Lower bound

Complexity of problems

- Let $T_A(X)$ denote the running time of algorithm $\boldsymbol{A}$ given input X.
- The worst-case running time of $\boldsymbol{A}$ as a function of the input size

$$T_A(n) := \max_{|X|=n} T_A(X).$$

- The worst-case complexity of a *problem* $\boldsymbol{\Pi}$ is the worst-case running time of the *fastest* algorithm for solving it :

$$T_\Pi(n) := \min_{A \text{ solves } \Pi} T_A(n) = \min_{A \text{ solves } \Pi} \max_{|X|=n} T_A(X).$$

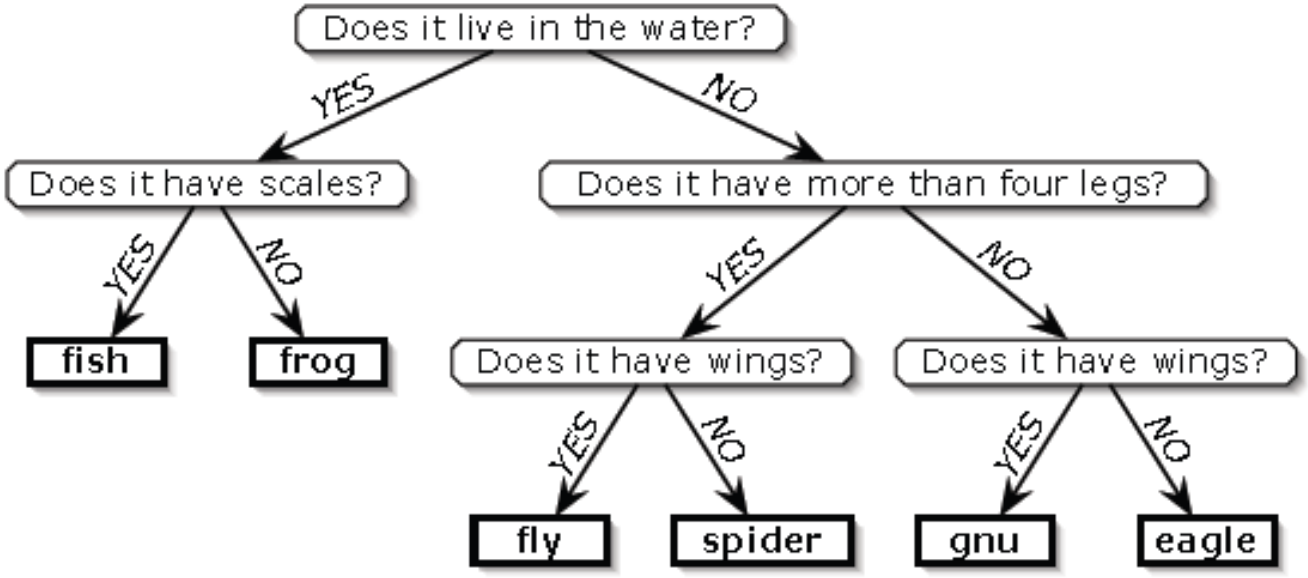
- Any algorithm $\boldsymbol{A}$ that solves $\boldsymbol{\Pi}$ immediately implies an upper bound on the complexity of $\boldsymbol{\Pi}$: $\boldsymbol{T_\Pi(n) \leq T_A(n)}$ follows directly from the definition of T_Π
- Faster algorithms gives us better (smaller) upper bounds.

Complexity of problems

- How can we argue that certain problems are *hard* (by proving *lower bounds* on their complexity) ?
- To prove $T_{\Pi}(n) = \Omega(f(n))$, we must prove that *every* algorithm that solves Π has a worst-case running time $\Omega(f(n))$, or equivalently, that *no* algorithm runs in $o(f(n))$ time.
- "all algorithms?" – need to specify precisely what kinds of algorithms we will consider and how to measure their running time : this specification is called a *model of computation*.

