

Unit I – Important problem types and analysis framework

V. Balasubramanian

Algorithm properties

1. Finiteness

∴ terminates after a finite number of steps

2. Definiteness

∴ rigorously and unambiguously specified

3. Input

∴ valid inputs are clearly specified

4. Output

∴ can be proved to produce the correct output given a valid input

5. Effectiveness

∴ steps are sufficiently simple and basic

Examples Page 42 problem 3

- ❧ 1. Which of the following formulas can be considered an algorithm for computing the area of a triangle whose side lengths are given positive numbers a , b and c ?
- (a) $S = \sqrt{p(p-a)(p-b)(p-c)}$ where $p = (a+b+c)/2$.
 - (b) $S = 1/2 \ bc \sin A$ where A is the angle between sides b and c .
 - (c) $S = 1/2 \ a h_a$ where h_a is the height to base a .

Solution

- ❧ The formula in (a) is a valid algorithm, because it unambiguously specifies HOW to calculate the area S from the given inputs a , b and c .
- ❧ The formula in (b) can NOT be considered a valid algorithm, because it does not clearly specify how to compute the angle A , even if we have a subroutine to compute $\text{Sin}(A)$.
- ❧ The formula in (c) does not specify how to calculate the height ha , and thus it can not be considered an algorithm.

Example 2 page 42 no 9

- 2. Consider the following algorithm for finding the distance between the two closest elements in an array of numbers.
- ALGORITHM** MinDistance($A[0 \dots n-1]$)
- // Input: Array $A[0..n-1]$ of numbers
- // Output: Minimum distance between two of its elements
- $D_{min} \leftarrow \infty$
- for $i \leftarrow 0$ to $n-1$ do
- for $j \leftarrow 1$ to $n-1$ do
- if $i \neq j$ and $|A[i] - A[j]| < d_{min}$
- $d_{min} \leftarrow |A[i] - A[j]|$
- return d_{min}

Contd...

- ⌚ **Make as many improvements as you can in this algorithmic solution to the problem. (If you need to, you may change the algorithm altogether; if not, improve the implementation given.)**

Solution

ALGORITHM MinDistance($A[0 \dots n-1]$)

// Inout: Array $A[0..n-1]$ of numbers

// Output: Minimum distance between two of its elements

$d_{\min} \leftarrow \infty$

for $i \leftarrow 0$ **to** $n-2$ **do**

for $j \leftarrow i+1$ **to** $n-1$ **do**

if $|A[i] - A[j]| < d_{\min}$

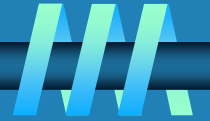
$d_{\min} \leftarrow |A[i] - A[j]|$

return d_{\min}

Contd...

- ⌚ A further improvement could be made to design a new algorithm for the problem: First use an $O(n \log n)$ sorting algorithm to first sort the array A and then use a linear algorithm to scan through the sorted array A and compute dmin.

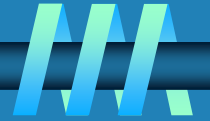
Important problem types



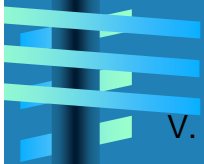
- ∩ sorting
- ∩ searching
- ∩ string processing
- ∩ graph problems
- ∩ combinatorial problems
- ∩ geometric problems
- ∩ numerical problems



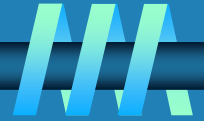
Sorting



- ❧ We rearrange the items of a given list in ascending / descending order.
- ❧ The relation between elts are in total ordering.
- ❧ To sort, we need a piece of information, called key to guide sorting.
- ❧ Good sorting algorithms $O(n \log n)$
- ❧ Sorting algorithm should have 2 properties
 - Stable
 - In place



Contd...



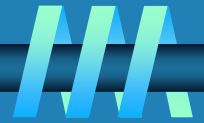
❧ Stable:

- Sorting algo is stable if it preserves the relative order of any 2 equal elements in its input.
- i.e if the input list contains 2 equal elements in position i and j where $i < j$, then in sorted list they have to be in positions i' and j' .
- Example: students name and GPA.
if we sort according to GPA, students with same GPA will still be in sorted alphabetically.

❧ In place: An algorithm is said to be in place if it does not require extra memory.



Example 3 page 47 exercise 1



- 3. Consider the algorithm for the sorting problem that sorts an array by counting, for each of its elements, the number of smaller elements, and then uses this information to put the element in its appropriate position in the sorted array:
- First for loop $i=0$ to $n-1$ & one more for loop after count for $i=0$ to $n-2$

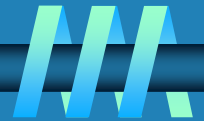
ALGORITHM ComparisonCountingSort($A[0..n-1]$)

// Input: Array $A[0..n-1]$ of orderable values

//Output: Array $S[0..n-1]$ of A 's elements sorted in non-descending order

```
for  $i \leftarrow 0$  to  $n - 2$  do
    Count[i]  $\leftarrow 0$ 
for  $j \leftarrow i + 1$  to  $n - 1$  do
    if  $A[i] < A[j]$ 
        Count[j]  $\leftarrow$  Count[j] + 1
    else Count[i]  $\leftarrow$  Count[i] + 1
for  $i \leftarrow 0$  to  $n - 1$  do
    S[count[i]]  $\leftarrow$  A[i]
```

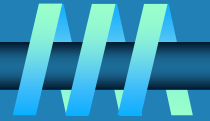
Correct algo



```
Algorithm ComparisonCountingSort( $A[0..n-1]$ ,  $S[0..n-1]$ )  
//Sorts an array by comparison counting  
//Input: Array  $A[0..n-1]$  of orderable values  
//Output: Array  $S[0..n-1]$  of  $A$ 's elements sorted in nondecreasing order  
for  $i \leftarrow 0$  to  $n-1$  do  
     $Count[i] \leftarrow 0$   
    for  $i \leftarrow 0$  to  $n-2$  do  
        for  $j \leftarrow i+1$  to  $n-1$  do  
            if  $A[i] < A[j]$   
                 $Count[j] \leftarrow Count[j] + 1$   
            else  $Count[i] \leftarrow Count[i] + 1$   
    for  $i \leftarrow 0$  to  $n-1$  do  
         $S[Count[i]] \leftarrow A[i]$ 
```



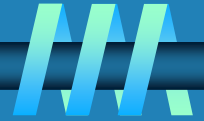
Contd...



- ❧ Apply this algorithm to sorting the list [60, 35, 81, 98, 14, 47]
- ❧ Is this algorithm stable?
- ❧ Is the algorithm in place?



Solution



Q Answer:

Q (a) Count = [3, 1, 4, 5, 0, 2]

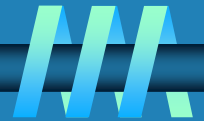
Q S[0..5] = [14, 35, 47, 60, 81, 98]

Q (b) No. The algorithm is NOT stable. When $A[i] = A[j]$ ($i < j$), the algorithm (in the else part) will actually increment the count for $A[i]$ and thus the Count[i] will be bigger than Count[j]. So it will put $A[i]$ to a location $S[k]$ and $A[j]$ to a location $S[l]$ with $k > l$.

Q (c) No, the algorithm is NOT in place, because it uses two extra arrays S and Count.



solution



a. Sorting 60, 35, 81, 98, 14, 47 by comparison counting will work as follows:

Array $A[0..5]$

60	35	81	98	14	47
----	----	----	----	----	----

Initially

Count[]

0	0	0	0	0	0
---	---	---	---	---	---

After pass $i = 0$

Count[]

3	0	1	1	0	0
---	---	---	---	---	---

After pass $i = 1$

Count[]

	1	2	2	0	1
--	---	---	---	---	---

After pass $i = 2$

Count[]

		4	3	0	1
--	--	---	---	---	---

After pass $i = 3$

Count[]

			5	0	1
--	--	--	---	---	---

After pass $i = 4$

Count[]

				0	2
--	--	--	--	---	---

Final state

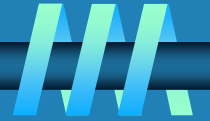
Count[]

3	1	4	5	0	2
---	---	---	---	---	---

Array $S[0..5]$

14	35	47	60	81	98
----	----	----	----	----	----

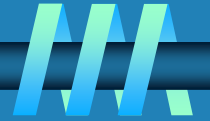
solution



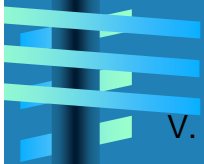
- b. The algorithm is not stable. Consider, as a counterexample, the result of its application to $1', 1''$.
- c. The algorithm is not in place because it uses two extra arrays of size n : *Count* and *S*.



