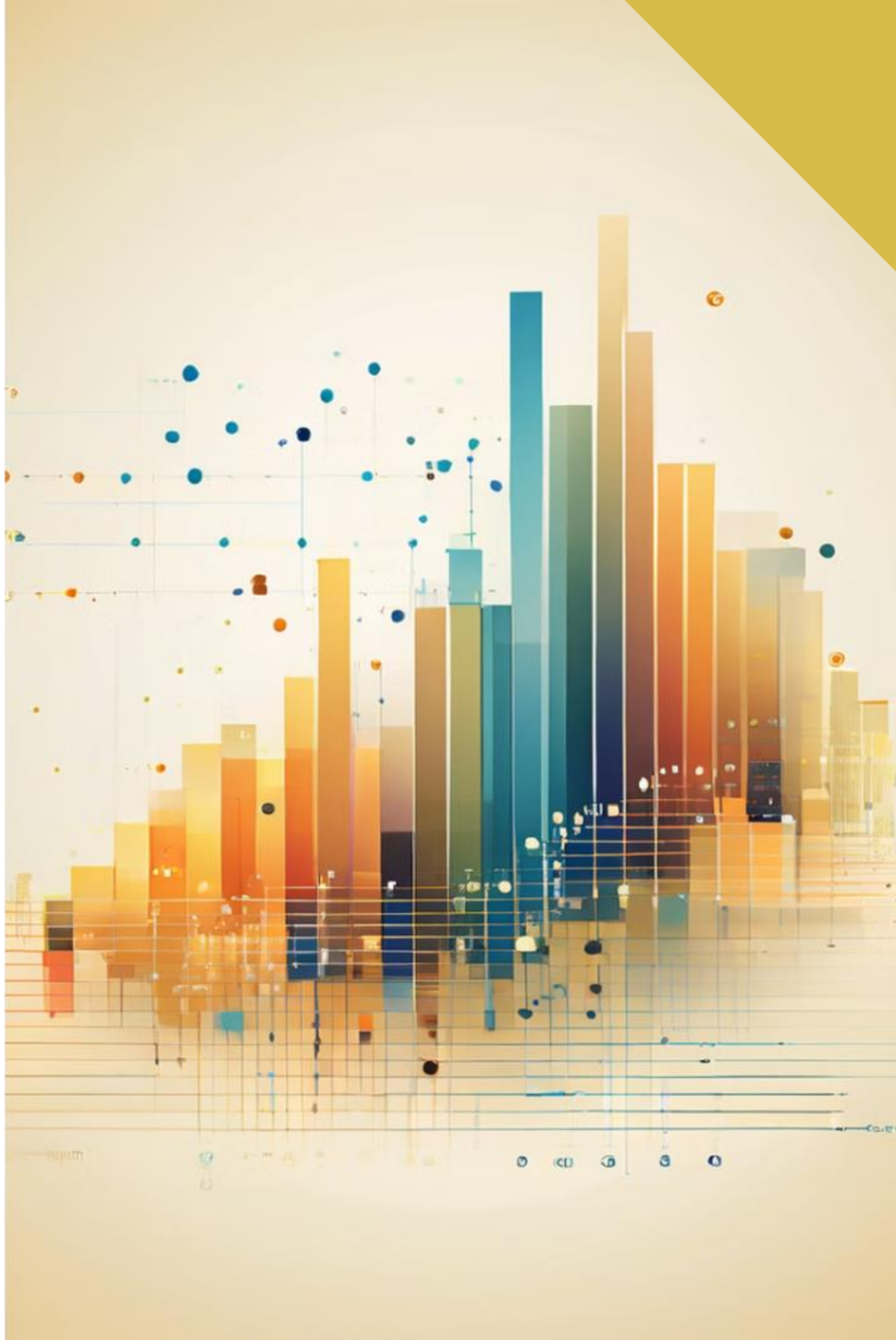
Analysis of Algorithms

Dedan Kimathi University
of Technology

Introduction

Analysis of Algorithms is a field at the boundary of computer science and mathematics. It is an investigation of an algorithm's efficiency with respect to running time (time complexity) and memory space (space complexity). The goal is to obtain a precise understanding of the asymptotic, average-case characteristics of algorithms and data structures using probabilistic, combinatorial, and analytic methods.

G



Introduction to Algorithm Analysis

1

What is Analysis?

Analysis is the separation of an intellectual or substantial whole into parts for individual study.

2

Time Complexity

Time complexity is how fast the algorithm works, measuring the running time as the input size grows.

3

Space Complexity

Space complexity is the amount of additional memory units required by an algorithm, beyond the space for inputs and outputs.

Importance of Algorithm Analysis

Scalability

Algorithms help us understand scalability - how well a solution can handle increasing amounts of data or users.

Feasibility

Performance often determines what is feasible and what is impossible for a given problem.

Program Behavior

Algorithmic mathematics provides a language for talking about and predicting program behavior.

