cphbusiness

COPENHAGEN BUSINESS ACADEMY

Algorithms and Data Structure

Topics for the week

- Efficiency of algorithms
  - Big O

- Classic Algorithms
  - Sorting
  - Searching
  - Recursion

- Data Structures
  - Java collection framework
  - ArrayList
  - LinkedList
  - Binary Tree
  - Hash Table/Hash map
  - Binary Search Tree
  - Tree map

Day 1 Monday

- Efficiency of algorithms
  - Big O

- Insertion/Selection/Bubble sorts

- Binary search

- Data Structures
  - Introduction to Java collection framework
  - ArrayList
  - LinkedList

Efficiency of algorithms

- Think about fundamental operations computer does
  - Access
  - Insert
  - Delete
  - Find/Search
  - Sort

Efficiency of algorithms

- What are the complexities in achieving those operations?
  - Time
  - Money/Space
  - Ideal?
  - Big O - means the running time of the algorithm grows in proportion to "something"

What is the time complexity?

```
public static int sumOfThreeNum (int x, int y, int z ) {
    int  sum = 0;

    sum = x + y +z ;

    return sum ;
}
```

Total unit of time = O(C1 + C2 + C3) = O(C)

What is the time complexity?

```
public static int sumOfarray (int [] list ) {
    int total = 0;

    for (int i = 0; i < list . length ; i ++)
        total = total + list [i];

    return total ;
}
```

Total unit of time = O(1 + 2n + 2n + 1) = O(2 + 4n) = O(n)

ThreeSum example

```
public static nt count (int [] a) {
    int n = a. lenght ;
    int count = 0;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j ++)
            for ( int k = j + 1; k < n; k ++)
                if (a[i] + a[j] + a[k] == 0)
    count ++
return count ;
}
```

Total unit of time = ?

```
public static Comparable linearSearch (Comparable[] list,
                                       Comparable target)
  {
     int index = 0;
     boolean found = false;

     while (!found && index < list.length)
     {
        if (list[index].equals(target))
           found = true;
        else
           index++;
     }

     if (found)
        return list[index];
     else
        return null;
  }
```

Logarithms

How many times can we **half** N before we only have 1

- Log_2 - logarithm function with base 2
  - The inverse function to the exponential function with base 2:
    $f(x) = 2^x$

- Log_2
  - How does it look - graphically?

  - O(n) < O(n.logn) < O(n^2)