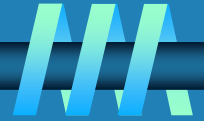
Searching



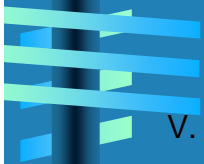
- ❧ It deals with finding a given value, called key.
- ❧ In searching algorithms, searching has to be considered in conjunction with addition and deletion of an data item.



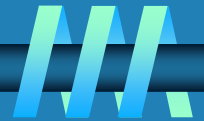
String matching



- ⌚ **Non-numeric data processing**
- ⌚ **String is a sequence of characters from an alphabet**
- ⌚ **String matching is particularly important to find a word or pattern**
- ⌚ **DNA alphabet { A,C,G,T }**



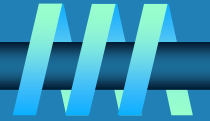
Graph problem



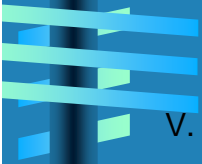
- ⌚ It is a collection of points called vertices and edges.
- ⌚ Real life applications – commn networks, project scheduling
- ⌚ Graph traversal algorithms – travelling salesman, shortest path algo, topological sorting
- ⌚ Adjacency list, matrix
- ⌚ Graph coloring problem
- ⌚ sudoko



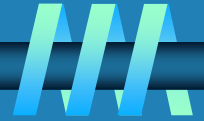
Combinatorial problem



- ⌚ Most difficult problems in computing
- ⌚ Branch of pure maths
- ⌚ Grows extremely fast
- ⌚ Ex: deck of cards ordering $52! = 8.0658 \times 10^{67}$



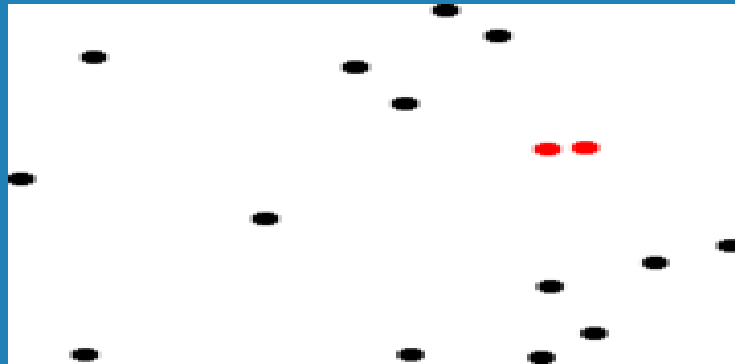
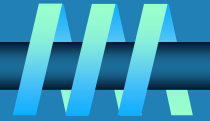
Geometrical problem



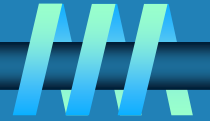
- ⌚ Deals with points, lines and polygons
- ⌚ Simple shapes also triangle, circles.
- ⌚ CAD/CAM
- ⌚ Robotics – motor planning and visibility
- ⌚ GIS (geometric location)
- ⌚ Closest pair: if given n points, in n -dimensional space find a pair of points with smallest distance.



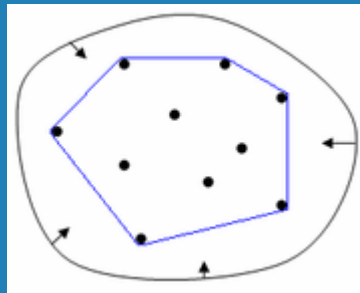
Closest pair



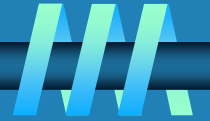
Convex Hull



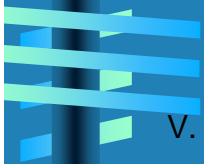
It finds smallest polygon that would include all points of a given set.



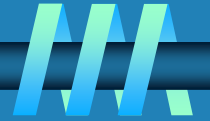
Numerical problems



- ⌚ It involves mathematical objects of continuous nature
- ⌚ Computing definite integrals
- ⌚ It is solved only by approximation algorithm.



Fundamental data structures



∩ list

- array
- linked list
- string

∩ stack

∩ queue

∩ priority queue

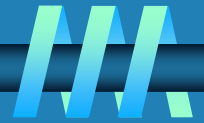
∩ graph

∩ tree

∩ set and dictionary



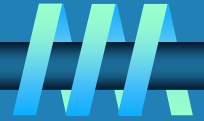
Examples



- ⌚ For each of the following algorithms, indicate (i) a natural size metric for its inputs; (ii) its basic operation; (iii) whether the basic operation count can be different for inputs of the same size:
 - ⌚ a. computing the sum of n numbers
 - ⌚ b. computing $n!$
 - ⌚ c. finding the largest element in a list of n numbers
 - ⌚ d. Euclid's algorithm
 - ⌚ e. sieve of Eratosthenes
 - ⌚ f. pen-and-pencil algorithm for multiplying two n -digit decimal integers



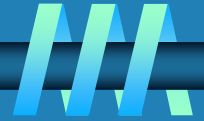
solution



- Ω a. (i) n ; (ii) addition of two numbers; (iii) no
- Ω b. (i) the magnitude of n , i.e., the number of bits in its binary representation;
 Ω (ii) multiplication of two integers; (iii) no
- Ω c. (i) n ; (ii) comparison of two numbers; (iii) no (for the standard
 Ω list scanning algorithm)
- Ω d. (i) either the magnitude of the larger of two input numbers, or the
 Ω magnitude of the smaller of two input numbers, or the sum of the
magnitudes of two input numbers; (ii) modulo division; (iii) yes
- Ω e. (i) the magnitude of n , i.e., the number of bits in its binary representation;
 Ω (ii) elimination of a number from the list of remaining candidates
 Ω to be prime; (iii) no
- Ω f. (i) n ; (ii) multiplication of two digits; (iii) no



Exercise 2



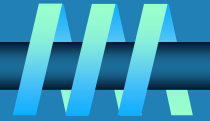
- ⌚ a. Consider the definition-based algorithm for adding two n -by- n matrices. What is its basic operation? How many times is it performed as a function of the matrix order n ? As a function of the total number of elements in the input matrices?
- ⌚ b. Answer the same questions for the definition-based algorithm for matrix multiplication.



Solution

- ⌚ Addition of two numbers. It's performed n^2 times (once for each of n^2 elements in the matrix being computed). Since the total number of elements in two given matrices is $N = 2n^2$, the total number of additions can also be expressed as $n^2 = N/2$.

Exercise 3



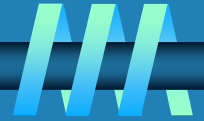
- Consider a variation of sequential search that scans a list to return the number of occurrences of a given search key in the list. Will its efficiency differ from the efficiency of classic sequential search?



Solution

- ⌚ This algorithm will always make n key comparisons on every input of size n ,

Exercise



Q Determine whether following is true or false.

- $100n + 5 \in O(n^2)$

Q Solution:

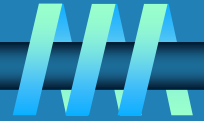
Q $100n + 5 \leq 100n + n$ for all $n \geq 5$

Q $101n \leq 101n^2$

Q Where $c = 101$ and $n_0 = 5$.



Ex 2



Q Determine whether following is true or false.

- $n^2 + 10n \in O(n^2)$

Q Solution:

Q When $c=2$, $n_0=10$,

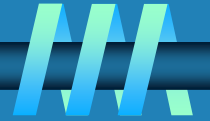
Q Because for

Q $n^2 + 10n \leq 2 n^2$ for all $n \geq n_0$

Q $n^2 + 10n \leq 2 n^2$ for all $n \geq 10$



Ex 3

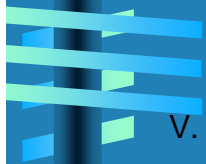


Q Show that $5n^2 \in O(n^2)$

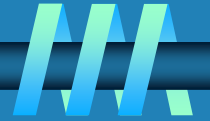
Q Solution:

Q For $n \geq 0$, we can take $c = 5$, $n_0 = 0$.

Q $5n^2 \leq 5n^2$ for $n \geq 0$



Ex 4

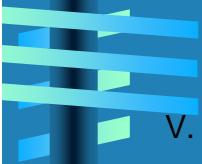


Q Show that $n(n-1)/2 \in O(n^2)$

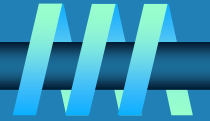
Q Solution:

Q For $n \geq 0$, $n(n-1)/2 \leq n(n)/2 \leq n^2$

Q $c = 1/2$ and $n_0 = 0$.



