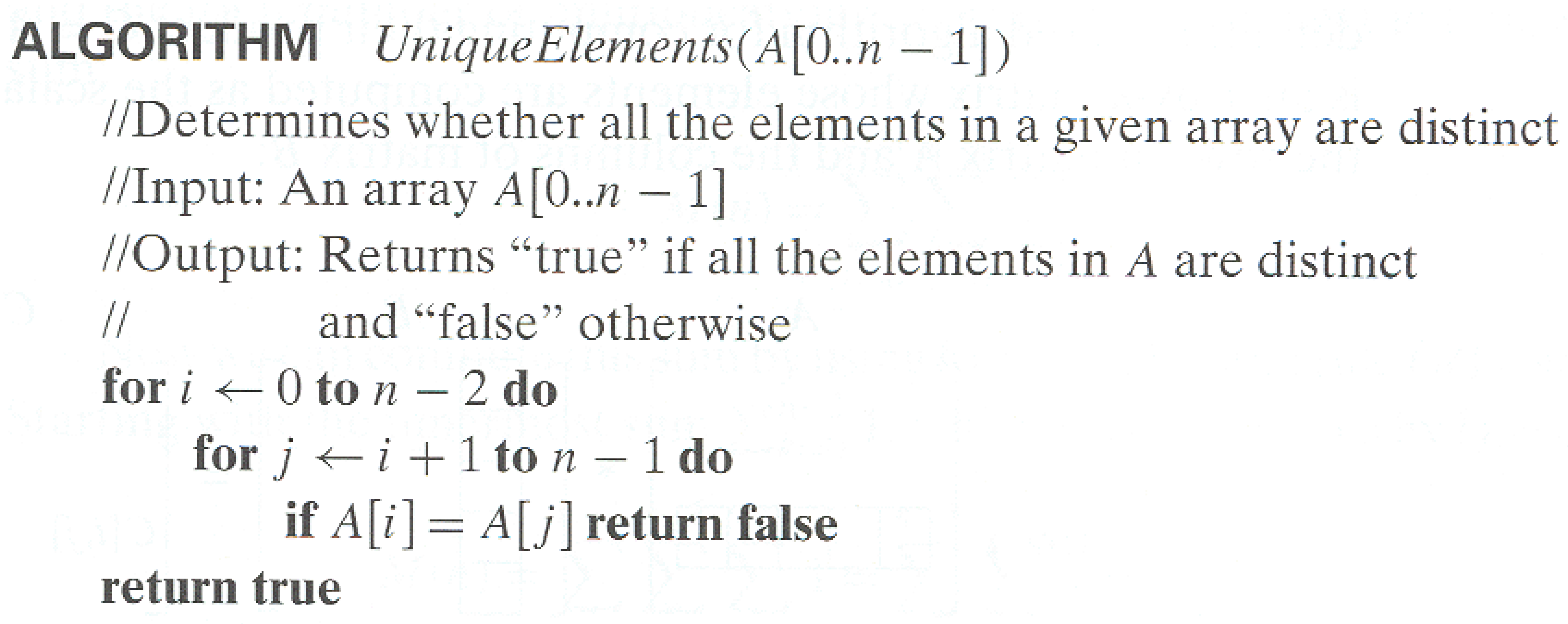


```
ALGORITHM  $MaxElement(A[0..n-1])$
    //Determines the value of the largest element in a given array
    //Input: An array $A[0..n-1]$ of real numbers
    //Output: The value of the largest element in $A$
    $maxval \leftarrow A[0]$
    for $i \leftarrow 1$ to $n-1$ do
        if $A[i] > maxval$
            $maxval \leftarrow A[i]$
    return $maxval$
```

Analysis

- Two Basic Operation:
  - Comparison and assignment
  - Comparison is done always.
- the comparison to be the algorithm’s basic operation.
- The number of comparisons will be the same for all arrays of size n;
- Every Case Time complexity.

Unique Elements



```
ALGORITHM  $UniqueElements(A[0..n-1])$
    //Determines whether all the elements in a given array are distinct
    //Input: An array $A[0..n-1]$
    //Output: Returns "true" if all the elements in $A$ are distinct
    //          and "false" otherwise
    for $i \leftarrow 0$ to $n-2$ do
        for $j \leftarrow i+1$ to $n-1$ do
            if $A[i] = A[j]$ return false
    return true
```

Analysis

- Input Size: number of elements in the array n.
- Basic Operation: Since the innermost loop contains a single operation (the comparison of two elements

Analysis

- worst-case: Inputs for which the algorithm does not exit the loop prematurely / arrays with no equal elements and arrays in which the last two elements are the only pair of equal elements.

