Design and Analysis of Algorithms

Vandana M L
Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Analysis Framework

Slides courtesy of **Anany Levitin**

Vandana M L

Department of Computer Science & Engineering

Analysis Framework

What do you mean by analysing an algorithm?

Investigation of Algorithm's efficiency with respect to two resources

- Time
- Space

What is the need for Analysing an algorithm?

- To determine resource consumption
  - CPU time
  - Memory space
- Compare different methods for solving the same problem before actually implementing them and running the programs.
- To find an efficient algorithm

Complexity of an Algorithm

- A measure of the performance of an algorithm
- An algorithm's performance depends on
  - *internal* factors
    - Time required to run
    - Space (memory storage)required to run
  - *external* factors
    - Speed of the computer on which it is run
    - Quality of the compiler
    - Size of the input to the algorithm

Performance of Algorithm

important Criteria for performance:

- Space efficiency - the memory required, also called, space complexity
- Time efficiency - the time required, also called time complexity

Space Complexity

S(P)=C+SP(I)

- Fixed Space Requirements (C)
  Independent of the characteristics of the inputs and outputs
  - instruction space
  - space for simple variables, fixed-size structured variable, constants
- Variable Space Requirements (SP(I))
  dependent on the instance characteristic I
  - number, size, values of inputs and outputs associated with I
  - recursive stack space, formal parameters, local variables, return address

$S(P)=C+S_P(I)$

```
float rsum(float list[ ], int n)
{
  if (n)
     return rsum(list, n-1) + list[n-1]
  return 0
}
```

$S_{sum}(I)=S_{sum}(n)=6n$

Type	Name	Number of bytes
parameter: float parameter: integer return address:(used internally)	list [ ] n	2 2 2
TOTAL per recursive call		6

Time Complexity

$T(P)=C+T_P(I)$

- Compile time (C)
  independent of instance characteristics

- run (execution) time TP

How to measure time complexity?

- Theoretical Analysis
- Experimental study

Experimental study

- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Get an accurate measure of the actual running time
- Use a method like System.currentTimeMillis()
- Plot the results

Limitations of Experimental study

- It is necessary to implement the algorithm, which may be difficult
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used
- Experimental data though important is not sufficient

Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, n.
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Two approaches:

1.Order of magnitude/asymptotic categorization –

This uses coarse categories and gives a general idea of performance.
If algorithms fall into the same category, if data size is small,
or if performance is critical, use method 2

2.Estimation of running time -

1. *operation counts* - select operation(s) that are executed most frequently and determine how many times each is executed.
2. *step counts* - determine the total number of steps, possibly lines of code, executed by the program.

Analysis Framework

- Measuring an input’s size
- Measuring running time
- Orders of growth (of the algorithm’s efficiency function)
- Worst-base, best-case and average efficiency

Measuring an input's size

Efficiency is defined as a function of input size.

Input size depends on the problem.

Example 1, what is the input size of the problem of sorting n numbers?

Example 2, what is the input size of adding two n by n matrices?

Units for Measuring Running Time

- Measure the running time using standard unit of time measurements, such as seconds, minutes?
  Depends on the speed of the computer.

- count the number of times each of an algorithm's operations is executed.
  (step count method)
  Difficult and unnecessary

- count the number of times an algorithm's basic operation is executed.

  Basic operation: the most important operation of the algorithm, the operation contributing the most to the total running time.

  For example, the basic operation is usually the most time-consuming operation in the algorithm's innermost loop.

