Bounds on Comparison Sorts

CSE 373 Winter 2020

Instructor: Hannah C. Tang

Teaching Assistants:

Aaron Johnston
Amanda Park
Anish Velagapudi
Brian Chan
Elena Spasova
Ethan Knutson
Farrell Fileas
Howard Xiao
Jade Watkins
Lea Quan
Nathan Lipiarski
Sam Long
Yifan Bai
Yuma Tou

Announcements

- Remember : no late days for HW8 (Seam Carving)!
- You can always make appointments with staff (TAs or me) to discuss anything: homework, concepts, imposter syndrome, and more
  - Crucially, if you need health accommodations for the final please reach out!

Lecture Outline

- **Reading Review: Twiddling with Constants!**
- Asymptotic Analysis Practice
- Theoretical Lower Bound for Comparison Sorts

Feedback from the Reading Quiz

- How were 47 and 67 decided upon?
- What is a run (in the sorting context)?
- If QuickSort looks for runs, doesn't that mean it's doing extra work?
- Why is the best possible sorting algorithm Ω(N)?

Reference Types vs Primitive Types

- Java uses MergeSort* for reference types because MergeSort is stable

- Java uses adaptive QuickSort for primitive types because stability doesn't matter for these types, and QuickSort's constants are better than MergeSort's
  - However: InsertionSort's constants are better for small arrays
  - However: MergeSort's constants are better for partially-sorted arrays

* technically, this is a MergeSort variant known as TimSort

Java's QuickSort Adapts to its Input

- At the beginning of the sort (ie, only once), sort() checks whether the input is partially sorted by looking for *runs*
  - 9 7 1 2 4 8 6 3 5
  - This is Θ(N) work, so still dominated by our N log N runtimes
  - If there are long-enough runs, switches to MergeSort. Done.
- If not, it picks two pivots, partitions the input, and recursively sorts each partition
  - When the partition is small enough, switches to InsertionSort. Done.
- Why is "small enough" defined as <47?
  - Performance testing on InsertionSort and QuickSort using randomized input; the "break point" happened at 47

* technically, this is a MergeSort variant known as TimSort

Lecture Outline

- Reading Review: Twiddling with Constants!
- **Asymptotic Analysis Practice**
- Theoretical Lower Bound for Comparison Sorts

Problem #1

- Consider the functions N! and $(N/2)^{N/2}$. Is $N! \in \Omega((N/2)^{N/2})$? Prove your answer.

```
N!        = N   * (N-1)* … * (N/2 + 1) * N/2 * … * 2 * 1
(N/2)^N/2 = N/2 * N/2 * … *    N/2     * N/2
```

- $N! > (N/2)^{N/2}$ for large N, therefore $N! \in \Omega((N/2)^{N/2})$
- Demo: https://www.desmos.com/calculator/7lahriir6s

Problem #2

- Now, let's consider the functions log N! and N log N. Is log N! $\in \Omega$(N log N)? Prove your answer.

- From problem #1, we know N! > $(N/2)^{N/2}$
  - Taking the log of both sides: log N! > log $(N/2)^{N/2}$
  - log N! > N/2 log (N/2)
  - log N! > N/2 (log N – log 2)

- Therefore, log N! $\in \Omega$(N log N)

- Demo: https://www.desmos.com/calculator/4ptk1kcmss

Problem #3

- Show N log N ∈ Ω(log N!)

```
log N!   = log(N   *     (N-1) * … *      1)
         = log(N) + log(N-1) + … + log 1
N log N  = log(N) + log(N)     + … + log(N)
```

- log N! < N log N for large N, therefore N log N ∈ Ω(log N!)

- Demo: https://www.desmos.com/calculator/4jeakr9vvb

Poll Everywhere

pollev.com/uwcse373

- Given that $N \log N \in \Omega(\log N!)$ and $\log N! \in \Omega(N \log N)$, which of the following statements are true?

