# LECTURE #3

CS 170

Spring 2021

So far we have applied divide and conquer to arithmetic problems:

- Karatsuba: faster integer multiplication $(n^2 \to n^{\log_2 3})$
- Strassen: faster matrix multiplication $(n^3 \to n^{\log_2 7})$

Today we apply divide and conquer to common tasks on lists:

① sorting

② finding the median

**SORTING:** given a list of numbers $a_1, \dots, a_n$, output them in increasing order.
(or decreasing)

Idea is to (i) split the list into two halves
(ii) recursively sort each half
(iii) merge the two sorted halves

MERGESORT$(a_1, \dots, a_n)$ := 1. if $n=1$, return $a_1$
2. $S_L :=$ MERGESORT$(a_1, \dots, a_{n/2})$
3. $S_R :=$ MERGESORT$(a_{n/2+1}, \dots, a_n)$
4. $S =$ merge$(S_L, S_R)$
5. return $S$

How to implement merge? Take smaller element from the two sorted lists and repeat.

Ex:

$$\left.\begin{matrix} 3 & 7 & 10 & 13 & 15 \\ 2 & 6 & 11 & 14 & 15 \end{matrix}\right\} \quad 2 \quad 3 \quad 6 \quad 7 \quad 10 \quad 11 \quad 13 \quad 14 \quad 15 \quad 15$$

Hence merge$(S_L, S_R)$ runs in time $O(|S_L| + |S_R|)$.

(Each iteration compares two elements and removes one.)

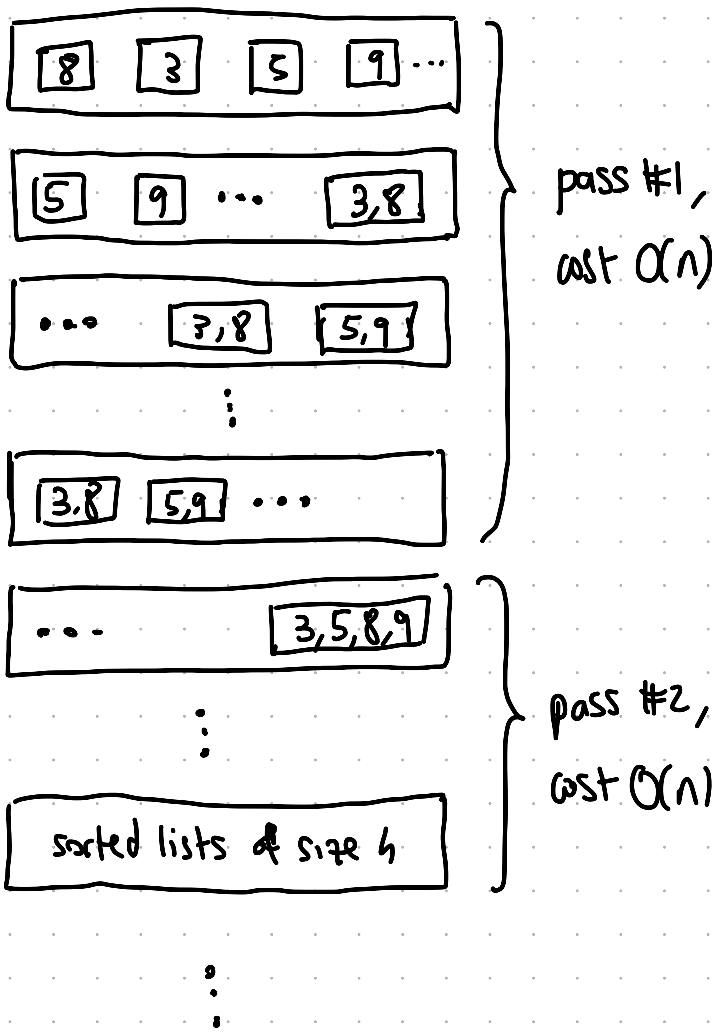
balanced case:
work at each of $\log_b n$ levels is $n^d$

The running time is $T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$.

