Design and Analysis of Algorithm (DAA)

Introduction

[Module 1]

Dr. Dayal Kumar Behera

School of Computer Engineering
KIIT Deemed to be University, Bhubaneswar, India

Definition

- **A step-by-step problem-solving procedure.**
- A sequence of instructions that tells how to solve a particular problem.
- A set of instructions for solving a problem, especially on a computer.
- A computable sequence of actions to get the desired result.

However, most agree that algorithm has something to do with defining generalized processes to get "output" from the "input".

Definition

Algorithm can be defined as

Sequence of well-defined computational steps that transform the input into the output of a given problem statement.

Analysis:

- **Correctness** of the algorithm
- **Efficiency**: predict the cost of an algorithm in terms of resources and performance.
    - Time Complexity
    - Space Complexity

CPU Time vs Memory Footprint

Design: design algorithms which minimize the cost

- Incremental
- Divide-and-Conquer
- Greedy
- Dynamic
- Backtracking
- Branch and Bound
- Brute force

Algorithm : Goals and Representation

Basic goals for an algorithm:

- always correct
- always terminates
- Performance (Most important for this course)

Representation of Algorithm:

1. Give a description in your own language, e.g. English, Hindi, …
2. Pseudo code
3. Graphical

General Concepts

- Algorithm strategy
  - Approach to solving a problem
  - May combine several approaches
- Algorithm structure
  - Iterative $\Rightarrow$ execute action in loop
  - Recursive $\Rightarrow$ reapply action to subproblem(s)
- Data structure
- Problem type
  - Satisfying $\Rightarrow$ find any satisfactory solution
  - Optimization $\Rightarrow$ find best solutions

Problem Size of an Algorithm

The problem size depends on the nature of the problem.

Example:

Problem	**Problem Size**
Search in an array of size n	n (array size)
Merge two arrays of size m and n	m + n
Compute n^{th} factorial	n

Basic of Algorithm Analysis

- Algorithm Complexity Theory:

  It's a branch of algorithm study that deals with analysis of algorithm in terms of usage of computational resource such as Time and Space (Performance)

  Time Complexity :

  Time complexity deals with finding out how the computational time of an algorithm changes with the change in size of the input.

