SNS College of Engineering
Coimbatore - 641107

Limitations of Algorithmic Power

Limitations of Algorithmic Power

- Introduction
- Lower Bounds
- P, NP, NP-complete and NP-hard Problems

Introduction

Algorithm efficiency:

- **Logarithmic**
- **Linear**
- **Polynomial with a lower bound**
- **Exponential**

Some problems cannot be solved by any algorithm

Question: how to compare algorithms and their efficiency

Lower Bounds

Lower bound: **an estimate on a minimum amount of work needed to solve a given problem**

Lower bound can be

an exact count

an efficiency class (Ω)

Tight lower bound: there exists an algorithm with the same efficiency as the lower bound

Example

Problem	Lower bound	Tightness
sorting	$\Omega(n\log n)$	yes
searching in a sorted array	$\Omega(\log n)$	yes
element uniqueness	$\Omega(n\log n)$	yes
n-digit integer multiplication	$\Omega(n)$	unknown
multiplication of n-by-n matrices	$\Omega(n^2)$	unknown

Methods for Establishing Lower Bounds

- ***trivial lower bounds***
- ***information-theoretic arguments (decision trees)***
- ***adversary arguments***
- ***problem reduction***

Trivial Lower Bounds

- **Based on counting the number of items that must be processed in input and generated as output**

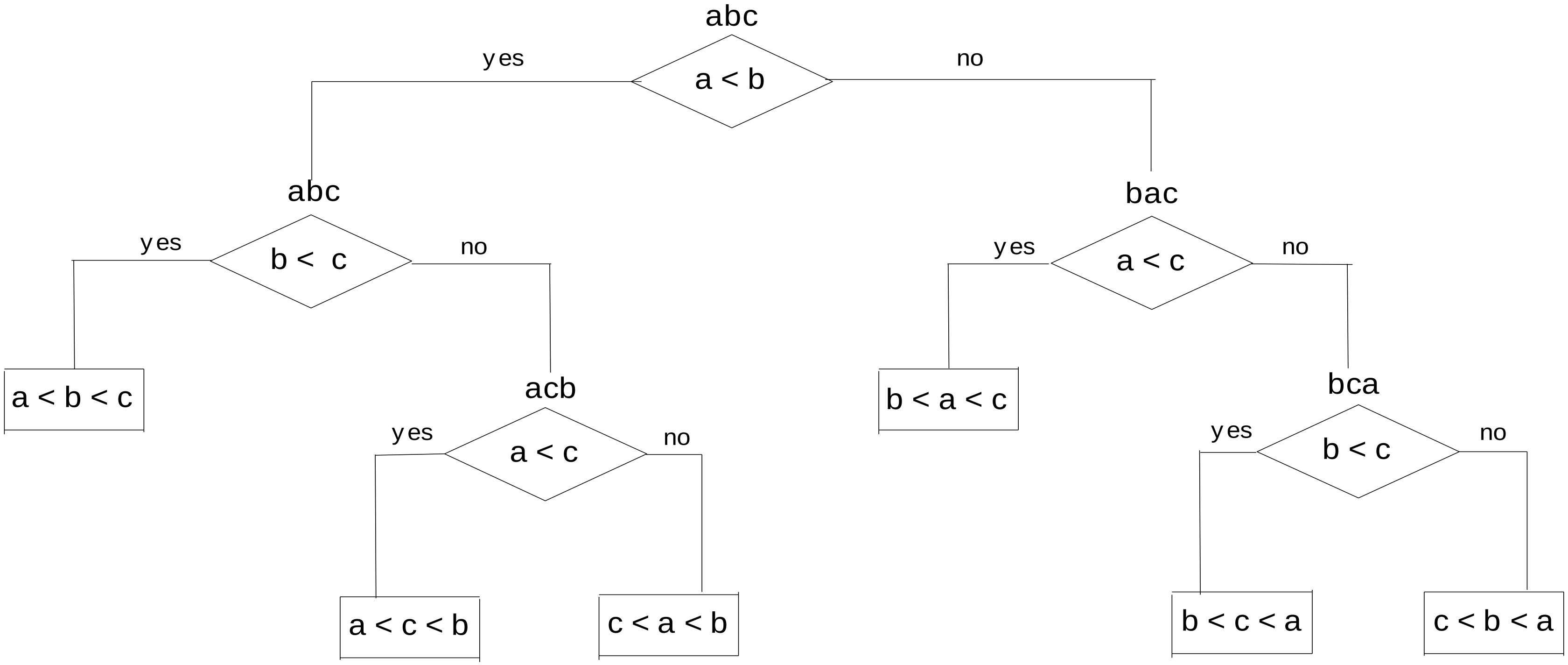
- **Examples**
  - ❑ **finding max element**
  - ❑ **sorting**
  - ❑ **element uniqueness**