Decision tree model

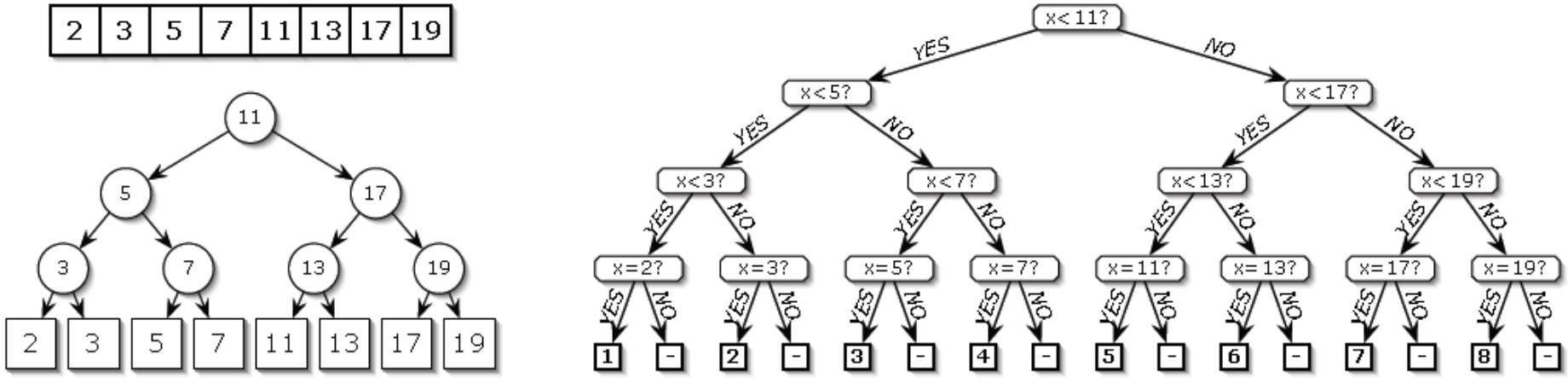
- Internal node : labeled by a query – a question about the input
- Edges : possible answers to the query
- Leaf : output



A decision tree to choose one of six animals.

Dictionary problem

- Given an array *A* with *n* numbers and a number *x*, we want to determine the position of *x* in the array *A*, if any.
- Sort *A* and use binary search.



Left: A binary search tree for the first eight primes.
Right: The corresponding binary decision tree for the dictionary problem (- = 'none').

Decision tree model

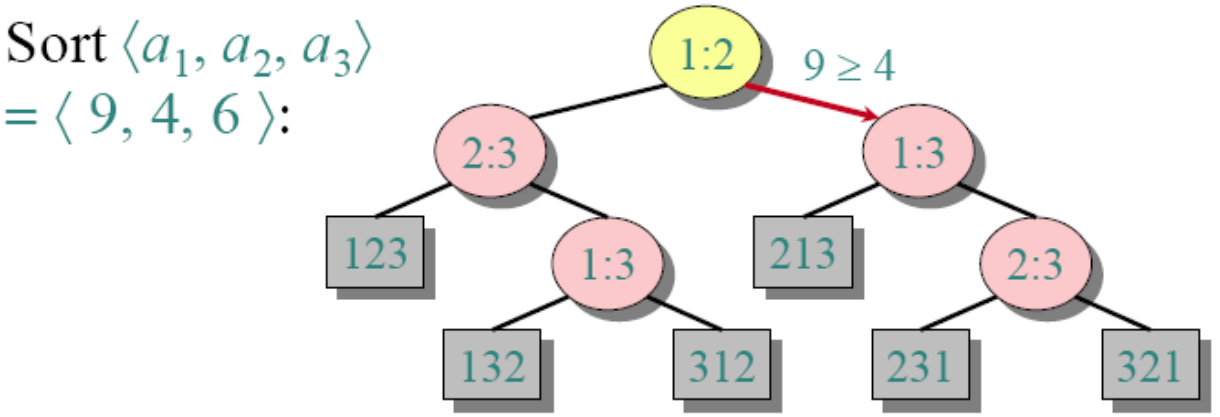
- Define the running time of a decision tree algorithm for a given input to be the number of queries in the path from the root to the leaf.
- We can generalize binary decision trees to *k*-ary decision trees where every query has (at most) *k* different answers.
- Observation: the answers to the queries must give you enough information to specify any possible output.
- If the problem has *N* different outputs, then the decision tree must have at least *N* leaves.
- Thus, if every query has at most *k* possible answers, the depth of the decision tree must be at least $\lceil \log_k N \rceil = \Omega(\log N)$.
- Dictionary problem : $n+1$ possible outputs -> at least $n+1$ leaves -> has depth at least $\lceil \log_k (n+1) \rceil = \Omega(\log n)$.
- So the complexity of the dictionary problem in the decision-tree model of computation is $\Omega(\log n)$ which matches the upper bound $O(\log n)$ from a perfectly-balanced binary search tree. Thus, binary search algorithm is *optimal*.