Analysis

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

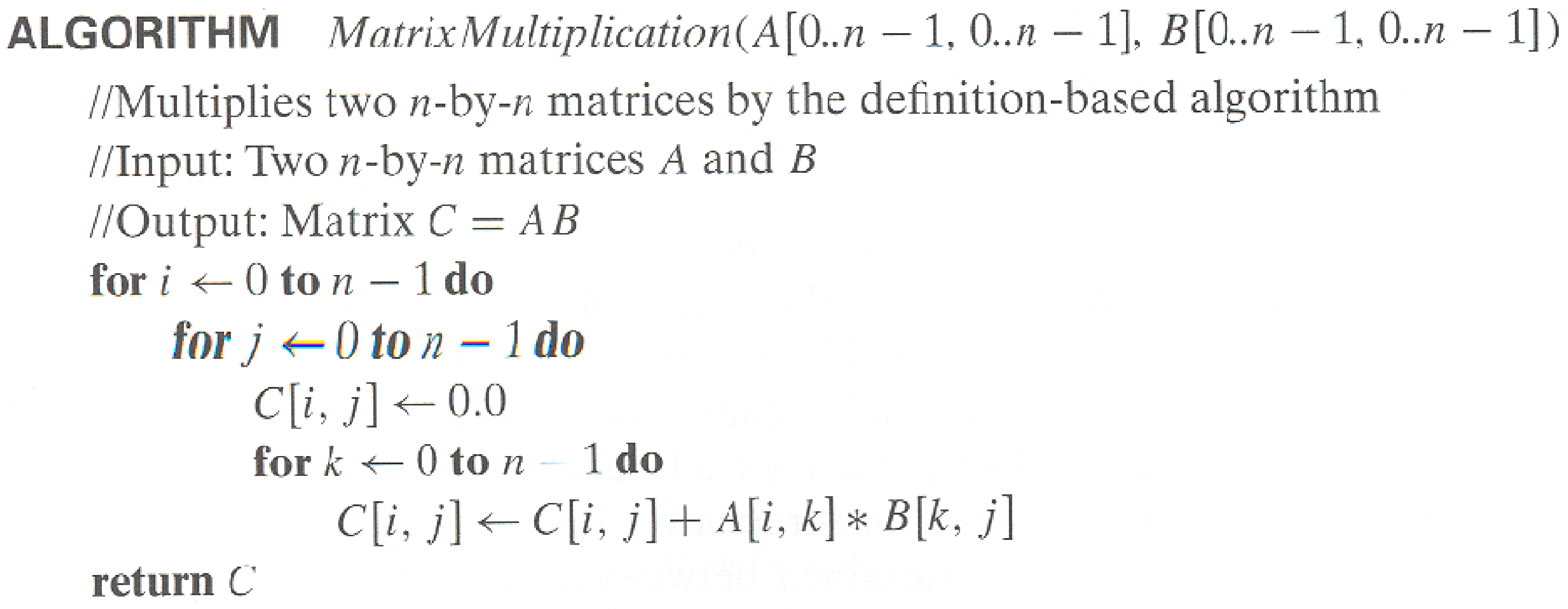
$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2).$$

We also could have computed the sum $\sum_{i=0}^{n-2}(n-1-i)$ faster as follows:

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

Matrix Multiplication



```
ALGORITHM  MatrixMultiplication($A[0..n-1, 0..n-1], B[0..n-1, 0..n-1]$)
    //Multiplies two $n$-by-$n$ matrices by the definition-based algorithm
    //Input: Two $n$-by-$n$ matrices $A$ and $B$
    //Output: Matrix $C = AB$
    for $i \leftarrow 0$ to $n-1$ do
        for $j \leftarrow 0$ to $n-1$ do
            $C[i, j] \leftarrow 0.0$
            for $k \leftarrow 0$ to $n-1$ do
                $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
    return $C$
```

Analysis

- Input Size: Matrix order n.
- Basic Operation: There are two arithmetical operations in the innermost loop here—multiplication and addition.
- No additional property, hence every case time complexity

Analysis

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1.$$

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

Decimal to Binary

```
ALGORITHM  Binary(n)
    //Input: A positive decimal integer n
    //Output: The number of binary digits in n's binary representation
    count ← 1
    while n > 1 do
        count ← count + 1
        n ← ⌊n/2⌋
    return count
```

Analysis

- Input Size: No of Bits to store the number.
- Basic operation: Division
- Since the value of n is about halved on each repetition of the loop, the answer should be about $\log_2 n$

Exercise Problems

- 1 + 3 + 5 + 7 + ...+…+ 999

1. a. $1+3+5+7+...+999 = \sum_{i=1}^{500}(2i\text{-}1) = \sum_{i=1}^{500} 2i\text{-}\sum_{i=1}^{500} 1 = 2\frac{500*501}{2}\text{-}500 = 250,000.$

Exercise problem

b. $2 + 4 + 8 + 16 + \cdots + 1024$

c. $\sum_{i=3}^{n+1} 1$ **d.** $\sum_{i=3}^{n+1} i$ **e.** $\sum_{i=0}^{n-1} i(i+1)$

f. $\sum_{j=1}^{n} 3^{j+1}$ **g.** $\sum_{i=1}^{n} \sum_{j=1}^{n} ij$ **h.** $\sum_{i=1}^{n} 1/i(i+1)$

Solution

b. $2+4+8+16+\ldots+1{,}024=\sum_{i=1}^{10}2^i=\sum_{i=0}^{10}2^i-1=(2^{11}-1)-1=2{,}046.$
(Or by using the formula for the sum of the geometric series with $a=2$, $q=2$, and $n=9$: $a\frac{q^{n+1}-1}{q-1}=2\frac{2^{10}-1}{2-1}=2{,}046.$)

c. $\sum_{i=3}^{n+1}1=(n+1)-3+1=n-1.$

d. $\sum_{i=3}^{n+1}i=\sum_{i=0}^{n+1}i-\sum_{i=0}^{2}i=\frac{(n+1)(n+2)}{2}-3=\frac{n^2+3n-4}{2}.$

e. $\sum_{i=0}^{n-1}i(i+1)=\sum_{i=0}^{n-1}(i^2+i)=\sum_{i=0}^{n-1}i^2+\sum_{i=0}^{n-1}i=\frac{(n-1)n(2n-1)}{6}+\frac{(n-1)n}{2}$

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

Contd…

f. $\sum_{j=1}^{n} 3^{j+1} = 3\sum_{j=1}^{n} 3^{j} = 3[\sum_{j=0}^{n} 3^{j} - 1] = 3[\frac{3^{n+1}-1}{3-1} - 1] = \frac{3^{n+2}-9}{2}.$

