Sorting Lower Bound

Tufts University

Warm-up question

Philosophical-question: What difference in input makes the algorithm behave differently?

Previously ...

CountingSort and RadixSort are fast

- CountingSort works on limited values
- RadixSort repeatedly runs CountingSort ℓ many times

☺ $O(n+k)$ or $O(\ell(n+r))$ respectively

☹ Some limitations on data

Can we apply this idea to improve *classic* algorithms? NO

Today's goal

Theorem

Any algorithm that sorts n numbers will need $\Omega(n \log n)$ time

Roadmap:

1. COTD
2. Decision tree model
3. Number of outcomes in sorting problem
4. Runtime of decision tree
5. Glueing all pieces together

Cheat of the day

Lemma (Stirling's approximation)

$n! = \Theta(\sqrt{2\pi n}\frac{n^n}{e^n})$

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

Second cheat of the day

Lemma

Let T be a binary tree with $\geq n$ leaves. T has height $h \geq \lfloor \log_2 n \rfloor$.

Decision tree model

Represents any comparison-based algorithm:

- **Unknown** algorithm

  Goal is known

  n is fixed
- Don't care about local variables
- Compares and decides how to act

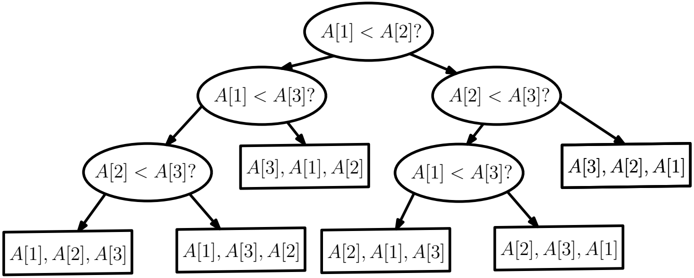
  If $x \leq y$ then do this, else do that

  x and y are input values/local variables.
- No bit manipulation

Example: sorting three numbers

Other decision tree?

Outcomes



- Algorithm stops comparing when we reach a leaf
- We give an *outcome* (i.e, $A[1] < A[3] < A[2] < A[4]$)
- Sorting n numbers → Number of outcomes =

Runtime of DT algorithms

Lemma

For any algorithm A, consider its decision tree representation T_A. The (worst-case) runtime of A is at least the height of T_A.

Proof.

- Height $T_A \Rightarrow$ worst case we do T_A many comparisons
- Each comparison needs $\Omega(1)$ time
- Time spent in local variables is ignored $\Rightarrow$ lower bound

$\square$

Glueing it all together

Theorem
Any (comparison-based) algorithm that correctly sorts n numbers will need $\Omega(n \log n)$ time

Proof.

let A be **any** sorting algorithm, let T_A be its decision tree

T_A is a binary tree of height h

Runtime of A is $\Omega(h)$

if A is correct, the number of leaves of T_A is $\geq n!$

$h \geq \log_2 n! = \Omega(n \log n)$

$\square$

Magic!

Discussion

- Any sorting strategy will need $\Omega(n \log n)$

  ANY! Even those that have not been discovered yet!

- What about $\textsc{CountingSort}$ or $\textsc{RadixSort}$?

  Technically does not apply

  In practice it does

  If all n numbers are distinct you need $\ell = \Omega(\log n)$ bits

Practice

(Fall'19 exam) While walking along the halls of Halligan, you find n McGuffins (for some $n > 0$). All of the McGuffins look and feel identical, but there are two that are authentic (and the rest are fake). The Guardian of the McGuffins is a robot with a small screen. Each time you ask a question, it will answer with a single digit (a number from 0 to 9).

1. What is the **minimum** number of questions that someone would need to ask to be sure of finding the real McGuffins?
2. (Just for fun, outside 160) Design a strategy to find the real McGuffin

Lower bound

Lemma

Any correct strategy for finding the real McGuffin will need at least $\log_{10}\binom{n}{2}$ *many questions*

Proof.

- Model ANY strategy with decision tree T
- 10 possible answers $\Rightarrow$ a node branches ≤ 10 ways
- Nodes have ≤ 10 children $\Rightarrow$ height$(T) \geq \log_{10}(\#\text{of leaves})$
- $\binom{n}{2}$ different solutions $\Rightarrow$ # of leaves $\geq \binom{n}{2} = \frac{n(n-1)}{2} \approx n^2/2$
- # Questions $\geq$ height$(T) \geq \log_{10}(\#\text{of leaves}) \approx 2\log_{10}(n/2)$

$\square$

Strategy?

Just for fun! Outside scope of 160

- If $n = 2$ the remaining two are the real McGuffins
- Otherwise, split n McGuffins into 4 balanced groups
- Ask guardian: Say a digit according to the following formula:

  0 if both real McGuffins are in the first group
  1 if both real McGuffins are in the second group
  ...
  3 if both real McGuffins are in the fourth group
  4 if you have a real McGuffins in both groups 1 and 2
  5 if you have a real McGuffins in both groups 1 and 3
  ...
  9 if you have a real McGuffins in both groups 3 and 4
- Recurse on the 1-2 groups containing the real McGuffins
- Recurrence: $T(n) \leq T(\frac{2n}{4}) + 1 = T(n/2) + 1$
  - base case $T(2) = 0$
  - Solves to $T(n) = \log_2 n$ (by substitution)

Not optimal! Can you improve?