CIT 3106: Design and Analysis of Algorithms

Dedan Kimathi University
of Technology

Goals of the course

Equip learners with skills

Equip learners with skills to design and analyze computer algorithms.

Describe different categories

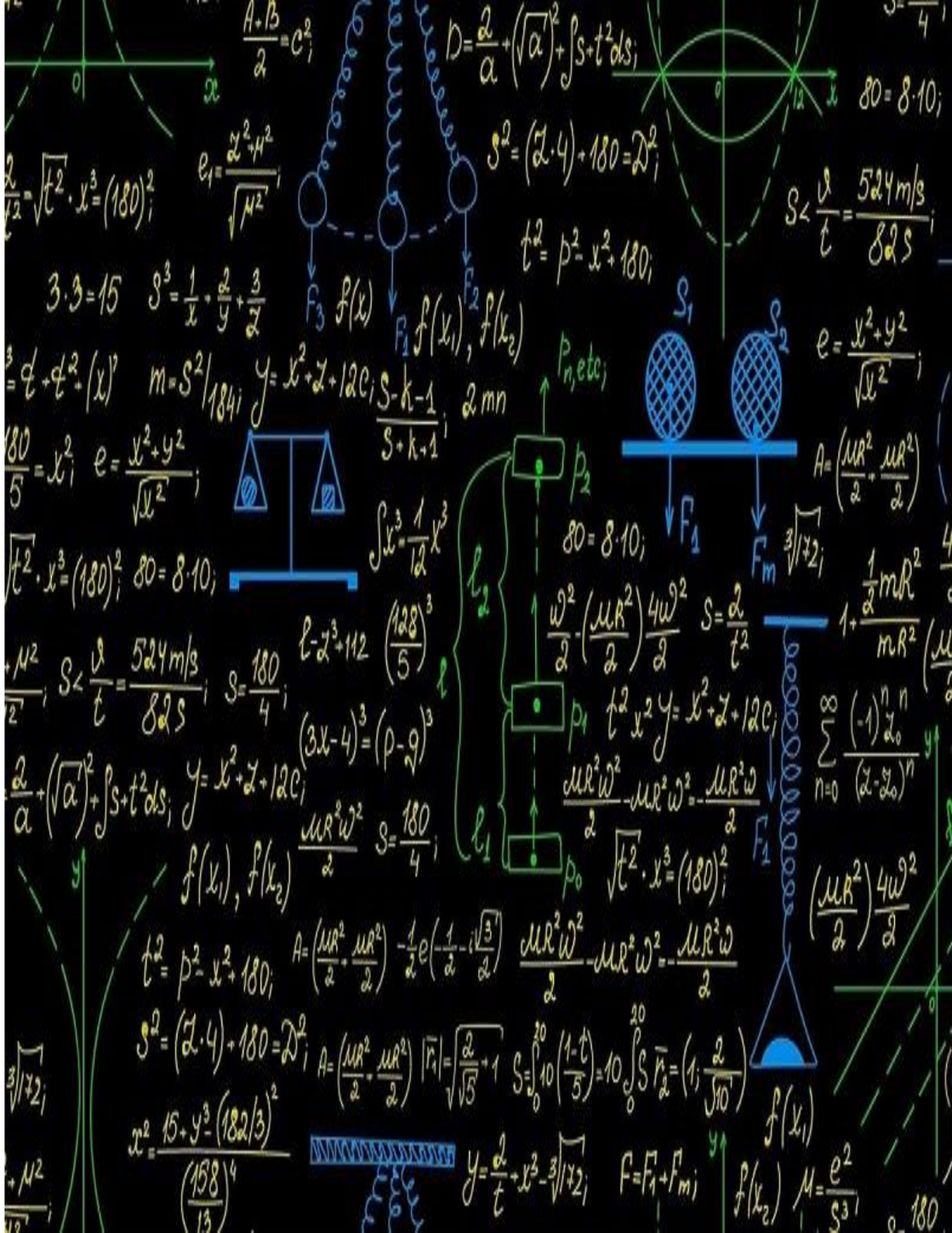
Describe different categories of algorithms.

