Design and Analysis of Algorithms

Lecture 1-2

Instructor: Dr. G P Gupta

Course Goals

- As a result of successfully completing this course, students will:
  - **Learn fundamentals of Algorithm design and Analysis:**
    - How to devise algorithms?
    - How to validate algorithm?
    - How to analyze performance of an algorithm?

Design and Analysis of Algorithms

- ***Design:*** design algorithms which minimize the cost
- ***Analysis:*** predict the cost of an algorithm in terms of resources and performance

Syllabus & Course Outlines

- **Course resources :** Piazza webpage
  - https://piazza.com/
  - **Piazza is designed to connect students, and Professors so that every student can get the help they need when they need it.**
- **Course Outlines**

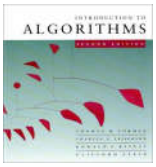
Course Outlines

Unit No.	Lecture Topics	Reference
1.	Analyzing algorithms, Algorithm types, Recurrence Equations, **Growth function: Asymptotic notation**, Standard notation & common functions, Recurrence relation, different methods of solution of recurrence equations with examples.	Lecture notes and PPT, Ref-1,Ref-2.
2.	Introduction to **Divide and Conquer paradigm**, Quick and merge sorting techniques, Linear time selection algorithm, **the basic divide and conquer algorithm** for matrix multiplication Strassen Multiplication and, Red Black tree, Binary Search tree , heap sort, shell & bucket sort.	Lecture notes & PPT, Ref-1,Ref-2.
3.	Overview of the **greedy paradigm** examples of exact optimization solution (minimum cost spanning tree), Knapsack problem, Single source shortest paths. Overview, difference between **dynamic programming** and divide and conquer, Applications: Shortest path in graph, Matrix multiplication, Traveling salesman Problem, longest Common sequence.	Lecture notes and PPT, Ref-1,Ref-2, Ref-3

Course Outlines cont..

Unit No.	Lecture Topics	Reference
4.	**Representational issues in graphs**, Depth first search & Breath first search on graphs, Computation of biconnected components and strongly connected components using DFS, Topological sorting of nodes of an acyclic graph & applications, Shortest Path Algorithms , Bellman-Ford algorithm, Dijkstra's algorithm & Analysis of Dijkstra's algorithm using heaps, Floyd-Warshall's all pairs shortest path algorithm	Lecture notes and PPT, Ref-1,Ref-2, Ref-3
5.	The general **string problem** as a finite automata, Knuth Morris and Pratt algorithms, Linear time analysis of the KMP algorithm, The Boyer-Moore algorithm. **Backtracking & Recursive backtracking**, Applications of backtracking paradigm ,Complexity measures, **Polynomial Vs Non-polynomial time complexity**; NP- hard and NP-complete classes, examples.	Lecture notes and PPT, Ref-1,Ref-2, Ref-3

Textbook and References



1. T.H. **Cormen**, C.E. Leiserson, R.L. Rivest and C. Stein, **"Indtroduction to Algoritms"**, Prentice-Hall of India, **3ed**, 2015.
2. Ellis Horowitz, Sartaj Sahni, " **Fundamnetals of Computer Algorithms**" 2ed, University Press (ndia), 2008.
3. A.V. Levitin, **"Introduction to the Design and Analysis of algorithms"**, Pearson Education, 2006.

Introduction

- The **methods of algorithm design** form ***one of the core practical technologies*** of computer science.
- **Origin of Algorithms**

Algorithms

- The word *algorithm* comes from the name of a **Persian mathematician**:
  - **Abu Ja'far Mohammed ibn-i Musa al Khowarizmi.**
- In computer science, ***this word refers to a special method useable by a computer for solution of a problem***.
- The statement of the problem specifies in general terms the desired input/output relationship.
  - For example, sorting a given sequence of numbers into nondecreasing order provides fertile ground for introducing many standard design techniques and analysis tools.

What is course about?

The theoretical study of design and analysis of computer algorithms

Basic goals for an algorithm:

- always correct
- always terminates
- performance
  - Performance often draws the line between what is possible and what is impossible.

L1.10

Algorithm Definition

- **A sequence of computational steps** that transform the input to the desired output
- *An algorithm* ***must halt within finite time with the right output***
- **Example:**

L1.11

Algorithm Definition cont..

