Algorithms: Searching

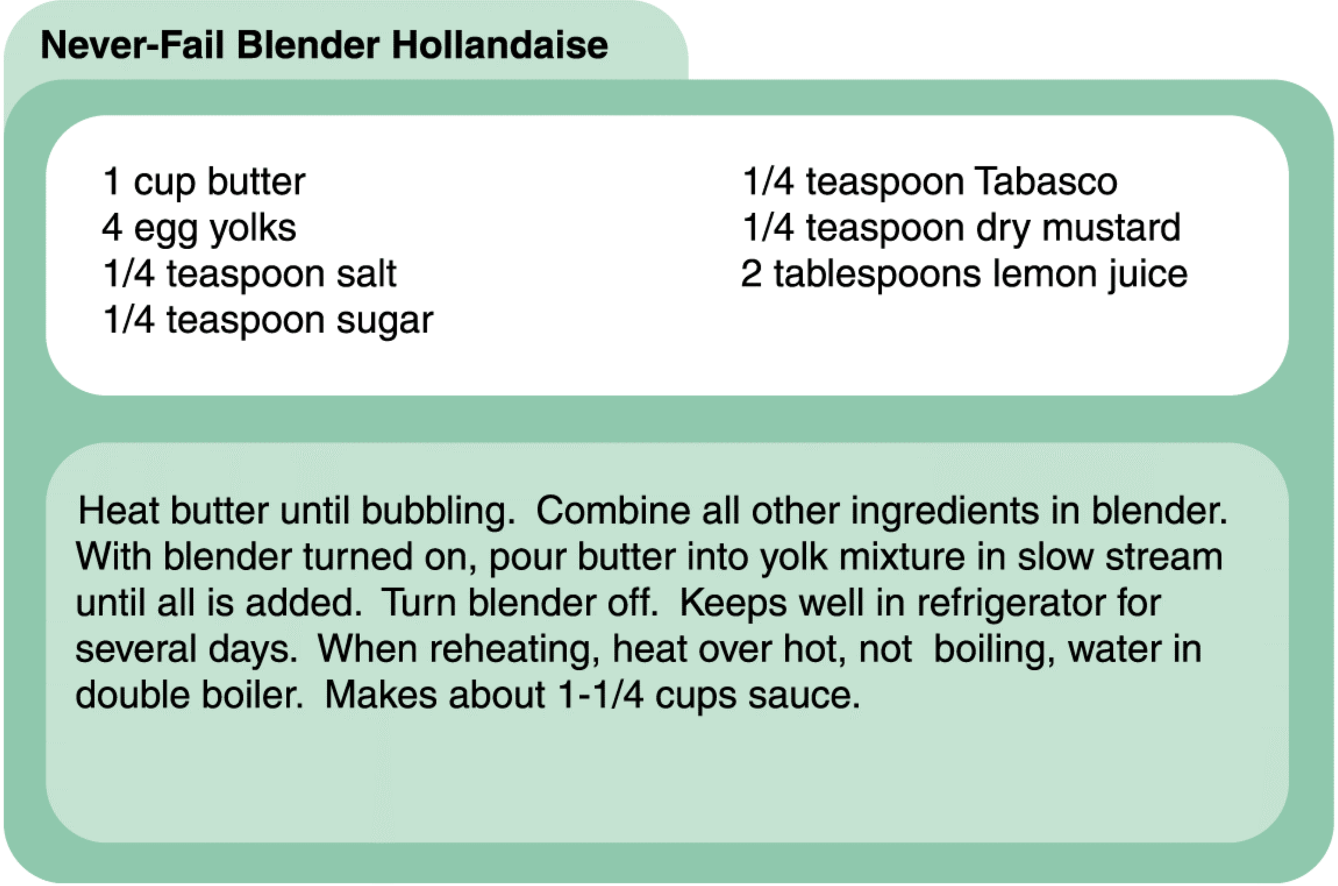
Nell Dale & John Lewis
(adaptation by Michael
Goldwasser)

Algorithms

- An **algorithm** is set of instructions for solving a problem or subproblem in a finite amount of time using a finite amount of data
- The instructions are unambiguous