Lower bounds for sorting

- $\Omega(n)$ to examine all the input.
- All sorts seen so far are ***comparison sorts*** - the sorted order are only determined by comparisons between the input elements.
- All sorts seen so far are $\Omega(n \lg n)$.
- ***Is O(n lg n) the best we can do?***
- We'll show that $\Omega(n \lg n)$ is a lower bound for comparison sorts.

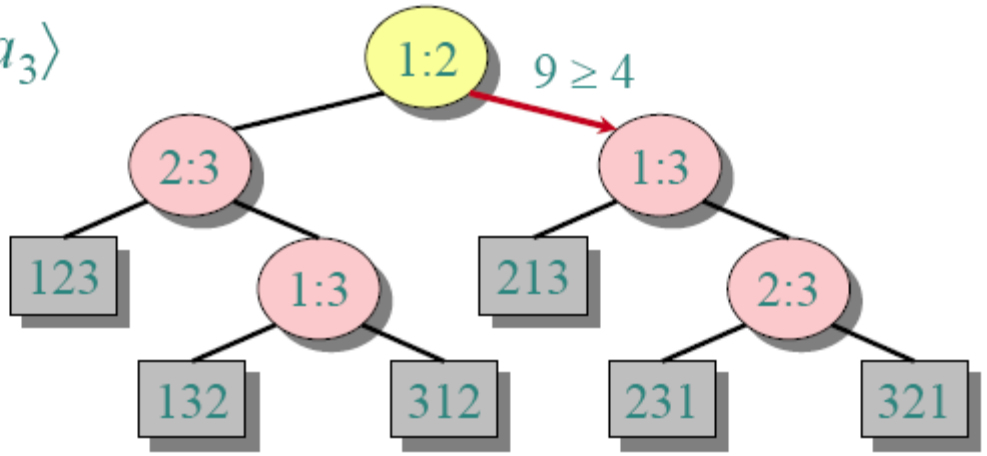
Decision tree

- Abstraction of any comparison sort.
- Represents comparisons made by
  - a specific sorting algorithm
  - on inputs of a given size.
- Abstracts away everything else (control, data movement.)
- Count only comparisons.
- Internal nodes : i:j - comparisons $a_i \leq a_j$
  - Left subtree gives subsequent comparisons if $a_i \leq a_j$
  - Right subtree gives subsequent comparisons if $a_i > a_j$
- Leaf : permutation $<\pi(1), \pi(2), \ldots, \pi(n)>$



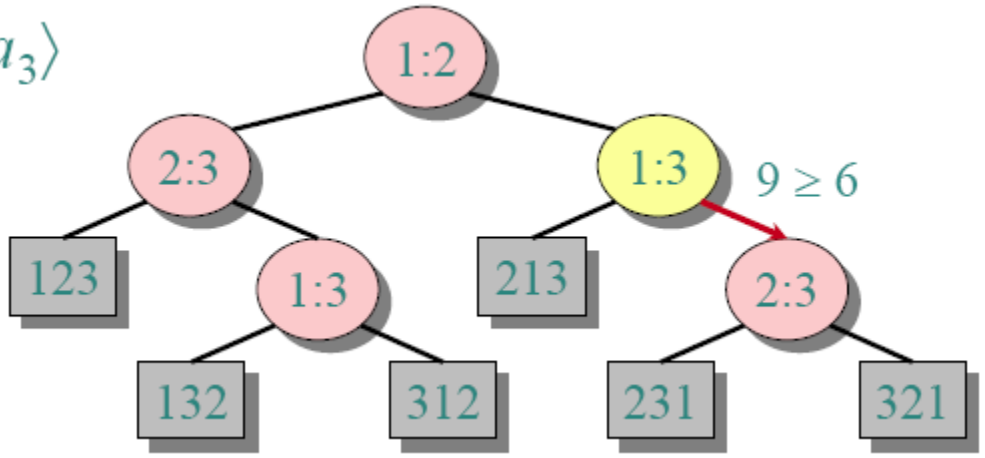
Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