Ex



Q Show that $n^2 \in O(n^2 + 10n)$

Q Solution:

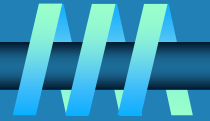
Q For $n \geq 0$

Q $n^2 \leq 1 * (n^2 + 10n)$

Q $c = 1$ and $n_0 = 0$.



Ex



Q Show that $n \in O(n^2)$

Q Solution:

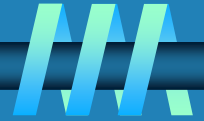
Q For $n \geq 1$,

Q $n \leq 1 * n^2$

Q $c = 1$ and $n_0 = 1$.



Ex



Q Find the order of the function $3n + 2$

Q Solution:

Q $t(n) = 3n + 2$

Q $t(n) \leq cn$

Q $3n + 2 \leq cn$

Q $3n + 2 \leq 4n$ where $c = 4$

Q If $n = 1$,

Q $3 \times 1 + 2 \leq 4$ i.e $5 \leq 4$

Q If $n = 2$,

Q $3 \times 2 + 2 \leq 8$ i.e $8 \leq 8$

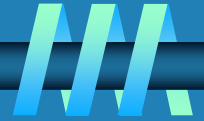
Q If $n = 3$,

Q $3 \times 3 + 2 \leq 12$ i.e $11 \leq 12$

Q $t(n) \in O(n)$ where $c = 4$ and $n_0 = 2$, which is called breakeven point.



Ex



Find the order of the function

$$t(n) = 10n^2 + 4n + 2$$

Solution:

$$t(n) \leq cg(n) \text{ for all } n \geq n_0$$

$$10n^2 + 4n + 2 \leq cn^2 \text{ where } g(n) = n^2$$

$$10n^2 + 4n + 2 \leq cn^2 \text{ where } c = 11$$

$$\text{If } n = 1, 16 \leq 11$$

$$\text{If } n = 2, 50 \leq 44$$

$$\text{If } n = 3, 104 \leq 99$$

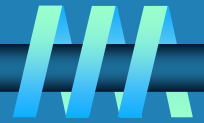
$$\text{If } n = 4, 178 \leq 176$$

$$\text{If } n = 5, 272 \leq 275$$

$$\text{If } n = 6, 386 \leq 396$$

the given function $10n^2 + 4n + 2 \in O(n^2)$ when $c = 11$ and $n_0 = 5$, which is breakeven point.

$O(n^2)$



$$3 \log n + 8$$

$$5n + 7$$

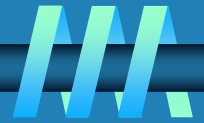
$$2 \log n$$

$$4n^2$$

$$6n^2 + 9$$

$$5n^2 + 2n$$

Omega(n^2)



$$4 n^2$$

$$6 n^2 + 9$$

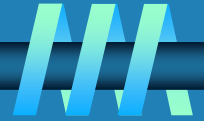
$$5 n^2 + 2 n$$

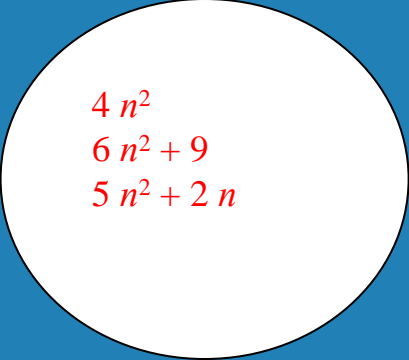
$$4 n^3 + 3 n^2$$

$$6 n^6 + 3 n^4$$

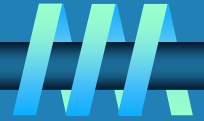


Theta n^2




$$\begin{aligned} &4 n^2 \\ &6 n^2 + 9 \\ &5 n^2 + 2 n \end{aligned}$$

Exercise

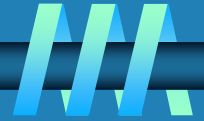


For each of the following functions, indicate how much the function's value will change if its argument is increased fourfold.

- a. $\log_2 n$ b. \sqrt{n} c. n d. n^2 e. n^3 f. 2^n



solution



a. $\log_2 4n - \log_2 n = (\log_2 4 + \log_2 n) - \log_2 n = 2.$

b. $\frac{\sqrt{4n}}{\sqrt{n}} = 2.$

c. $\frac{4n}{n} = 4.$

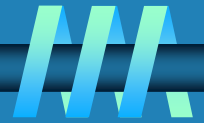
d. $\frac{(4n)^2}{n^2} = 4^2.$

e. $\frac{(4n)^3}{n^3} = 4^3.$

f. $\frac{2^{4n}}{2^n} = 2^{3n} = (2^n)^3.$



Exercise



Indicate whether the first function of each of the following pairs has a smaller, same, or larger order of growth (to within a constant multiple) than the second function.

a. $n(n+1)$ and $2000n^2$

b. $100n^2$ and $0.01n^3$

c. $\log_2 n$ and $\ln n$

d. $\log_2^2 n$ and $\log_2 n^2$

e. 2^{n-1} and 2^n

f. $(n-1)!$ and $n!$



solution

a. $n(n+1) \approx n^2$ has the same order of growth (quadratic) as $2000n^2$ to within a constant multiple.

b. $100n^2$ (quadratic) has a lower order of growth than $0.01n^3$ (cubic).

c. Since changing a logarithm's base can be done by the formula

$$\log_a n = \log_a b \log_b n,$$

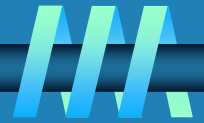
all logarithmic functions have the same order of growth to within a constant multiple.

d. $\log_2^2 n = \log_2 n \log_2 n$ and $\log_2 n^2 = 2 \log n$. Hence $\log_2^2 n$ has a higher order of growth than $\log_2 n^2$.

e. $2^{n-1} = \frac{1}{2}2^n$ has the same order of growth as 2^n to within a constant multiple.

f. $(n-1)!$ has a lower order of growth than $n! = (n-1)!n$.

Exercise

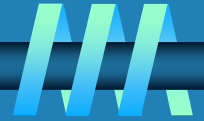


Use the most appropriate notation among O , Θ , and Ω to indicate the time efficiency class of sequential search (see Section 2.1)

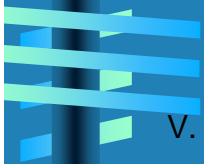
- a. in the worst case.
- b. in the best case.
- c. in the average case.



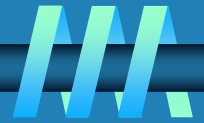
Solution



- a. Since $C_{worst}(n) = n$, $C_{worst}(n) \in \Theta(n)$.
- b. Since $C_{best}(n) = 1$, $C_{best}(1) \in \Theta(1)$.
- c. Since $C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p) = (1 - \frac{p}{2})n + \frac{p}{2}$ where $0 \leq p \leq 1$, $C_{avg}(n) \in \Theta(n)$.



Exercise



Use the informal definitions of O , Θ , and Ω to determine whether the following assertions are true or false.

a. $n(n+1)/2 \in O(n^3)$

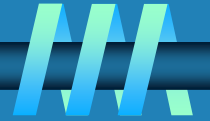
b. $n(n+1)/2 \in O(n^2)$

c. $n(n+1)/2 \in \Theta(n^3)$

d. $n(n+1)/2 \in \Omega(n)$

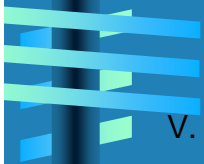


Solution

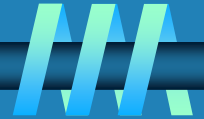


$n(n+1)/2 \approx n^2/2$ is quadratic. Therefore

- a. $n(n+1)/2 \in O(n^3)$ is true.
- b. $n(n+1)/2 \in O(n^2)$ is true.
- c. $n(n+1)/2 \in \Theta(n^3)$ is false.
- d. $n(n+1)/2 \in \Omega(n)$ is true.



Exercise



For each of the following functions, indicate the class $\Theta(g(n))$ the function belongs to. (Use the simplest $g(n)$ possible in your answers.) Prove your assertions.

a. $(n^2 + 1)^{10}$

b. $\sqrt{10n^2 + 7n + 3}$

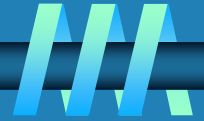
c. $2n \lg(n + 2)^2 + (n + 2)^2 \lg \frac{n}{2}$

d. $2^{n+1} + 3^{n-1}$

e. $\lfloor \log_2 n \rfloor$



solution



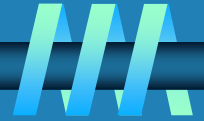
a. Informally, $(n^2 + 1)^{10} \approx (n^2)^{10} = n^{20} \in \Theta(n^{20})$ Formally,

$$\lim_{n \rightarrow \infty} \frac{(n^2 + 1)^{10}}{n^{20}} = \lim_{n \rightarrow \infty} \frac{(n^2 + 1)^{10}}{(n^2)^{10}} = \lim_{n \rightarrow \infty} \left(\frac{n^2 + 1}{n^2} \right)^{10} = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n^2} \right)^{10} = 1.$$

Hence $(n^2 + 1)^{10} \in \Theta(n^{20})$.