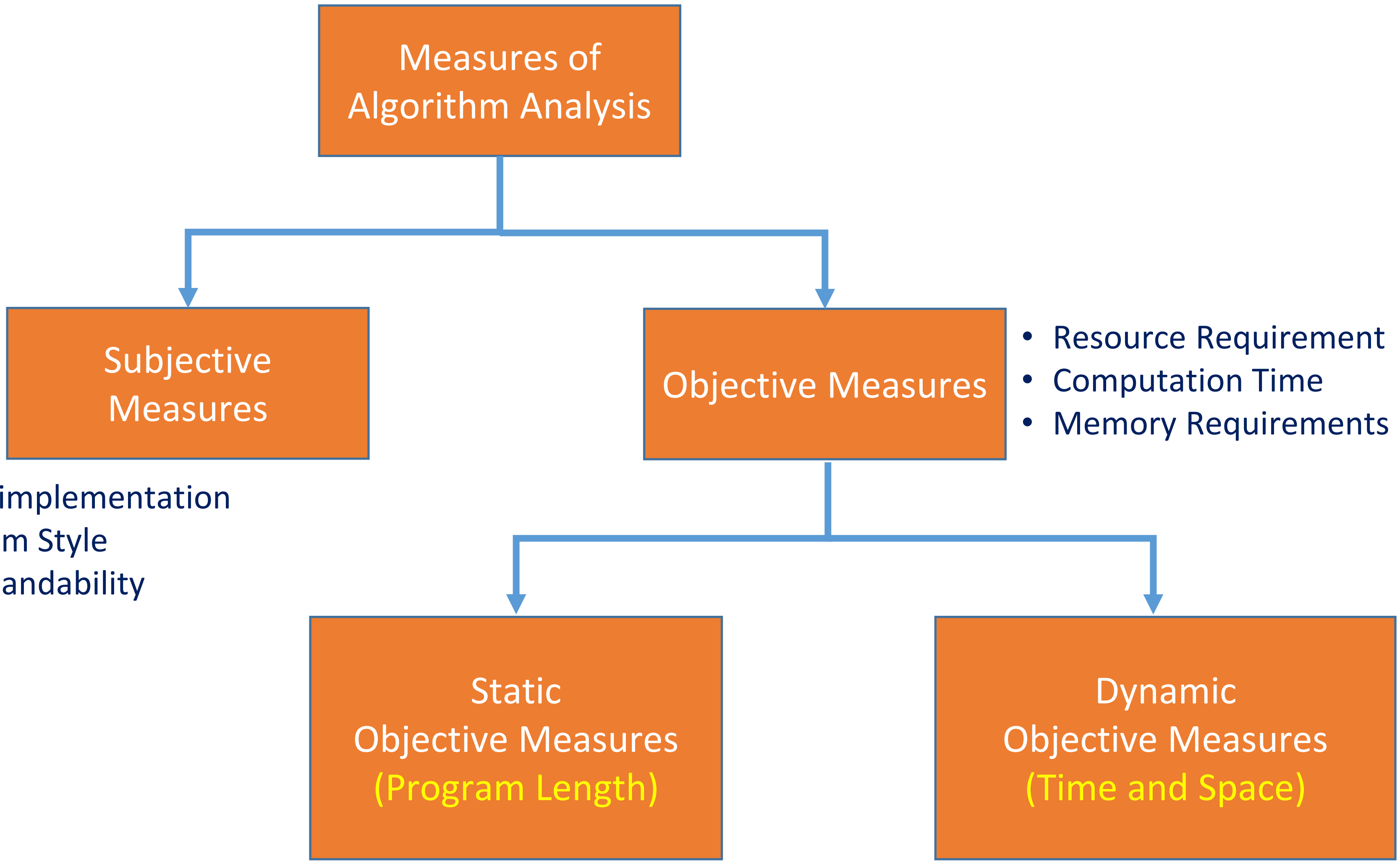
  Space Complexity:

  Space complexity deals with finding out how much (extra) space would be required by the algorithm with change in the input size.

  Not Much importance is given to space complexity:

  - Less Cost/ Can be easily increased
  - In case of Embedded System or memory constraint applications this is emphasized