- an algorithm is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**.
- ***All algorithm must satisfy the following Criteria:***
  - ***Input:*** ***zero or more***
  - ***Output:*** ***at least one***
  - ***Definiteness:*** ***each instruction is clear and unambiguous***
  - ***Finiteness:*** ***algorithm must terminates after a finite number of steps***
  - ***Effectiveness:*** ***each instruction must be very basic—so that can be evaluated using paper & pencil.***

L1.12

Many Real World Applications

- Bioinformatics
  - ➢Determine/compare DNA sequences
- Internet
  - ➢Manage/manipulate/route data
- Information retrieval
  - ➢Search and access information in large data
- Security
  - ➢Encode & decode personal/financial/confidential data
- Electronic design automation
  - ➢Minimize human effort in chip-design process

L1.13

Analysis of Algorithms

- What is the goal of analysis of algorithms?
  - To compare algorithms mainly in terms of ***running time*** but also in terms of other factors (e.g., memory requirements, programmer's effort etc.)
- What do we mean by ***running time analysis***?
  - **Determine how running time increases as the size of the problem increases**.

L1.14

Performance Analysis

- Space Complexity:
  - amount of memory, algorithm needs to run to completion.
- Time Complexity:
  - Amount of computer time algorithm needs to run to completion.

L1.15

Input Size

- Input size (number of elements in the input)
  - size of an array
  - polynomial degree
  - # of elements in a matrix
  - # of bits in the binary representation of the input
  - vertices and edges in a graph

L1.16

Types of Analysis

- **Worst case**
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are.
- **Best case**
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest

$$Lower\ Bound \leq Running\ Time \leq Upper\ Bound$$

- **Average case**
  - Provides a prediction about the running time
  - Assumes that the input is random

L1.17

Our Machine Model

- Instructions are ***executed one after another***. **No concurrent operations**.
- Set of primitive operations:
  - **Arithmetic:** add, subtract, multiply, divide, remainder, floor, ceiling). Also,shift left/shift right (good for multiplying/dividing by 2 k).
  - **Data movement:** load, store, copy.
  - **Control**: conditional/unconditional branch, subroutine call and return.
  - **Each of these instructions takes a constant amount of time.**
- Simplifying assumption: all ops cost 1 unit
  - *Eliminates dependence on the speed of our computer, otherwise impossible to verify and to compare*

L1.18

How do we compare algorithms?

- We need to define a number of objective measures.

(1) ***Compare execution times?***

Not good: times are specific to a particular computer !!

(2) ***Count the number of statements executed?***

Not good: number of statements ***vary with the programming language*** as well as the ***style of the individual programmer***.

L1.19

Ideal Solution

- Express running time as a ***function of the input size n***
  - (i.e., *f(n)*).
- Compare different functions corresponding to running times.
- **Such an analysis *is* *independent* of machine time, programming style, etc.**

L1.20

Example

- Associate a "cost" with each statement.
- Find the "total cost“ by ***finding the total number of times each statement is executed***.

Algorithm 1

	Cost
arr[0] = 0;	c_1
arr[1] = 0;	c_1
arr[2] = 0;	c_1
...	...
arr[N-1] = 0;	c_1

	$c_1+c_1+...+c_1 = c_1 \times N$

Algorithm 2

	Cost
for(i=0; i<N; i++)	c_2
arr[i] = 0;	c_1

	$(N+1) \times c_2 + N \times c_1 =$
	$(c_2 + c_1) \times N + c_2$

L1.21

Another Example

- ***Algorithm 3*** *Cost*

sum = 0;	c_1
for(i=0; i<N; i++)	c_2
for(j=0; j<N; j++)	c_2
sum += arr[i][j];	c_3

$c_1 + c_2 \times (N+1) + c_2 \times N \times (N+1) + c_3 \times N^2$

L1.22

Getting Started

- Study two sorting algorithms as examples
- Insertion sort and its analysis
  - *Incremental* algorithm
- Merge sort with its analysis
  - *Divide-and-conquer*

L1.23

The problem of sorting

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

Example:

Input: 8 2 4 9 3 6

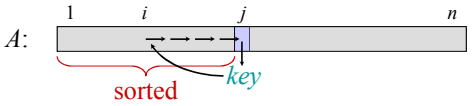
Output: 2 3 4 6 8 9

L1.24

Insertion sort

"pseudocode"

```
INSERTION-SORT (A, n)    ⊳ A[1 . . n]
  for j ← 2 to n
      do key ← A[ j]
         i ← j – 1
         while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
         A[i+1] = key
```