Contd...



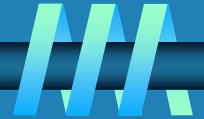
b. Informally, $\sqrt{10n^2 + 7n + 3} \approx \sqrt{10n^2} = \sqrt{10}n \in \Theta(n)$. Formally,

$$\lim_{n \rightarrow \infty} \frac{\sqrt{10n^2 + 7n + 3}}{n} = \lim_{n \rightarrow \infty} \sqrt{\frac{10n^2 + 7n + 3}{n^2}} = \lim_{n \rightarrow \infty} \sqrt{10 + \frac{7}{n} + \frac{3}{n^2}} = \sqrt{10}.$$

Hence $\sqrt{10n^2 + 7n + 3} \in \Theta(n)$.



Contd...



c. $2n \lg(n+2)^2 + (n+2)^2 \lg \frac{n}{2} = 2n2\lg(n+2) + (n+2)^2(\lg n - 1) \in \Theta(n \lg n) + \Theta(n^2 \lg n) = \Theta(n^2 \lg n).$

d. $2^{n+1} + 3^{n-1} = 2^n 2 + 3^n \frac{1}{3} \in \Theta(2^n) + \Theta(3^n) = \Theta(3^n).$

e. Informally, $\lfloor \log_2 n \rfloor \approx \log_2 n \in \Theta(\log n)$. Formally, by using the inequalities $x - 1 < \lfloor x \rfloor \leq x$ (see Appendix A), we obtain an upper bound

$$\lfloor \log_2 n \rfloor \leq \log_2 n$$

and a lower bound

$$\lfloor \log_2 n \rfloor > \log_2 n - 1 \geq \log_2 n - \frac{1}{2} \log_2 n \text{ (for every } n \geq 4) = \frac{1}{2} \log_2 n.$$

Hence $\lfloor \log_2 n \rfloor \in \Theta(\log_2 n) = \Theta(\log n).$



Exercise

TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

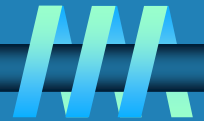
a. Table 2.1 contains values of several functions that often arise in analysis of algorithms. These values certainly suggest that the functions

$$\log n, \quad n, \quad n \log n, \quad n^2, \quad n^3, \quad 2^n, \quad n!$$

are listed in increasing order of their order of growth. Do these values prove this fact with mathematical certainty?

b. Prove that the functions are indeed listed in increasing order of their order of growth.

solution



$$\text{b. } \lim_{n \rightarrow \infty} \frac{\log_2 n}{n} = \lim_{n \rightarrow \infty} \frac{(\log_2 n)'}{(n)'} = \lim_{n \rightarrow \infty} \frac{\frac{1}{n} \log_2 e}{1} = \log_2 e \lim_{n \rightarrow \infty} \frac{1}{n} = 0.$$

$$\lim_{n \rightarrow \infty} \frac{n}{n \log_2 n} = \lim_{n \rightarrow \infty} \frac{1}{\log_2 n} = 0.$$

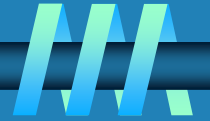
$$\lim_{n \rightarrow \infty} \frac{n \log_2 n}{n^2} = \lim_{n \rightarrow \infty} \frac{\log_2 n}{n} = (\text{see the first limit of this exercise}) = 0.$$

$$\lim_{n \rightarrow \infty} \frac{n^2}{n^3} = \lim_{n \rightarrow \infty} \frac{1}{n} = 0.$$

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{n^3}{2^n} &= \lim_{n \rightarrow \infty} \frac{(n^3)'}{(2^n)'} = \lim_{n \rightarrow \infty} \frac{3n^2}{2^n \ln 2} = \frac{3}{\ln 2} \lim_{n \rightarrow \infty} \frac{n^2}{2^n} = \frac{3}{\ln 2} \lim_{n \rightarrow \infty} \frac{(n^2)'}{(2^n)'} \\ &= \frac{3}{\ln 2} \lim_{n \rightarrow \infty} \frac{2n}{2^n \ln 2} = \frac{6}{\ln^2 2} \lim_{n \rightarrow \infty} \frac{n}{2^n} = \frac{6}{\ln^2 2} \lim_{n \rightarrow \infty} \frac{(n)'}{(2^n)'} \\ &= \frac{6}{\ln^2 2} \lim_{n \rightarrow \infty} \frac{1}{2^n \ln 2} = \frac{6}{\ln^3 2} \lim_{n \rightarrow \infty} \frac{1}{2^n} = 0. \end{aligned}$$

$$\lim_{n \rightarrow \infty} \frac{2^n}{n!} = (\text{see Example 3 in the section}) 0.$$

Exercise

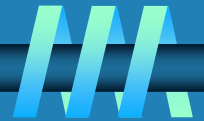


Order the following functions according to their order of growth (from the lowest to the highest):

$(n-2)!$, $5 \lg(n+100)^{10}$, 2^{2n} , $0.001n^4 + 3n^3 + 1$, $\ln^2 n$, $\sqrt[3]{n}$, 3^n .



solution



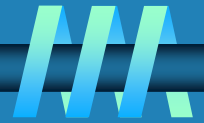
$(n-2)! \in \Theta((n-2)!)$, $5 \lg(n+100)^{10} = 50 \lg(n+100) \in \Theta(\log n)$, $2^{2n} = (2^2)^n \in \Theta(4^n)$, $0.001n^4 + 3n^3 + 1 \in \Theta(n^4)$, $\ln^2 n \in \Theta(\log^2 n)$, $\sqrt[3]{n} \in \Theta(n^{\frac{1}{3}})$, $3^n \in \Theta(3^n)$. The list of these functions ordered in increasing order of growth looks as follows:

$5 \lg(n+100)^{10}$, $\ln^2 n$, $\sqrt[3]{n}$, $0.001n^4 + 3n^3 + 1$, 3^n , 2^{2n} , $(n-2)!$

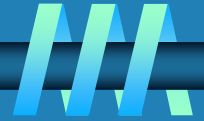


TABLE 2.2 Basic asymptotic efficiency classes

Class	Name	Comments
1	<i>constant</i>	Short of best-case efficiencies, very few reasonable examples can be given since an algorithm's running time typically goes to infinity when its input size grows infinitely large.
$\log n$	<i>logarithmic</i>	Typically, a result of cutting a problem's size by a constant factor on each iteration of the algorithm (see Section 5.5). Note that a logarithmic algorithm cannot take into account all its input (or even a fixed fraction of it): any algorithm that does so will have at least linear running time.
n	<i>linear</i>	Algorithms that scan a list of size n (e.g., sequential search) belong to this class.
$n \log n$	<i>"n-log-n"</i>	Many divide-and-conquer algorithms (see Chapter 4), including mergesort and quicksort in the average case, fall into this category.
n^2	<i>quadratic</i>	Typically, characterizes efficiency of algorithms with two embedded loops (see the next section). Elementary sorting algorithms and certain operations on n -by- n matrices are standard examples.
n^3	<i>cubic</i>	Typically, characterizes efficiency of algorithms with three embedded loops (see the next section). Several nontrivial algorithms from linear algebra fall into this class.
2^n	<i>exponential</i>	Typical for algorithms that generate all subsets of an n -element set. Often, the term "exponential" is used in a broader sense to include this and larger orders of growth as well.
$n!$	<i>factorial</i>	Typical for algorithms that generate all permutations of an n -element set.



