Data Structures and Algorithms

Dr. L. Rajya Lakshmi

Sorted or unsorted lists

- Make a choice between sorted or unsorted implementation
- Depends upon the operations to be supported by the list and relative importance of these operations
- To search for an element in a sorted list, we can use binary search

  Ex: A = {2, 3, 6, 8, 12, 23, 33, 45, 65}

  We want to search "65"

  Initialize L=0 and R = 8

  compute m = floor((L+R)/2)=4; Since 12 < 65, update L as: L = 5

  compute m = floor((L+R)/2)=6; Since 33 < 65, update L as: L = 7

  compute m = floor((L+R)/2)=7; Since 45 < 65, update L as: L = 8

  compute m = floor((L+R)/2)=8; since 65 found at position 8 return the same

Sorted or unsorted lists

- To search for an element in an unsorted list, we have to use sequential search

  Ex: A = {2, 3, 6, 8, 12, 23, 33, 45, 65}

  We want to search “65”
- Similar is the case with deletion operation
- Depending upon the relative importance of the operations we can make a choice between sorted and unsorted lists

Analysing Algorithms

- Analysing the dependency of running time on the size of input
- A general methodology that associates a function f(n) to characterize the running time in terms of input
  - A language for describing algorithms
  - A computational model that algorithms execute within
  - A metric for measuring algorithm running time
  - An approach for characterizing running times

Pseudo-code

- The programming language constructs that will be used:
  - Use standard mathematical symbols to describe numeric and Boolean operations/expressions
  - Use “←” for assignment instead of “=“
  - Use “=“ for equality relationship
  - Method declaration: Algorithm name(param1, param2)
  - Decision structures: if … then … [else …]
  - while-loops: while . . . do
  - for-loops: for . . . do
  - Array indexing: A[i], A[i,j]
  - Method calls: object.method(args)
  - Method return: return value
  - Use indentation to signify the beginning of a new loop

Analytic approach

- Define a set of high-level primitive operations independent of programming language
  - Data movement (assign)
  - Switching control (branch, subroutine call, return)
  - Logical and arithmetic operations (addition, comparison)
  - Indexing into an array
- The execution time of these primitive operations is dependent on the hardware and software environment (constant)
- Count the number of primitive operations executed by the algorithm and use that count as a high-level estimate of the running time

Count the primitive operations

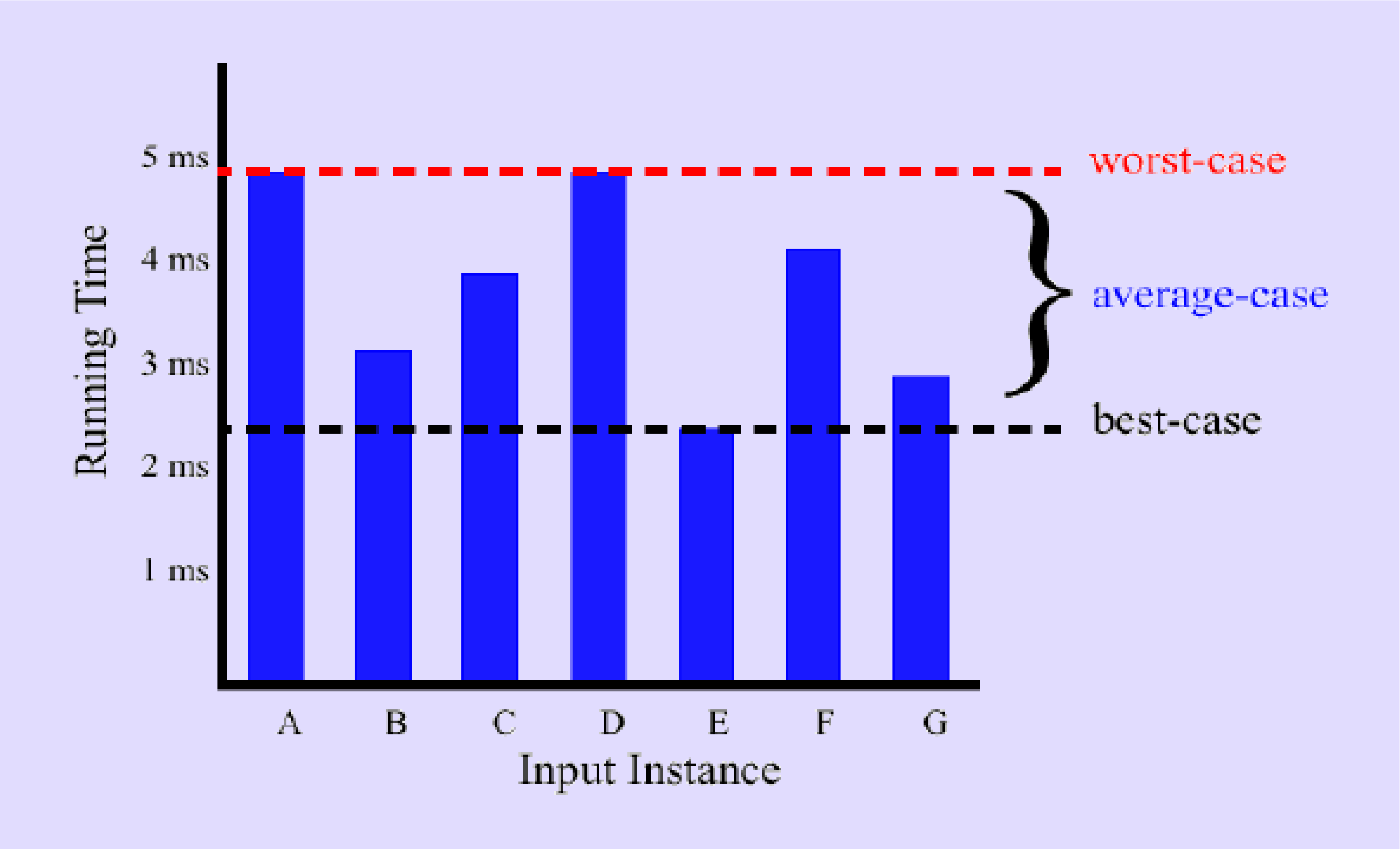
Algorithm arrayMax(A, n)	
Input: An array A storing n integers and the size	
Output: The maximum element in A	
currentMax ← A[0]	2 units
for i←1 to n-1 do	1 + n units
if currentMax < A[i] then	2(n-1) units
currentMax ← A[i]	0-2(n-1) units
	2(n-1) units
return currentMax	1 unit