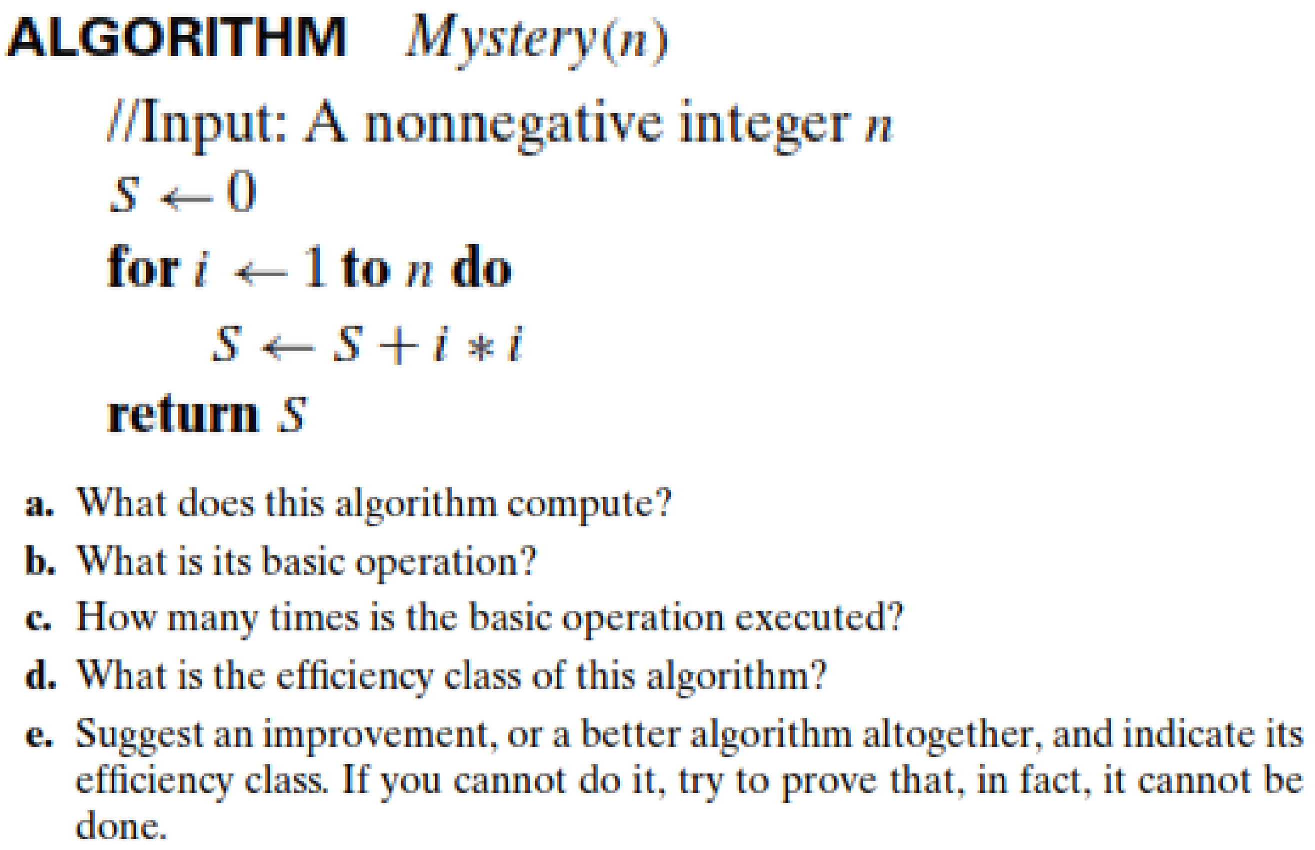
g. $\sum_{i=1}^{n}\sum_{j=1}^{n} ij = \sum_{i=1}^{n} i \sum_{j=1}^{n} j = \sum_{i=1}^{n} i\frac{n(n+1)}{2} = \frac{n(n+1)}{2}\sum_{i=1}^{n} i = \frac{n(n+1)}{2}\frac{n(n+1)}{2}$

$= \frac{n^2(n+1)^2}{4}.$

h. $\sum_{i=1}^{n} 1/i(i+1) = \sum_{i=1}^{n}(\frac{1}{i} - \frac{1}{i+1})$

$= (\frac{1}{1} - \frac{1}{2}) + (\frac{1}{2} - \frac{1}{3}) + ... + (\frac{1}{n-1} - \frac{1}{n}) + (\frac{1}{n} - \frac{1}{n+1}) = 1 - \frac{1}{n+1} = \frac{n}{n+1}.$

Exercise



```
ALGORITHM Mystery(n)
    //Input: A nonnegative integer n
    $S \leftarrow 0$
    for $i \leftarrow 1$ to $n$ do
        $S \leftarrow S + i * i$
    return $S$
```

a. What does this algorithm compute?

b. What is its basic operation?

c. How many times is the basic operation executed?

d. What is the efficiency class of this algorithm?

e. Suggest an improvement, or a better algorithm altogether, and indicate its efficiency class. If you cannot do it, try to prove that, in fact, it cannot be done.

Solution

- It computes i^2
- Basic Operation: Multiplication
- $M(n) = n$

Exercise Problem

```
ALGORITHM  Secret(A[0..n − 1])
    //Input: An array A[0..n − 1] of n real numbers
    minval ← A[0]; maxval ← A[0]
    for i ← 1 to n − 1 do
        if A[i] < minval
            minval ← A[i]
        if A[i] > maxval
            maxval ← A[i]
    return maxval − minval
```

Solution

a. Computes the range, i.e., the difference between the array's largest and smallest elements.

b. An element comparison.