Analysis of algorithms



❧ Issues:

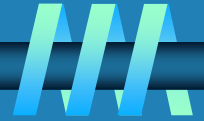
- correctness
- time efficiency
- space efficiency
- optimality

❧ Approaches:

- theoretical analysis
- empirical analysis

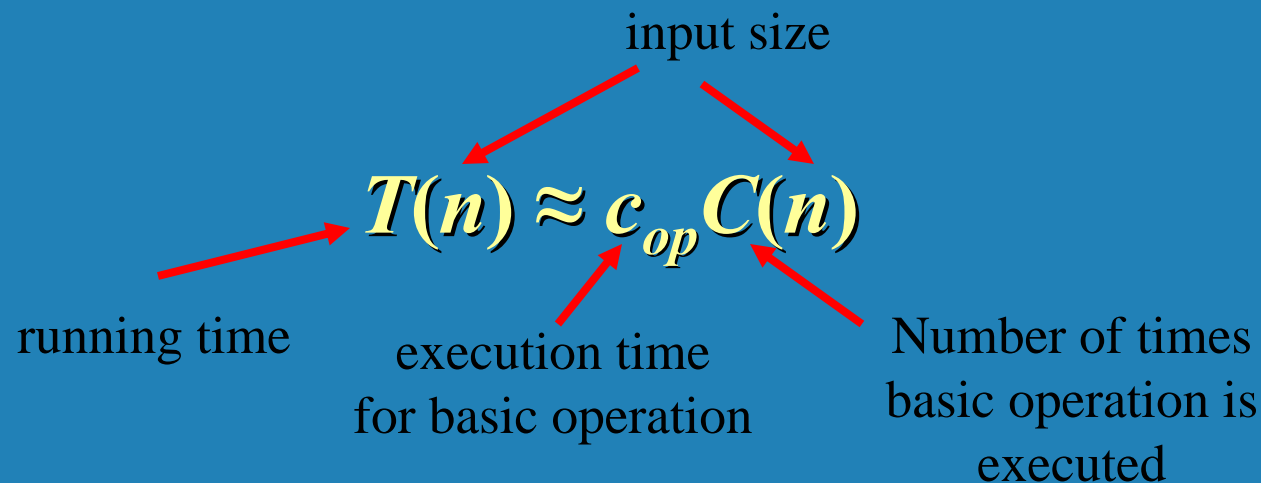


efficiency

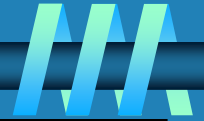


Time efficiency is analyzed by determining the number of repetitions of the basic operation as a function of input size

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

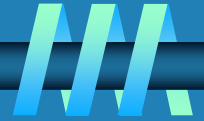


examples



<i>Problem</i>	<i>Input size measure</i>	<i>Basic operation</i>
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

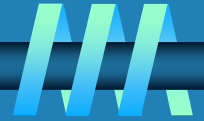
efficiency



- ⌚ 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



case



For some algorithms efficiency depends on form of input:

❧ **Worst case:** $C_{\text{worst}}(n)$ – maximum over inputs of size n

❧ **Best case:** $C_{\text{best}}(n)$ – minimum over inputs of size n

❧ **Average case:** $C_{\text{avg}}(n)$ – “average” over inputs of size n

- Number of times the basic operation will be executed on typical input
- NOT the average of worst and best case
- Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs



Example: 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**

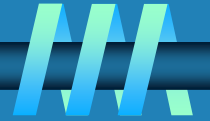
$i \leftarrow i + 1$

if $i < n$ **return** i

else return -1

- Worst case
- Best case

Types of formulas for basic operations count



⌚ Exact formula

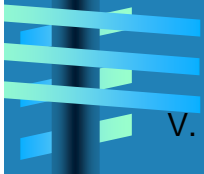
e.g., $C(n) = n(n-1)/2$

⌚ Formula indicating order of growth with specific multiplicative constant

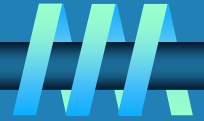
e.g., $C(n) \approx 0.5 n^2$

⌚ Formula indicating order of growth with unknown multiplicative constant

e.g., $C(n) \approx cn^2$



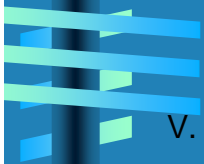
Order of growth



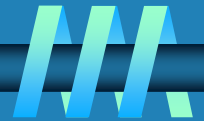
⌚ **Most important: Order of growth within a constant multiple as $n \rightarrow \infty$**

⌚ **Example:**

- **How much faster will algorithm run on computer that is twice as fast?**
- **How much longer does it take to solve problem of double input size?**



$n \rightarrow \infty$

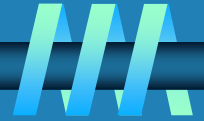


n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Table 2.1 Values (some approximate) of several functions important for analysis of algorithms



Asymptotic order of growth



A way of comparing functions that ignores constant factors and small input sizes

- ❧ $O(g(n))$: class of functions $f(n)$ that grow no faster than $g(n)$
- ❧ $\Theta(g(n))$: class of functions $f(n)$ that grow at same rate as $g(n)$
- ❧ $\Omega(g(n))$: class of functions $f(n)$ that grow at least as fast as $g(n)$



