Analysis Of Algorithms

Lecture No. 1

Introduction

Objective of course

In this course we will cover the following topics:

- Understand foundations of algorithms && design and analysis various variants algorithms

  Accuracy

  Efficiency

  Comparing efficiencies

- Make use of skills to understand mathematical notations in algorithms and their simple mathematical proofs

- Gain familiarity with a number of classical problems that occur frequently in real-world applications

Algo Analysis

COURSE OUTLINE

- Foundations of Algorithms, Problem solving, Proving correctness of algorithm using Loop Invariants
- Asymptotic Notations: Worst, Best and Average Case Behavior of Algorithms; Big O notation;
- Complexity Classes i.e. Constant, Linear, Quadratic; Empirical Measurements of Performance,
- Time and Space Tradeoffs in Algorithms,
- Recurrence Algorithms; Analysis of Iterative and Recurrence Relations;
- Master Theorem;
- Divide and Conquer;
- Recursive Backtracking;
- Worst Case Quadratic Sorting Algorithms, Worst or Average Case Sorting Algorithms (Quick, Heap & Merge Sort),
- Representation of Graphs, Depth First and Breadth First Traversal,
- Brute Force Algorithms;
- Greedy Algorithms; Approximation Algorithms,
- Dynamic Programming; Branch-and-Bound Techniques;
- Heuristics; Reductions: Transform and Conquer; Basic Computability:
- The Complexity Classes P and NP; Introduction to NP Complete Problem.

Algo Analysis

COURSE OUTLINE

- Tree Algorithms: understanding and making different types of trees and their algorithms.
- Graph Algorithms: understanding and making different types of graphs and their algorithms for finding shortest paths
- Heap Algorithms : developing heaps, binomial heaps, disjoint structures, and their algorithms.
- Divide and Conquer: Multiplication of Integers, Binary Search, Sorting Algorithms and Matrix Multiplication.
- Greedy Algorithms : General Characteristics of Greedy Algorithms, Minimum Spanning Trees, Shortest Graph Paths, Huffman Codes and Scheduling.
- NP-Completeness: understanding of P, NP and NP-completeness and use of algorithms for some problems

Today's Lecture

- What is algorithms
- What are different techniques used for algorithms
- Model of Computation

Definition: Algorithm

- **A computer algorithm is a detailed step by step method for solving the problem**
- **An algorithm is a sequence of Clear-cut instructions for solving the problem in a finite amount of time**
- **An algorithm is a set of well defined computational procedure that takes some values, or set of values, as input and produces some value or set of values as output.**

Definition: Algorithm

-
- **More Generally an algorithm is any well defined computational procedure that takes collection of elements as input and procedures collection of elements as output.**

Definition: Data Strcuture

-
- **DATA STRUCTURE**
- **A data structure is systematic way of organizing and accessing data with a specific relationship between the elements**

Advantage: Algorithms

- **Effective Communication:** Since it is written in a natural language like English, it becomes easy to
understand the step-by-step delineation of a solution to any particular problem.
• **Easy Debugging:** A well-designed algorithm facilitates easy debugging to detect the logical errors
that occurred inside the program.
• **Easy and Efficient Coding:** An algorithm is nothing but a blueprint of a program that helps
develop a program.
• **Independent of Programming Language:** Since it is a language-independent, it can be easily
coded by incorporating any high-level language(BASIC,C/C++).

Disadvantage: Algorithms

- Developing algorithms for complex problems would be time-consuming and difficult to understand.
  • It is a challenging task to understand complex logic through algorithms.

  Pseudocode

  Pseudo means Not genuine, artificial.

  It Looks like code to some extent, but it is not code in some specific programming language.

  It is some type of Pre-code which you can easily convert into any programming language.

  In generally pseudocodes are easily understandable by programmers and they can convert these into the any programming language.

  Advantages of Pseudocode

  • Since it is similar to a programming language, it can be quickly transformed into the actual programming language than a flowchart.
  • The layman can easily understand it.
  • Easily modifiable as compared to the flowcharts.
  • Its implementation is beneficial for structured, designed elements.
  • It can easily detect an error before transforming it into a code.

Difference: Algorithm & Pseudocode

- An algorithm is simply a problem-solving process, which is used not only in computer science to write a program but also in our day to day life.

- It is nothing but a series of instructions to solve a problem or get to the problem's solution.
- It not only helps in simplifying the problem but also to have a better understanding of it.

- However, Pseudocode is a way of writing an algorithm.
- Programmers can use informal, simple language to write pseudocode without following any strict syntax.
  - It encompasses semi-mathematical statements.