A. $N \log N \in \Theta(\log N!)$

B. $\log N! \in \Theta(N \log N)$

C. Both A and B

D. Neither A nor B

E. I'm not sure …

Lecture Outline

- Reading Review: Twiddling with Constants!
- Asymptotic Analysis Practice
- **Theoretical Lower Bound for Comparison Sorts**

Comparison Sorts Review

	Best-Case Time	Worst-Case Time	Space	Stable?	Notes
SelectionSort	$\Theta(N^2)$	$\Theta(N^2)$	$\Theta(1)$	No	
In-Place HeapSort	$\Theta(N)$	$\Theta(N \log N)$	$\Theta(1)$	No	Slow in practice
MergeSort	$\Theta(N \log N)$	$\Theta(N \log N)$	$\Theta(N)$	Yes	Fastest stable sort
In-Place InsertionSort	$\Theta(N)$	$\Theta(N^2)$	$\Theta(1)$	Yes	Best for small or partially-sorted input
Naïve QuickSort	$\Theta(N \log N)$	$\Theta(N^2)$	$\Theta(N)$	Yes	>=2x slower than MergeSort
Dual-Pivot QuickSort	$\Omega(N)$	$O(N^2)$	?	No	Fastest comparison sort

Best Case != Worst Case

- Our best-cases are linear but our worst-cases are N log N

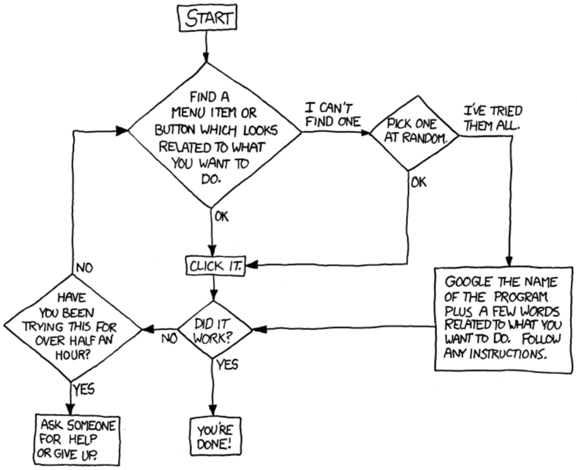
- We spent all of last lecture *and* the reading twiddling with real-world constants to speed up N log N, but we didn't ask ourselves

*Does there exist a **comparison-based sorting** algorithm whose worst-case is faster than N log N?*

- Let's ask that now. Call this theoretical algorithm "OptimalSort"
  - Next, we will describe the constraints on OptimalSort and then try to derive its worst-case runtime

