CS1200: Intro. to Algorithms and their Limitations	Anshu & Vadhan
Lecture 3: Measuring efficiency (cont.)	
Harvard SEAS - Fall 2024	*2024-09-10*

1 Announcements

- Lecture 2 detailed notes posted (but to be posted again later today with some small corrections/additions).
- Section 0 recording available on Course Schedule Doc.
- Handout: Lecture notes 3 (PDF on Ed)
- Avi Wigderson (Abel Prize '21, Turing Award '23) lecture Friday 3:45pm, in this room "The Value of Errors in Proofs". Highly recommended!
- Salil OH: Thu 11-12 in SEC 3.327, Anurag OH: Fri 1:30-2:30 SEC 3.323.
- Sender–Receiver exercise today (partway through class)! Followed by a 5min in-class reflection survey (required for your participation grade).
- Anurag will be available to support DCE students for the SRE in the Study Lounge, Fri 2:30-3pm.
- Don't burn yourself out on psets. Mistakes are part of learning, hence the possibility of revision videos. On psets 0 and 1, any grade can be revised even up to an R.

2 Loose Ends from Lecture 2

We spent most of the class completing material from Lecture 2 (Computational Problems and their Complexity), as outlined below, before turning to the Sender-Receiver Exercise on SingletonBucketSort. See Lecture Notes 2 for details.

Definition 2.1. Let $h, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. We say:

- $h = O(g)$ if
- $h = \Omega(g)$ if
  Equivalently:
- $h = \Theta(g)$ if
- $h = o(g)$ if
  Equivalently:
- $h = \omega(g)$ if
  Equivalently:

Informal def: *computational complexity* of a problem Π = smallest possible growth rate of runtime among algorithms that solve Π.

2.1 Computational Complexity of Sorting

Let's analyze the runtime of the sorting algorithms covered so far (`ExhaustiveSearchSort`, `InsertionSort`, `MergeSort`).

$T_{exhaustsort}(n) =$

$T_{insertsort}(n) =$

And when n is a power of 2, $T_{mergesort}$ satisfies the following recurrence:

$T_{mergesort}(n) \leq$

From this, we can derive that when n is a power of 2,

$T_{mergesort}(n) =$

And the same holds true when n is not a power of 2 by rounding n up to the next-larger power of 2.

Exercise 2.2. Order $T_{exhaustsort}, T_{insertsort}, T_{mergesort}$ from fastest to slowest, i.e. T_0, T_1, T_2 such that $T_0 = o(T_1)$ and $T_1 = o(T_2)$.

In the detailed lecture notes for Lecture 2, you can find a proof (optional to read) that `MergeSort` is *asymptotically optimal* among *comparison-based* sorting algorithms:

Theorem 2.3. *If A is a comparison-based algorithm that correctly solves the sorting problem on arrays of length n in time $T(n)$, then $T(n) = \Omega(n \log n)$. Moreover, this lower bound holds even if the keys are restricted to be elements of $[n]$ and the values are all empty.*

We will be interested in three very coarse categories of running time:

(at most) exponential time $T(n) = 2^{n^{O(1)}}$ (slow)

(at most) polynomial time $T(n) = n^{O(1)}$ (reasonably efficient)

(at most) nearly linear time $T(n) = O(n \log n)$ or $T(n) = O(n)$ (fast)