Fundamentals of Data Structure

Mahesh Shirole

VJTI, Mumbai-19

Slides are prepared from
1. Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014
2.Data Structures and Algorithms in Java, by Robert Lafore, Second Edition, Sams Publishing

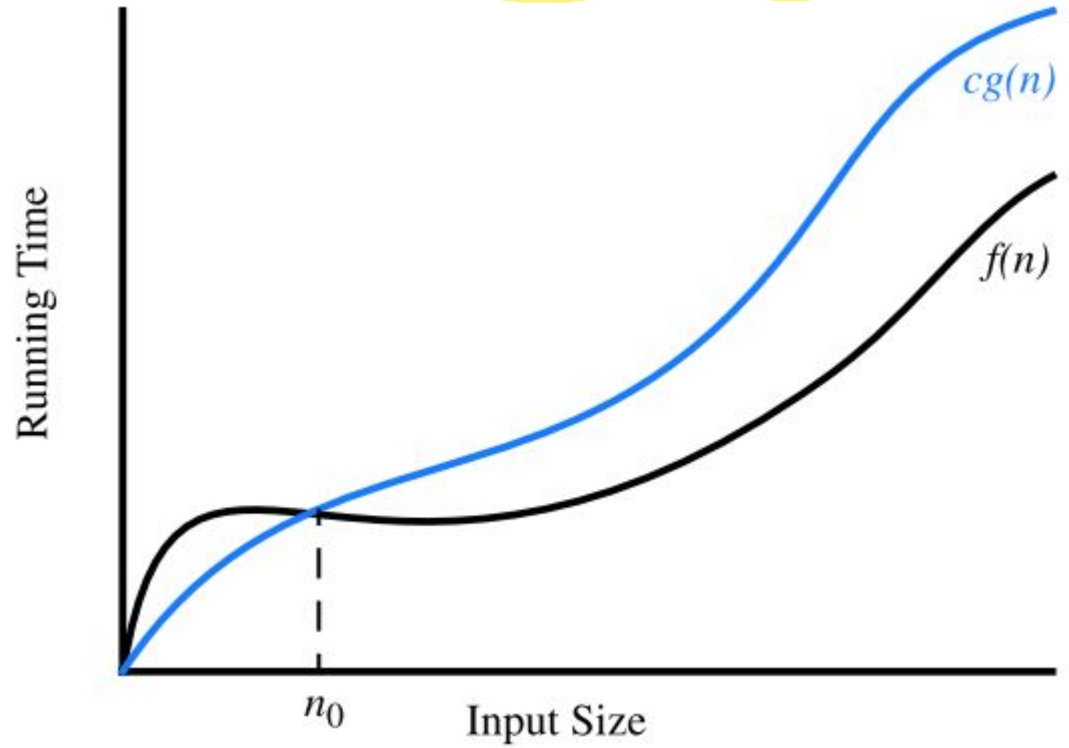
Asymptotic Analysis

- In algorithm analysis, we focus on the growth rate of the running time as a function of the input size ***n***, taking a "big-picture" approach
- We analyze algorithms using a mathematical notation for functions that disregards constant factors
- We can perform an analysis of an algorithm by estimating the number of primitive operations executed up to a constant factor, rather than getting bogged down in language-specific or hardware-specific analysis of the exact number of operations that execute on the computer

The "Big-Oh" Notation

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $O(g(n))$ if there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that

$$f(n) \leq c \cdot g(n), \quad \text{for} \quad n \geq n_0.$$



Example

- The function 8n+5 is ***O(n)***
- By the big-Oh definition, we need to find a real constant ***c*** ***>0*** and an integer constant ***n0*** ***≥ 1*** such that $8n+5 \le cn$ for every integer ***n ≥ n0***
- It is easy to see that a possible choice is c = 9 and n0 = 5
- The big-Oh notation allows us to say that a function ***f(n)*** is "less than or equal to" another function ***g(n)*** up to a constant factor and in the asymptotic sense as ***n*** grows toward infinity
- *" **f(n) = O(g(n))**"* ==>*" **f(n) is O(g(n))**"*
- Alternatively, we can say " ***f(n) is order of g(n)***"

Some Properties of the Big-Oh Notation

- The big-Oh notation allows us to ignore constant factors and lower-order terms and focus on the main components of a function that affect its growth

Example 4.7: *$5n^4 + 3n^3 + 2n^2 + 4n + 1$ is $O(n^4)$.*

Justification: Note that $5n^4 + 3n^3 + 2n^2 + 4n + 1 \leq (5+3+2+4+1)n^4 = cn^4$, for $c = 15$, when $n \geq n_0 = 1$. ■

- In fact, we can characterize the growth rate of any polynomial function

Proposition 4.8: *If $f(n)$ is a polynomial of degree d, that is,*

$$f(n) = a_0 + a_1 n + \cdots + a_d n^d,$$

and $a_d > 0$, then $f(n)$ is $O(n^d)$.