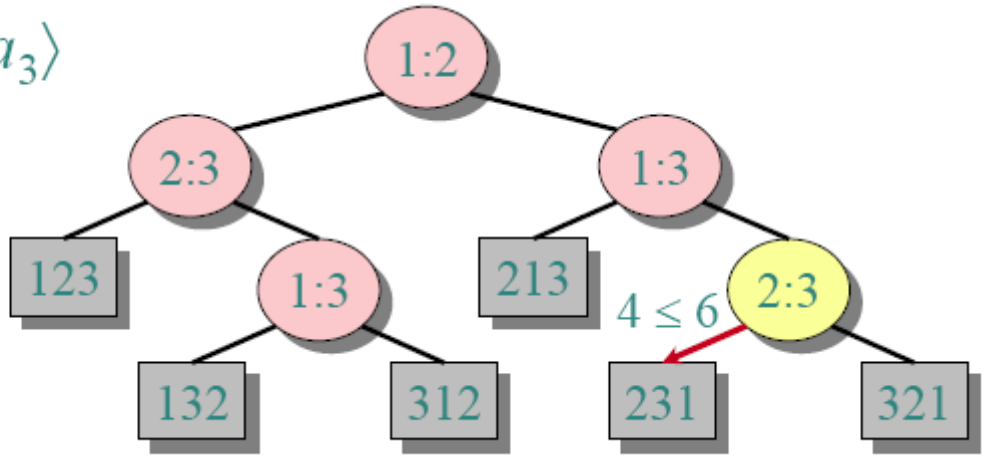
Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



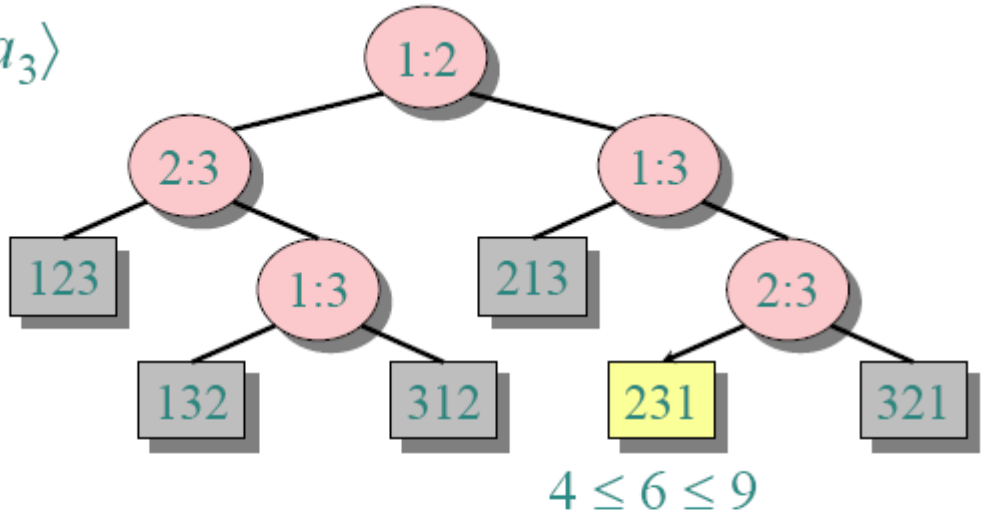
Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ has been established.

Decision-tree model

- A decision tree can model the execution of any comparison sort:
  - 1 tree for each input size *n*.
  - View the tree as if the algorithm splits in two at each node, based on the information it has determined up to that point.
  - The tree models all possible execution traces.
  - The running time of the algorithm = the length of the path taken.
  - Worst-case running time = height of tree

A lower bound for the worst case

- Theorem : Any comparison sort algorithm requires $\Omega(n \lg n)$ comparisons in the worst case.

Pf) There are $l \geq n!$ leaves on the decision tree, because every permutation appears at least once.

- Need to determine the height of a decision tree in which each permutation appears as a reachable leaf.
- Any binary tree of height h has $\leq 2^h$ leaves.
- $n! \leq l \leq 2^h$
- $h \geq \lg(n!)$ (since the lg function is monotonically increasing)
- $\lg(n!) = \Theta(n \lg n)$ (Stirling's approximation)

$$n! = \left(\frac{n}{e}\right)^n \sqrt{2\pi n}\left(1 + \Theta\left(\frac{1}{n}\right)\right) > \left(\frac{n}{e}\right)^n.$$

- $h = \Omega(n \lg n)$.

Lower bound for comparison sorting

- Corollary : Heapsort and merge sort are asymptotically optimal comparison sorts.

- How about *randomized* algorithms?
  - At least one tree for each n.
  - Proof still works because every tree works!
  - So, randomized quicksort is asymptotically optimal in expectation.

Definition: (*Lower bound*)

- A lower bound in time complexity for solving a problem is the least amount of time to solve the most difficult instance of the problem.
- (Alternatively) A lower bound in time complexity for solving a problem *P* is said to be time required to solve *P*.

Definition: (*Upper bound*)

- An upper bound in time complexity for solving a problem *P* is time required for an algorithm to solve *P*.

Existence of an upper bound for a problem

$\Rightarrow$ Existence of an algorithm

$\therefore$ Algorithm - dependent !!!

However, a lower bound is inherent to a problem.