ISSUE

- We can compute upper-bounds on the costs of a specific algorithmic solution to a problem.
- To design better algorithms, it is also important to compute lower-bounds on the cost of solving a problem algorithmically, i.e. the minimum cost of any algorithmic solution to a particular problem.
- In other words, in addition to computing the **O() cost of an algorithm**, it is useful to be able to calculate the **Ω() cost of a problem**
- If a problem is known to have a Ω(f) lower bound cost and it has a known algorithmic solution which is O(f), then the bound f is said to be **tight**.

Problem	Lower bound	Tightness
Sorting an array of n elements	$\Omega(n \log n)$	Yes
Sorting an array of n elements	$\Omega(n)$	
Searching in a sorted array of n elements	$\Omega(\log n)$	
Element uniqueness of n elements	$\Omega(n \log n)$	
n-digit integer multiplication	$\Omega(n)$	
Addition of two $n \times n$ matrices	$\Omega(n^2)$	
Multiplication of two $n \times n$ matrices	$\Omega(n^2)$	

TRIVIAL LOWER BOUNDS

Approach

- For any problem which must calculate m outputs out of n inputs, any algorithmic solution will at least "read" the n inputs and "write" the m outputs.
- The minimum cost of an algorithmic solution to a problem with n inputs and m outputs is max(n,m).
- Be careful in deciding how many elements must be processed – e.g. searching for an element in a sorted array

Examples

- Finding maximum of n elements: n inputs, 1 output
- Sorting an array of n elements: n inputs, n outputs
- Evaluating the polynomial $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x^1 + a_0 x^0$
- Any problem that generates an n×m matrix
- Generating all the permutations of n elements
- Generating all the subsets of a set of n elements

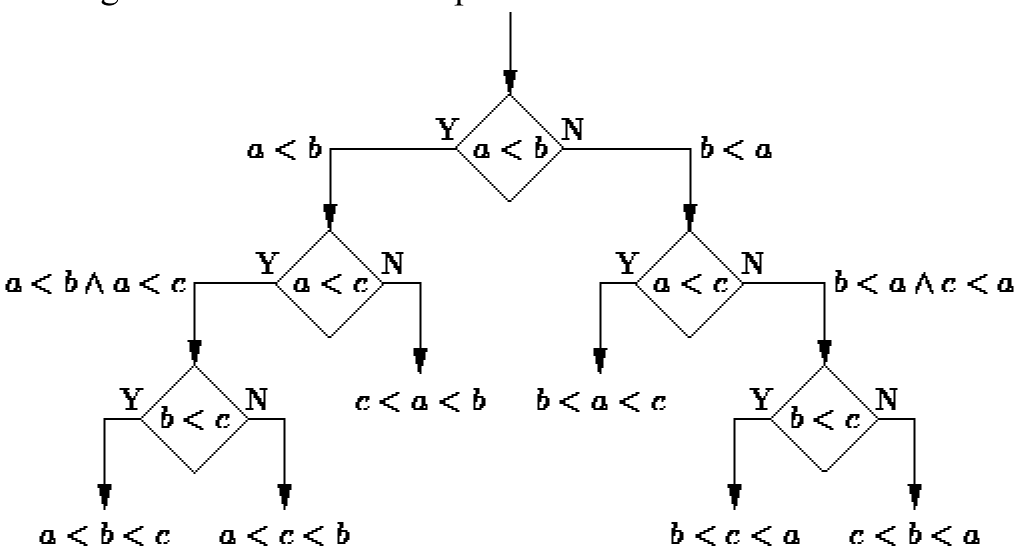
DECISION TREES

Approach

- Some algorithms compare the inputs against each other
- Build a **decision tree**: tree which shows all the possible comparisons and their outcomes.
  - Internal nodes represent comparisons.
  - Leaves represent outcomes.
- Number of comparisons in worst case = depth of tree = longest length between root and any of its leaves.

Example - Sorting

- Sorting: decision tree for comparison based sort of 3 distinct elements



  - Any comparison-based sorting algorithm can be represented by a decision tree
  - Number of leaves (possible outcomes) $\geq$ n! (# of permutations)
  - 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 (merge sort)

ADVERSARY ARGUMENTS

Definition

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

Approach

- Algorithm A is a correct algorithm to solve a problem. It has an adversary D
- A asks D a set of questions, and D is allowed to answer in such a way to make A ask as many questions are possible.
- The number of questions represents the worst case performance of the algorithm

Example - Finding number in a set

- Problem: "Guess " a number between 1 and n with yes/no comparison questions
- Adversary D: Puts the number in a larger of the two subsets generated by last question
- D can force A to ask $\lceil \log_2 n \rceil$ questions

  $\rightarrow$ Searching through a sorted list costs $\Omega(\log_2 n)$

Example - Merging two sorted lists of size n

- Problem: Merging two sorted lists of size n

  $a_1 < a_2 < \ldots < a_n$ and $b_1 < b_2 < \ldots < b_n$
- Adversary: $a_i < b_j$ iff $i < j$
- Output $b_1 < a_1 < b_2 < a_2 < \ldots < b_n < a_n$

  requires 2n-1 comparisons of adjacent elements

PROBLEM REDUCTION

<u>Approach</u>

- Need a lower bound for problem P
- You know that problem P is at least as hard as problem Q
  (i.e. P might be harder/more expensive than Q)
- You know that problem Q has a lower bound $\Omega(f)$
  (i.e. Q is at least as expensive as C.f for some constant C)
- You can conclude that $\Omega(f)$ is also a lower bound for P

<u>Example - Euclidian Minimum Spanning Tree</u>

- Problem P: Find Minimum Spanning Tree for n points in Cartesian plane.
  Note that this is not the same problem as finding the MST of a graph.
  We need a lower bound for this problem.
- Problem Q:
  - Element uniqueness problem: are there duplicates in a set of n values?
  - It is known that cost(Q) $\in \Omega$ (n logn)
- Want to show that P is at least as hard as Q, i.e. Cost(P) $\geq$ Cost(Q)
  to conclude that cost(P) $\in \Omega$ (n logn)
- Reduce Q to P, i.e. Solve Q with P
  - Transform each element x of Q into a point (x,0) $Cost_1 \in \theta(n)$
  - Find MST for these points $Cost_2 = Cost(P)$
  - Traverse MST to look for a zero-length edge $Cost_3 \in \theta(n)$
    because MST has n-1 edges
- Reasoning by contradiction:
  - If Cost(P) can be asymptotically lower that Ω (n logn)
    i.e Cost(P) $\in$ o(n logn)
  - Then Q can be solved at that lower cost + $\theta(n)$
  - But any cost which is $\theta(n)$ is also o(n logn)
  - i.e Q can be solved using two components which are both o(n log n)
    i.e. Q $\in$ o(n logn)
    this contradicts the fact that cost(Q) $\in \Omega$ (n logn)