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

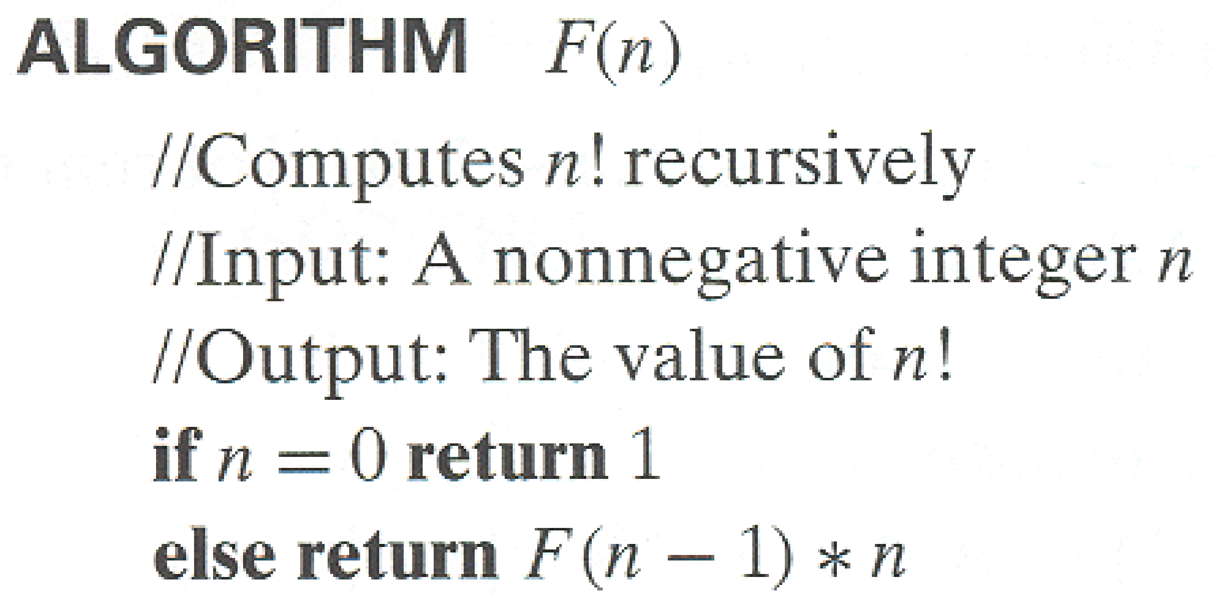
d. $\Theta(n)$.

e. An obvious improvement for some inputs (but not for the worst case) is to replace the two if-statements by the following one:

Plan for Analysis of Recursive Algorithms

- Decide on a parameter indicating an input’s size.
- Identify the algorithm’s basic operation.
- Check whether the number of times the basic op. is executed may vary on different inputs of the same size.
- Set up a recurrence relation with an appropriate initial condition expressing the number of times the basic op. is executed.
- Solve the recurrence (or, at the very least, establish its solution’s order of growth) by backward substitutions or another method.

Factorial



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

Analysis

- Input Size: N bit representation
- Basic Operation: Multiplication

$$M(n) = \underbrace{M(n-1)}_{\substack{\text{to compute}\\ F(n-1)}} + \underbrace{1}_{\substack{\text{to multiply}\\ F(n-1)\text{ by } n}} \quad \text{for } n > 0.$$

- Basis condition: M(0) = 0 No multiplication is required.

Backward Substitution

$$
\begin{aligned}
M(n) &= M(n-1) + 1 && \text{substitute } M(n-1) = M(n-2) + 1 \\
&= [M(n-2) + 1] + 1 = M(n-2) + 2 && \text{substitute } M(n-2) = M(n-3) + 1 \\
&= [M(n-3) + 1] + 2 = M(n-3) + 3.
\end{aligned}
$$

