Algorithm Lower Bounds

Fatemeh Panahi
Department of Computer Science
University of Toronto
CSC263-Fall 2017
Lecture 12

Today

- Lower bounds

  Techniques:

  - ✓ Information theory lower bounds
  - ✓ Adversary arguments
  - ✓ Reductions

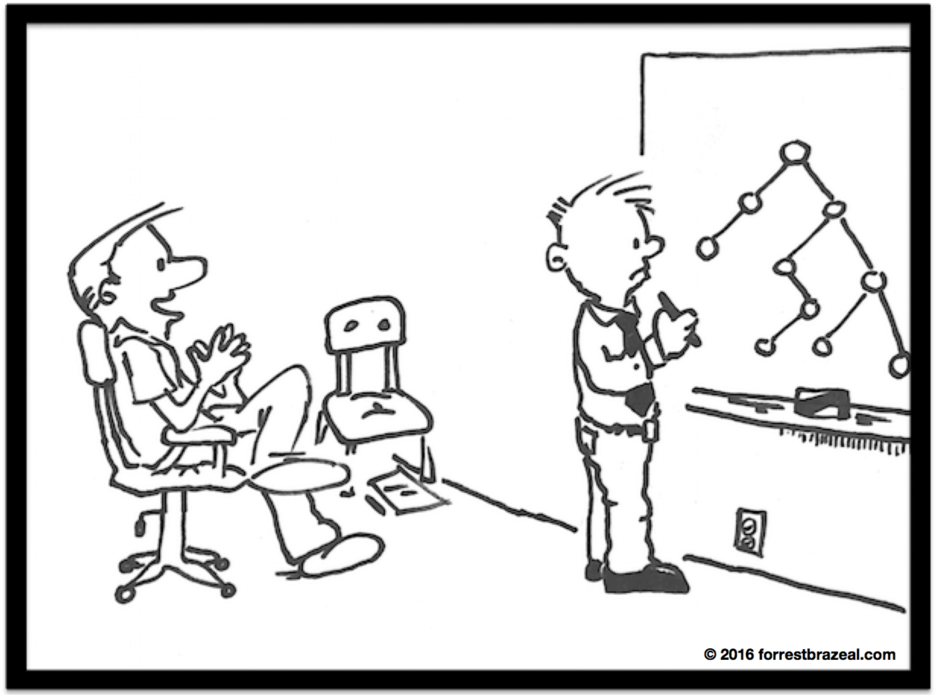
- Review of the course

Reading Assignments

Sections 8.1, 9.1. 

Can we do better? Why not?

- Lower bounds prove that we cannot hope for a better algorithm, no matter how smart we are.
- Only very few lower bound proofs are known.
- Most notorious open problems in theoretical computer science are related to proving lower bounds for very important problems.



Lower bound:

- For a problem P, the worst case complexity of P is

$$T(P) = \min\{\, T_A \mid A \text{ is any algorithm that solves } P\}$$

Where T_A is the running time of using algorithm A.

Techniques:

- ✓ Adversary arguments
- ✓ Information theory lower bounds
- ✓ Reductions

1- Adversary argument

How they play (argue)?

An smart algorithm is playing an information game against an adversary.

- The algorithm wants to get as much information as possible in order to get as much work done as effective as possible.
- The adversary wants to give as least information and modifies the input as possible to give the algorithm the worst case.
- The rule of the game is consistency.
  The adversary can trick but cannot cause inconsistency in the given information.

smart algorithm 

 Adversary

Adversary argument to prove a lower bound:

To prove a lower bound $L(n)$ for the time complexity of problem P

- show that for each algorithm A that solves P and for each input size n, an adversary can choose an input of size n on which algorithm A must take at least $L(n)$ steps.

- **Example:** Guessing a number between 1 and n using yes/no questions.
  - What is the minimum number of questions you need to guess the number in the worst case?

Adversary argument: Compute sum

Smart algorithm : computes the sum of n numeric values without looking at all the n inputs.