Justification: Note that, for $n \geq 1$, we have $1 \leq n \leq n^2 \leq \cdots \leq n^d$; hence,

$$a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d \leq (|a_0| + |a_1| + |a_2| + \cdots + |a_d|)\, n^d.$$

We show that $f(n)$ is $O(n^d)$ by defining $c = |a_0| + |a_1| + \cdots + |a_d|$ and $n_0 = 1$.

Characterizing Functions in Simplest Terms

- In general, we should use the big-Oh notation to characterize a function as closely as possible
- While it is true that the function $f(n) = 4n^3 + 3n^2$ is $O(n^5)$ or even $O(n^4)$, it is more accurate to say that $f(n)$ is $O(n^3)$

Big-Omega

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers. We say that $f(n)$ is $\Omega(g(n))$, pronounced "$f(n)$ is big-Omega of $g(n)$," if $g(n)$ is $O(f(n))$, that is, there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that

$$f(n) \geq cg(n), \quad \text{for} \quad n \geq n_0.$$

This definition allows us to say asymptotically that one function is greater than or equal to another, up to a constant factor.

Example 4.14: $3n\log n - 2n$ is $\Omega(n\log n)$.

Justification: $3n\log n - 2n = n\log n + 2n(\log n - 1) \geq n\log n$ for $n \geq 2$; hence, we can take $c = 1$ and $n_0 = 2$ in this case. ■

Big-Theta

In addition, there is a notation that allows us to say that two functions grow at the same rate, up to constant factors. We say that $f(n)$ is $\Theta(g(n))$, pronounced "$f(n)$ is big-Theta of $g(n)$," if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$, that is, there are real constants $c' > 0$ and $c'' > 0$, and an integer constant $n_0 \geq 1$ such that

$$c'g(n) \leq f(n) \leq c''g(n), \quad \text{for} \quad n \geq n_0.$$

Example 4.15: $3n\log n + 4n + 5\log n$ *is* $\Theta(n\log n)$.

Justification: $3n\log n \leq 3n\log n + 4n + 5\log n \leq (3+4+5)n\log n$ for $n \geq 2$. ■