Design an algorithm

Design an algorithm for a given problem.

Introduction

This chapter provides an overview of the fundamental concepts and principles of algorithms. It explores the importance of studying algorithms, defines key terminology, and presents examples of computational problems that can be solved through algorithmic approaches.



Why Study Algorithms?

1

Practical Reasons

We need to understand a standard set of algorithms and be able to analyze and design new ones for practical applications.

2

Theoretical Importance

Algorithms are a cornerstone of computer science. Computer programs would not exist without algorithms.

3

Analytical Skills

Studying algorithms helps develop analytical and problem-solving skills.

Definitions

This is something we want to solve or a question we want to answer. It is defined as a infinite set of possible instances of a question usually to be solved by a computer. It is a set of three parameters P=(I,O,R) where I – input set, O- A set of the possible outputs, and R- Relationship between input set and output set.

Examples

Searching a sequence of integers

Given a sequence of integers, s, and a search value x (an integer), the goal is to output the index i such that s[i] = x, if one exists, otherwise output -1.

Searching an ordered sequence of integers

Given a sequence of integers, s, in ascending order, and a search value x (an integer) known to be in the sequence, the goal is to output the index i such that s[i] = x.

Sorting integers

Given a sequence of integers, s, the goal is to create a new sequence s' with the same elements as s, but in ascending order.

What is an Algorithm

- This is a sequence of unambiguous instruction for solving a given computational problem.
- An **algorithm is a sequence of unambiguous instructions for solving a** problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
- Algorithm involves taking some inputs, performing some steps and finally producing some output. i.e Algorithm involve obtaining output from any given input that is correct in a finite number of times.

Points to note on algorithms

Procedural Solutions

We can consider algorithms to be procedural solutions to problems.

Input Range Specification

The range of inputs for which an algorithm works has to be specified carefully.

Multiple Algorithms

There may exist several algorithms for solving the same problem.

Nonambiguity Requirement

The nonambiguity requirement for each step of an algorithm cannot be compromised.

Multiple Representations

The same algorithm can be represented in several different ways.

Varying Speeds

Algorithms for the same problem can be based very different ideas and can solve the problem with dramatically different speeds.

Example of an Algorithm

Euclid's Algorithm

One way of solving the problem of finding the greatest common divisor (gcd) of two non-negative, not both zero integers m and n is Euclid's algorithm.

Consecutive Integer Checking

Another way of solving the gcd problem is the consecutive integer checking algorithm.

Middle School Procedure

The third way of solving the gcd problem is the middle-school procedure.

Euclid's algorithm

1 Step 1

2 Step 2

3 Step 3

Euclid's algorithm for computing gcd

1. If **n** = 0, return the value of **m** as the answer and stop; otherwise, proceed to Step 2.
2. Divide **m** by **n** and assign the value of the remainder to **r**.
3. Assign the value of **n** to **m** and the value of **r** to **n**. Go to Step 1.

Alternatively, we can express the same algorithm in pseudocode:

Euclid's algorithm pseudocode

The pseudocode for Euclid's algorithm to compute the greatest common divisor (gcd) of two nonnegative integers m and n is as follows:

ALGORITHM *Euclid(m, n)*

- //Computes gcd_(m, n)_ by Euclid's algorithm

- //Input: Two nonnegative, not-both-zero integers *m* and *n*

- //Output: Greatest common divisor of *m* and *n*

while *n* = 0 **do**

r ← *m* mod *n*

m ← *n*

n ← *r*

return m

Consecutive integer checking algorithm

1. The definition of the greatest common divisor of **m** and **n** as the largest integer that divides both numbers evenly.
2. Obviously, such a common divisor cannot be greater than the smaller of these numbers, which we will denote by **t** = min{**m, n**}.
3. So we can start by checking whether **t** divides both **m** and **n**: if it does, **t** is the answer; if it does not, we simply decrease **t** by 1 and try again.
4. (How do we know that the process will eventually stop?) For example, for numbers 60 and 24, the algorithm will try first 24, then 23, and so on, until it reaches 12, where it stops.