$S = \{a_1, a_2, \dots, a_n\}$

Adversary goes and modifies the input not looked at, then run the algorithm again.

Then this is not the correct answer.

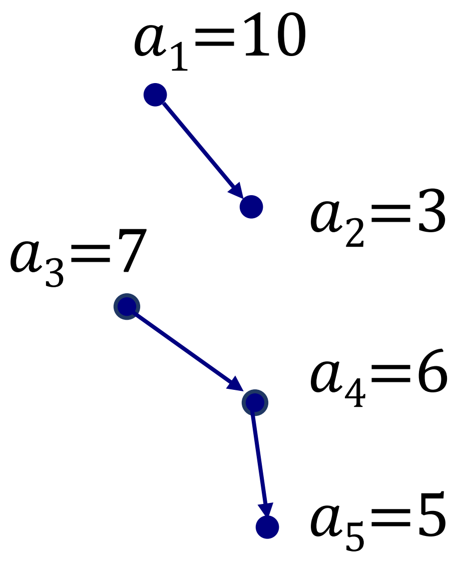
Lower bound: Finding the minimum

Theorem: to find a minimum of n element we need to ask at least $n-1$ questions.
Adversary Argument needs $n-1$ questions

Is $a_1 < a_2$? No.

Is $a_3 < a_4$? No.

Is $a_4 > a_5$? Yes.

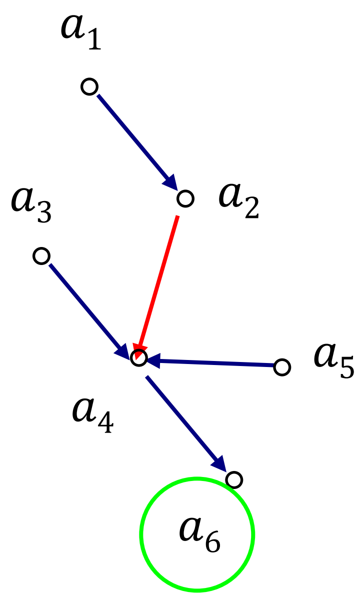


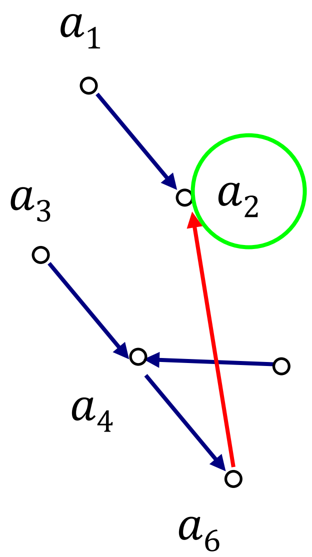
Minimum is a_5!

Wrong! It is a_2!

Lower bound: Finding the minimum

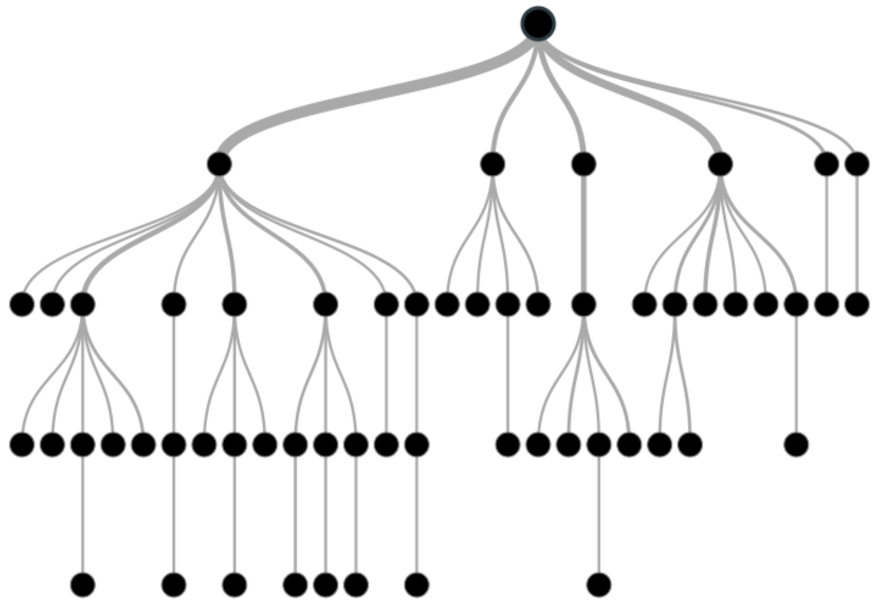
If less than $n - 1$ questions, the graph of comparisons is disconnected