Count the primitive operations

- Total running time of arrayMax is:
  - Best case: elements are in sorted decreasing order: 2+1+n+4(n-1)+1 = 5n
  - Worst case: elements are sorted increasing order: 2+1+n+6(n-1)+1 = (7n-2)
  - Average case: elements are partially sorted: between 5n and (7n-2)

Best, average, and worst case

- Expected running time based on a given input distribution



Best, average, and worst case

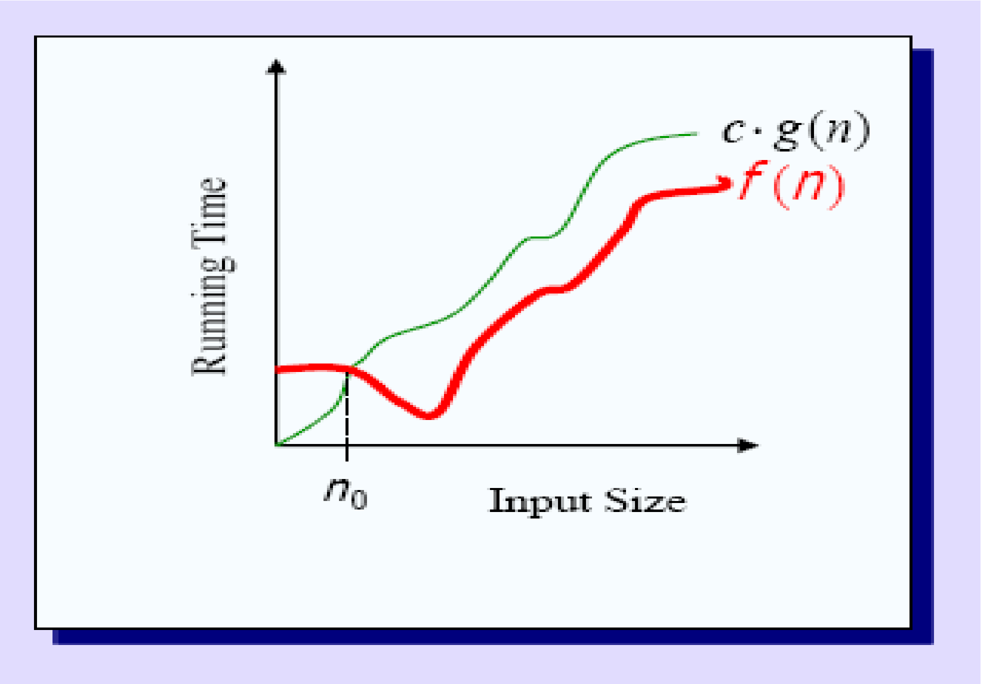
- We are mainly interested in worst-case bound
- Need to identify the worst-case scenario
- An algorithm that performs best in the worst-case scenario also performs best in the best case scenario (expectation)

Asymptotic notation

- The approach of counting primitive operations would be cumbersome to analyse complicated algorithms
- A simplified analysis that estimates the number primitive operation executed by an algorithm up to a constant factor by counting the steps of the algorithm
- Asymptotic notation facilitates analysis by getting rid of details