Decision Trees

A convenient model of algorithms involving comparisons in which:

- **internal nodes represent comparisons**
- **leaves represent outcomes**

Decision tree for 3-element insertion sort



Decision Trees and Sorting Algorithms

- **Any comparison-based sorting algorithm can be represented by a decision tree**
- **Number of leaves (outcomes) $\geq n!$**
- **Height of binary tree with $n!$ leaves $\geq \lceil \log_2 n! \rceil$**
- **Minimum number of comparisons in the worst case $\geq \lceil \log_2 n! \rceil$ for any comparison-based sorting algorithm**
- **$\lceil \log_2 n! \rceil \approx n \log_2 n$**
- **This lower bound is tight (mergesort)**

Adversary Arguments

Adversary argument: **a method of proving a lower bound by playing role of adversary that makes algorithm work the hardest by adjusting input**

Example: **"Guessing" a number between 1 and *n* with yes/no questions**

Adversary: **Puts the number in a larger of the two subsets generated by last question**

Lower Bounds by Problem Reduction

Idea: If problem *P* is at least as hard as problem *Q*, then a lower bound for *Q* is also a lower bound for *P.*

Hence, find problem *Q* with a known lower bound that can be reduced to problem *P* in question. Then any algorithm that solves P will also solve Q.

Example of Reduction

Problem Q: Given a sequence of boolean values, does at least one of them have the value "true"?

Problem P: Given a sequence of integers, is the maximum of integers positive?

$f(x_1, x_2, \ldots x_n) = y_1, y_2, \ldots y_n$

where $y_i = 0$ if x_i = false, $y_i = 1$ if x_i = true

P, NP, NP-complete, and NP-hard Problems

- **Decision and Optimization problems**
- **Decidable, semi-decidable and undecidable problems**
- **Class P, NP, NP-complete and NP-hard problems**

Class P

- **P**: the class of decision problems that are solvable in **O(*p*(*n*))** time, where ***p*(*n*)** is a polynomial of problem's input size ***n*.** Problems in this class are called **tractable**
- **Examples:**
  - **searching**
  - **graph connectivity**

Class NP

- **NP** (nondeterministic polynomial): class of decision problems whose **proposed solutions can be verified in polynomial time** = solvable by a ***nondeterministic polynomial algorithm***.
- Problems in this class are called **intractable**

Problems in NP

- **Hamiltonian circuit existence**
- **Partition problem:** Is it possible to partition a set of *n* integers into two disjoint subsets with the same sum?
- **Decision versions of TSP, knapsack problem, graph coloring,** and many other combinatorial optimization problems. (Few exceptions include: MST, shortest paths)

P and NP

- All the problems in ***P*** can also be solved in this manner (but no guessing is necessary), so we have:

$$\boldsymbol{P \subseteq NP}$$

- ***P=NP ?***

P = NP ?

- *P* = *NP* would imply that **every problem** in *NP,* including all *NP*-complete problems, could be solved in **polynomial time**

- If a polynomial-time algorithm for just one *NP*-complete problem is discovered, then every problem in *NP* can be solved in polynomial time, i.e., *P* = *NP*
- **Most but not all researchers believe that $P \neq NP$, i.e. *P* is a proper subset of *NP***

NP-Hard Problems

NP-hard problems are NP-complete but not necessarily in NP.

Examples – the optimization versions of the NP problems