Consecutive integer checking algorithm

1

Step 1

Assign the value of min{_m, n_} to _t._

2

Step 2

Divide _m_ by _t._ If the remainder of this division is 0, go to Step 3; otherwise, go to Step 4.

3

Step 3

Divide _n_ by _t._ If the remainder of this division is 0, return the value of _t_ as the answer and stop; otherwise, proceed to Step 4.

4

Step 4

Decrease the value of _t_ by 1. Go to Step 2.

Middle-school procedure for computing gcd_(m, n)_

1

Step 1

Find the prime factors of **m**.

2

Step 2

Find the prime factors of **n**.

3

Step 3

Identify all the common factors in the two prime expansions found in Step 1 and Step 2. (If **p** is a common factor occurring **pm** and **pn** times in **m** and **n**, respectively, it should be repeated min{**pm**, **pn**} times.)

4

Step 4

Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

Example of middle-school procedure

Thus, for the numbers 60 and 24, we get:

60 = 2 * 2 * 3 * 5
24 = 2 * 2 * 2 * 3

gcd(60, 24) = 2 * 2 * 3 = 12.

Design and analysis process

The design and analysis of algorithms is a fundamental aspect of computer science. It involves the systematic approach to creating efficient and effective algorithms to solve computational problems.

Design and analysis process

- **Computational means:** there are exact algorithms and approximation algorithms. Sometimes solving the problem exactly can be slow due to its complexity.
- **Design:** An **algorithm design technique (or "strategy" or "paradigm") is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.** Choosing the right data structure. Specifying the problem can include flow charts or pseudocode.

Design and Analysis of Algorithms

- **Proving correctness:** Prove that the algorithm yields a required result for every legitimate input in a finite amount of time.
- **Analyzing algorithms:** Show efficiency. Time efficiency and space efficiency.
- **Coding:** Translate algorithm to code.

Algorithm correctness

One counter-example is enough to disapprove correctness. NB: correctness of an algorithm does not necessarily guarantee the correctness of the program that implement the algorithm.

An algorithm chosen to solve a given problem is correct if for every instance of the input (that is, the specified properties of the input are satisfied), the algorithm halts, and the outputs produced satisfy the specified input/output relation. Correctness must be proven.

Characteristics of an Algorithm

Input

An algorithm takes one or more input from an external source.

Output

For every input(s) provided there should be an output.

Finiteness

An algorithm must terminate for any number of input(s) supplied.

Definiteness

Each step of a given algorithm must be clear and unambiguous.

Effectiveness

An algorithm should solve the problem it was designed to solve. And it should be possible to demonstrate that the algorithm converges with just a paper and a pen.

Types of Problems

1 **Sorting**

Arranging data in ascending or descending order.

2 **Searching**

Finding an element in a data set.

3 **String Processing**

String matching and manipulation.

4 **Graph Problems**

Modeling and solving problems related to transportation, communication, social and economic networks, project scheduling, and games.

Types of Problems (cont.)

1

Combinatorial Problems

Finding a combinatorial object (e.g., permutation, combination, or subset) that satisfies certain constraints, such as maximum or minimum.

2

Geometric Problems

Dealing with geometric objects such as points, lines, and polygons, with applications in computer graphics, robotics, and tomography.