Example: Algorithm & Pseudocode

1. **Problem: Suppose there are 60 students in the class. How will you calculate the number of absentees in the class?**

- Pseudo Approach:
- Initialize a variable called as **Count** to zero, **absent** to zero, **total** to 60
- FOR EACH Student PRESENT DO the following:
  Increase the **Count** by One
- Then Subtract **Count** from **total** and store the result in **absent**
- Display the number of absent students
- Algorithmic Approach:
- Count <- 0, absent <- 0, total <- 60
- REPEAT till all students counted
  Count <- Count + 1
- absent <- total - Count
- Print "Number absent is:" , absent

Algorithm:Most Popular

-
- **Sorting Algorithms**
- **Searching algorithms**
- **Factors Of Dependence**
  - **No of items to be sorted**
  - **The extent to which elements are already sorted**
  - **Possible restrictions on Item values**
  - **The kind of storage**
- **No Algo is better than any algorithm**

One Problem many Algorithms

- Problem “The statement of the problem specify, in general terms the desired input /output relationship”

- Algorithm :The algorithm specifies the computational procedure for achieving input/output relationships or objective of the problem.

- For example: sorting of number in descending order we need number of algorithms like merge sort, heap sort, counting sort, radix sort and quick sort etc.

Criteria for Satisfaction of an Algorithm

- Supply of input quantities (zero or more)
- Production of output quantities (one or more)
- Each instruction must be clear and unambiguous
- Must terminate after a finite number of steps

PHRASES FOR WRITING ALGORITHM

Phrases such as mentioned below are used to link sequences of steps if required:-

(i) if…… then

_____

_____

_____

else

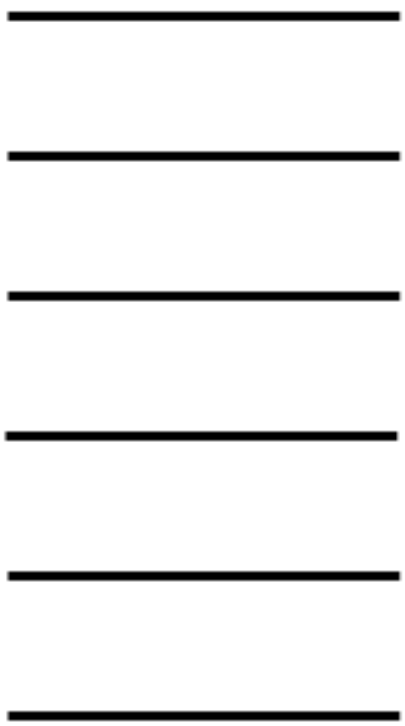
_____

_____

_____

PHRASES FOR WRITING ALGORITHM

(ii) Repeat



until

PHRASES FOR WRITING ALGORITHM

(iii) while …….. do

_____

_____

_____

(iv) do

_____

_____

_____

while

ALGORITHM FOR USING PUBLIC TELEPHONE

```
Repeat
        Get money ready
        Lift receiver
        If dialing tone
        then Dial number
                If number obtainable
                        and not engaged
                then Repeat
                        wait
                Until someone answers
                        or waited long enough
                If answered
                then Insert money
```