$$M(n) = M(n-1) + 1 = \cdots = M(n-i) + i = \cdots = M(n-n) + n = n.$$

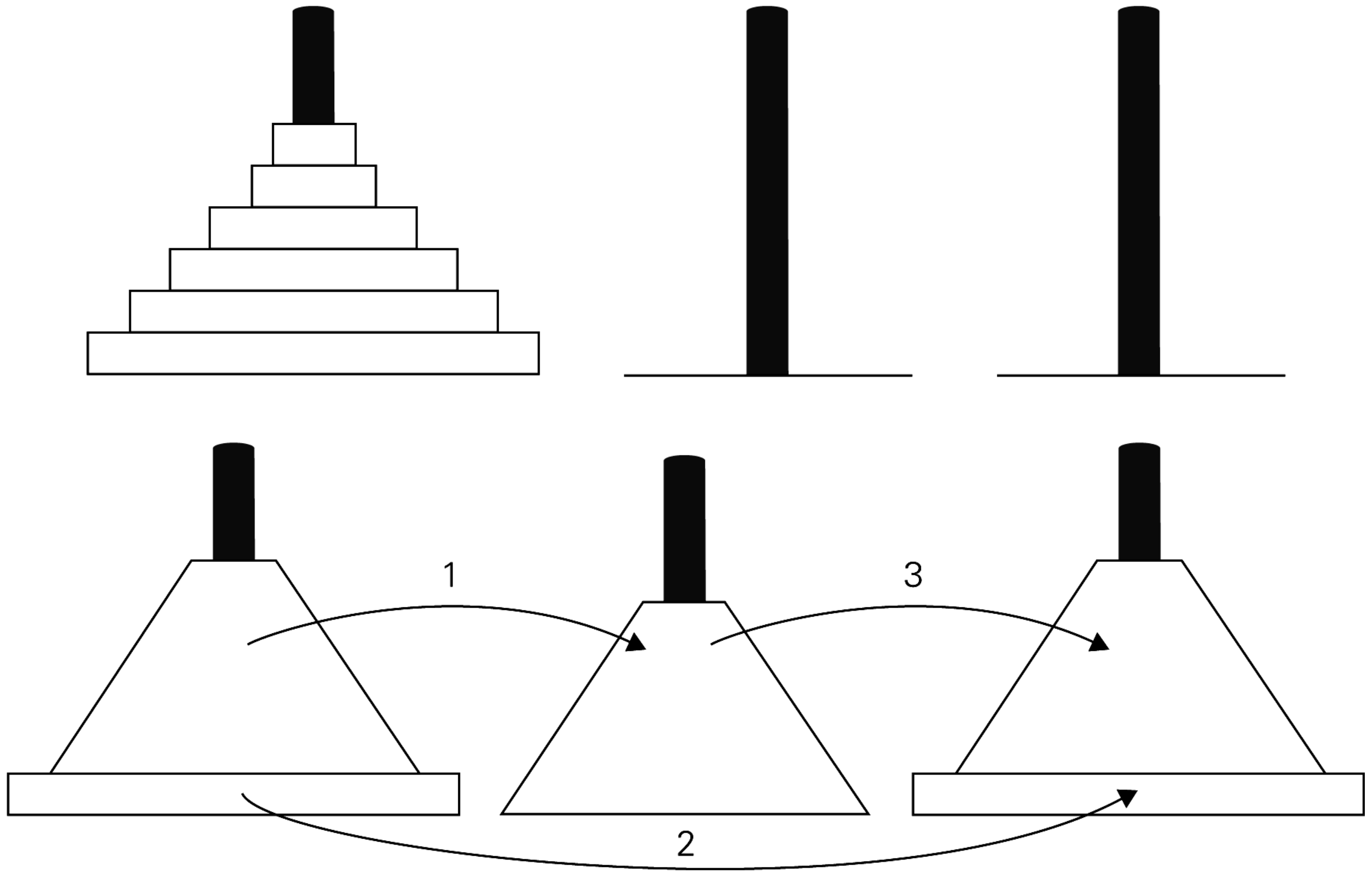


FIGURE 2.4 Recursive solution to the Tower of Hanoi puzzle

Analysis

$$M(n) = M(n-1) + 1 + M(n-1) \quad \text{for } n > 1.$$

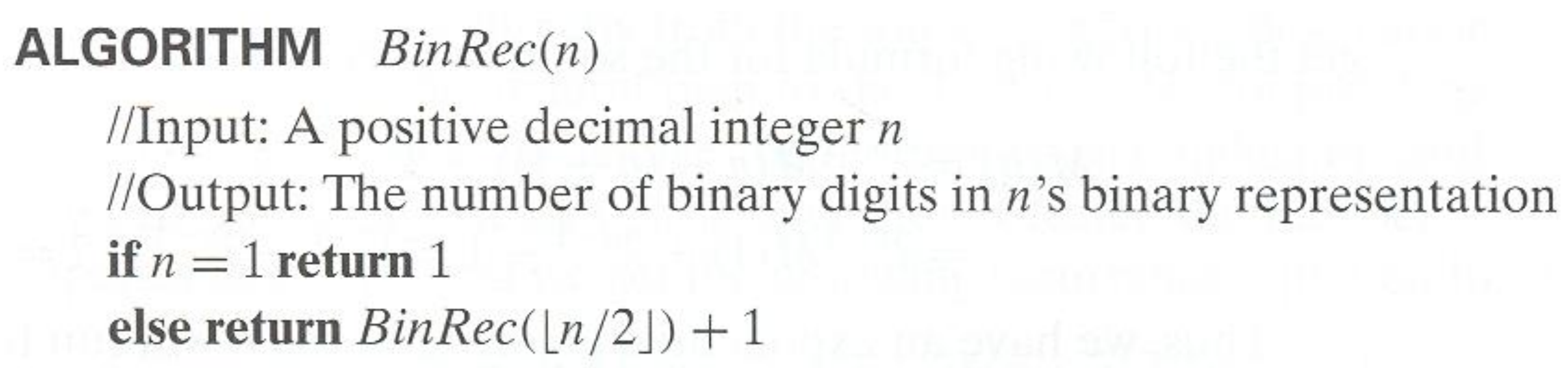
$$M(n) = 2M(n-1) + 1 \quad \text{for } n > 1,$$
$$M(1) = 1.$$

$$M(n) = 2M(n-1) + 1 \qquad \text{sub. } M(n-1) = 2M(n-2) + 1$$
$$= 2[2M(n-2) + 1] + 1 = 2^2M(n-2) + 2 + 1 \quad \text{sub. } M(n-2) = 2M(n-3) + 1$$
$$= 2^2[2M(n-3) + 1] + 2 + 1 = 2^3M(n-3) + 2^2 + 2 + 1.$$

$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 = 2^i M(n-i) + 2^i - 1.$$

$$M(n) = 2^{n-1}M(n-(n-1)) + 2^{n-1} - 1$$
$$= 2^{n-1}M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1.$$

Decimal to Binary digits



```
ALGORITHM  BinRec(n)
    //Input: A positive decimal integer n
    //Output: The number of binary digits in n's binary representation
    if $n = 1$ return 1
    else return $BinRec(\lfloor n/2 \rfloor) + 1$
```

Analysis

- Input Size: No of bits
- Basic operation: additions made in computing *BinRec(n/2) plus one more addition*

$$A(n) = A(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1.$$

$$A(1) = 0.$$

Smoothness rule

$$A(2^k) = A(2^{k-1}) + 1 \quad \text{for } k > 0,$$
$$A(2^0) = 0.$$

Now backward substitutions encounter no problems:

$$\begin{aligned}
A(2^k) &= A(2^{k-1}) + 1 && \text{substitute } A(2^{k-1}) = A(2^{k-2}) + 1 \\
&= [A(2^{k-2}) + 1] + 1 = A(2^{k-2}) + 2 && \text{substitute } A(2^{k-2}) = A(2^{k-3}) + 1 \\
&= [A(2^{k-3}) + 1] + 2 = A(2^{k-3}) + 3 && \dots \\
&\dots \\
&= A(2^{k-i}) + i \\
&\dots \\
&= A(2^{k-k}) + k.
\end{aligned}$$

