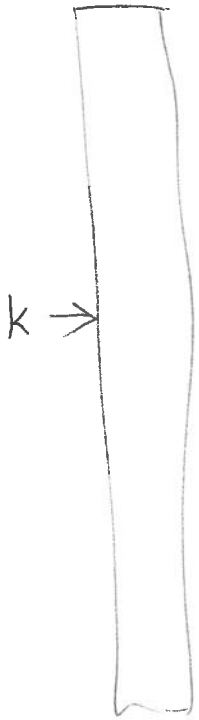
CSCE 750
9/28/2023

Lower bound for comparison-based sorting
Hash tables

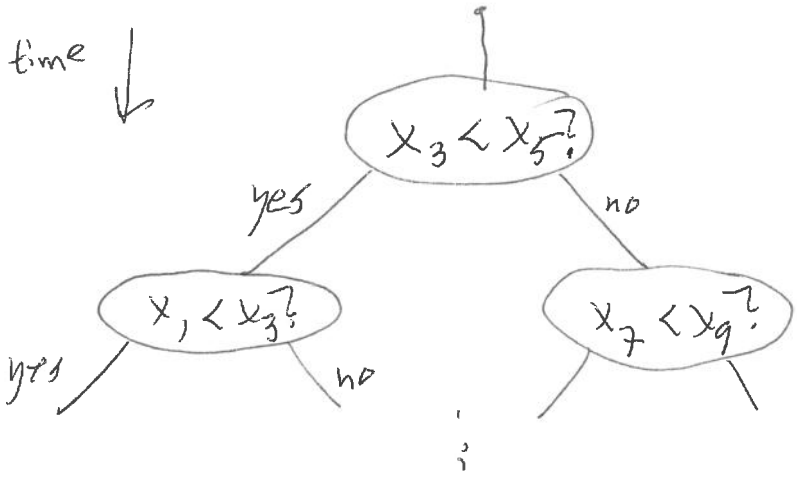


Theorem: Every comparison-based sorting algorithm takes worst-case time $\Omega(n \lg n)$ to sort n elements.

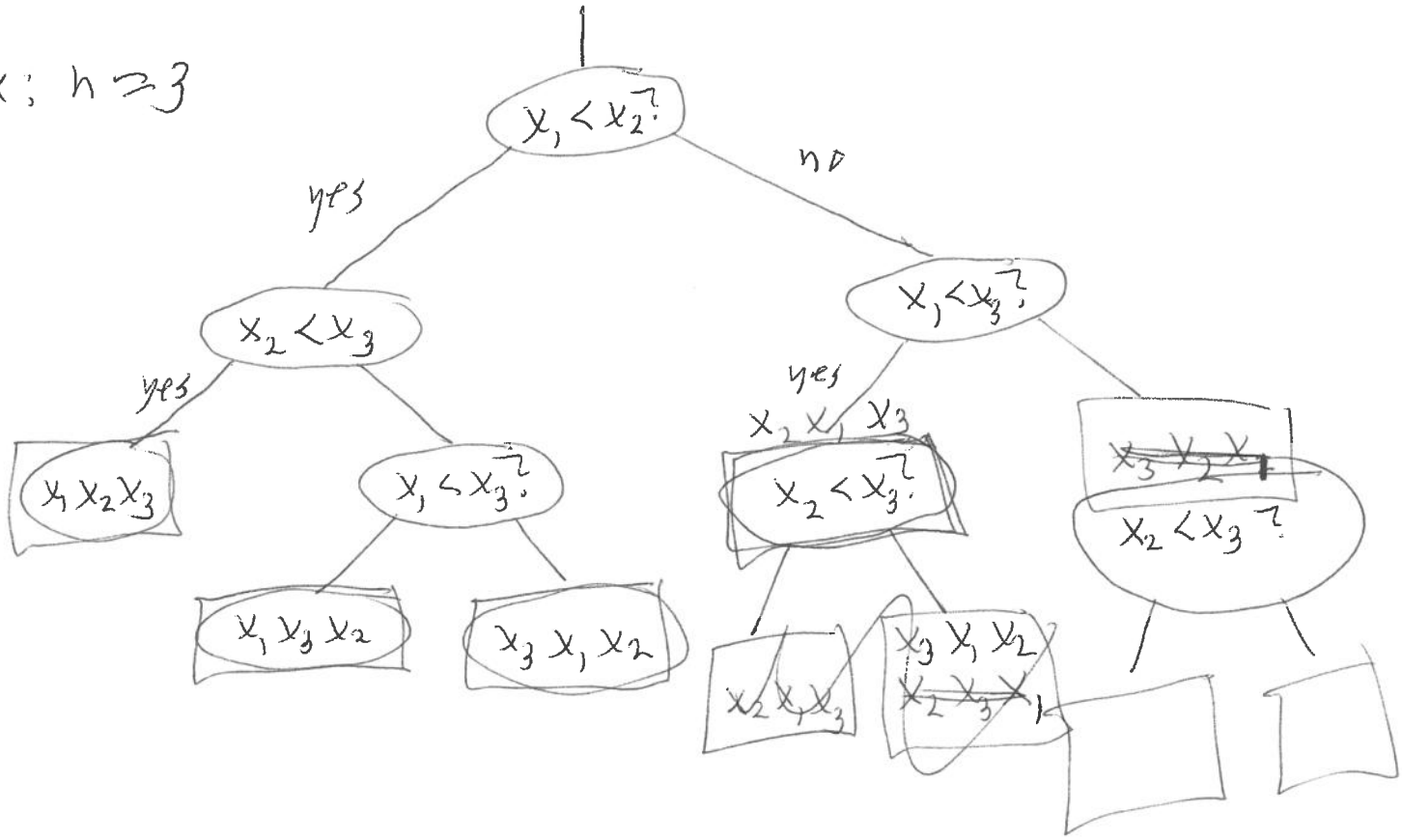
Cor: Mergesort & Heapsort (& deterministic Quicksort) are asymptotically optimal.