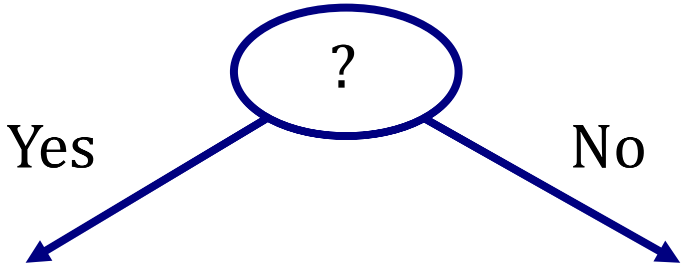
The adversary can re-arrange the data so that the answer is different

2- Information Theory bound



Information theory: Decision Trees

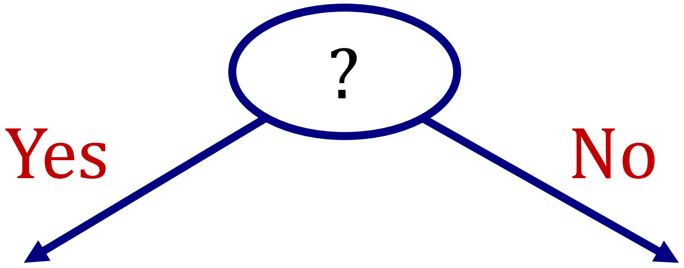
Binary Decision Trees

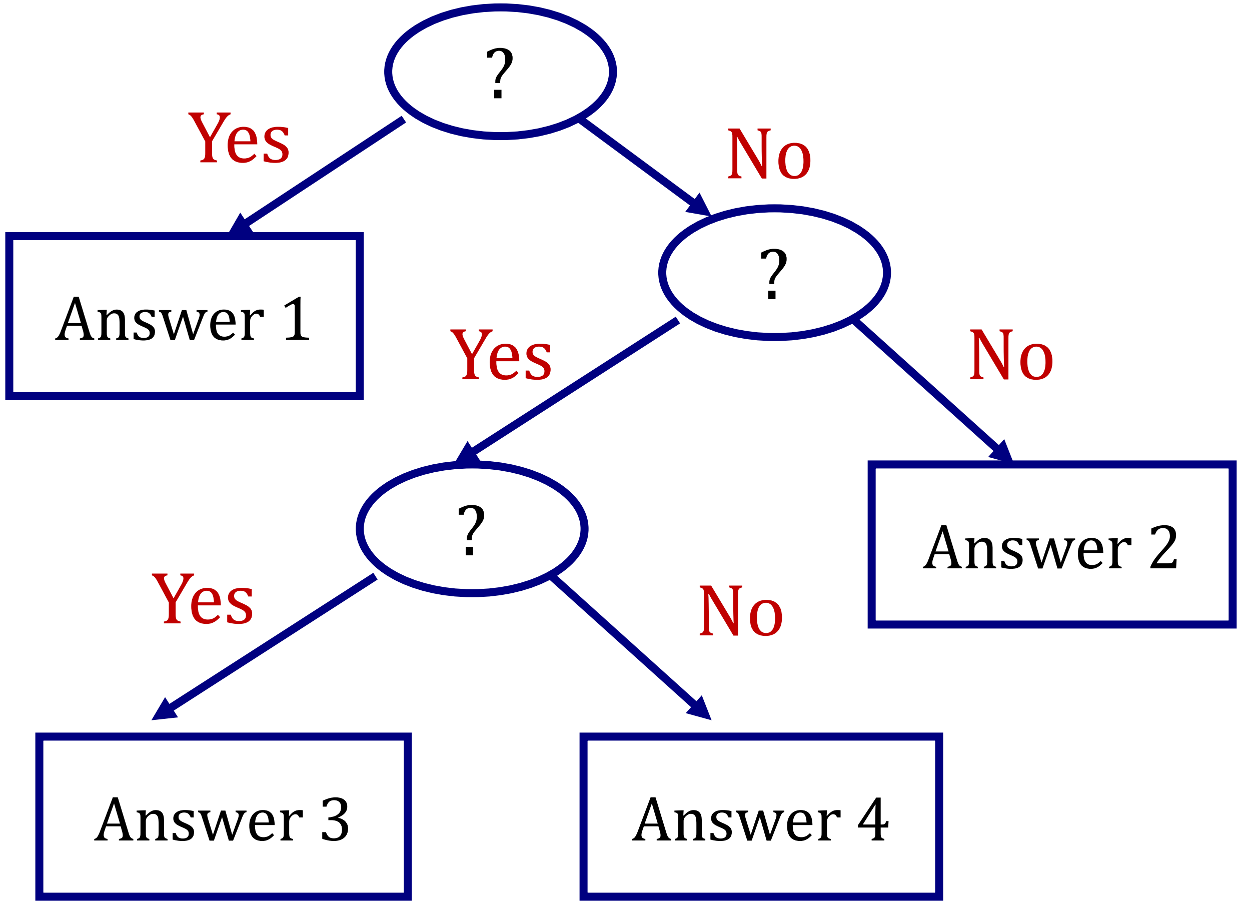


Model algorithms based on successive answers to yes/no questions

Binary Decision Trees

Model algorithms based on successive answers to yes/no questions





Information Theory Lower Bound

- A t-ary tree of height h has $< \ t^h$ leaves
- A t-ary tree with n leaves must have depth at least $\lceil \log_t n \rceil$
- This gives a lower bound on the worst case time to find an answer
- If the number of possible answers is n, then the algorithm MUST ask at least $\log_t n$ questions