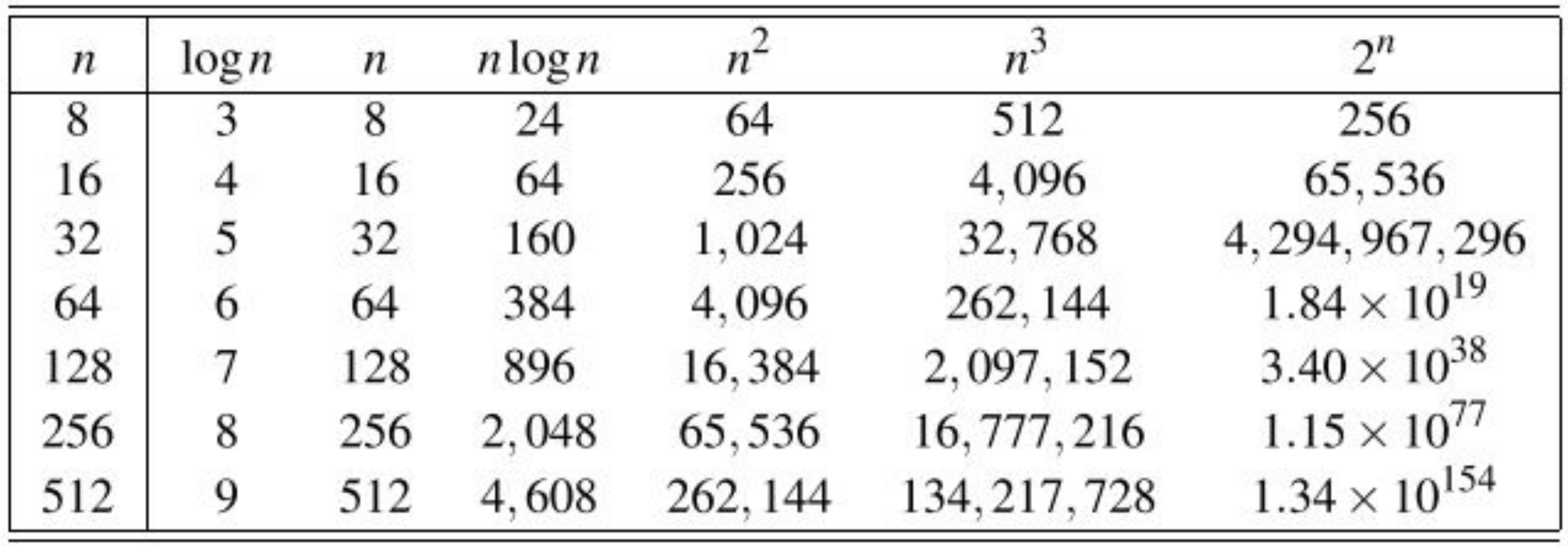
Comparative Analysis

- The big-Oh notation is widely used to characterize running times and space bounds in terms of some parameter ***n***, which is defined as a chosen measure of the "**size**" of the problem.
- Suppose two algorithms solving the same problem are available: an algorithm **A**, which has a running time of ***O(n)***, and an algorithm **B**, which has a running time of ***O(n^2)***.
- Which algorithm is better?
- We know that ***n*** is ***O(n^2)***, which implies that algorithm **A** is asymptotically better than algorithm **B**, although for a small value of ***n***, **B** may have a lower running time than **A**

Comparative Analysis

- We can use the big-Oh notation to order classes of functions by asymptotic growth rate
- Our seven functions are ordered by increasing growth rate in the following sequence, such that ***f(n)*** is ***O(g(n))*** if function ***f(n)*** precedes function ***g(n)***:

1 <= logn <= n <= nlogn <= n2 <= n3 <= 2n.

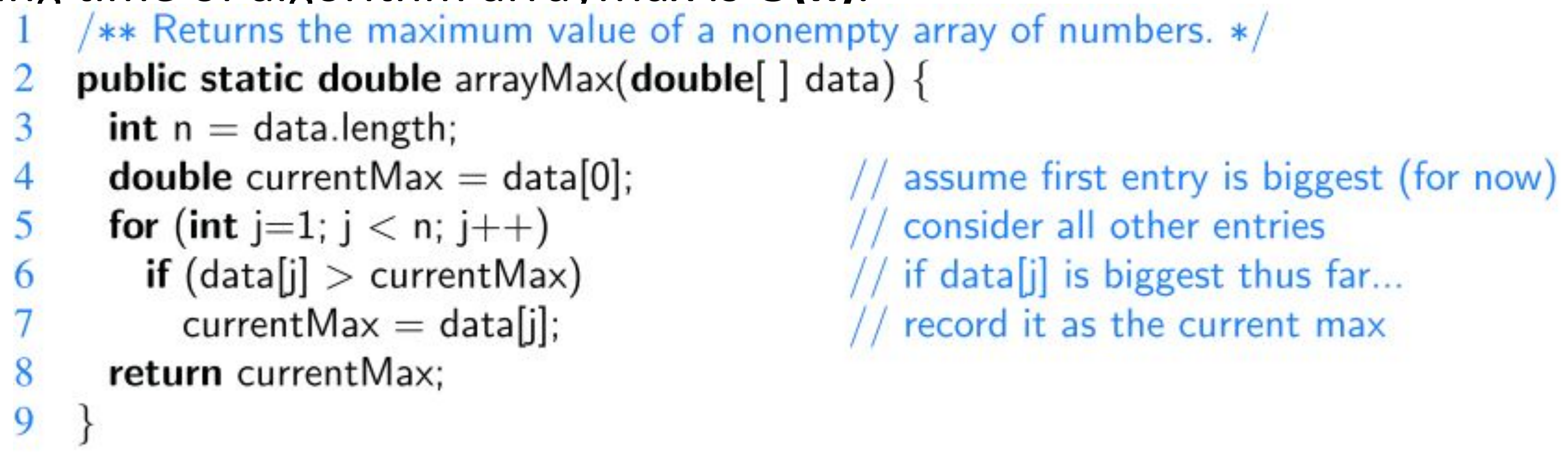


n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}

Table 4.3: Selected values of fundamental functions in algorithm analysis.