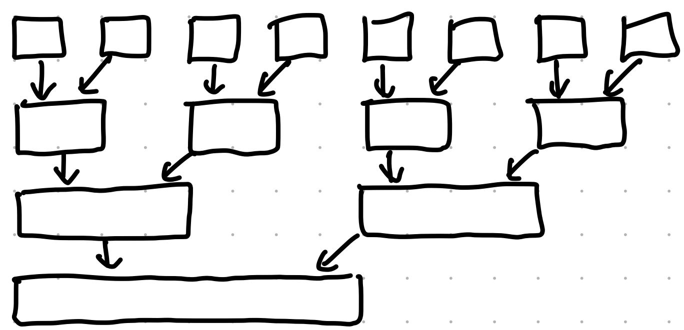
By the Master Theorem on Recurrences: $a=2, b=2, d=1 \Rightarrow \frac{a}{b^d} = \frac{2}{2^1} = 1 \Rightarrow O(n^d \log_b n) = O(n \log n)$.

All the "real work" is in merging, as nothing happens till the recursion hits the base case.

This naturally leads to an iterative algorithm that maintains a queue on lists:




There are $O(\log n)$ passes, each taking time $O(n)$, which double the size of the lists on the queue.



Q: can we do better than mergesort? No and Yes

No: mergesort is a comparison sort, i.e., an algorithm in which the only operation performed on the input elements are comparisons (their values are otherwise ignored)

> **theorem:** Any comparison sorting algorithm requires $\Omega(n \log n)$ comparisons to sort lists of $n$ elements.

So mergesort is optimal among comparison sorting algorithms.

Yes: there are sorting algorithms that are not solely based on comparisons.

For example, if the elements are $w$ bits long, then:

- radix sort uses $O(w \cdot n)$ bit operations
- merge sort uses $O(w \cdot n \cdot \log n)$ bit operations (a comparison costs $O(w)$ bit operations)

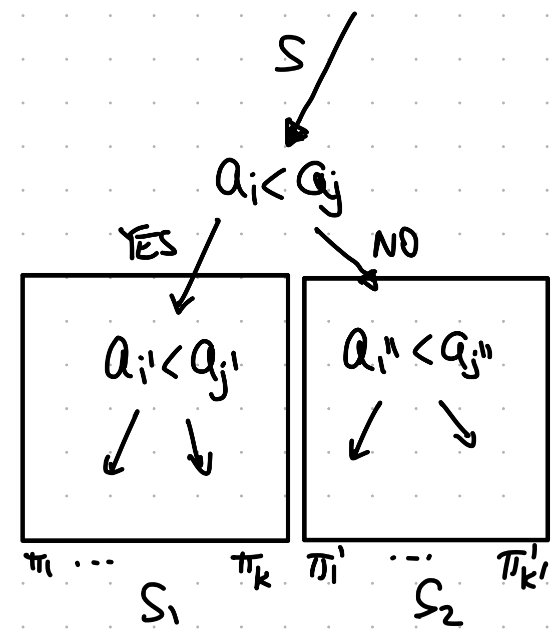
There are many sorting algorithms and the "best" one depends on the application.
(Data resides in RAM vs disk, mergesort works better on linked lists, ...)

Back to **theorem:** Any comparison sorting algorithm requires $\Omega(n\log n)$ comparisons to sort lists of $n$ elements.

Fix an algorithm $A$. WLOG focus on input lists $a_1,\dots,a_n$ where elements are distinct.
The computation of $A$ on $a_1,\dots,a_n$ defines a permutation $\pi:[n]\to[n]$ (the output is $a_{\pi(1)},a_{\pi(2)},\dots,a_{\pi(n)}$).
Every permutation is a possible output.
Let $S$ denote the set of possible permutations at a given point in $A$'s computation.

Before algorithm starts: $|S| = |\{\text{all possible permutations}\}| = n!$

At each comparison: if $a_i < a_j$ then $S \mapsto S_1$
if $a_i > a_j$ then $S \mapsto S_2$




Since $S_1 \cup S_2 = S$, we know that $|S_1| \geq |S|/2$ or $|S_2| \geq |S|/2$.
So a comparison divides possible outputs by at most 2.