Following an Algorithm

- Preparing a Hollandaise sauce



Never-Fail Blender Hollandaise

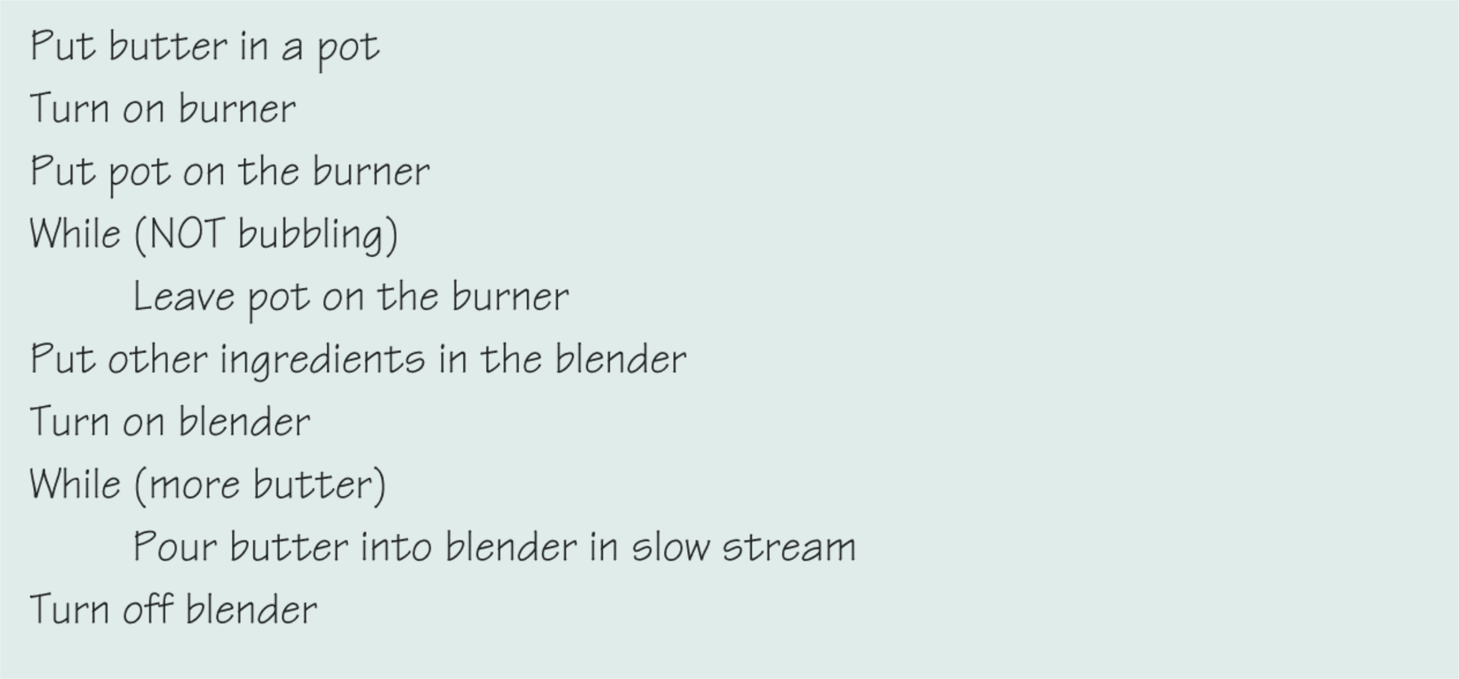
1 cup butter
4 egg yolks
1/4 teaspoon salt
1/4 teaspoon sugar
1/4 teaspoon Tabasco
1/4 teaspoon dry mustard
2 tablespoons lemon juice

Heat butter until bubbling. Combine all other ingredients in blender. With blender turned on, pour butter into yolk mixture in slow stream until all is added. Turn blender off. Keeps well in refrigerator for several days. When reheating, heat over hot, not boiling, water in double boiler. Makes about 1-1/4 cups sauce.

Figure 6.4

Following an Algorithm (cont.)

- Preparing a Hollandaise sauce



```
Put butter in a pot
Turn on burner
Put pot on the burner
While (NOT bubbling)
        Leave pot on the burner
Put other ingredients in the blender
Turn on blender
While (more butter)
        Pour butter into blender in slow stream
Turn off blender
```

Page 150

Algorithmic Paradigms

- **Iterative**

  (repetitive structure based on loops)

- **Recursive**

  (repetitive structure based on recursion)

Searching

Goal: given a collection of items, determine whether a particular item is in the collection.

Approach: For starters, lets assume that all of the items are stored in an array.

Sequential Search (Iterative)

Items in Unsorted Array:

```
boolean Search(Item)
   int i = 0;
   boolean success = FALSE;
   while ((i < Length(A)) AND (NOT success)) do
        if (A[i] = Item) then set success = TRUE
        add 1 to i
    return success
```

In worst case, search time is proportional to Length(A)

A Better Way to Search

Items in Sorted Array:

There is a much better way!