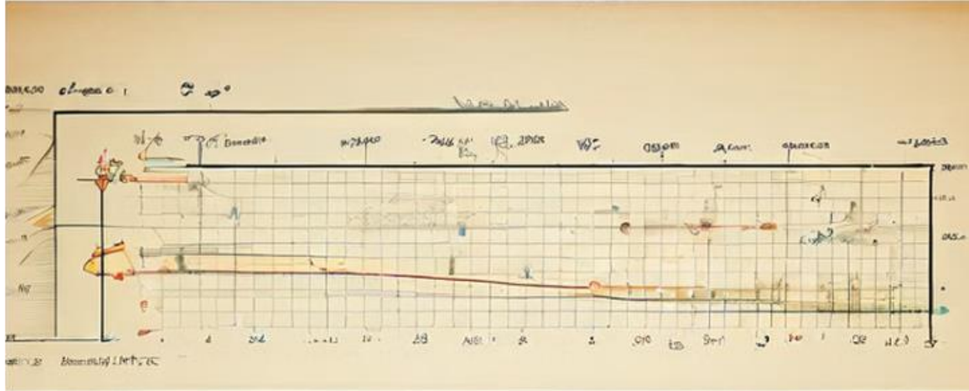
3

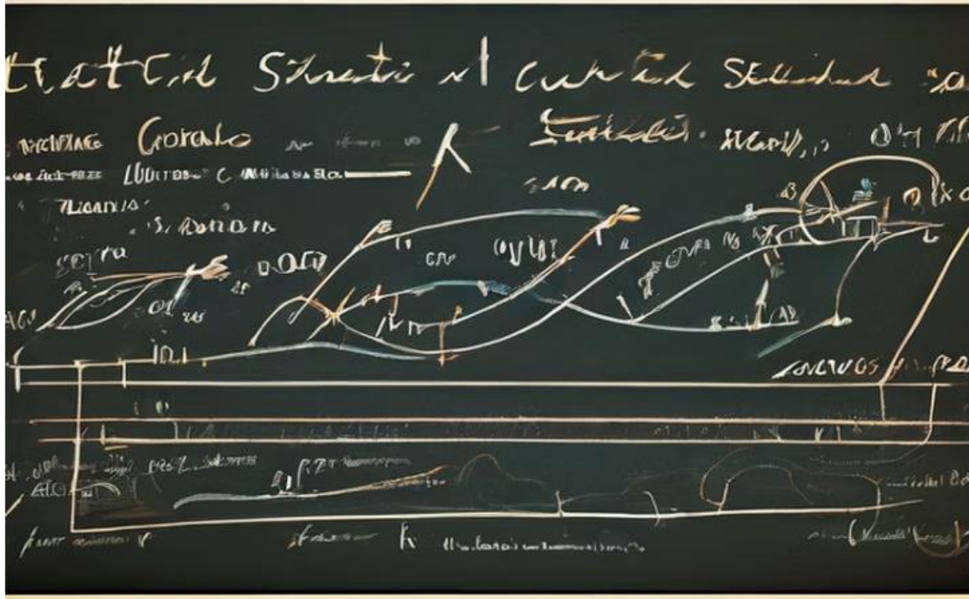
Numerical Problems

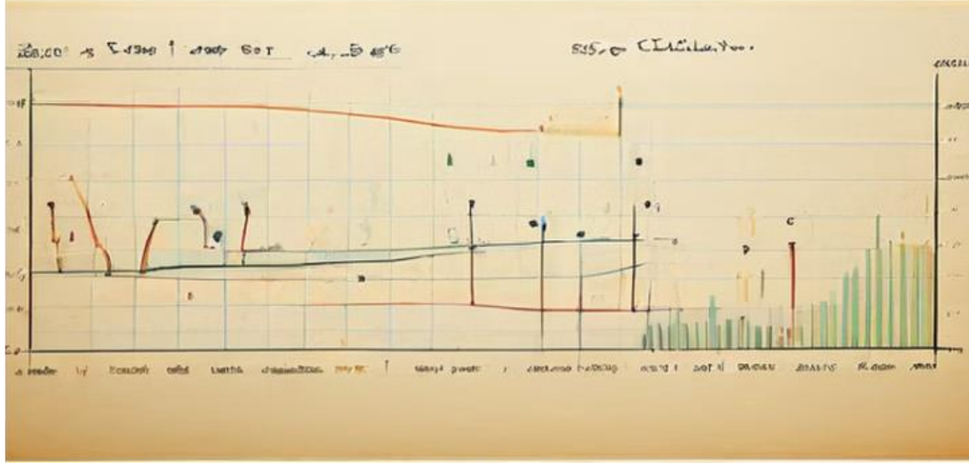
Involving mathematical objects of continuous nature, such as solving equations and systems of equations, computing definite integrals, and evaluating functions.

Numerical problems

Numerical problems are problems that involve mathematical objects of continuous nature: solving equations and systems of equations, computing definite integrals, evaluating functions, and so on.