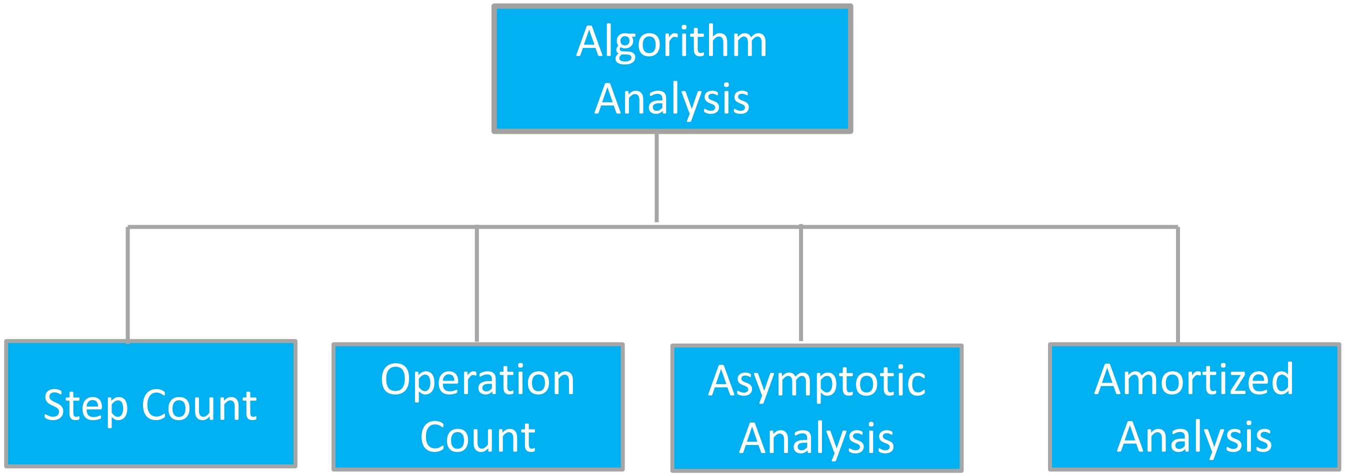
Measures of Algorithm Analysis



- Resource Requirement
- Computation Time
- Memory Requirements

- Ease of implementation
- Algorithm Style
- Understandability

Analysis of Iterative Algorithms



Step Count

Sample Algorithm

Simple(a, b, c)
Begin
a= b+1
c=a+2
d=a+b
End

Step no	Program	Step per execution	Frequency	Total
1	*Simple(a, b, c)*	0		
2	*Begin*	0		
3	*a= b+1*	1	1	1
4	*c=a+2*	1	1	1
5	*d=a+b*	1	1	1
6	*End*	0		
	Total			**3**

$T(n) = 3$

Note# Assume that each step takes one unit of time.

Step Count: Another Example

Sample Algorithm

Sum( )

Begin
sum=0.0
for i=1 to n do
sum = sum +1
end for
return sum
End

Step no	Program	Step per execution	Frequency	Total
1	**Sum( )**	0		
2	Begin	0		
3	sum=0.0	1	1	1
4	for i=1 to n do	1	n+1	n+1
5	sum = sum +1	1	n	n
6	end for	0	0	0
7	return sum	1	1	1
8	End	0	0	0
	Total			**2n+3**

Note# Assume that each step takes one unit of time.

$$T(n) = 2n+3$$

Operation Count

Step no	Program	Operations	Cost of Operation per Execution	Frequency	Total Cost
1	**Sum( )**		0		
2	Begin		0		
3	sum=0.0	Assignment	c1	1	c1
4	for i=1 to n do	Initialization, compare, update	c2	n+1	(n+1)c2
5	sum = sum +1	Perform sum	c3	n	nc3
6	end for		0	0	0
7	return sum		c4	1	C4
8	End		0	0	0
	Total			**c1 + (n+1)c2 + nc3+c4**	

Comparison Operation Count: T(n) = c2(n+1)

Total Steps: T(n) =(c2+c3)n+c1+c2+c4

Analysis as dominant operation

```
for  i ← 1 to n do
     for j ← 1 to n do
          Stmt1
     end
end
for i  ← 1 to n do
      Stmt2
end
Stmt3
```

$T(n) = c1\, n^2 \underline{+ c2\, n + c3}$

Can be ignored for larger value of n

Time Complexity Analysis

- Mathematical Analysis (a priori analysis/ Theoretical Analysis)
  - This is done before the algorithm is translated into a program
  - Estimates complexity in terms of step count or operation count
  - Independent of particular Machine, Programming Language, OS or compiler.

- Empirical Analysis (Posteriori analysis)
  - This is done after the algorithm is translated into a program
  - Program is analyzed with real-time datasets
  - Advantage is finding actual speed of the program in the field.

Note# Priori Analysis is beneficial over Posteriori Analysis.

Priori Analysis

- The principle of Priori Analysis was introduced by "*Donald Knuth*".

- In this analysis, the basic operations (or instructions) in the algorithms need to be identified and counted. This count is used as a figure of merit in doing analysis.

- There are two types of analysis based on **step/operation count**:

    - **Micro Analysis:** Perform the instruction count for all operations
    - **Macro Analysis:** Perform the instruction count for only for dominant operations.