Analysis in the RAM Model

SmartFibonacci(n)	*cost*	*times* ($n > 1$)
if $n = 0$	c_1	1
then return 0	c_2	0
elseif $n = 1$	c_3	1
then return 1	c_4	0
else $pprev \leftarrow 0$	c_5	1
$prev \leftarrow 1$	c_6	1
for $i \leftarrow 2$ **to** n	c_7	n
do $f \leftarrow prev + pprev$	c_8	$n - 1$
$pprev \leftarrow prev$	c_9	$n - 1$
$prev \leftarrow f$	c_{10}	$n - 1$
return f	c_{11}	1

$$T(n) = c_1 + c_3 + c_5 + c_6 + c_{11} + nc_7 + (n-1)(c_8 + c_9 + c_{10})$$

$T(n) = nC_1 + C_2 \quad \Rightarrow T(n)$ is a *linear function* of n

Input Size and Basic Operation Examples

Problem	***Input size measure***	***Basic operation***
Search for a key in a list of *n* items	Number of items in list, *n*	Key comparison
Add two *n* by *n* matrices	Dimensions of matrices, *n*	addition
multiply two matrices	Dimensions of matrices, *n*	multiplication

Theoretical Analysis of Time Efficiency: Basic operation count

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

input size

$$T(n) \approx c_{op} C(n)$$

running time

execution time
for the basic operation

Number of times the
basic operation is
executed

C(n) Basic Operation Count

- The efficiency analysis framework ignores the multiplicative constants of C(n) and focuses on the orders of growth of the C(n).
- Simple characterization of the algorithm's efficiency by identifying relatively significant term in the C(n).

Why do we care about the order of growth of an algorithm's efficiency function, i.e., the total number of basic operations?

- Because, for smaller inputs, it is difficult to distinguish inefficient algorithms vs. efficient ones.
- For example, if the number of basic operations of two algorithms to solve a particular problem are n and n^2 respectively, then
  - if n = 2, Basic operation will be executed 2 and 4 times respectively for algorithm1 and 2.

    Not much difference!!!
  - On the other hand, if n = 10000, then it does makes a difference whether the number of times the basic operation is executed is n or n^2.