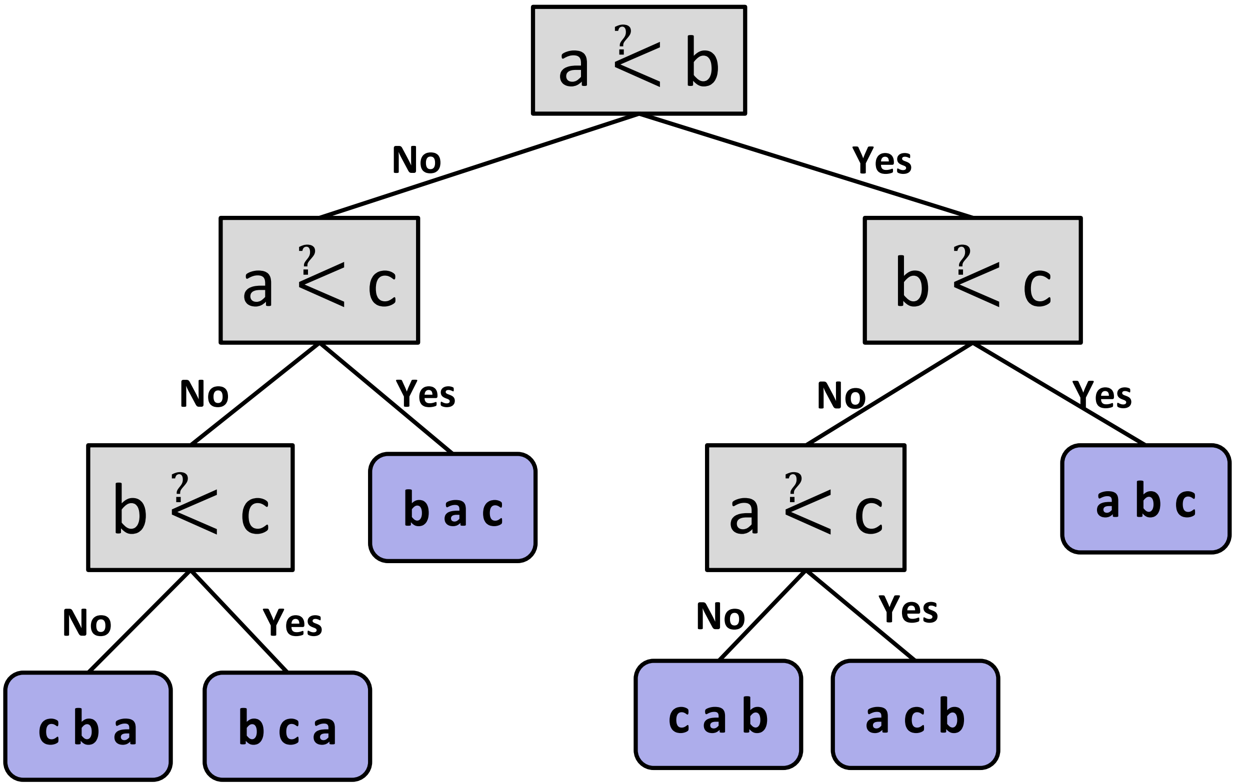
OptimalSort on 3 Items

- Given 3 items a, b, c, what is the minimum number of *comparisons* OptimalSort needs to order them?
- We don't know in what order OptimalSort would do the comparisons, but we can *model* those comparisons as a decision-tree



https://xkcd.com/627/

OptimalSort Decision Tree: N=3



Poll Everywhere

pollev.com/uwcse373

- How many possible *permutations* exist for a list of N=4 elements?

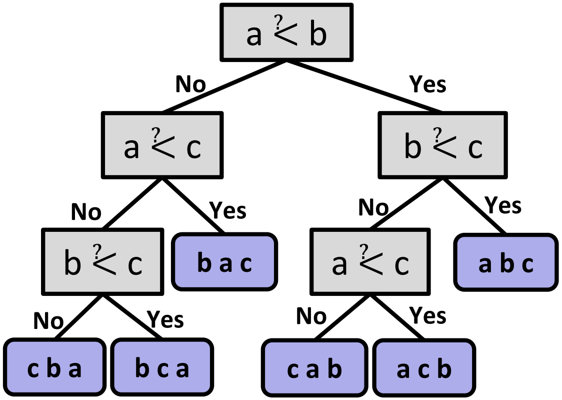
A. 16

B. 24

C. 32

D. 36

E. I’m not sure …



Poll Everywhere

pollev.com/uwcse373

- In the worst case, how many *comparisons* would OptimalSort make for a list of N=4 elements?

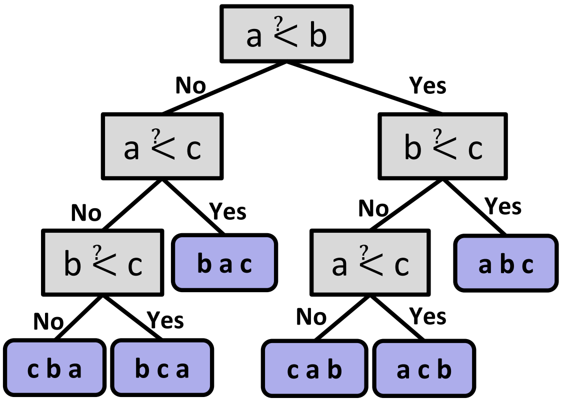
A. 3

B. 4

C. 5

D. 6

E. I’m not sure …



OptimalSort for all N

- OptimalSort needs to decide between N! permutations (ie, N! leaves) in a list of N elements
- The height of OptimalSort's decision tree is $\log_2 N!$, rounded up
- Therefore, OptimalSort's worst-case requires Ω(log N!) comparisons
  - So its total runtime must be Ω(log N!)
    - Because we still need to do swaps, merges, partitions, etc
  - … which is equivalent to Ω(N log N)
  - … which means that OptimalSort's worst-case runtime is Ω(N log N)

Comparison Sorts Review

	Best-Case Time	Worst-Case Time	Space	Stable?	Notes
SelectionSort	$\Theta(N^2)$	$\Theta(N^2)$	$\Theta(1)$	No	
In-Place HeapSort	$\Theta(N)$	$\Theta(N \log N)$	$\Theta(1)$	No	Slow in practice
MergeSort	$\Theta(N \log N)$	$\Theta(N \log N)$	$\Theta(N)$	Yes	Fastest stable sort
In-Place InsertionSort	$\Theta(N)$	$\Theta(N^2)$	$\Theta(1)$	Yes	Best for small or partially-sorted input
Naïve QuickSort	$\Theta(N \log N)$	$\Theta(N^2)$	$\Theta(N)$	Yes	>=2x slower than MergeSort
Dual-Pivot QuickSort	$\Omega(N)$	$O(N^2)$	?	No	Fastest comparison sort

- HeapSort, MergeSort, and Dual-Pivot QuickSort are asymptotically optimal
  - Mathematically impossible to make *asymptotically fewer* comparisons
  - That's why we focus on optimizing their constants