- While analyzing the algorithm based on Priori Analysis Principles, three different cases are identified.
    - **Worst-case**
    - **Average-case**
    - **Best-case**

Worst, Best and Average Case

Worst-Case Complexity of an algorithm:

- Maximum number of computational steps required for the execution of an algorithm over all possible inputs of same size
- Provides an upper bound on the complexity of an algorithm
- The complexity that is observed for the most difficult input instances
- The most Pessimistic view

Worst, Best and Average Case

Best-Case Complexity of an algorithm:

- Minimum number of computational steps required for the execution of an algorithm over all possible inputs of same size
- Provides a lower bound on the complexity of an algorithm
- The complexity that is observed when one of the easiest possible input instances is picked as input
- The most Optimistic view

Worst, Best and Average Case

Average Case Complexity of an algorithm:

- The average amount of resources the algorithm consumes assuming some plausible frequency of occurrences of each input instance
- Possibly the most meaningful one among the complexity measures
- However, Figuring out the average cost is much more difficult than figuring out the worst case or the best-case costs!

- We have to assume a probability distribution for the types of input instances to the algorithm

- **Difficulty:** What is the distribution of real-world instances!!!

Worst-Case Time Complexity

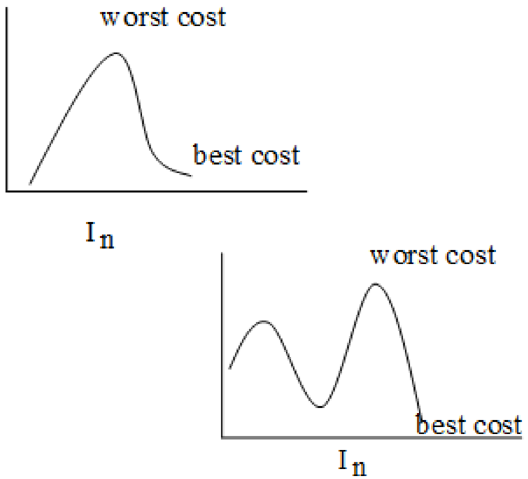
Worst-case time complexity of an algorithm is the function W(n) such that W(n) equals to the maximum value of T(I).

$$W(n) = Max\,\{\,T\,(\,I\,)\,|\;I\;\in S_n\,\}$$

$T\,(\,I\,)$: number of basic operations performed for input instance I .

S_n : Set of all inputs

n : input size



I_n – all possible instances of size n

Best-Case Time Complexity

Best case time complexity of an algorithm is the function B(n) such that B(n) equals to the minimum value of T(I).

$$B(n) = Min\,\{\,T\,(\,I\,)\mid\,I\,\in S_n\,\}$$

$T\,(\,I\,)$: number of basic operations performed for input instance I .

S_n : Set of all inputs

Average-Case Time Complexity

Average case time complexity of an algorithm is the function A(n) such that

$$A(n) = \sum_{I \in S_n} T(I)\, P(I) = E(T)$$

$P(I)$: probability of basic operation for input instance I .

S_n : Finite input sets

E : Expected value/mean

NOTE

To find the expected value, $E(X)$, or mean μ of a discrete random variable X, simply multiply each value of the random variable by its probability and add the products. The formula is given as $E(X) = \mu = \sum xP(x)$.

Here x represents values of the random variable X, $P(x)$ represents the corresponding probability, and symbol $\sum$ represents the sum of all products $xP(x)$. Here we use symbol μ for the mean because it is a parameter. It represents the mean of a population.

Example: Linear Search

Complexity of an algorithm is the function which gives the running time and/or space in terms of input size.

Complexity not only depends upon size of input but also distribution of input data.

Input Scenario	Remarks	Complexity
Worst-case	Key value is present as the last value in the array or it is not present.	W(n) = O(n)
Best-case	Key value is found at the beginning	B(n) = O(1)
Average-case	Kay value is likely to occur at any position	A(n) = O(n)

For linear search the probability of presence of item in an array is $P_i = \frac{1}{n}$

So $A(n) = 1.\frac{1}{n} + 2.\frac{1}{n} + 3.\frac{1}{n} + \ldots + n.\frac{1}{n}$

$= \frac{1}{n}(1+2+3\ldots+n) = \frac{1}{n}\left(\frac{n(n+1)}{2}\right) = \frac{n+1}{2}$


Each of your
actions will
have an
impact on your
future.


Once you know
who is walking
with you on your path.
you will never
be afraid.

Thank you