ALGORITHM FOR USING PUBLIC TELEPHONE

If right number

and person you want is available

then Have conversation

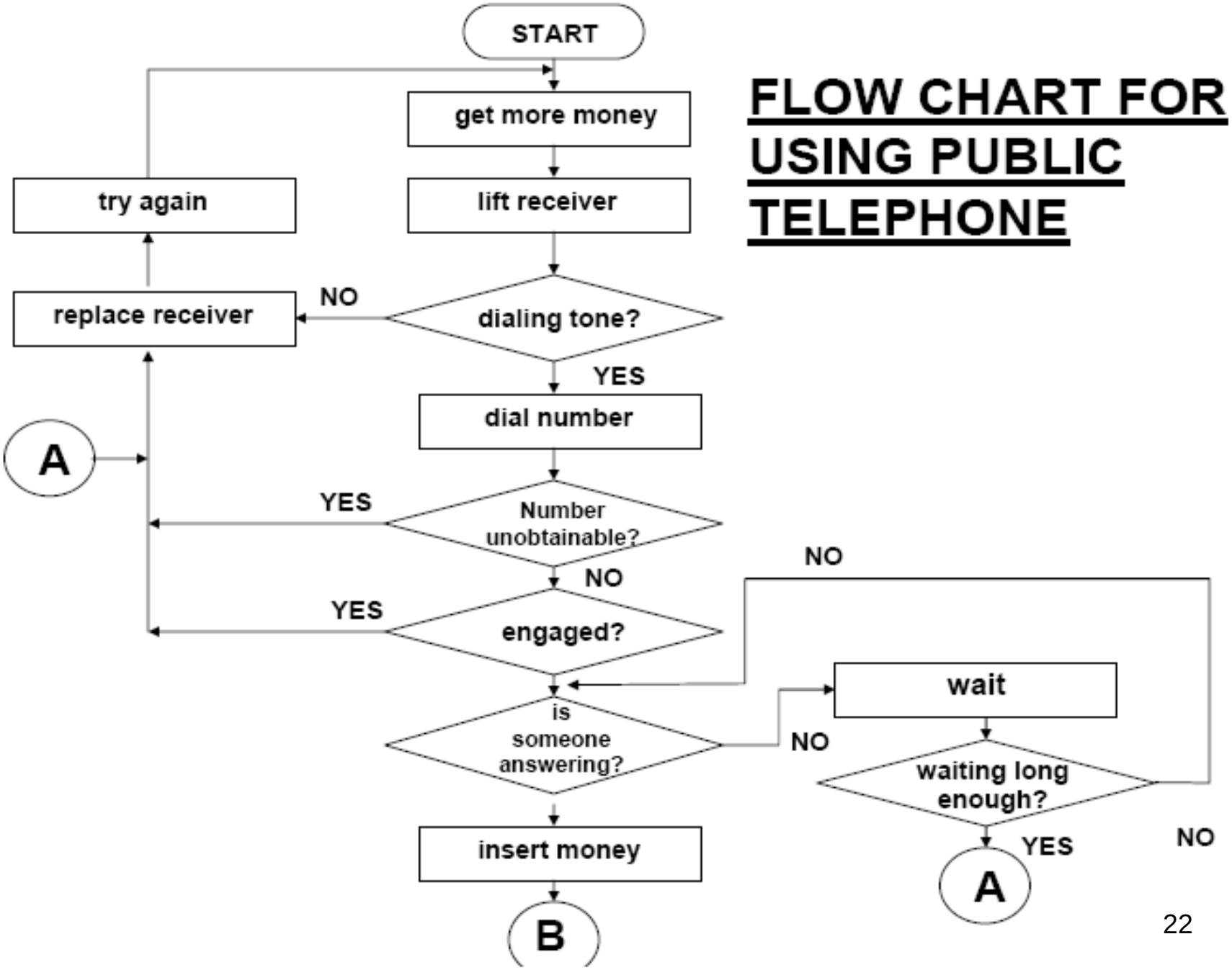
Replace receiver

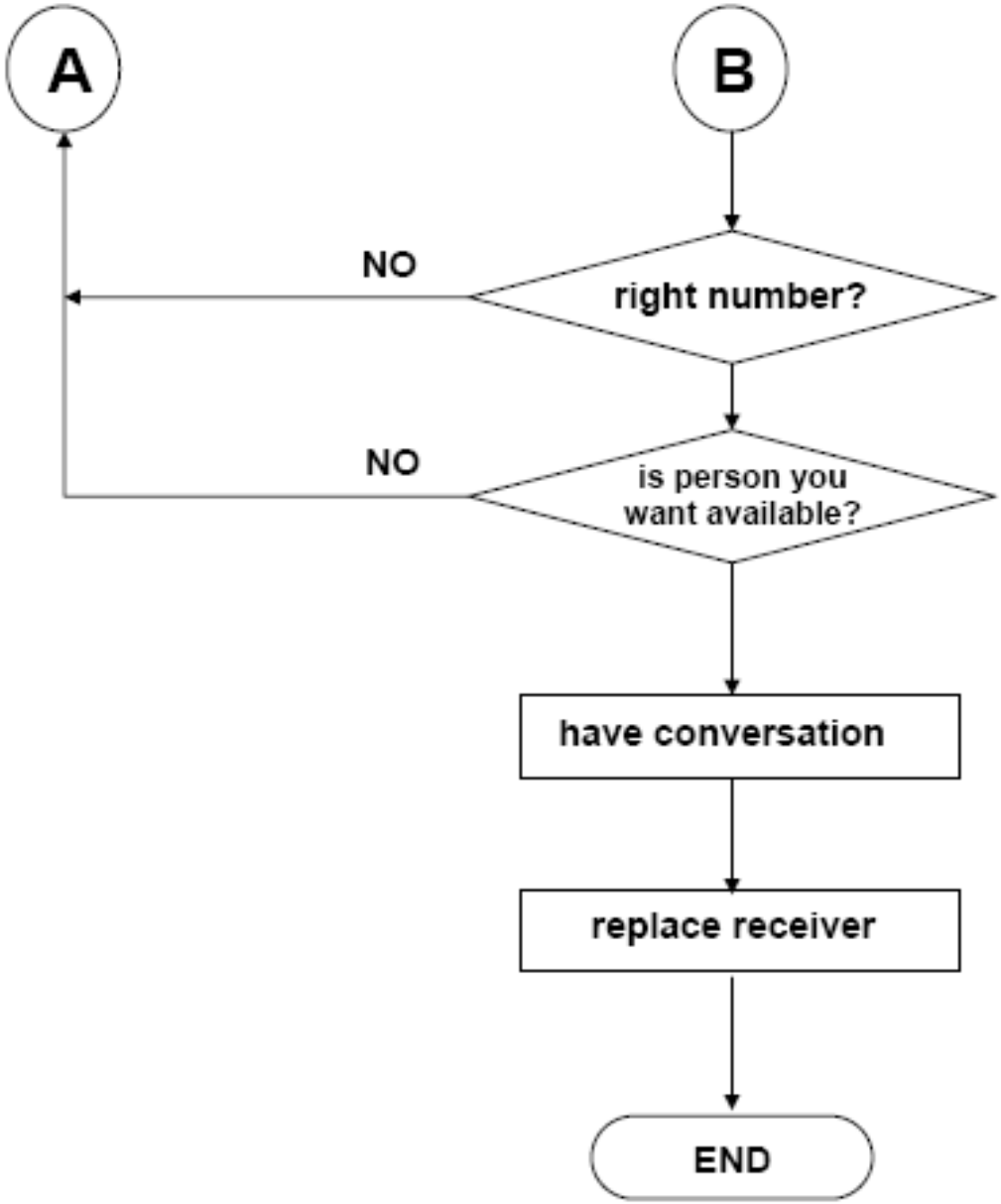
else Replace receiver

Try again later

Until conversation completed

FLOW CHART FOR USING PUBLIC TELEPHONE




A
B
NO
right number?
NO
is person you want available?
have conversation
replace receiver
END

Important Techniques used for Algorithms Design and Analysis

- **Brute Force** : Exhaustive search
- Straight Forward, Naïve/Raw Approach, Most Expensive.

  E.g prime number (check 2 to n-1)

  No technique used, no efficiency is used.
- All possible scenarios are checked.
- Final result depend on exhaustive check

Important Techniques used for Algorithms Design and Analysis

Divide and Conquer Approach

(Divide problem into smaller equal sub approach)

Decompose until solved components are reach. Then start merging solved components and final results are achieved.

Advantage: Efficient than Brute Force Approach

Disadvantage: Dependency of Components

Important Techniques used for Algorithms Design and Analysis

Iterative Approach

- One component solved and move to next component. Size start reduces.

- **Transform And Conquer**
- Modify problem first and then solve it

Important Techniques used for Algorithms Design and Analysis

- **Greedy Approach** :Used to solve optimization problem in which you want to find, not just *a* solution, but the *best* solution.
- A “greedy algorithm” sometimes works well for optimization problems