Big-oh

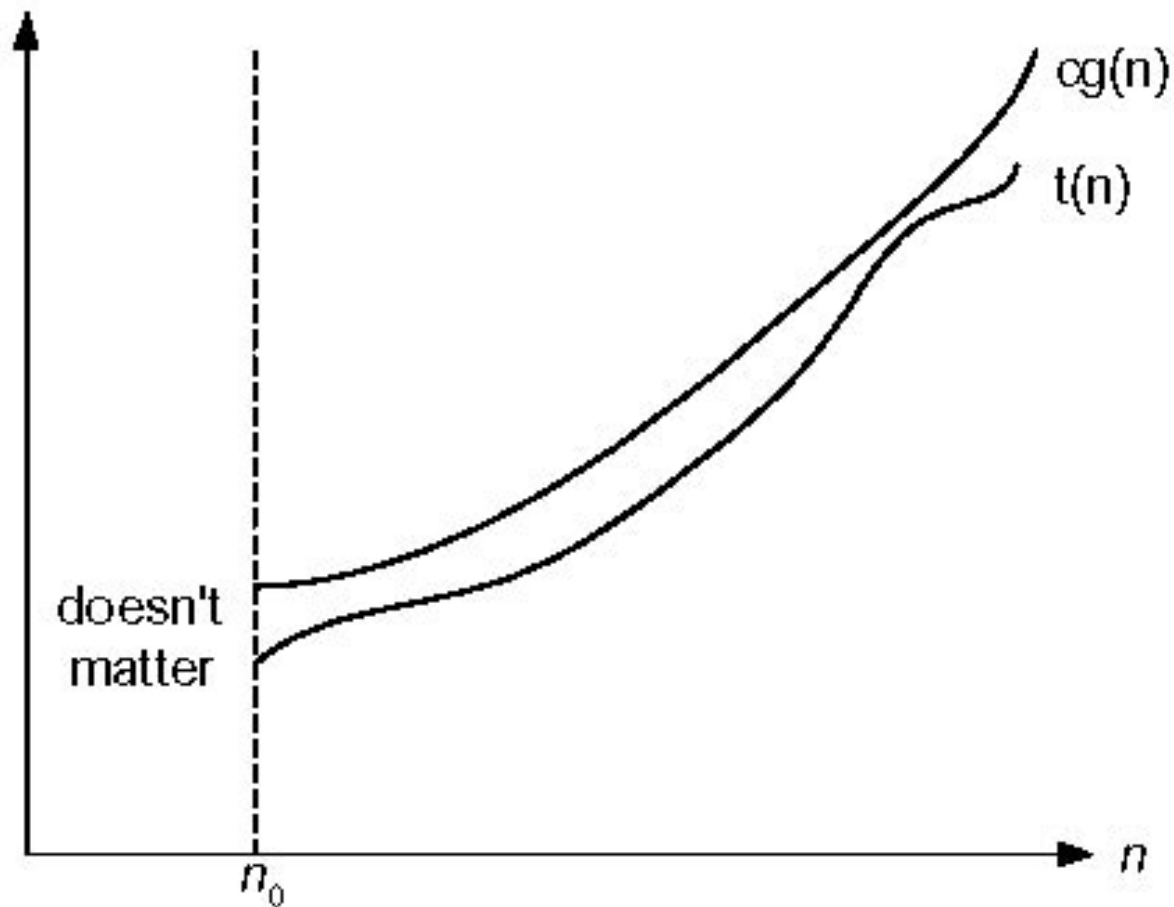
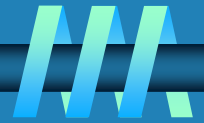
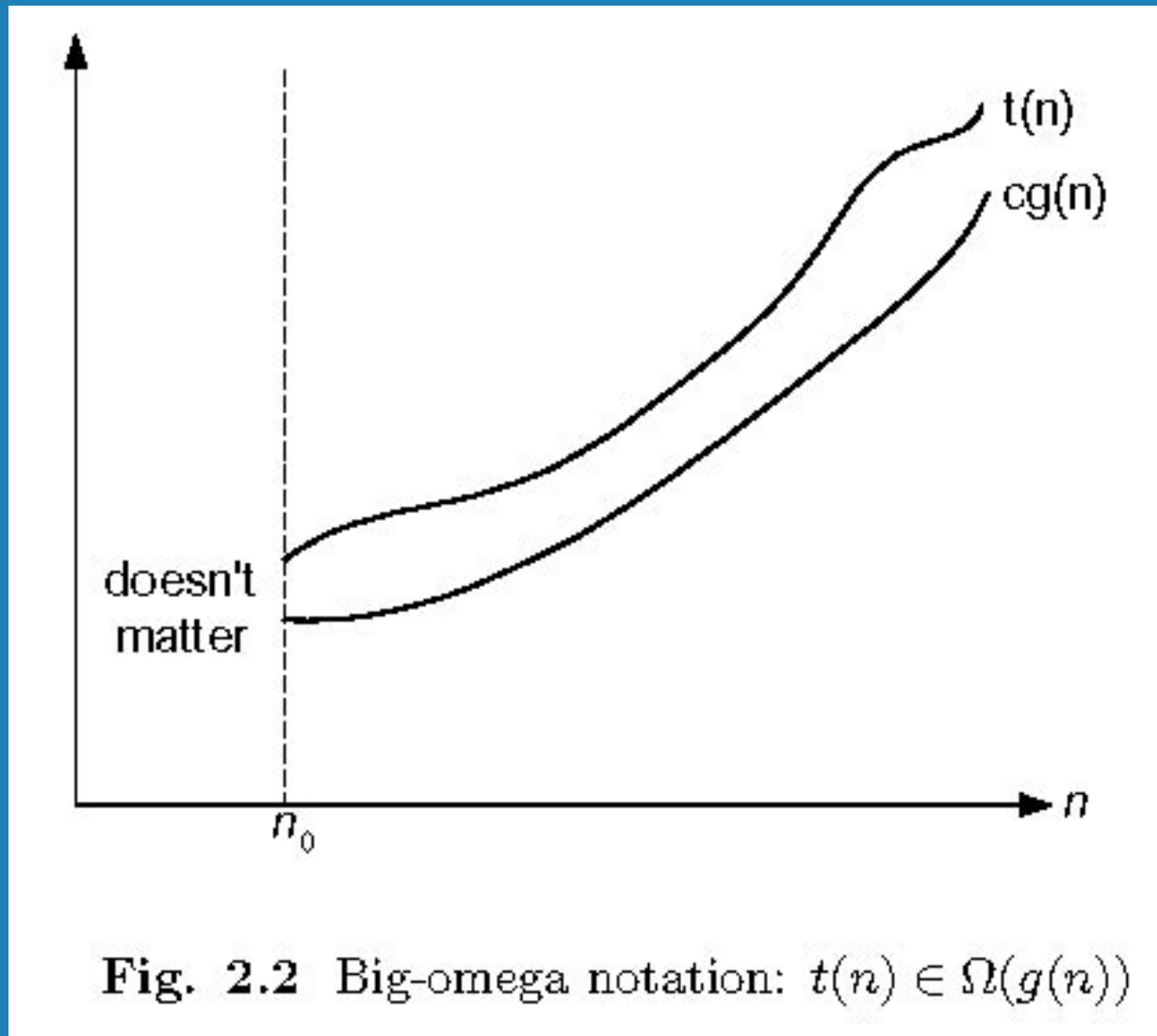
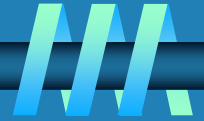
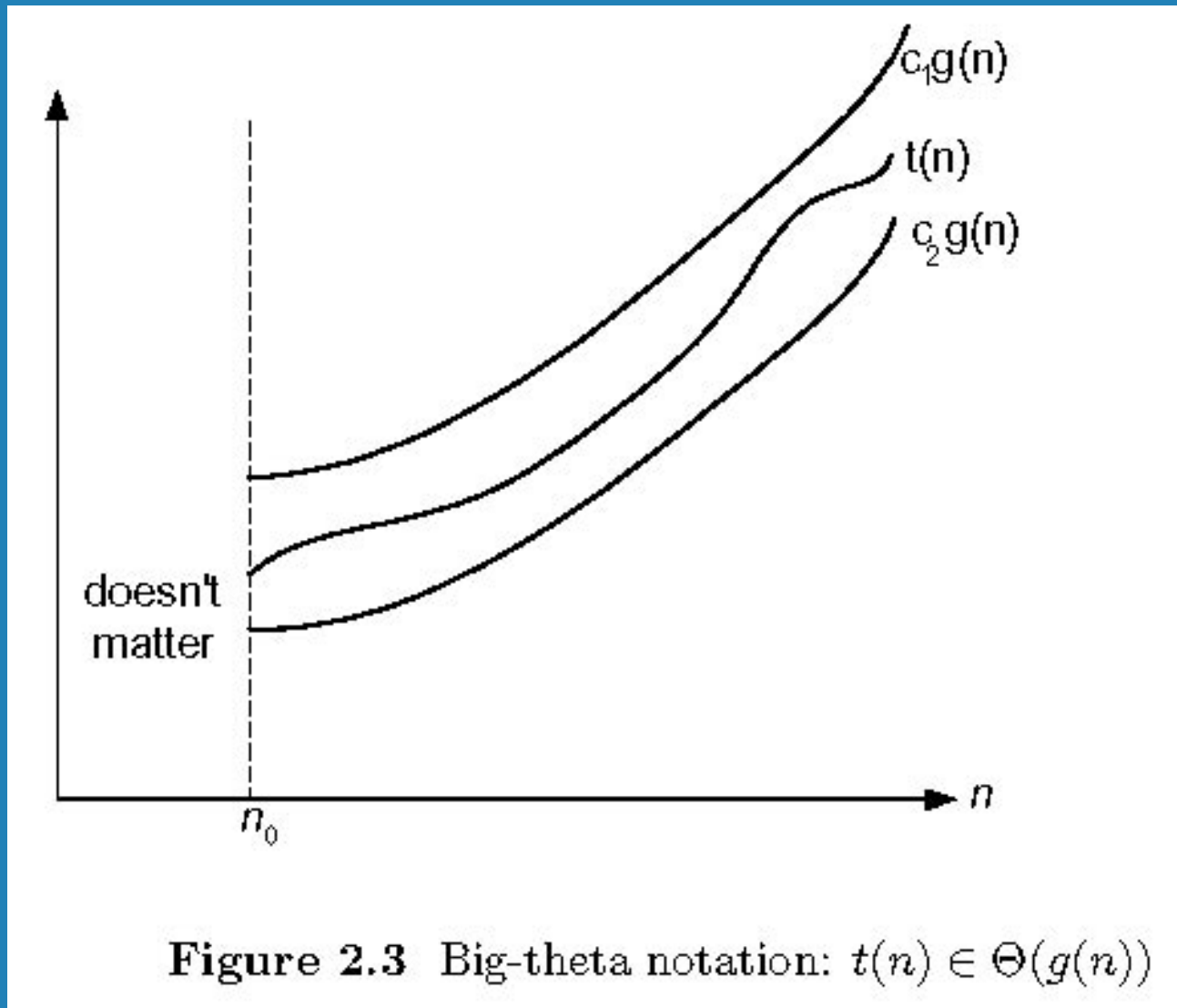
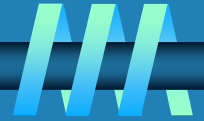


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

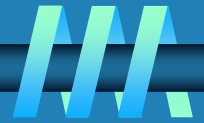
Big-omega



Big-theta



definition



Definition: $f(n)$ is in $O(g(n))$ if order of growth of $f(n) \leq$ order of growth of $g(n)$ (within constant multiple), i.e., there exist positive constant c and non-negative integer n_0 such that

$$f(n) \leq c g(n) \text{ for every } n \geq n_0$$

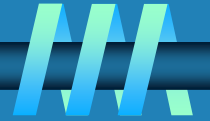
Examples:

❧ $10n$ is $O(n^2)$

❧ $5n+20$ is $O(n)$



growth



⌚ $f(n) \in O(f(n))$

⌚ $f(n) \in O(g(n))$ iff $g(n) \in \Omega(f(n))$

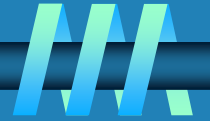
⌚ If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$

Note similarity with $a \leq b$

⌚ If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then
 $f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$



limits

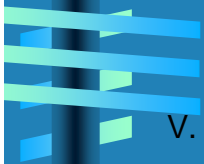


$$\lim_{n \rightarrow \infty} T(n)/g(n) = \begin{cases} 0 & \text{order of growth of } T(n) < \text{order of growth of } g(n) \\ c > 0 & \text{order of growth of } T(n) = \text{order of growth of } g(n) \\ \infty & \text{order of growth of } T(n) > \text{order of growth of } g(n) \end{cases}$$

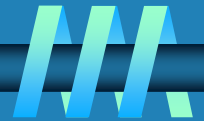
Examples:

• $10n$ vs. n^2

• $n(n+1)/2$ vs. n^2



formula



L'Hôpital's rule: If $\lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} g(n) = \infty$ and the derivatives f', g' exist, then

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$$

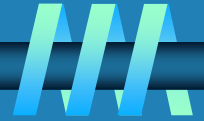
Example: $\log n$ vs. n

Stirling's formula: $n! \approx (2\pi n)^{1/2} (n/e)^n$

Example: 2^n vs. $n!$



Orders of growth of some important functions



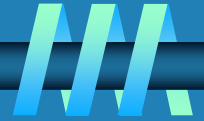
- ⌚ All logarithmic functions $\log_a n$ belong to the same class $\Theta(\log n)$ no matter what the logarithm's base $a > 1$ is
- ⌚ All polynomials of the same degree k belong to the same class:
 $a_k n^k + a_{k-1} n^{k-1} + \dots + a_0 \in \Theta(n^k)$
- ⌚ Exponential functions a^n have different orders of growth for different a 's
- ⌚ $\text{order } \log n < \text{order } n^\alpha \ (\alpha > 0) < \text{order } a^n < \text{order } n! < \text{order } n^n$



classes

1	constant
$\log n$	logarithmic
n	linear
$n \log n$	n-log-n
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

algorithms

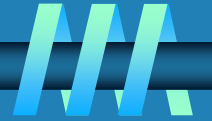


General Plan for Analysis

- ❧ 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
- ❧ Simplify the sum using standard formulas and rules (see Appendix A)



Useful summation formulas and rules



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

In particular, $\sum_{1 \leq i \leq n} 1 = n - 1 + 1 = n \in \Theta(n)$

$$\sum_{1 \leq i \leq n} i = 1 + 2 + \dots + n = n(n+1)/2 \approx n^2/2 \in \Theta(n^2)$$

$$\sum_{1 \leq i \leq n} i^2 = 1^2 + 2^2 + \dots + n^2 = n(n+1)(2n+1)/6 \approx n^3/3 \in \Theta(n^3)$$

$$\sum_{0 \leq i \leq n} a^i = 1 + a + \dots + a^n = (a^{n+1} - 1)/(a - 1) \text{ for any } a \neq 1$$

In particular, $\sum_{0 \leq i \leq n} 2^i = 2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1 \in \Theta(2^n)$

$$\sum (a_i \pm b_i) = \sum a_i \pm \sum b_i \quad \sum c a_i = c \sum a_i \quad \sum_{l \leq i \leq u} a_i = \sum_{l \leq i \leq m} a_i + \sum_{m+1 \leq i \leq u} a_i$$



Example 1: Maximum element

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 $maxval$

problem

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

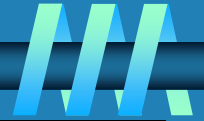
for $i \leftarrow 0$ **to** $n - 2$ **do**

for $j \leftarrow i + 1$ **to** $n - 1$ **do**

if $A[i] = A[j]$ **return false**

return true

1 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**

$C[i, j] \leftarrow 0.0$

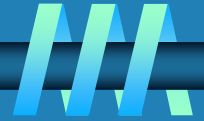
for $k \leftarrow 0$ **to** $n - 1$ **do**

$C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$

return C



Example 1: Gaussian elimination



Algorithm *GaussianElimination*($A[0..n-1,0..n]$)

//Implements Gaussian elimination of an n -by- $(n+1)$ matrix A

for $i \leftarrow 0$ to $n - 2$ do

 for $j \leftarrow i + 1$ to $n - 1$ do

 for $k \leftarrow i$ to n do

$A[j,k] \leftarrow A[j,k] - A[i,k] * A[j,i] / A[i,i]$

Find the efficiency class and a constant factor improvement.



digits

ALGORITHM *Binary(n)*

//Input: A positive decimal integer n

//Output: The number of binary digits in n 's binary representation

$count \leftarrow 1$

while $n > 1$ **do**

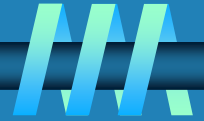
$count \leftarrow count + 1$

$n \leftarrow \lfloor n/2 \rfloor$

return $count$

It cannot be investigated the way the previous examples are.

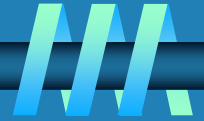
Algorithms



- ❧ **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. (If it may, the worst, average, and best cases must be investigated separately.)**
- ❧ **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.**



Example 1: Recursive Evaluation of $n!$



Definition: $n! = 1 * 2 * \dots * (n-1) * n$ for $n \geq 1$ and $0! = 1$

ALGORITHM $F(n)$

//Computes $n!$ recursively

//Input: A nonnegative integer n

//Output: The value of $n!$

if $n = 0$ **return** 1

else return $F(n - 1) * n$

$n! : F(n) = F(n-1) * n$

$$F(0) = 1$$



Solving the recurrence for $M(n)$

$$M(n) = M(n-1) + 1, \quad M(0) = 0$$

Puzzle

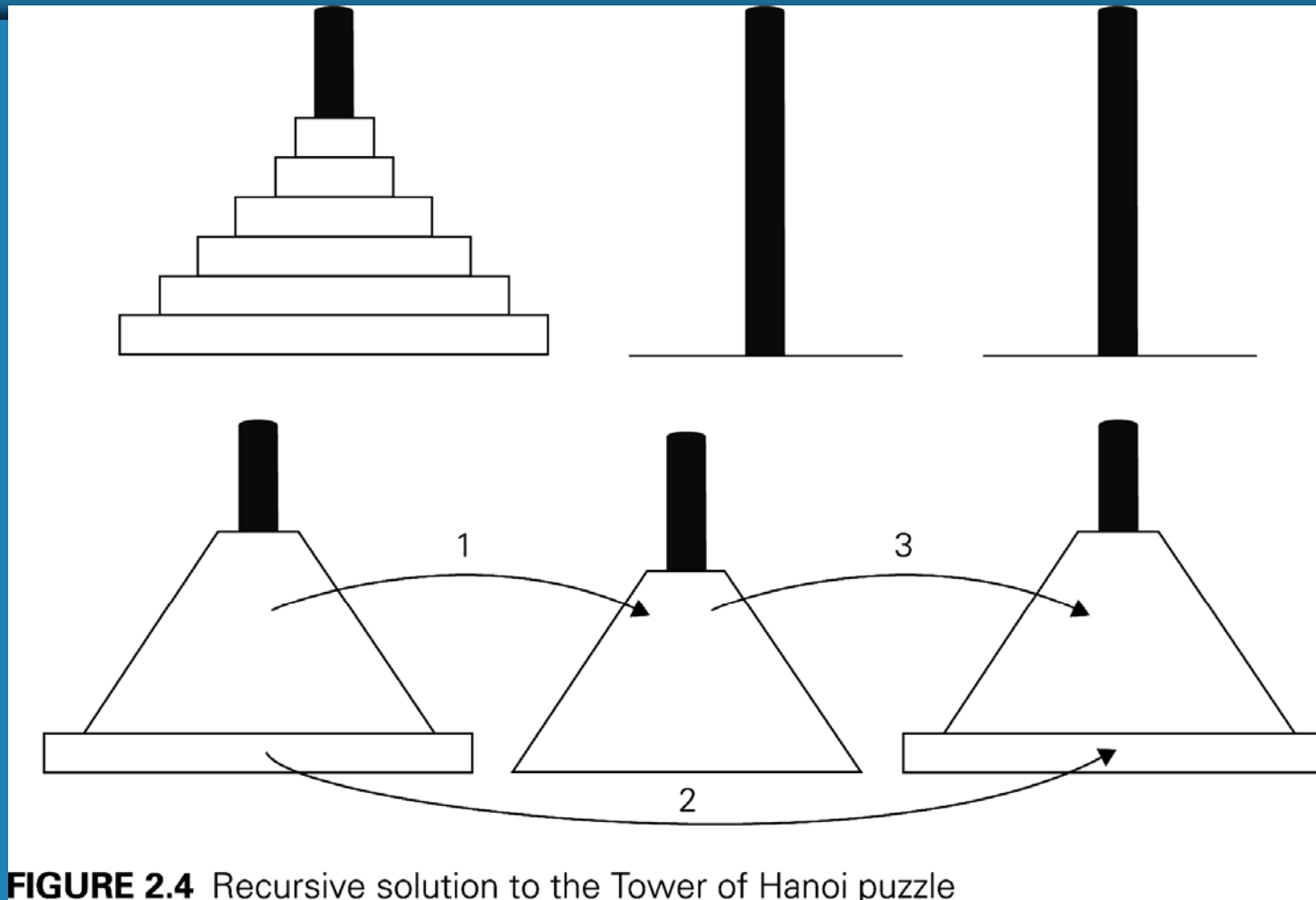
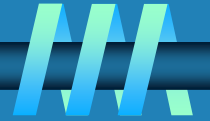