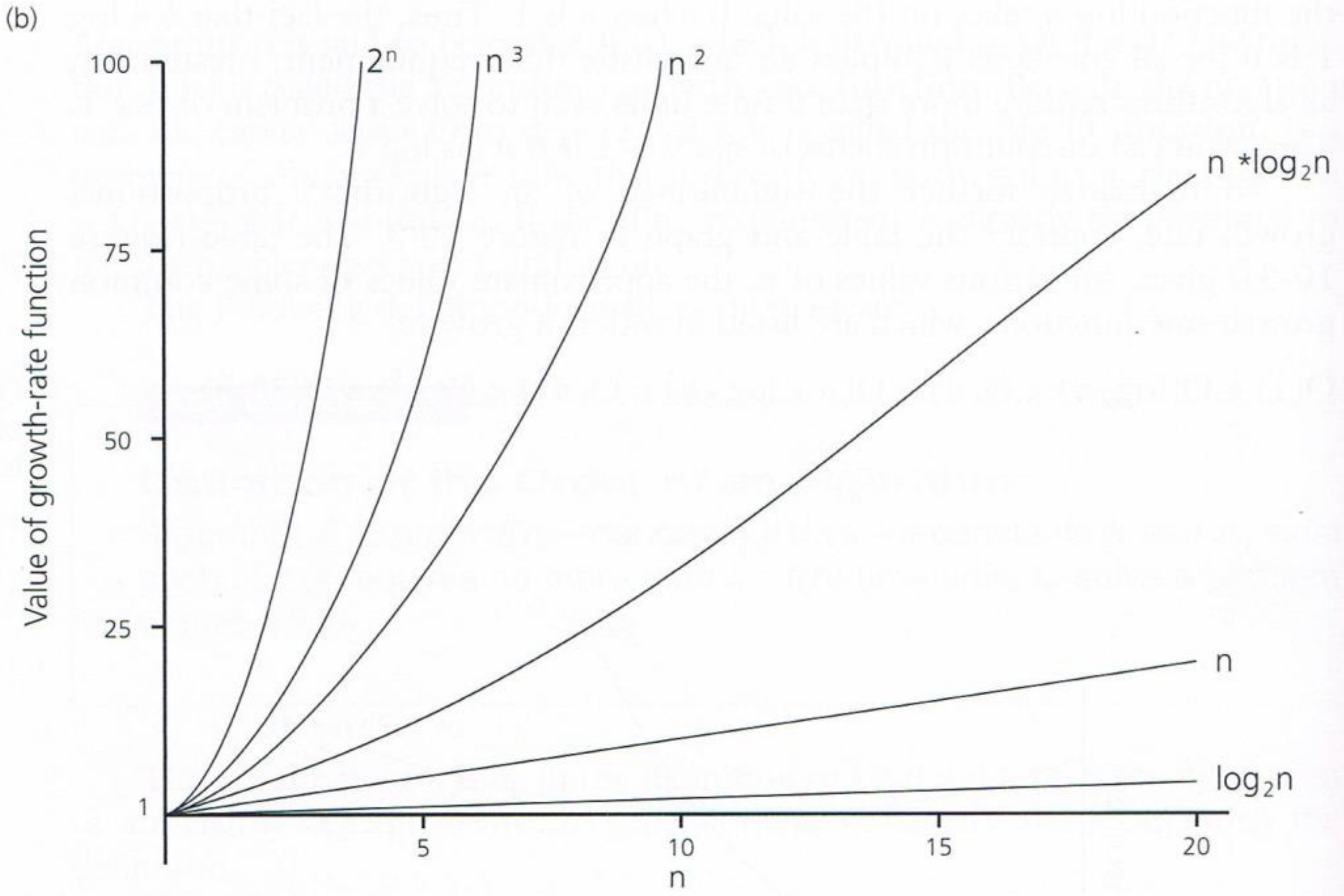
