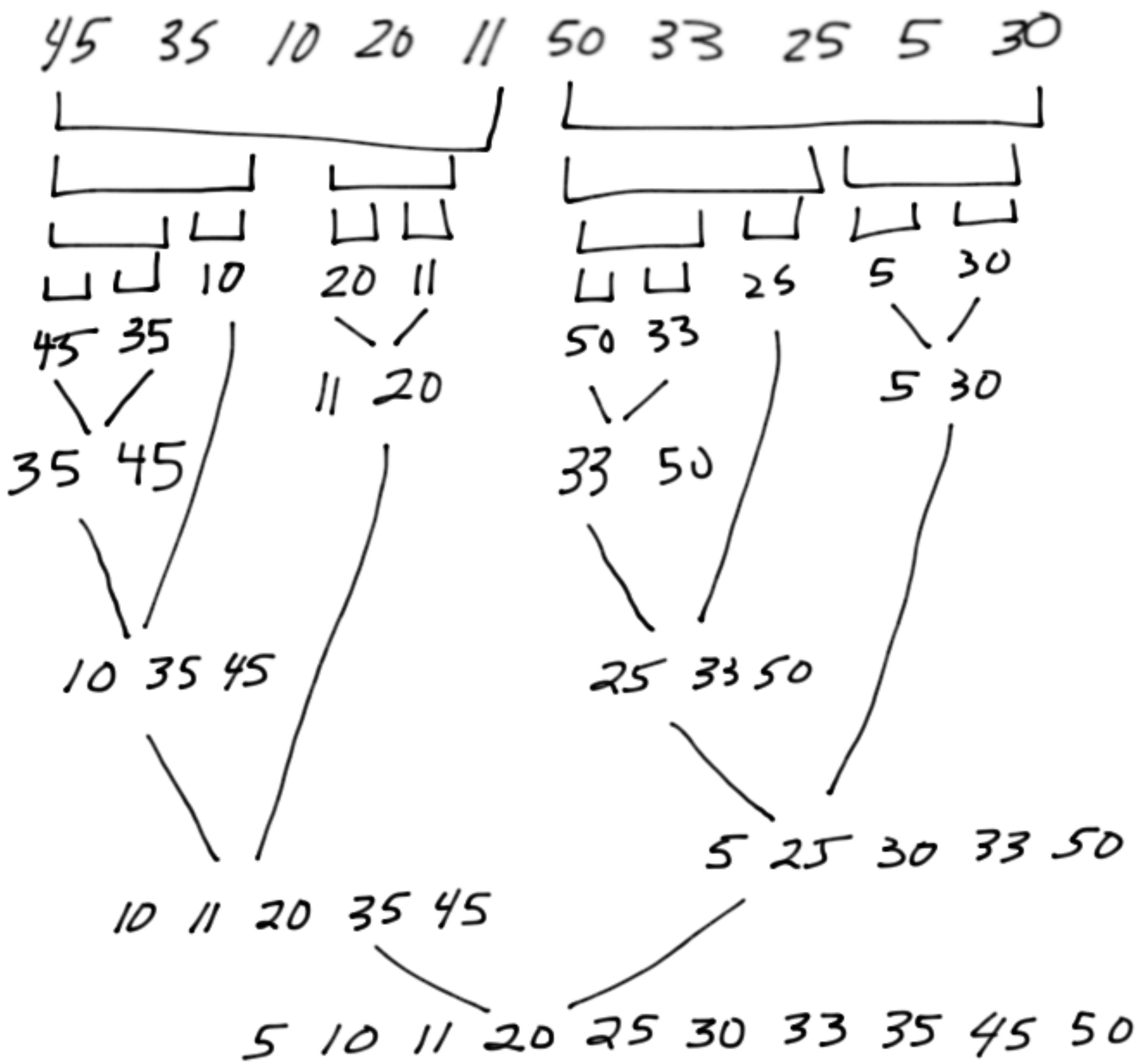
Today - Lecture 18 - CS163

1) Topic #13 - Recursive Sorting Algs
   - Merge Sort
   - Quicksort
2) Summarize the performance of each sorting algorithm
3) Review efficiency of the various data structures used for table abstractions

Announcements:

* Practice Trees, Graphs, Efficiency, and $\boxed{\text{MOST}}$ importantly $\boxed{\text{recursion}}$

Merge Sort



45 35 10 20 11 50 33 25 5 30
10
20 11
25
5 30
45 35
50 33
11 20
5 30
35 45
33 50
10 35 45
25 33 50
5 25 30 33 50
10 11 20 35 45
5 10 11 20 25 30 33 35 45 50

Quick Sort

~~45~~ 35 10 20 11 50 ~~33~~ 25 5 30

$\boxed{33}$ $\underbrace{35}_{R}$ 45

PV

* Swap 10 with the leftmost item in the Right partition (35)

$\boxed{33}$ $\underbrace{10}_{L}$ | $\underbrace{35}_{R}$ 20

$\underbrace{10 \quad 20}_{L}$ | $\underbrace{35}_{R}$ 11

$\boxed{33}$ $\underbrace{10 \quad 20 \quad 11}_{L}$ | $\underbrace{35 \quad 50 \quad 45}_{R}$ 25

$\boxed{33}$ $\underbrace{10 \quad 20 \quad 11 \quad 25}_{L}$ | $\underbrace{50 \quad 45 \quad 35}_{R}$ 5