How do you search in a phone book?

Describe your method precisely as if you were teaching it to someone else

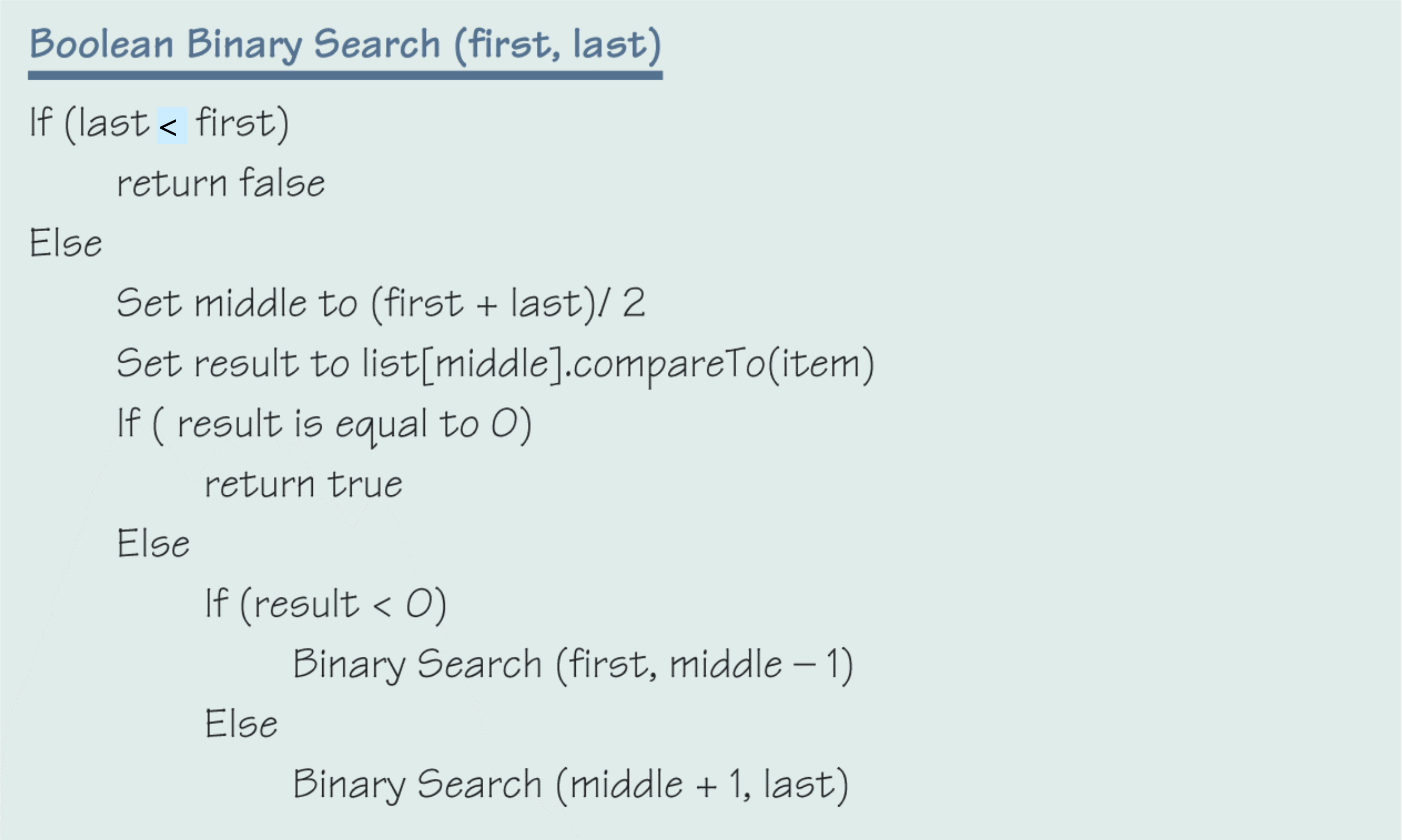
Binary Search

- A divide-and-conquer (recursive) strategy
  - If no items for consideration, search is a failure
  - Otherwise, compare the “middle” entry to the target item
  - If you found a match, great!
  - Otherwise,