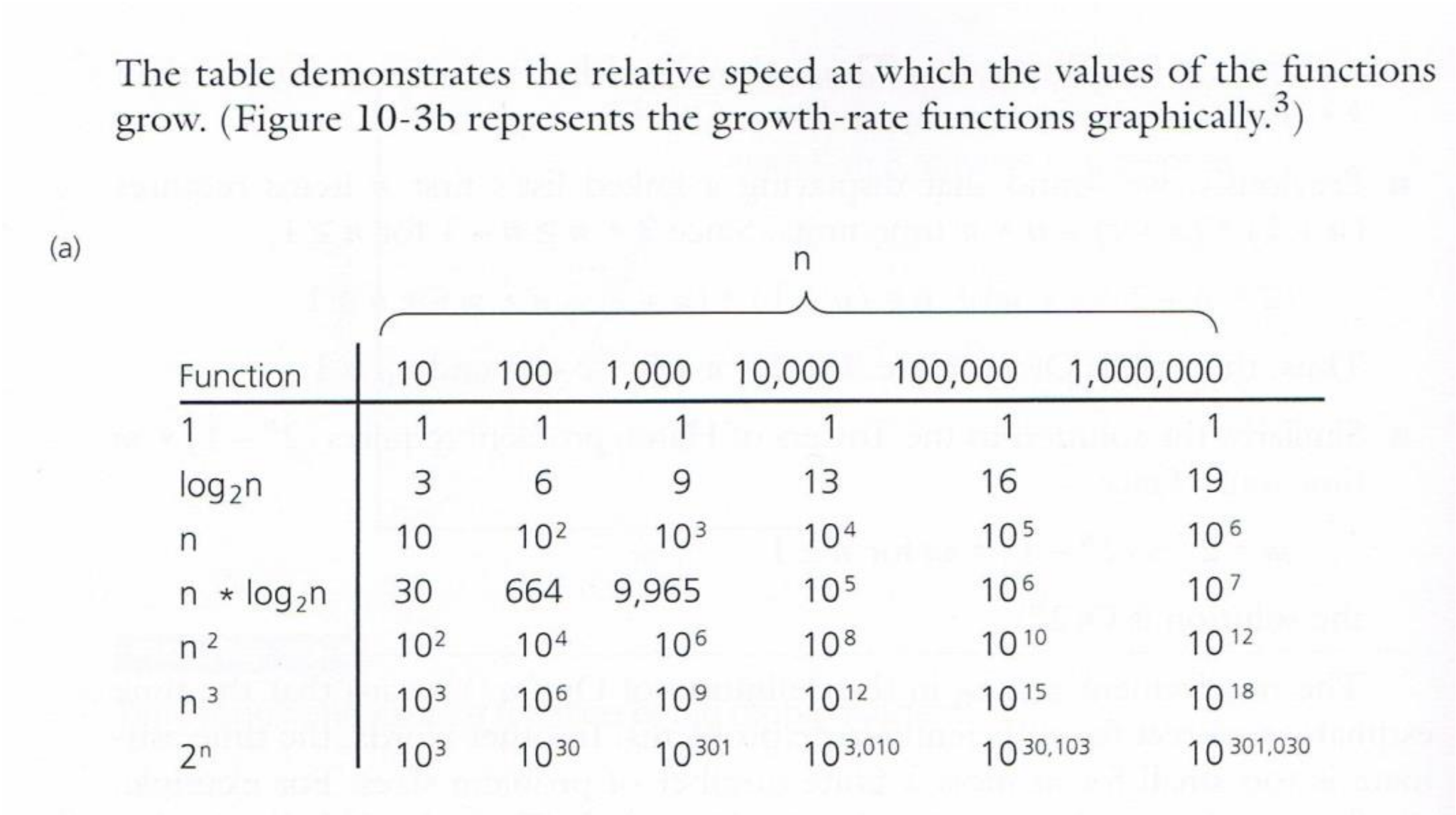