Resource Efficiency

Performance is the currency of computing - efficient algorithms maximize the use of limited resources like time and memory.

The Sorting Problem

Input

A sequence of numbers: <a1, a2, ..., an>

Example: 8 2 4 9 3 6

1

2

Output

A permutation <a'1, a'2, ..., a'n> such that a'1 ≤ a'2 ≤ ... ≤ a'n

Example: 2 3 4 6 8 9

Insertion Sort Algorithm

1

Initialize

For j = 2 to n, set key = A[j]

2

Insert

While i > 0 and A[i] > key, shift elements right

3

Place Key

Insert key into the correct position

Running Time Analysis

1

Input Dependence

The running time depends on the input data - some inputs are easier to sort than others.

2

Input Size

Generally, we parameterize the running time by the input size n, since larger inputs take longer to process.

3

Worst Case Analysis

We often seek upper bounds (worst cases) to guarantee performance for all possible inputs.

Important Functions in Analysis

O(1)	Constant time
O(log n)	Logarithmic time
O(n)	Linear time
O(n log n)	Linearithmic time
O(n^2)	Quadratic time
O(2^n)	Exponential time

Worst, Average and Best Cases

Worst Case

Calculate the upper bound on running time for the most difficult input case. For linear search, this is when the element is not present.

Average Case

Calculate the expected running time over all possible inputs, assuming a known statistical distribution.

Best Case

Calculate the lower bound on running time for the easiest input case. For linear search, this is when the element is first.

Typical Analysis

Worst case analysis is most common, as it provides a guaranteed upper bound on performance.

Asymptotic Performance

1

Asymptotic Efficiency

Focuses on how the running time increases as the input size grows towards infinity.

2

Comparing Growth Growth Rates

An algorithm with O(n) time complexity will always beat an O(n^2) algorithm for large enough inputs.

3

Balancing Factors

Asymptotic analysis helps structure thinking, but real-world design requires balancing multiple objectives.

Insertion Sort Analysis

Worst Case

Input is reverse sorted, requiring maximum possible comparisons and shifts.

Average Case

All permutations are equally likely, so average behavior depends on the number of inversions.

Performance

Insertion sort is reasonably fast for small input sizes n, but very slow for large n due to its quadratic O(n^2) time complexity.

Merge Sort Algorithm

1

Base Case

If input size n = 1, return the single element (sorted).

2

Recursion

Recursively sort the first n/2 elements and the last n/2 elements.

3

Merge

Merge the two sorted sublists into one sorted list.

Merge Sort Analysis

1

Divide

The divide step takes O(1) time, as it simply computes the midpoint.

2

Recursive Calls

There are two recursive calls on n/2 elements each, so the work at this level is O(n).

3

Merge

The merge step takes O(n) time to merge the two sorted sublists.

Merge Sort Time Complexity

1

Recurrence Relation

T(n) = 2T(n/2) + O(n), with base case T(1) = O(1)

2

Master Theorem

Using the Master Theorem, we can solve this recurrence to get T(n) = O(n log n).

3

Comparison

Since O(n log n) grows more slowly than O(n^2), merge sort asymptotically beats insertion sort.

Analyzing Non-Recursive Functions

Identify Input Size

Decide on the parameter(s) indicating the input size.

Basic Operation

Identify the algorithm's basic, most frequently executed operation.

Set Up Sum

Express the number of basic operation executions as a summation.

Analyze Growth

Find a closed-form formula or determine the order of growth.

Finding the Maximum Element

Algorithm

```
maxval ← A[0]
for i ← 1 to n-1
  if A[i] > maxval
    maxval ← A[i]
return maxval
```

Analysis

The basic operation is the comparison A[i] > maxval.
This is executed n-1 times, once per loop iteration.
Therefore, the time complexity is O(n).

Analyzing Recursive Functions

Identify Input Size

Decide on the parameter(s) indicating the input size.

Basic Operation

Identify the algorithm's basic, most frequently executed operation.

Set Up Recurrence

Set up a recurrence relation for the basic operation count.

Solve Recurrence

Solve the recurrence or determine its order of growth.

Factorial Function Analysis

Algorithm

F(n):
if n = 0 return 1
else return F(n-1) * n

Analysis

The basic operation is multiplication.
Let M(n) be the number of multiplications.
M(n) = M(n-1) + 1, with M(0) = 0
Solving, we get M(n) = n, so the time complexity is O(n).

Empirical Analysis

1

Experiment Purpose

Understand the goal of the empirical analysis experiment.

2

Efficiency Metric

Decide what to measure (operation count, time, etc.) and the units.

3

Input Characteristics

Determine the input sample range, size, and other properties.

4

Implementation

Prepare a program implementing the algorithm(s) to be tested.

Empirical Analysis Process

1

Generate Inputs

Generate a sample of inputs according to the specified characteristics.

2

Run Algorithm

Run the algorithm(s) on the sample inputs and record observations.

3

Analyze Data

Analyze the empirical data to draw conclusions about performance.

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***