    if (“middle” > target) search first half of group

    if (“middle” < target) search second half of group

Binary Search

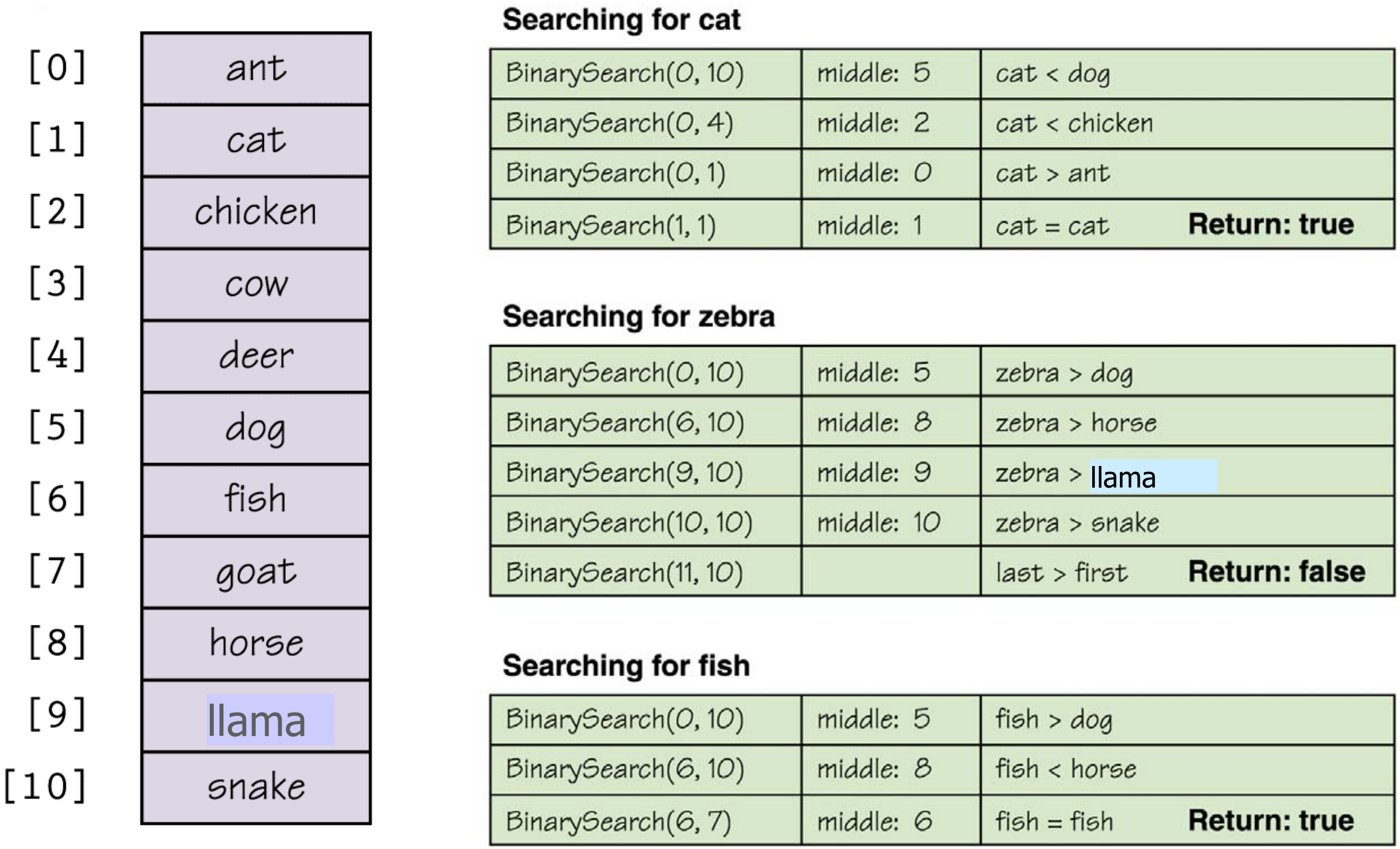


Boolean Binary Search (first, last)

```
If (last < first)
    return false
Else
    Set middle to (first + last)/ 2
    Set result to list[middle].compareTo(item)
    If ( result is equal to 0)
        return true
    Else
        If (result < 0)
            Binary Search (first, middle – 1)
        Else
            Binary Search (middle + 1, last)
```

Page 296

Binary Search



[0]	ant
[1]	cat
[2]	chicken
[3]	cow
[4]	deer
[5]	dog
[6]	fish
[7]	goat
[8]	horse
[9]	llama
[10]	snake