Order of Growth

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}		

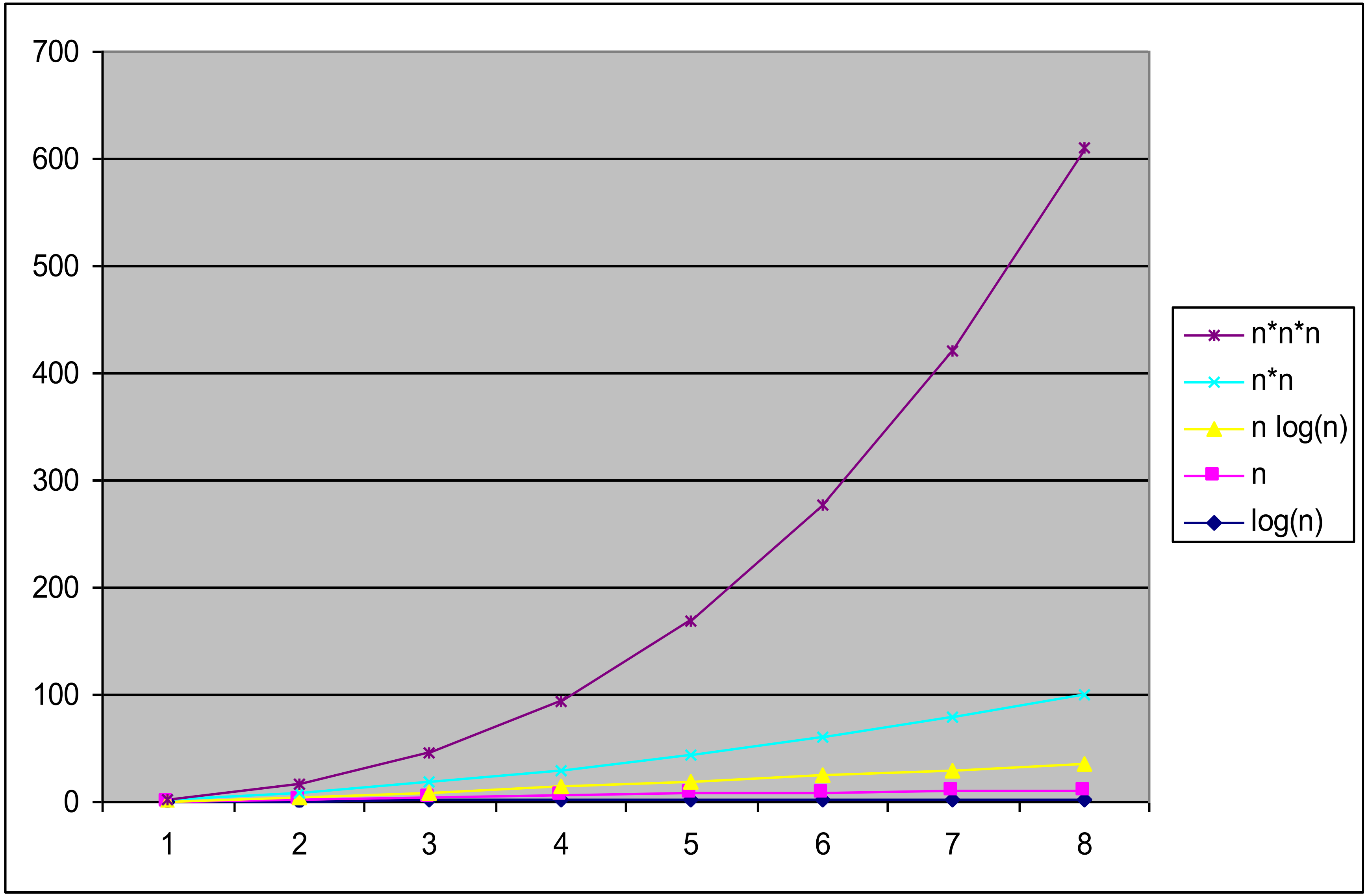
Exponential-growth functions

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

Orders of growth:

- consider only the leading term of a formula
- ignore the constant coefficient.

Design and Analysis of Algorithms
Order of Growth



Basic Efficiency 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

- Algorithm efficiency depends on the input size n
- For some algorithms efficiency depends on type of input.

Example: Sequential Search

Problem: Given a list of n elements and a search key K, find an element equal to K, if any.

Algorithm: Scan the list and compare its successive elements with K until either a matching element is found (successful search) or the list is exhausted (unsuccessful search)

Given a sequential search problem of an input size of n,
what kind of input would make the running time the longest?
How many key comparisons?

Best, Worst and Average case Analysis

- Worst case Efficiency
  - Efficiency (# of times the basic operation will be executed) for the worst case input of size n.
  - The algorithm runs the longest among all possible inputs of size n.
- Best case
  - Efficiency (# of times the basic operation will be executed) for the best case input of size n.
  - The algorithm runs the fastest among all possible inputs of size n.
- Average case:
  - Efficiency (#of times the basic operation will be executed) for a typical/random input of size n.
  - NOT the average of worst and best case
- How to find the average case efficiency?

```
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: Returns the index of the first element of A that
   matches K or –1 if there are no matching elements
   i ←0
   while i < n and A[i] ‡ K do
         i ← i + 1
   if i < n          //A[I] = K
         return i
   else
         return -1
```

Best, Worst and Average case Analysis: Sequential Search

- Worst-Case: Cworst(n) = n
- Best-Case: Cbest(n) = 1
- Average-Case

  from **(n+1)/2** to **(n+1)**

Average case Analysis: Sequential Search

Let **'p'** be the probability that key is found in the list

Assumption: All positions are equally probable

Case1: key is found in the list

$C_{avg,case1}(n) = p*(1 + 2 + \ldots + n) / n = p*(n + 1) / 2$

Case2: key is not found in the list

$C_{avg,\ case2}(n) = (1-p)*(n)$

$\mathbf{C_{avg}(n) = p}(n + 1) / 2 \mathbf{+ (1 - p)}(n)$

THANK YOU

Vandana M L
Department of Computer Science & Engineering
vandanamd@pes.edu