Note: Some sorting algos (e.g., radix sort, bucket sort) run faster, but not comparison-based.

Proof: Let A be any comparison-based sort.
On some input $\langle x_1, \ldots, x_n \rangle$ of n distinct comparable elements, the behavior of A can be described as a tree



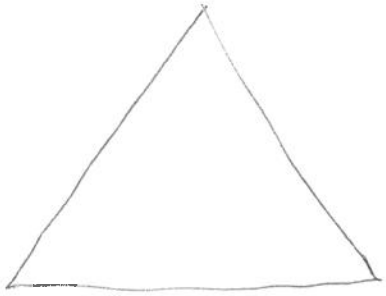
Ex: $n = 3$



For every permutation of n elements, there is a distinct leaf in A's decision tree.

$\therefore$ Tree has $\geq n!$ many leaves

Note: a binary tree of depth d has $\leq 2^d$ many leaves



Thus A's decision tree has some depth d such that $2^d \geq n!$

"Depth d" means A makes d comparisons on atleast one input.

$$2^d \geq n! = 1 \cdot 2 \cdot \cdots \cdot n$$

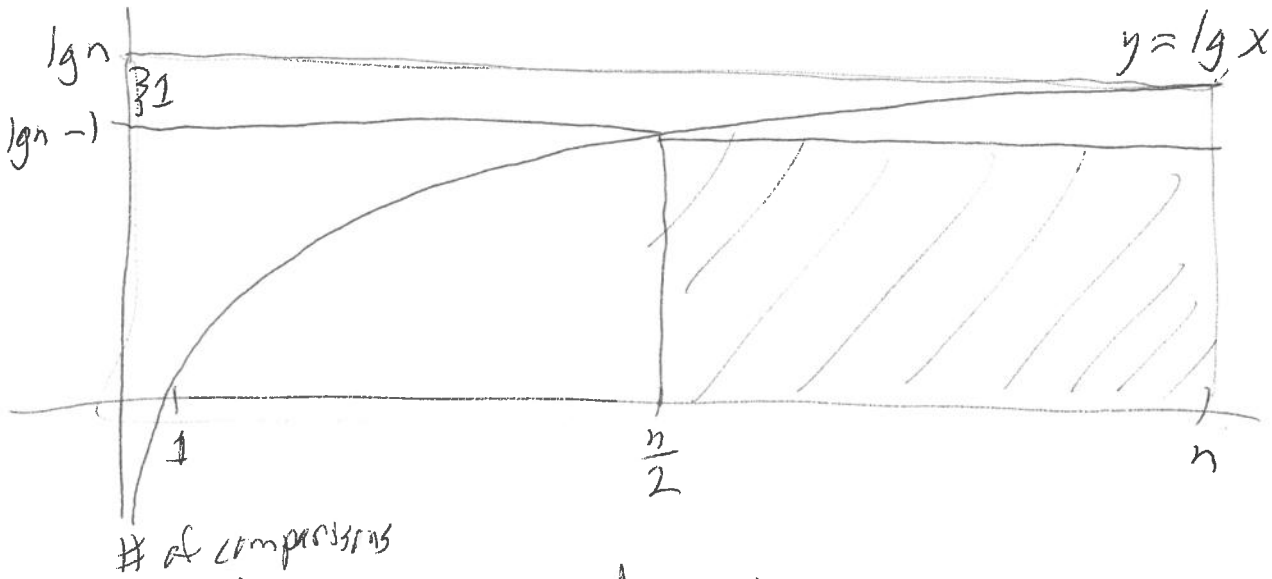
take lg of both sides:

$$d = \lg 2^d \geq \lg(n!) = \sum_{k=1}^{n} \lg k$$

$$\geq \sum_{k=\lceil \frac{n}{2} \rceil}^{n} \lg k \geq \sum_{k=\lceil \frac{n}{2} \rceil}^{n} \lg k \geq \sum_{k=\lceil \frac{n}{2} \rceil}^{n} \lg\left(\frac{n}{2}\right)$$

$$\geq \left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right) = \frac{n}{2}(\lg n - 1) = \frac{n}{2}\lg n - \frac{n}{2}$$

$$= \Omega(n \lg n) \qquad \square$$



Best case for comp-based sorting
= depth of shallowest leaf

Book exercise; $n-1$ comparisons in best case is a tight lower bound.

Hash tables — A hash table is a collection of items with keys

that supports the operations

- Insert(H, x) — insert key x into hash table H
- Lookup(H, x) — return item with key x in H (H is unchanged)
- Delete(H, x) — remove item x from H

Assume: keys are drawn from some universe U (finite).

For simplicity Assume $U = \{0, \ldots, |U|-1\}$

Simple way: $H[0..(|U|-1)]$ is an array of keys of size $|U|$. Store item x at $H[x]$

All ops take $O(1)$ time.

Disadvantage if $|U|$ is really big: need $\Theta(|U|)$ space. (May not have enough space)

Instead: $H[0..(m-1)]$ array of items (keys) (for some $m << |U|$) to store the items.

Store item x at position $H[h(x)]$, where key

$h(x)$: hash of x

~~$h : \{0, \ldots, |\mathcal{U}|-1\}$~~

$h : \mathcal{U} \to \{0, \ldots, m-1\}$

is some function called a hash function.

Requirements for h:

- h should be easy to compute (fast)
- h should look "random"

Two examples: ~~$0 \le x \le |\mathcal{U}|-1$~~ ($x$ an integer)

$h(x) := x \bmod m$

$h(x) := \lfloor m(Ax - \lfloor Ax \rfloor) \rfloor$ for some $A \in \mathbb{R}$ or $A \in \mathbb{Z}$

($Ax - \lfloor Ax \rfloor$: fractional part of Ax in $[0,1)$; $m(\ldots)$ in $[0,m)$; $\lfloor \ldots \rfloor$ integer in $[0..(m-1)]$)

Me:

$m := 101$ (not too big, not too small, and prime)

$\mathcal{U}$ consists of ~~ASCII~~ ASCII strings

s is a ~~null~~-terminated string

```
h(char *s) {
    sum = 0;
    while (*s) { sum += (37*sum + (*s)) % 101;
                 s++; } return sum; }
```

Big use: symbol table in a compiler.
Also native to many programming environments.

If $m < |U|$, there will be distinct $x, y \in U$ such that $h(x) = h(y)$. I.e., h is not one-to-one.

A pair (x, y) s.t. $x \neq y$ but $h(x) = h(y)$ is called a collision.

Collisions are inevitable, so we need techniques to deal with them.