Examples of Algorithm Analysis- 1. Constant-Time Operations

- All of the primitive operations are assumed to run in constant time; formally, we say they run in O(1) time.
- Finding the Maximum of an Array: It is a classic example of an algorithm with a running time that grows proportional to n, we consider the goal of finding the largest element of an array.
- The account for the number of primitive operations being $c' \cdot (n-1)+c''$ for appropriate constants c' and c'' that reflect, respectively, the work performed inside and outside the loop body.
- The running time of algorithm arrayMax on an input of size n is at most $c' \cdot (n-1)+c'' = c' \cdot n+(c'' - c') \leq c' \cdot n$ if we assume, without loss of generality, that $c'' \leq c'$.
- The running time of algorithm arrayMax is **O(n)**.



```
/** Returns the maximum value of a nonempty array of numbers. */
public static double arrayMax(double[ ] data) {
  int n = data.length;
  double currentMax = data[0];                 // assume first entry is biggest (for now)
  for (int j=1; j < n; j++)                    // consider all other entries
    if (data[j] > currentMax)                  // if data[j] is biggest thus far...
      currentMax = data[j];                    // record it as the current max
  return currentMax;
}
```

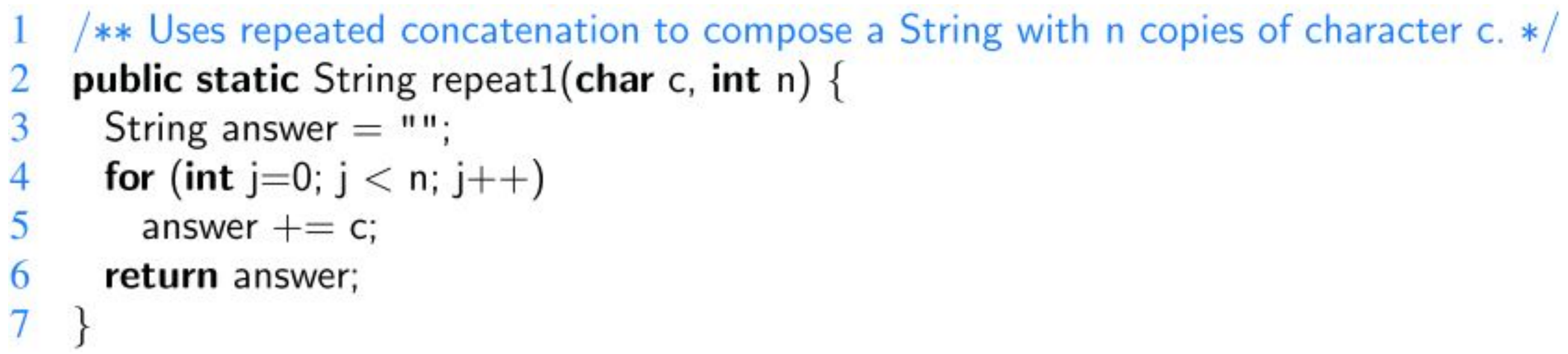
Examples of Algorithm Analysis-
2. Quadratic-Time Operations

- Consider a Java implementation of generating long string of some character
- The most important aspect of this implementation is that strings in Java are *immutable objects*
- The line 5 does not cause a new character to be added to the existing String instance; instead it produces a new String with the desired sequence of characters, and then it reassigns the variable, answer, to refer to that new string.
- The first time through this loop, the result has length 1, the second time through the loop the result has length 2, and so on, until we reach the final string of length n.

1+2+· · ·+n,

which we recognize as the familiar O(n2) summation

- Therefore, the total time complexity of the repeat1 algorithm is O(n2).



```
1  /** Uses repeated concatenation to compose a String with n copies of character c. */
2  public static String repeat1(char c, int n) {
3    String answer = "";
4    for (int j=0; j < n; j++)
5      answer += c;
6    return answer;
7  }
```

Examples of Algorithm Analysis-
3. Cubic-Time Operations

- Three-Way Set Disjointness: Suppose we are given three sets, A, B, andC, stored in three different integer arrays. The three-way set disjointness problem is to determine if the intersection of the three sets is empty, namely, that there is no element x such that x $\in$ A, x $\in$ B, and x $\in$C.
- This simple algorithm loops through each possible triple of values from the three sets to see if those values are equivalent.
- If each of the original sets has size n, then the worst-case running time of this method is O(n^3).

```
/** Returns true if there is no element common to all three arrays. */
public static boolean disjoint1(int[ ] groupA, int[ ] groupB, int[ ] groupC) {
  for (int a : groupA)
    for (int b : groupB)
      for (int c : groupC)
        if ((a == b) && (b == c))
          return false;                          // we found a common value
  return true;                                   // if we reach this, sets are disjoint
}
```

Class Assignment-03

- Find running time analysis of Bubble sort, Selection Sort, and Insertion sort for 10,100, 1000, and 10000 elements of array.
- Do asymptotic analysis of Bubble sort, Selection Sort, and Insertion sort Algorithms. Also no analysis for the number of comparisons and swap operations in each algorithm.

Lab Assignment- 02

- Extend your long integer array class with following new operations
- initArray(): void -- initialize all elements with random value [java.util.Random]
- bubbleSort()
- selectionSort()
- insertionSort()

- Write a test application for your class to demonstrate above operations