$\boxed{33}$ $\underbrace{10 \quad 20 \quad 11 \quad 25 \quad 5}_{L}$ | $\underbrace{45 \quad 35 \quad 50}_{R}$ 30

$\boxed{33}$ $\underbrace{10 \quad 20 \quad 11 \quad 25 \quad 5 \quad 30}_{L}$ | $\underbrace{35 \quad 50 \quad 45}_{R}$

$\boxed{33}\ \underbrace{10\ \ 20\ \ 11\ \ 25\ \ 5\ \ 30}_{L}\ \Big|\ \underbrace{35\ \ 50\ \ 45}_{R}$

Swap 33 with the Rightmost item in the Left partition

$\underbrace{30\ \ 10\ \ 20\ \ 11\ \ 25\ \ 5}\ \boxed{33}\ \ 35\ \underbrace{50\ \ 45}$

$\boxed{20}\ \underbrace{10}_{L}\ \Big|\ \underbrace{30}_{R}\ \ 11\ \ 25\ \ 5$

$\underbrace{10\ \ 11}_{L}\ \Big|\ \underbrace{30\ \ 25}_{R}\ \ 5$

$\boxed{20}\ \underbrace{10\ \ 11\ \ 5}_{L}\ \Big|\ \underbrace{25\ \ 30}_{R}$

$\underbrace{5\ \ 10\ \ 11}\ \boxed{20}\ \underbrace{25\ \ 30}\ \boxed{33}\ \underbrace{35\ \ 50\ \ 45}$

$\boxed{35}\ \underbrace{50\ \ 45}_{R}$

$\boxed{10}\ \underbrace{5}_{L}\ \Big|\ \underbrace{11}_{R}$

$45\ \boxed{50}$

$5\ \boxed{10}\ 11\ \boxed{20}\ \underbrace{25\ \ 30}$

$\boxed{25}\ \underbrace{30}_{R}$

$5\ \boxed{10}\ 11\ \boxed{20}\ \boxed{25}\ 30\ \boxed{33}\ \boxed{35}\ 45\ \boxed{50}$

Examine Efficiency of Algorithms

	Insertion sort	Selection sort	Exchange sort	Radix sort
compares	best O(N) worst O(N^2)	O(N^2)	best O(N) worst O(N^2)	None (but requires duplicate memory for at least pointers)
moves	best None worst O(N^2)	best None worst O(N)	best None worst O(N^2)	O(N*keylength)

	Mergesort	Quicksort
Compares	$O(N * \log_2 N)$	$O(N * \log_2 N) \longleftrightarrow (w)\ O(N^2)$
moves	$O(N * \log_2 N)$	$\emptyset \rightarrow O(N * \log_2 N)$

Understanding Efficiency for $\boxed{\text{Table}}$ ADTs

	Add	Remove	Search	Display
Sorted Array	+ Binary Search O(logN) - Shifting O(N) for each	+ Binary Search O(logN) - Shifting O(N)	**+ Binary Search O(logN)**	**+ Displays in Sorted Order automatically O(N)**
Unsorted Array	**+ Direct Access O(1) + No Shifting - Memory**	- Sequential Search O(N) - Shifting O(N)	- Sequential Search O(N)	- Must implement a sorting algorithm
Sorted LLL	- Sequential Search O(N) + No Shifting + Flexibility with Memory	- Sequential Search O(N) + No Shifting + Can Stop early if there is no match	- Sequential Search O(N)	**+ Supported O(N)**
Unsorted LLL	**+ Direct Access O(1) + No Shifting + Flexibility with Memory**	- Sequential Search O(N) + No Shifting + Flexibility with Memory	- Sequential Search O(N)	- Must implement a sorting algorithm

	Add	Remove	Search	Display
Sorted Array	+ Binary Search O(logN) - Shifting O(N) for each - Memory	+ Binary Search O(logN) - Shifting O(N)	**+ Binary Search O(logN)**	**+ Displays in Sorted Order automatically O(N)**
Sorted LLL	- Sequential Search O(N) + No Shifting + Flexibility with Memory	- Sequential Search O(N) + No Shifting + Can Stop early if there is no match	- Sequential Search O(N)	**+ Supported O(N)**
Hash Table using Chaining	**+ Instantaneous O(1) + Direct Access + Flexibility with Memory**	**+ Instantaneous O(1) + Direct Access + Flexibility with Memory**	**+ Instantaneous O(1) + Direct Access + Flexibility with Memory**	- Not Available (would need to use an alternate data structure)

	Add	Remove	Search	Display
BST	+ Binary Search best O(logN) worst O(N)	+ Binary Search best O(logN) worst O(N)	**+ Binary Search best O(logN) worst O(N)**	**+ Displays in Sorted Order automatically O(N)**
Balanced Tree	+ Binary Search O(logN)	+ Binary Search O(logN)	+ Binary Search O(logN)	**+ Displays in Sorted Order automatically O(N)**
Hash Table using Chaining	**+ Instantaneous O(1) + Direct Access + Flexibility with Memory**	**+ Instantaneous O(1) + Direct Access + Flexibility with Memory**	**+ Instantaneous O(1) + Direct Access + Flexibility with Memory**	- Not Available (would need to use an alternate data structure)