Searching for cat

BinarySearch(0, 10)	middle: 5	cat < dog
BinarySearch(0, 4)	middle: 2	cat < chicken
BinarySearch(0, 1)	middle: 0	cat > ant
BinarySearch(1, 1)	middle: 1	cat = cat **Return: true**

Searching for zebra

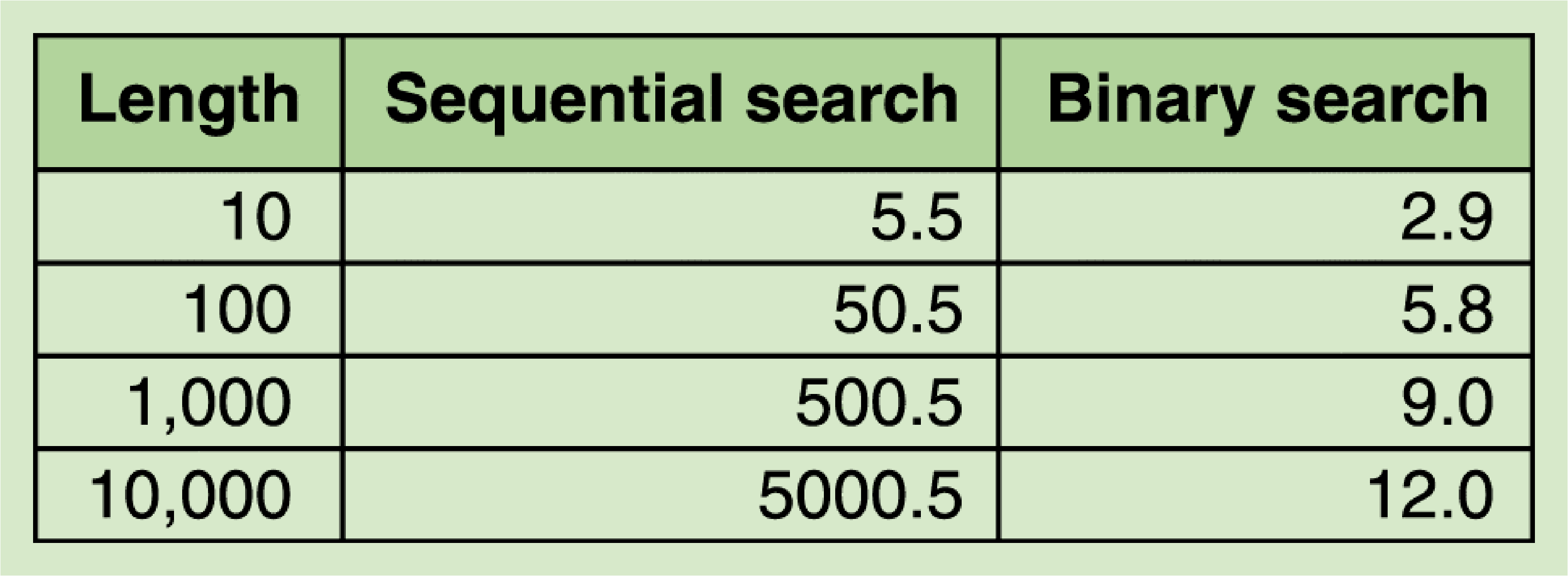
BinarySearch(0, 10)	middle: 5	zebra > dog
BinarySearch(6, 10)	middle: 8	zebra > horse
BinarySearch(9, 10)	middle: 9	zebra > llama
BinarySearch(10, 10)	middle: 10	zebra > snake
BinarySearch(11, 10)		last > first **Return: false**

Searching for fish

BinarySearch(0, 10)	middle: 5	fish > dog
BinarySearch(6, 10)	middle: 8	fish < horse
BinarySearch(6, 7)	middle: 6	fish = fish **Return: true**

Figure 9.14 **Trace of the binary search**

Efficiency



Length	Sequential search	Binary search
10	5.5	2.9
100	50.5	5.8
1,000	500.5	9.0
10,000	5000.5	12.0