Hence, "# of comparisons until $|S|=1$"

$$\geq \log_2(n!) \geq \log_2\left(\frac{n}{e}\right)^n = n\log n - n\log_2 e = \Omega(n\log n)$$

Note: this is a <u>worst-case</u> lower bound (depth of deepest leaf is $\Omega(n\log n)$), but can be improved to an <u>average-case</u> lower bound (average depth of leaf is $\Omega(n\log n)$)

# MEDIAN FINDING :

given $S = \{a_1, \dots, a_n\}$ output median$(S)$ := "$a \in S$ s.t. half of $S$ is smaller & half of $S$ is bigger"

How is median(S) different from average(S) = $(\sum_{i=1}^{n} a_i)/n$ ?

Ex:

| | | |
|---|---|---|
| (1,1,1) → | avg = 1 | median = 1 |
| (1,1,10) → | avg = 4 | median = 1 |
| (1,1,100) → | avg = 34 | median = 1 |
| (1,1,1000) → | avg = 334 | median = 1 |

median is one of the elements, and is less sensitive to outliers

How to compute median?

**Idea 1:** sort and take middle element — $O(n \log n)$

**Idea 2:** we do <u>NOT</u> care about the order of elements above and below the median

We use divide and conquer to solve a harder problem:

analogous to how strong induction can simplify recursion

| | | |
|---|---|---|
| <u>Selection</u> | input: | set of numbers $S$, index $k \in [n]$ |
| | output: | $k$-th smallest element in $S$ |

Note: $k = \frac{|S|}{2} = \frac{n}{2}$ is the median (some defs average two middles when $S$ is even)

Idea: pick $a \in S$ and split $S$ into $S_L := \{\text{elts in } S \text{ smaller than } a\}$
$S_a := \{\text{elts in } S \text{ equal to } a\}$
$S_R := \{\text{elts in } S \text{ greater than } a\}$

Then recurse in a straightforward way.

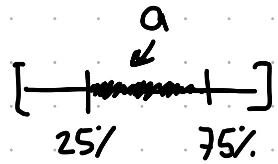
Select $(S,k)$ := • pick $a \in S$ and compute $S_L, S_a, S_R$ } can split in linear time
• if $k \leq |S_L|$ then Select $(S_L, k)$
• if $|S_L| < k \leq |S_L| + |S_a|$ then return $a$
• if $|S_L| + |S_a| < k$ then Select $(S_R, k - |S_L| - |S_a|)$

We go from list size $|S|$ to list size $\max\{|S_L|, |S_R|\}$.
How to pick $a$?

Bad case is if $a$ is always the largest (or smallest) element of the current set:
$$O(n) + O(n-1) + O(n-2) + \dots = O(n^2)$$
Good case is if $a$ always splits $S$ roughly in half: $|S_L|, |S_R| \approx |S|/2$
In this case we get the recurrence $T(n) = T(n/2) + O(n) = O(n)$

Problem: picking $a \in S$ as above requires... finding median!

**Idea: pick $a \in S$ at random!**

We say that $a$ is good if $a$ is between 25th and 75th percentiles: 


When $a$ is good, the new set shrinks by a constant factor:

$$\max\{|S_L|, |S_R|\} \le \frac{3}{4} \cdot |S|.$$

There are many good elements: $\Pr_{a \in S}[a \text{ is good}] = \frac{1}{2}$.

So in expectation it takes **2 tries to get a good $a$.**

The expected running time is:

$$
\begin{aligned}
\mathbb{E}\,T(n) &\le \mathbb{E}\,T\left(\tfrac{3}{4} \cdot n\right) + \mathbb{E}[\text{time to find good } a] \cdot \overbrace{O(n)}^{\text{time to check if } a \text{ is good}} + \overbrace{O(n)}^{\text{time to split } S \text{ according to the chosen } a} \\
&= \mathbb{E}\,T\left(\tfrac{3}{4} \cdot n\right) + 2 \cdot O(n) \\
&= \mathbb{E}\,T\left(\tfrac{3}{4} \cdot n\right) + O(n) \\
&= O(n)
\end{aligned}
$$

with $a \in S$ chosen at random until it is good

On any input $S$ and integer $k$, Select$(S,k)$ returns the correct answer in a number of steps that is $O(n)$ in expectation.