Fundamental data structures

A data structure can be defined as...

A **data structure** can be defined as a particular scheme of organizing related data items.

Linear data structures

Linear data structures include arrays, linked lists, stacks, and queues.

Graphs

Graphs are a fundamental data structure that represent relationships between objects.

Trees

Trees are a hierarchical data structure that represent parent-child relationships.

Comparing different algorithms

Availability in library

Two algorithms can solve the same problem, but they can be different though they satisfy the criteria of being correct algorithms. Since algorithms are solved using a computer, two algorithms can be compared to determine which is better using various criteria.

Space requirements

The amount of memory space needed to complete the task can be a factor in comparing different algorithms.

Resources

The resources required by each algorithm, such as memory space and time, can be used to compare which algorithm is more efficient.

Time Requirements

The time required for each algorithm to complete the task can be used to determine which algorithm is more efficient.

Cost of human time

- The **time required to develop and maintain the program** is the major concern when designing an algorithm
- With the advanced technology, computers have enough amount of memory, due to this the space requirement in most algorithms remain insignificant

Real Life Problems

The Human Genome Project

The Human Genome Project was a collaborative research program whose goal was the complete mapping and understanding of all the genes of human beings. All our genes together are known as our "genome".

Internet Searches

Algorithms power the search engines we use every day to find information on the internet.

E-Commerce

Algorithms are used in e-commerce to recommend products, optimize pricing, and streamline logistics.

Finding Shortest Routes

Algorithms are used to calculate the shortest or most efficient routes for transportation and delivery services.

Matrix Multiplication

Algorithms are used to perform complex matrix multiplication, which has applications in fields like image processing and machine learning.

Efficiency of Algorithms

A good algorithm is determined by two factors: it must work, and it must complete its task in a finite (reasonable) amount of time.

Measuring time requirements

Write and Run a program

This is a straight forward way of testing the speed of two different algorithms, where you write and run a program and probably use a timing device. Though this method is simple there are a number of problems associated with it.

Overhead

The actual implementation of the algorithm in a program may require some type of programming overhead that may influence the apparent speed of the actual algorithm.

Measuring Time Requirements

- The **speed of the algorithm** might be affected by the **actual coding of the program** and/or the **programming language used**. The difference might simply be the result of **better programming** or the **language**.
- The **speed of the program** might be influenced by the **computer running the program** and how it **executes particular sets of statements**.
- The **actual test data** can have a **major influence** on the **efficiency of a program**. This is probably the **most significant problem** and the **most difficult to overcome** (you can't test **ALL possible sets of data**).

Mathematical Analysis

Reliable Approach

This is the second approach which is reliable when it comes to analysis of the efficiency of algorithms, here time units are used to measure the speed of a given algorithm.

Common Time Units

Different types of operations require different time units, common time units include comparisons, assignments, I/O operations, and numeric operations (additions, subtractions, multiplications, etc.).

Ignoring implementation details

Sometime it is important to give an indication of running time for an algorithm that depends only on the algorithm itself and its input. In most cases the run time can be expressed as proportional to some fixed function of size of the input, and ignore the actual values of the input. Time α f(n) or equivalently Time=k*f(n) for some k that depends on implementation.

Analysis of Algorithms

- In analysis of algorithms, function **f(n)** is used to represent the number of units of time taken by the algorithm on any input size **n**.
- Algorithms are therefore analyzed by first grouping the size of inputs.
- Algorithms can be of different order depending on the size of the input they work on.

Algorithms and Other Techniques

Like other techniques (pipelining, OO etc.), algorithms are undergoing developments. Simple applications may not require algorithms but they rely on algorithms for facilities such as GUI, network routing, compilers which are based on algorithms. Knowledge of algorithms differentiate programmers from coders.

THANK YOU

Dedan Kimathi University of Technology

Dr Jane Kuria

Private Bag- Dedan kimathi, Nyeri-Mweiga road

Telephone: ***+254-721709511***

Email: ***jane.kuria@dkut.ac.ke***

Website: ***www.dkut.ac.ke***