FIGURE 10-3

A comparison of growth-rate functions: (a) in tabular form; (b) in graphical form

[3] The graph of $f(n) = 1$ is omitted because the scale of the figure makes it difficult to draw. It would, however, be a straight line parallel to the x axis through $y = 1$.



The table demonstrates the relative speed at which the values of the functions grow. (Figure 10-3b represents the growth-rate functions graphically.[3])

(a)

	n					
Function	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

```
public static void selectionSort (Comparable[] list)
  {
     int min;
     Comparable temp;

     for (int index = 0; index < list.length-1; index++)
     {
        min = index;
        for (int scan = index+1; scan < list.length; scan++)
           if (list[scan].compareTo(list[min]) < 0)
              min = scan;

        // Swap the values
        temp        = list[min];
        list[min]   = list[index];
        list[index] = temp;
     }
  }
```

Comparable <T> : Comparing objects

- Used widely for sorting objects in data structures

- `int compareTo(T obj)-` compare this object with `obj,` which is type T. It returns a negaive integer, zero or positive when this object is less than, equal, or greater `obj`
- `ObjectA.compareTo(ObjectB)`

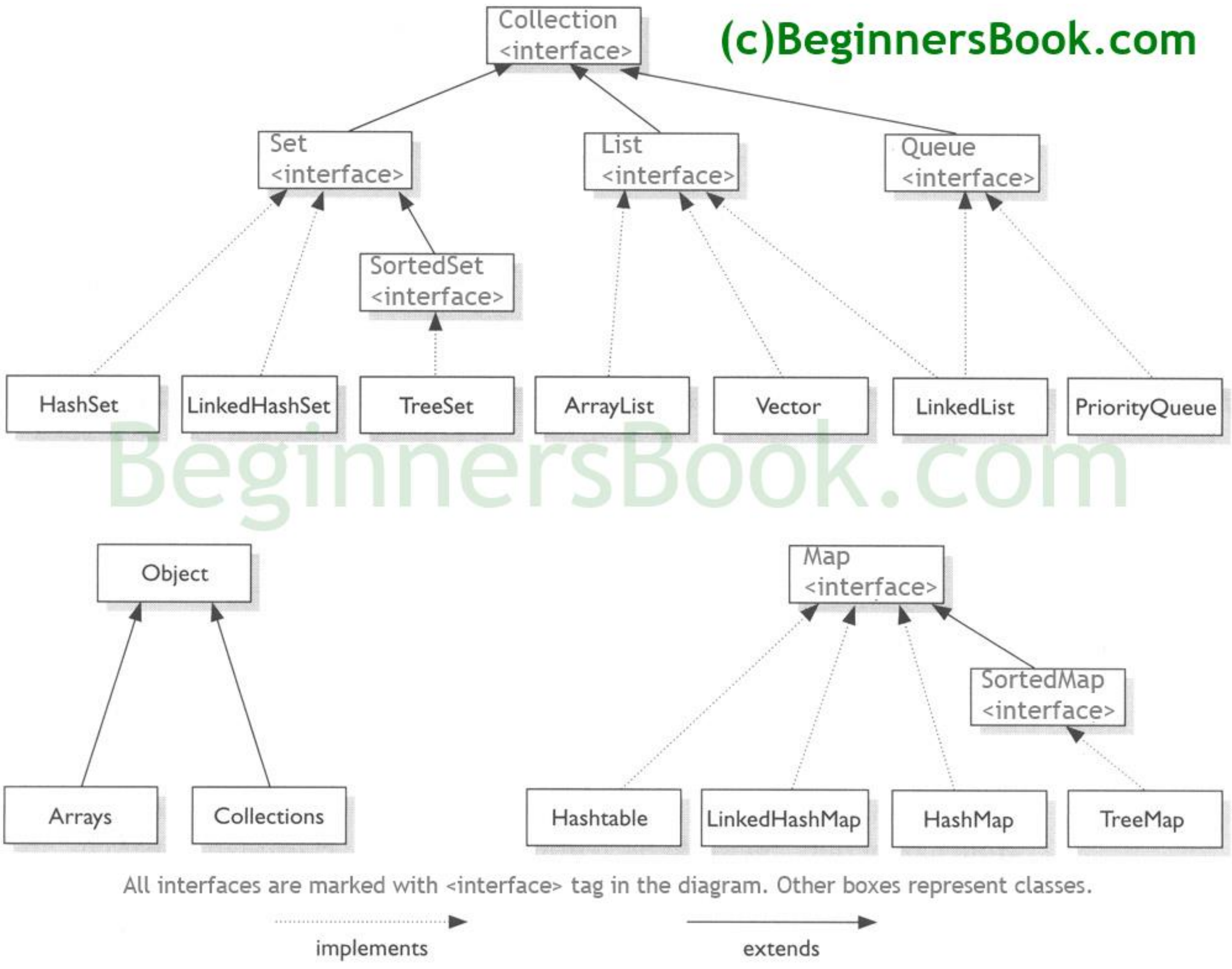
ObjectA	Less than	ObjectB	Negative Integer
ObjectA	Equal	ObjectB	Zero
ObjectA	Greater than	ObjectB	Positive Integer

Comparable and Comparator Interfaces

- Objects which implement Comparable in java can be used as keys in a TreeMap/TreeSet without implementing any other interface.

- Using Comparator interface, we can write different sorting based on different attributes of objects to be sorted.

Java Collection Framework



Classic algorithms
for manipulating a list

- Linear search: O(n)

- Binary search: O(log n)

- Selection Sort: $O(n^2)$ (same for Insertion and Bubble Sort)

- Quick Sort: O(n*log n) (average)
  $O(n^2)$ (worst)