L1.25

Example of insertion sort

8 2 4 9 3 6

L1.26

Example of insertion sort

8 2 4 9 3 6

L1.27

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

L1.28

Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

L1.29

Example of insertion sort

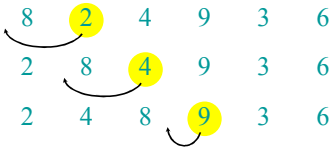
8 2 4 9 3 6

2 8 4 9 3 6

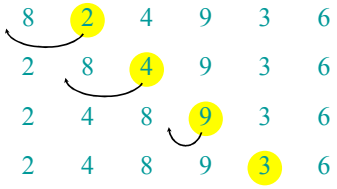
2 4 8 9 3 6

L1.30

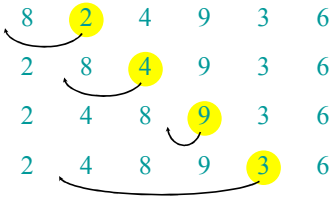
Example of insertion sort



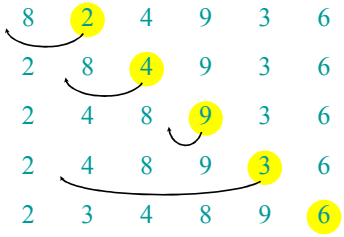
Example of insertion sort



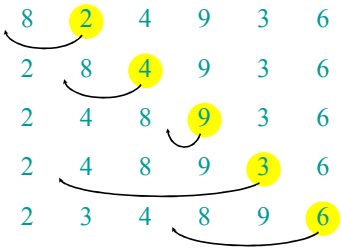
Example of insertion sort



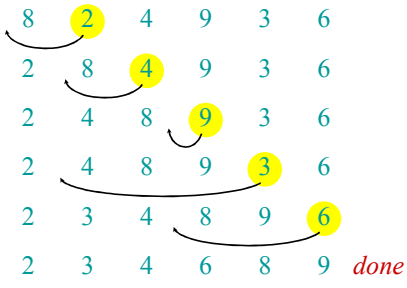
Example of insertion sort



Example of insertion sort



Example of insertion sort



Analysis of Insertion Sort

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

Analysis of Insertion Sort

- To compute T(n), the running time of INSERTION-SORT on an input of n values,
- **sum the products of the cost and times columns, obtaining :**

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1).$$

• Let t_j denote the number of times the while loop test in **line 5** is executed for that value of j .

Analysis of Insertion Sort cont..

- **Best Case:** occurs if the array is already sorted.
  - Value of t_j is 1.
- the best-case running time is

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

• running time as ***a*n+b*** for constants a and b
  • thus a ***linear function of n.***

Analysis of Insertion Sort cont..

- **Worst Case:** array is in reverse sorted order.
- Have to compare key with all elements to the left of the jth position
  - compare with j − 1 elements.
- $t_j = j$

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j \text{ and } \sum_{j=2}^{n}(t_j-1) = \sum_{j=2}^{n}(j-1).$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

Analysis of Insertion Sort cont..

- **Worst Case:**
  - Running time is

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

Analysis of Insertion Sort cont..

- **Average Case:**
- On average, the key in A[j] is less than half the elements in A[1 . . j − 1] and itís greater than the other half.
- On average, the while loop has to look halfway through the sorted subarray A[1 . . j − 1] to decide where to drop key.
- ⇒ $t_j = j/2$
- Although the average-case running time is approximately half of the worst-case running time, ***it is still a quadratic function of n.***

Kinds of analyses

Worst-case: (usually)
- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)
- $T(n)$ = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case: (NEVER)
- Cheat with a slow algorithm that works fast on *some* input.

L1.43

Machine-independent time

What is insertion sort's worst-case time?

BIG IDEAS:

- ***Ignore machine dependent constants,*** otherwise impossible to verify and to compare algorithms
- Look at ***growth*** of $T(n)$ as $n \to \infty$.

"Asymptotic Analysis"

L1.44

Insertion sort analysis

Worst case: Input reverse sorted.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \quad \text{[arithmetic series]}$$

Average case: All permutations equally likely.

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)$$

Is insertion sort a fast sorting algorithm?
- Moderately so, for small n.
- Not at all, for large n.

L1.45