- A greedy algorithm works in phases. At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum
- Efficient
- Easy to implement
- Every problem cannot be solved by greedy approach

Important Techniques used for Algorithms Design and Analysis

Dynamic Programming Approach

- Break problems into subproblems and combine their solutions into solutions to larger problems.
- The solution to a DP problem is typically expressed as a minimum (or maximum) of possible alternate solutions

Problem solving Process

- **Problem**
- **Strategy**
- **Algorithms**
  - **Input**
  - **outputs**
  - **steps**
- **After designing :**
- **Analysis(to find computational cost: better than any existing algorithm)**
  - **Correctness**
  - **Time and Space**
  - **Optimality**
- **Implementation (to prove whether best or not : not 100% to prove)**
- **Verification**

Analysis of Algorithm

The analysis is a process of estimating the efficiency of an algorithm. There are two fundamental
parameters based on which we can analysis the algorithm:

- **Space Complexity:** The space complexity can be understood as the amount of space required by
an algorithm to run to completion.
- **Time Complexity:** Time complexity is a function of input size **n** that refers to the amount of time needed by an algorithm to run to completion.
- Let's understand it with an example.
- Suppose there is a problem to solve in Computer Science, and in general, we solve a program by
writing a program. If you want to write a program in some programming language like C then
before writing a program, it is necessary to write a blueprint in an informal language.