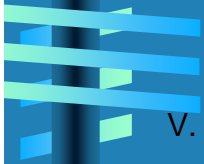
FIGURE 2.4 Recursive solution to the Tower of Hanoi puzzle

Recurrence for number of moves:

moves



$$M(n) = 2M(n-1) + 1, \quad M(1) = 1$$



Puzzle

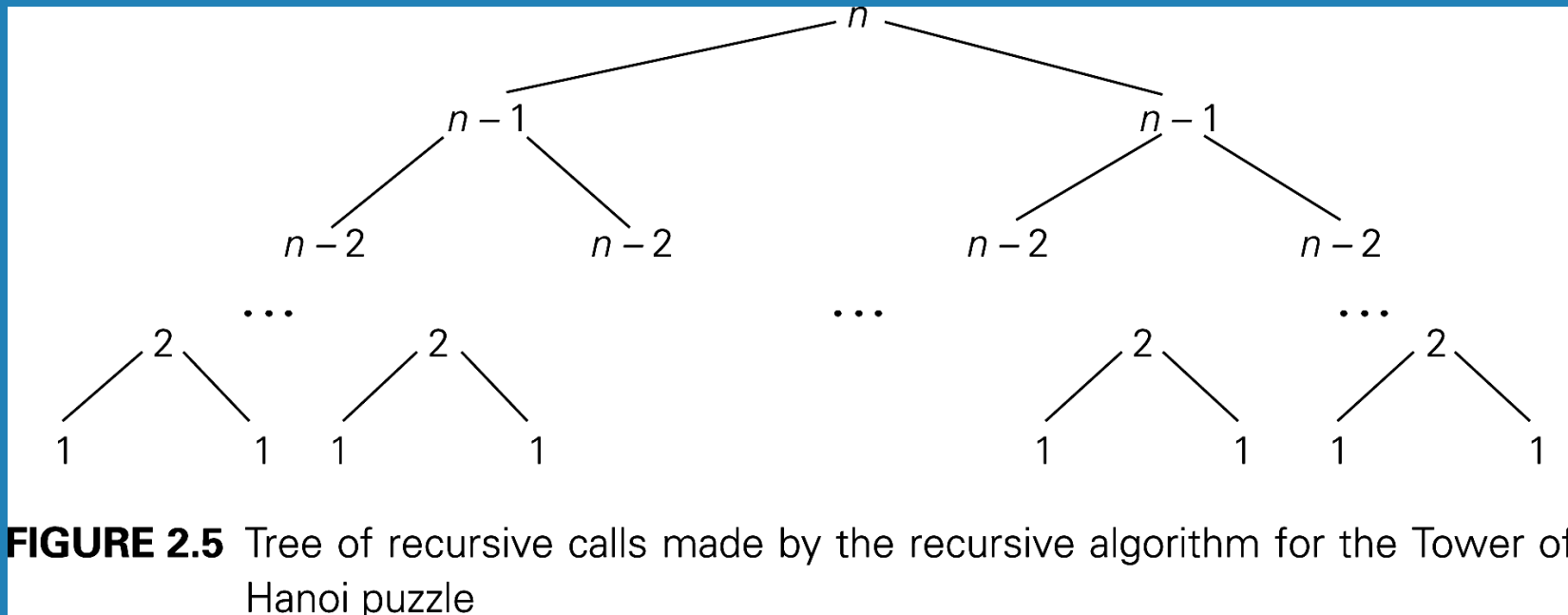
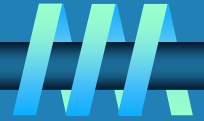
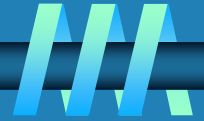


FIGURE 2.5 Tree of recursive calls made by the recursive algorithm for the Tower of Hanoi puzzle



Example 3: Counting #bits



ALGORITHM *BinRec*(n)

//Input: A positive decimal integer n

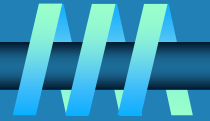
//Output: The number of binary digits in n 's binary representation

if $n = 1$ **return** 1

else return *BinRec*($\lfloor n/2 \rfloor$) + 1



Fibonacci numbers



The Fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

The Fibonacci recurrence:

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

$$F(0) = 0$$

$$F(1) = 1$$

General 2nd order linear homogeneous recurrence with constant coefficients:

$$aX(n) + bX(n-1) + cX(n-2) = 0$$



Tree for Fibonacci

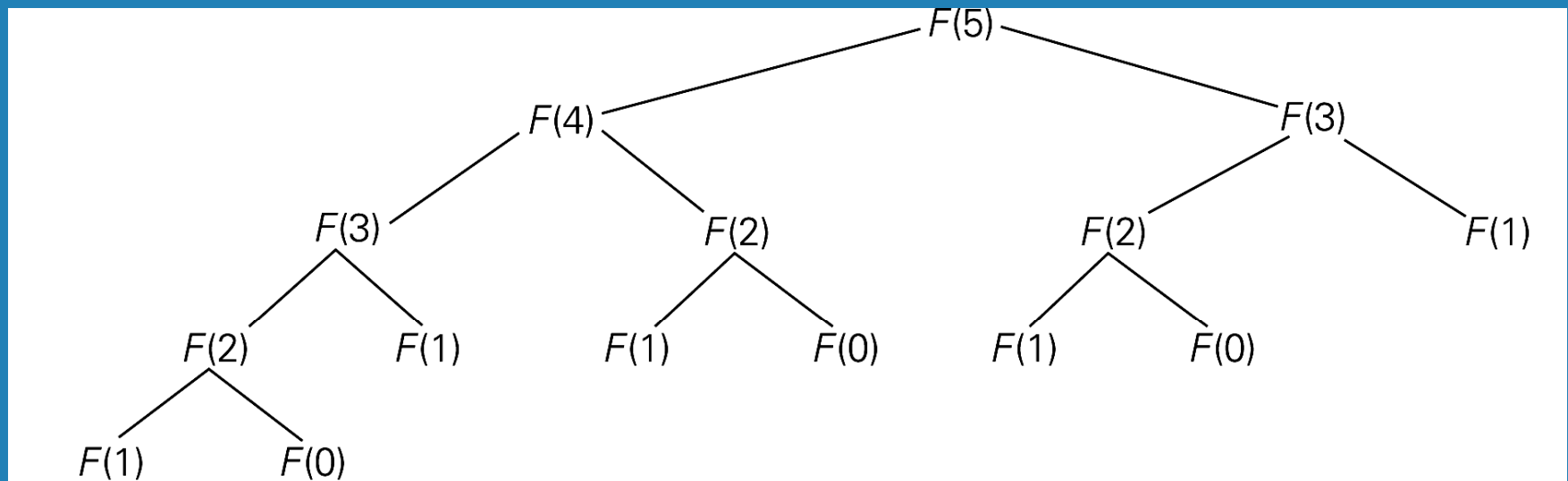
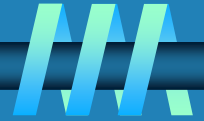
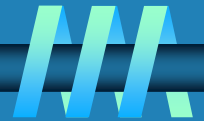


FIGURE 2.6 Tree of recursive calls for computing the 5th Fibonacci number by the definition-based algorithm



$$2) = 0$$



2) Set up the characteristic equation (quadratic)

$$ar^2 + br + c = 0$$

3) Solve to obtain roots r_1 and r_2

4) General solution to the recurrence

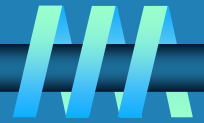
if r_1 and r_2 are two distinct real roots: $X(n) = \alpha r_1^n + \beta r_2^n$

if $r_1 = r_2 = r$ are two equal real roots: $X(n) = \alpha r^n + \beta n r^n$

5) Particular solution can be found by using initial conditions



numbers



$$F(n) = F(n-1) + F(n-2) \text{ or } F(n) - F(n-1) - F(n-2) = 0$$

Characteristic equation:

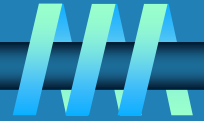
Roots of the characteristic equation:

General solution to the recurrence:

Particular solution for $F(0)=0$, $F(1)=1$:



Computing Fibonacci numbers



1. Definition-based recursive algorithm
2. Nonrecursive definition-based algorithm
3. Explicit formula algorithm
4. Logarithmic algorithm based on formula:

$$\begin{pmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$$

for $n \geq 1$, assuming an efficient way of computing matrix powers.