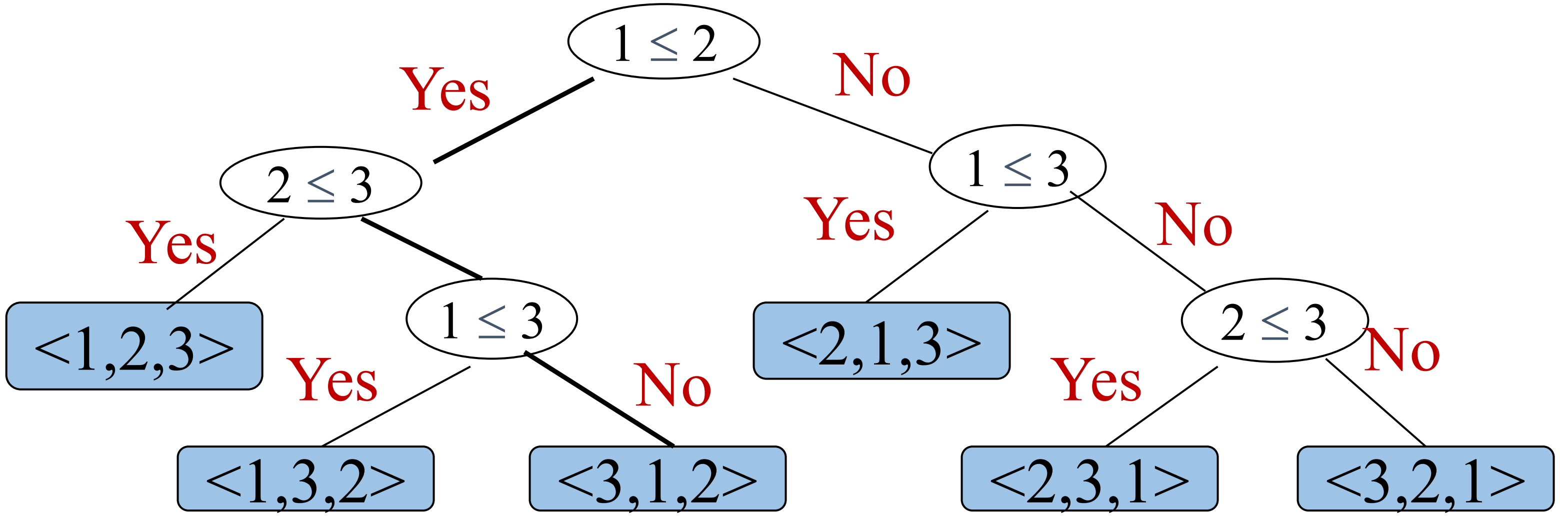
Lower-bound: Sorting

- Assume that we have a **comparison based** sorting algorithm.

- Number of possible sorted orders

  = number of all possible permutations of n elements

  = n!



<里面的都是index>

Lower-bound: Sorting

$\log n! \ = \Theta(n \log n)$

n/2 个

$$n! \ = \ 1 \times 2 \times \cdots \times n \geq \ \frac{n}{2} \times \left(\frac{n}{2} + 1\right) \times \ ... \times \ n \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$\Rightarrow \log n! \geq \log \left(\frac{n}{2}\right)^{\frac{n}{2}} = \frac{n}{2} \log \frac{n}{2} = \frac{n}{2} (\log n \ - \log 2) \ = \Omega(n \log n)$$

$$n! \ = \ 1 \times 2 \times \cdots \times n \leq \ \ n \times \ ... \times \ n = n^n \ \Rightarrow \log n! \leq \log n^n = O \ (n \log n)$$

Hence any comparison-based algorithm for sorting must take at least $log\ n! = \Theta(n \log n)$ time.

Lower-bound: Searching a sorted list

- Input: $A[1..n]$ where $A[1] \leq \ \ldots \leq \ A[n]$; key x.
- Output: index i such that $A[i] \ = \ x$ (or $i \ = \ 0$ if x not in A).

Information theory lower bound:

number of possible outputs $= \ n + 1$ so

every comparison tree has height $\geq \lceil \log(n+1) \rceil$

so every algorithm that uses only comparisons requires at least $\log(n+1)$ comparisons.

Lower-bound: Searching a sorted list

- Input: $A[1..n]$ where $A[1] \leq \ \ldots \leq \ A[n]$; key x.
- Output: index i such that $A[i] \ = \ x$ (or $i \ = \ 0$ if x not in A).

Information theory lower bound:

number of possible outputs $= \ n + 1$ so

every comparison tree has height $\geq \lceil \log(n+1) \rceil$

so every algorithm that uses only comparisons requires at least $\log(n+1)$ comparisons.

3-Reduction

Reduction

- If we know problem B can be solved by solving an instance of problem A, i.e., A is "harder" than B
- and we know that B has lower bound $L(n)$
- then A must also be lower-bounded by $L(n)$

Reduction: ExtractMax

Theorem: ExtractMax on a binary heap is lower bounded by $\Omega(\log n)$.

Proof. Suppose ExtractMax can be done faster than $\log n$, then HeapSort can be done faster than $n \log n$, because HeapSort is basically ExtractMax n times.

But HeapSort, as a comparison based sorting algorithm, has been proven to be lower bounded by $\Omega(n \log n)$.

$\rightarrow$ **Contradiction**, so ExtractMax must be lower bounded by $\Omega(\log n)$

Questions?