Thus, we end up with

$$A(2^k) = A(1) + k = k,$$

or, after returning to the original variable $n = 2^k$ and hence $k = \log_2 n$,

$$A(n) = \log_2 n \in \Theta(\log n).$$

Examples

Solve the following recurrence relations.

a. $x(n) = x(n-1) + 5$ for $n > 1$, $x(1) = 0$

b. $x(n) = 3x(n-1)$ for $n > 1$, $x(1) = 4$

c. $x(n) = x(n-1) + n$ for $n > 0$, $x(0) = 0$

d. $x(n) = x(n/2) + n$ for $n > 1$, $x(1) = 1$ (solve for $n = 2^k$)

e. $x(n) = x(n/3) + 1$ for $n > 1$, $x(1) = 1$ (solve for $n = 3^k$)

Solution

$$\begin{aligned}
x(n) &= x(n-1)+5 \\
&= [x(n-2)+5]+5 = x(n-2)+5\cdot 2 \\
&= [x(n-3)+5]+5\cdot 2 = x(n-3)+5\cdot 3 \\
&= \ldots \\
&= x(n-i)+5\cdot i \\
&= \ldots \\
&= x(1)+5\cdot(n-1) = 5(n-1).
\end{aligned}$$

Solution

$$\begin{aligned}
x(n) &= 3x(n-1) \\
&= 3[3x(n-2)] = 3^2x(n-2) \\
&= 3^2[3x(n-3)] = 3^3x(n-3) \\
&= \ldots \\
&= 3^ix(n-i) \\
&= \ldots \\
&= 3^{n-1}x(1) = 4 \cdot 3^{n-1}.
\end{aligned}$$

Solution

$$
\begin{aligned}
x(n) &= x(n-1)+n \\
&= [x(n-2)+(n-1)]+n = x(n-2)+(n-1)+n \\
&= [x(n-3)+(n-2)]+(n-1)+n = x(n-3)+(n-2)+(n-1)+n \\
&= \dots \\
&= x(n-i)+(n-i+1)+(n-i+2)+\dots+n \\
&= \dots \\
&= x(0)+1+2+\dots+n = \frac{n(n+1)}{2}.
\end{aligned}
$$

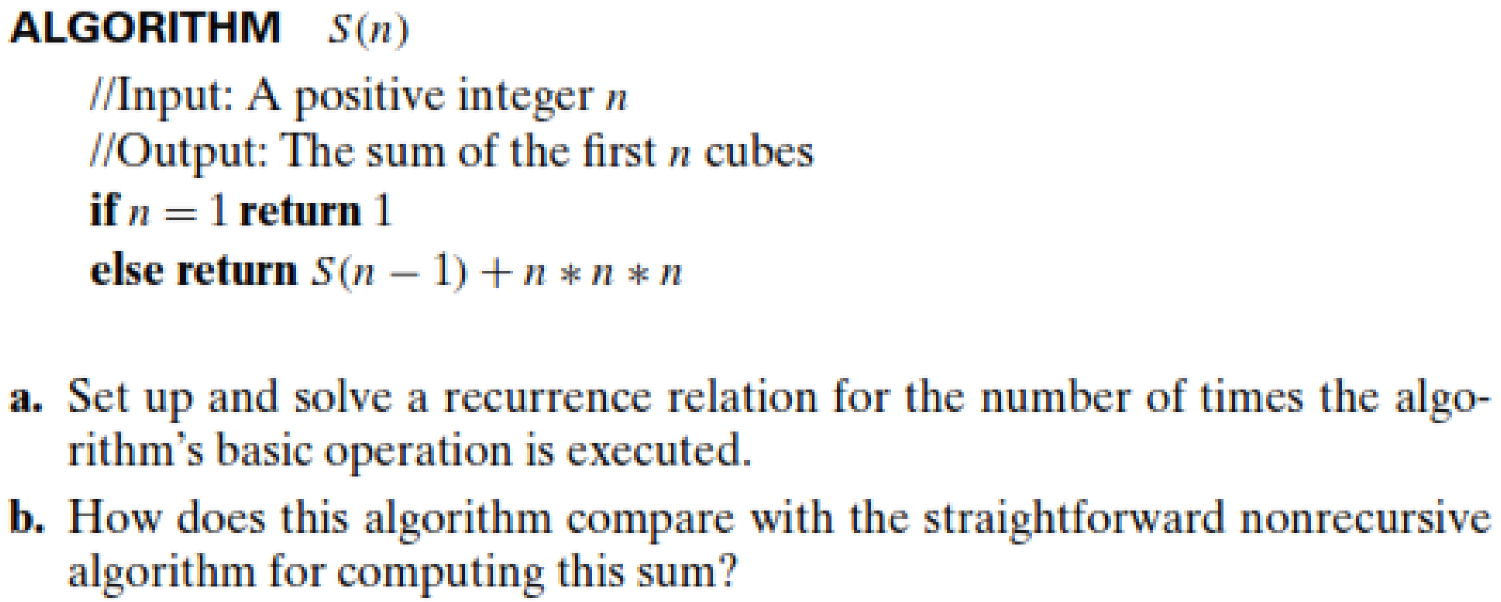
Solution

$$\begin{aligned}
x(2^k) &= x(2^{k-1}) + 2^k \\
&= [x(2^{k-2}) + 2^{k-1}] + 2^k = x(2^{k-2}) + 2^{k-1} + 2^k \\
&= [x(2^{k-3}) + 2^{k-2}] + 2^{k-1} + 2^k = x(2^{k-3}) + 2^{k-2} + 2^{k-1} + 2^k \\
&= \ldots \\
&= x(2^{k-i}) + 2^{k-i+1} + 2^{k-i+2} + \ldots + 2^k \\
&= \ldots \\
&= x(2^{k-k}) + 2^1 + 2^2 + \ldots + 2^k = 1 + 2^1 + 2^2 + \ldots + 2^k \\
&= 2^{k+1} - 1 = 2 \cdot 2^k - 1 = 2n - 1.
\end{aligned}$$