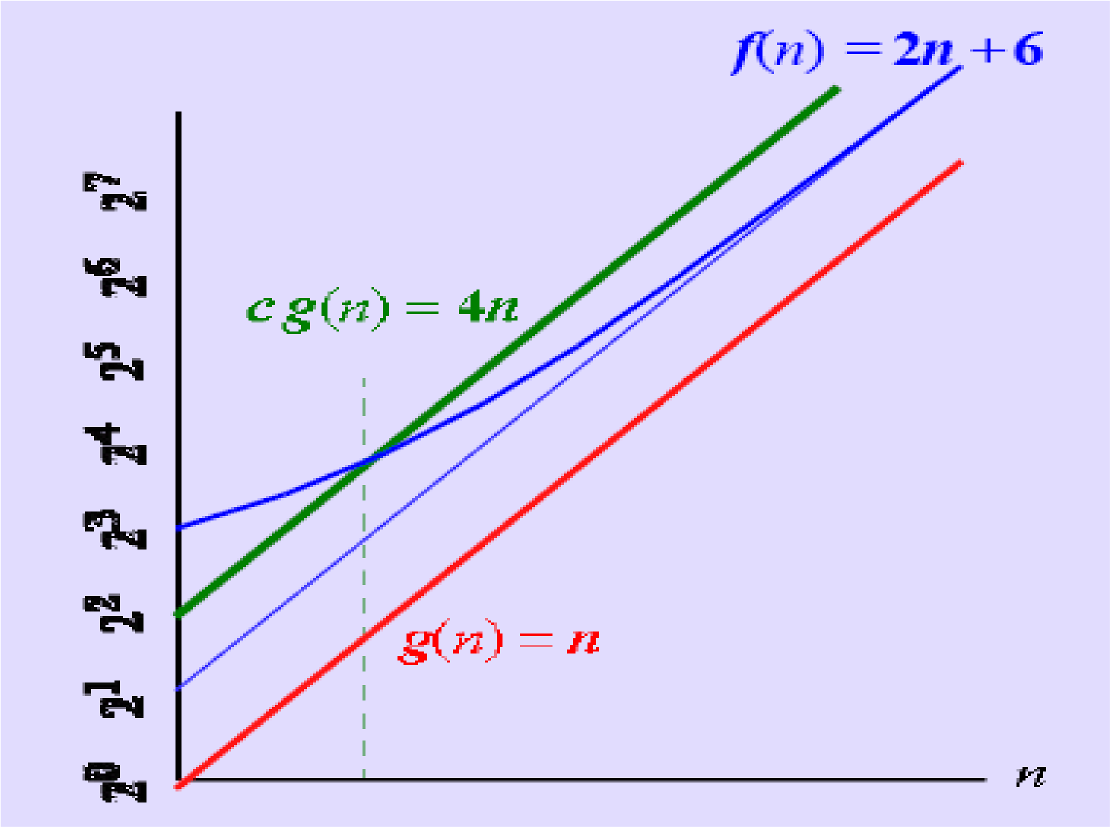
Asymptotic notation

- The "big-Oh" O-Notation
  - Provides asymptotic upper bound on the running time
  - f(n) is O(g(n)), if there exist constants "c" and "n_0" s.t. $f(n) \le cg(n)$ for $n \ge n_0$
  - f(n) and g(n) are functions over nonnegative integers and non-decreasing functions



Asymptotic notation

- f(n) = 2n +6 and g(n) = n
- f(n) is O(g(n)), with c= 4 and n_0 = 3



Asymptotic notation

- How to find the order of a function?
  - If $f(n)$ is a polynomial of degree "d", then $f(n)$ is $O(n^d)$
  - $\log n^x$ is $O(\log n)$ for any fixed $x > 0$
- Simple rule: drop lower order terms and constants

  Ex: $50 n \log n$ is $O(n \log n)$, $7n – 2$ is $O(n)$, $8n^2 \log n + 5n^2 + n$ is $O(n^2 \log n)$

  Note: Though $50 n \log n$ is $O(n^5)$, it is expected that such an approximation be of as small an order as possible

  Characterize the given function as closely as possible

Asymptotic analysis of running time

- Using O-notation, express the number of primitive operations executed as a function of input size
- How to compare asymptotic running times?
  - Algorithm that runs in $O(n)$ time is better than that runs in $O(n^2)$
  - $O(\log n)$ is better than that $O(n)$
  - Hierarchy of running times: $\log n < n < n^2 < n^3 < a^n$

Example of asymptotic analysis

Algorithm prefixAverages1(X):

Input: An n-element array X of numbers.

Output: An n-element array A of numbers such that A[i] is the average of elements X[0], ... , X[i].

```
for i ← 0 to n-1 do
    a ← 0
    for j ← 0 to i do
        a ← a + X[j]   ← 1 step
    A[i] ← a/(i+1)
return array A
```

Analysis: running time is $O(n^2)$

Example of asymptotic analysis

Algorithm prefixAverages2(X):

Input: An *n*-element array X of numbers.

Output: An *n*-element array A of numbers such that A[*i*] is the average of elements X[0], ... , X[*i*].

```
s ← 0
for i ← 0 to n-1 do
    s ← s + X[i]
    A[i] ← s/(i+1)
return array A
```

Analysis: Running time is O(n)

Classes of functions

- Logarithmic: $O(\log n)$
- Linear: $O(n)$
- Quadratic: $O(n^2)$
- Polynomial: $O(n^k)$ ($k \geq 1$)
- Exponential: $O(a^n)$ ($n > 1$)