Analysis

- Generally, we make three types of analysis, which is as follows:
- **Worst-case time complexity:** For '**n**' input size, the worst-case time complexity can be defined as the maximum amount of time needed by an algorithm to complete its execution. Thus, it is nothing but a function defined by the maximum number of steps performed on an instance having an input size of n.
- **Average case time complexity:** For '**n**' input size, the average-case time complexity can be defined as the average amount of time needed by an algorithm to complete its execution. Thus, it is nothing but a function defined by the average number of steps performed on an instance having an input size of n.
- **Best case time complexity:** For '**n**' input size, the best-case time complexity can be defined as the minimum amount of time needed by an algorithm to complete its execution. Thus, it is nothing but a function defined by the minimum number of steps performed on an instance having an input size of n.

Model of Computation

- It is a set of assumptions to what level we going to solve problem and at what level we analyze and design
- Design assumption
- Level of abstraction which meet our requirements
- E.g. real Number : Neither more nor Less [0, 1] Infinite continues level

Model of Computation

- **Analysis**
- Is independent of the variation in
- Machine
- Operating system
- Programming languages
- Compiler etc
- i.e. low level details are ignored in analysis of algorithms

Model of Computation

- Assumption that we have Random access machine (infinite memory and sequential memory instruction access)

- Further all basic operations(+,-,*,/,% etc) or any Boolean operation a>b, assignment i.e. a=2, accessing memory element take unit time

Model of Computation

- For huge input analysis is usually to check efficiency make use of asymptotic analysis
- Here units used to measure complexity analysis are Big O , Omega, Theta etc. Asymptotic notation are used because different implementation of algorithms may differ in their efficiency from each other

What about our model of computation

- Efficient of two given algorithms are related by a
  By a constant multiplicative factor called the hidden constant

ALGORITHMICS

It is the science that lets designers study and evaluate the effect of algorithms based on various factors so that the best algorithm is selected to meet a particular task in given circumstances. It is also the science that tells how to design a new algorithm for a particular job.

Few Classical Examples

Classical Multiplication Algorithms

- **English**
- **American**
- **A la russe**
- **Divide and Conquer**

CLASSIC MULTIPLICATION ALGORITHMS

(981 x 1234)

American	English
981	981
1234	1234
3924	981
2943	1962
1962	2943
981	3924
1210554	1210554

MULTIPLICATION (981 x 1234)

(*a la russe algorithm*)

981	1234	1234
490	2468	
245	4936	4936
122	9872	
61	19744	19744
30	39488	
15	78976	78976
7	157952	157952
3	315904	315904
1	631808	631808
		1210554

Approach:

- A divide-by-2 and multiply-by-2 method where one number is halved (discarding remainders) and the other is doubled.
- When the halved number is odd, the corresponding doubled value is added to the final sum.

MULTIPLICATION (981 x 1234)

(*Divide-and-Conquer Algorithm*)

	Multiply		Shift	Result
i)	09	12	4	108….
ii)	09	34	2	306..
iii)	81	12	2	972..
iv)	81	34	0	2754
				1210554

MULTIPLICATION (9 x 12)

(Divide-and-Conquer Algorithm)

	Multiply		Shift	Result
i)	0	1	2	0 . .
ii)	0	2	1	0 .
iii)	9	1	1	9 .
iv)	9	2	0	18
				108

Comparison

Algorithm	Method	Time Complexity	Efficiency
English Multiplication	Direct digit-wise multiplication	$O(n^2)$	Slow for large numbers
American Multiplication	Same as English (long multiplication)	$O(n^2)$	Slow for large numbers
A La Russe Multiplication	Halving & doubling	$O(\log n)$	Faster for large numbers
Divide and Conquer (Karatsuba)	Recursive multiplication	$O(n^{1.58})$	Much faster than schoolbook

PARAMETERS FOR SELECTION OF AN ALGORITHM

- Priority of Task
- Type of Available Computing Equipment
- Nature of Problem
- Speed of Execution
- Storage Requirement
- Programming Effort

A good choice can save both money and time, and can successfully solve the problem

Summery

- Objective of course
- Definitions of Algorithm
- Criteria for the satisfaction of Algorithm
- Important Techniques used for Algorithms Design and Analysis
- Problem solving Phases
- Model of Computation