Solution

$$\begin{aligned}
x(3^k) &= x(3^{k-1}) + 1 \\
&= [x(3^{k-2}) + 1] + 1 = x(3^{k-2}) + 2 \\
&= [x(3^{k-3}) + 1] + 2 = x(3^{k-3}) + 3 \\
&= \ldots \\
&= x(3^{k-i}) + i \\
&= \ldots \\
&= x(3^{k-k}) + k = x(1) + k = 1 + \log_3 n.
\end{aligned}$$

Exercise Probelm



```
ALGORITHM  $S(n)$
    //Input: A positive integer $n$
    //Output: The sum of the first $n$ cubes
    if $n = 1$ return 1
    else return $S(n - 1) + n * n * n$
```

a. Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed.

b. How does this algorithm compare with the straightforward nonrecursive algorithm for computing this sum?

Analysis

- Basic operation: Multiplication

  M(n) = M(n-1) + 2

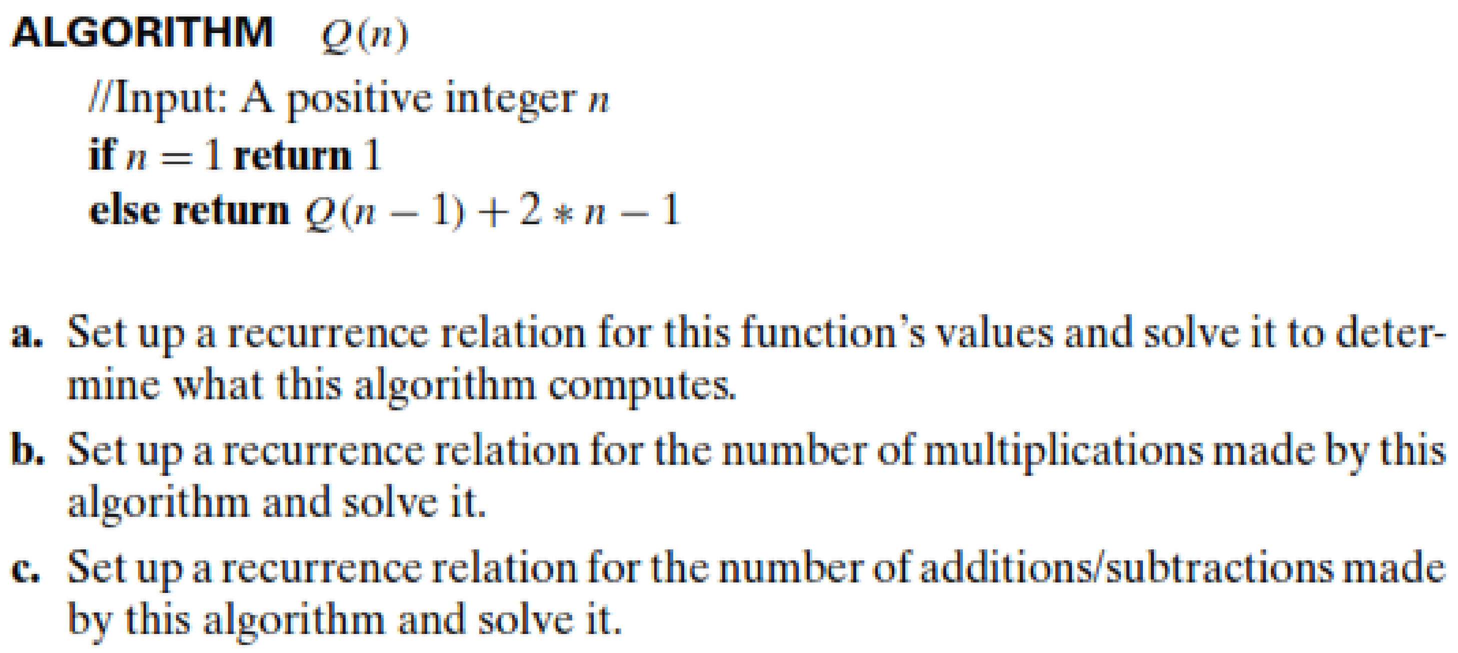
  M(1) = 0.

$$
\begin{aligned}
M(n) &= M(n-1)+2 \\
&= [M(n-2)+2]+2 = M(n-2)+2+2 \\
&= [M(n-3)+2]+2+2 = M(n-3)+2+2+2 \\
&= \ldots \\
&= M(n-i)+2i \\
&= \ldots \\
&= M(1)+2(n-1) = 2(n-1).
\end{aligned}
$$

Analysis

- Straightforward:
- M(n) = sum of (i=2 to n) 2 = 2(n-1)

Exercise Problem



```
ALGORITHM  Q(n)
    //Input: A positive integer n
    if $n = 1$ return 1
    else return $Q(n - 1) + 2 * n - 1$
```

a. Set up a recurrence relation for this function's values and solve it to determine what this algorithm computes.

b. Set up a recurrence relation for the number of multiplications made by this algorithm and solve it.

c. Set up a recurrence relation for the number of additions/subtractions made by this algorithm and solve it.