Table 9.1 **Average Number of Comparisons**

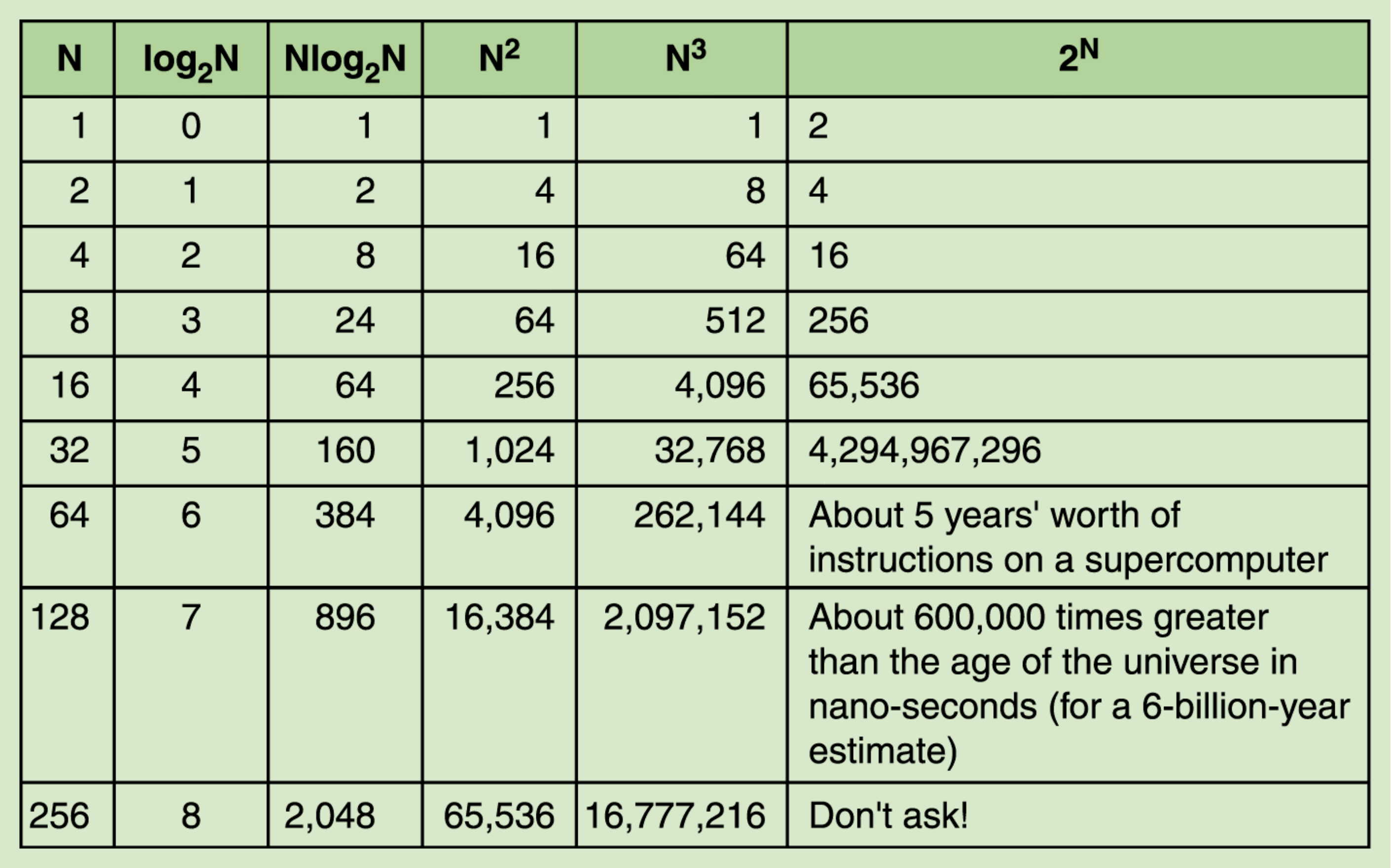
Big-O Analysis

- A function of the size of the input to the operation (for instance, the number of elements in the list to be summed)
- We can express an approximation of this function using a mathematical notation called order of magnitude, or **Big-O notation**

Big-O Analysis

- Common Orders of Magnitude
  - $O(1)$ *is called constant time*
  - $O(log_2N)$ is called logarithmic time
  - $O(N)$ is called linear is called linear time
  - $O(N\ log_2N)$
  - $O(N^2)$ is called quadratic time
  - $O(2^N)$ is called exponential time

Big-O Analysis



N	$\log_2 N$	$N\log_2 N$	N^2	N^3	2^N
1	0	1	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4,096	65,536
32	5	160	1,024	32,768	4,294,967,296
64	6	384	4,096	262,144	About 5 years' worth of instructions on a supercomputer
128	7	896	16,384	2,097,152	About 600,000 times greater than the age of the universe in nano-seconds (for a 6-billion-year estimate)
256	8	2,048	65,536	16,777,216	Don't ask!

Table 17.2
Comparison of rates of growth