Solution

$$Q(n) = Q(n-1) + 2n - 1 \quad \text{for } n > 1, \quad Q(1) = 1.$$

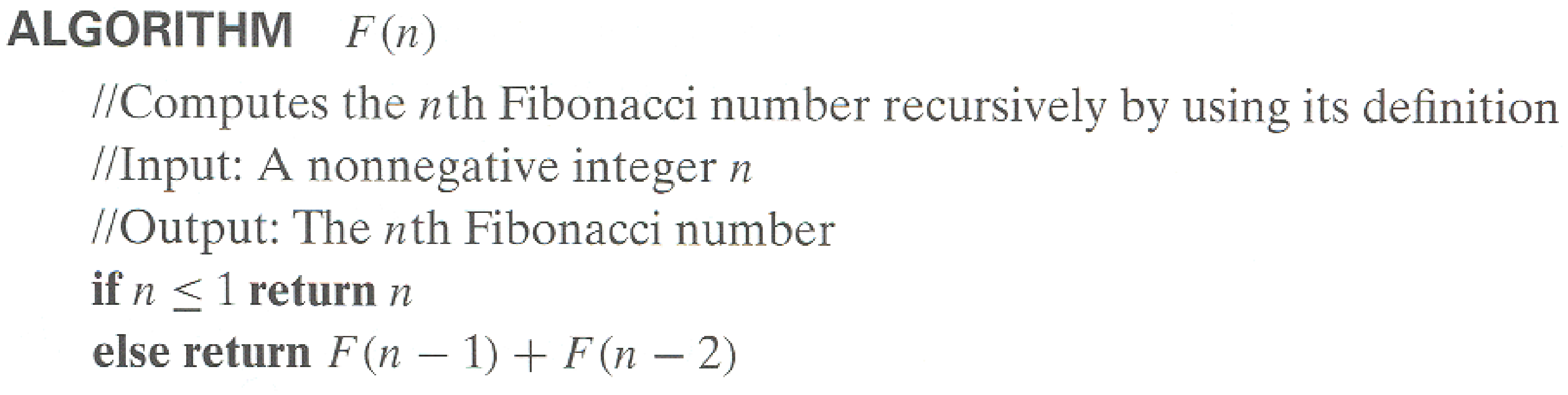
$$Q(n-1) + 2n - 1 = (n-1)^2 + 2n - 1 = n^2.$$

$$M(n) = M(n-1) + 1 \ \text{ for } n > 1, \ \ M(1) = 0.$$

$$C(n) = C(n-1) + 3$$

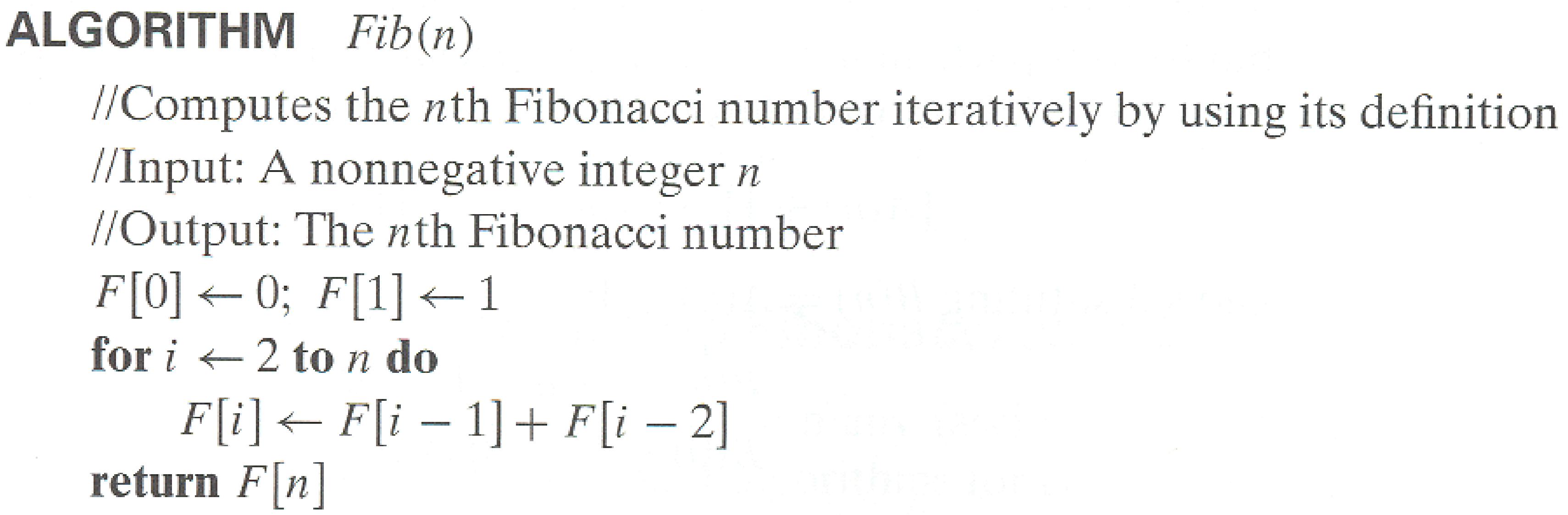
3 includes n-1, adding 2n, -
1

Fibonacci Series



```
ALGORITHM  $F(n)$
    //Computes the $n$th Fibonacci number recursively by using its definition
    //Input: A nonnegative integer $n$
    //Output: The $n$th Fibonacci number
    if $n \le 1$ return $n$
    else return $F(n-1) + F(n-2)$
```

Iterative algorithm



```
ALGORITHM  $Fib(n)$
    //Computes the $n$th Fibonacci number iteratively by using its definition
    //Input: A nonnegative integer $n$
    //Output: The $n$th Fibonacci number
    $F[0] \leftarrow 0;\ F[1] \leftarrow 1$
    for $i \leftarrow 2$ to $n$ do
        $F[i] \leftarrow F[i-1] + F[i-2]$
    return $F[n]$
```

Analysis

- Homogeneous linear recurrence relation.