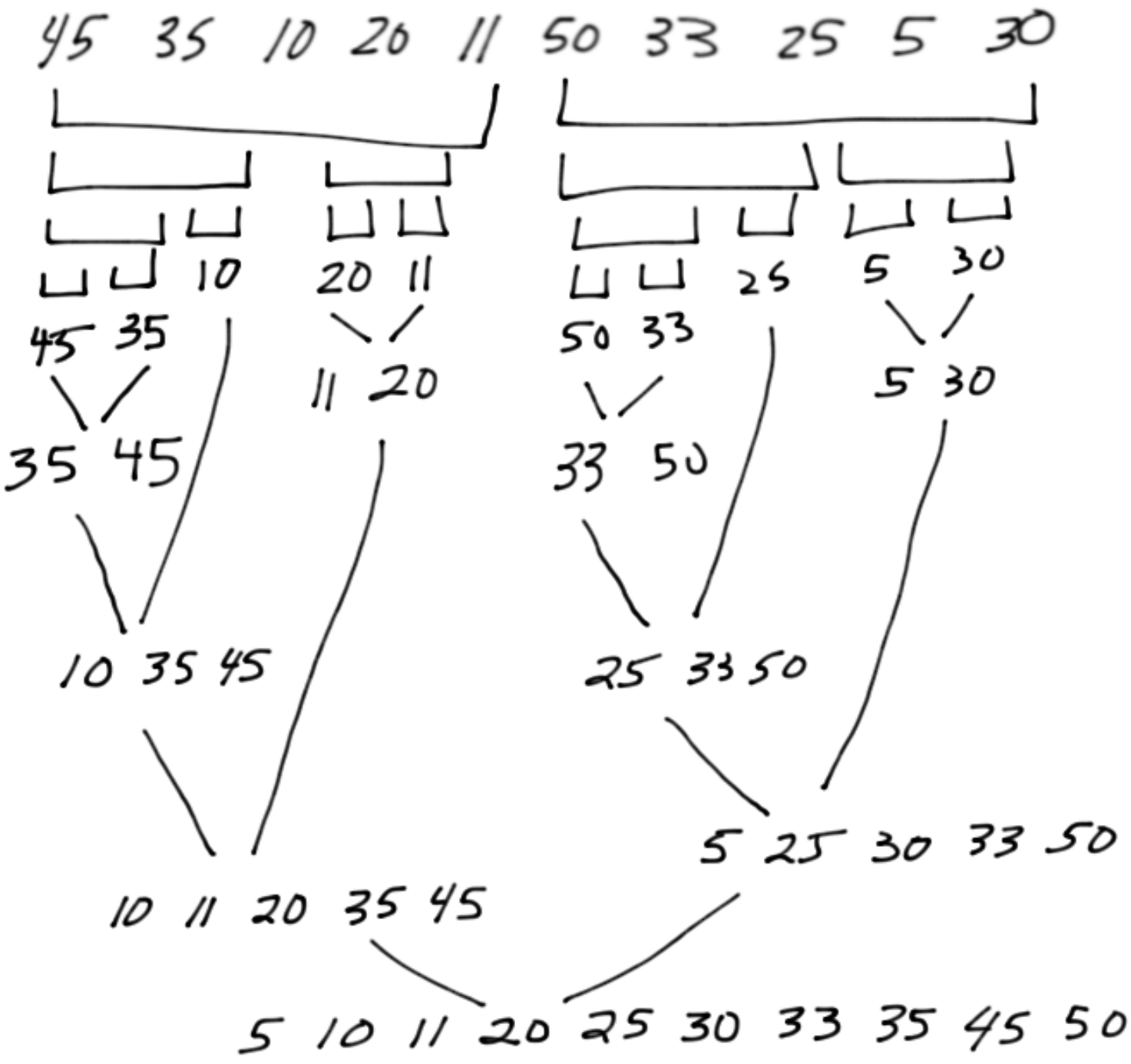
Today - Lecture 18 - CS163

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 MOST importantly recursion

Merge Sort


45 35 10 20 11 50 33 25 5 30
45 35 10
20 11
50 33
25
5 30
35 45
11 20
33 50
5 30
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 ~~37~~ 45 25 5 30

[33] — R

PV

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

[33] 10 (L) | 35 (R) 20

10 20 (L) | 35 (R) 11

[33] 10 20 11 (L) | 35 50 45 25 (R)

[33] 10 20 11 25 (L) | 50 45 35 5 (R)

[33] 10 20 11 25 5 (L) | 45 35 50 30 (R)

[33] 10 20 11 25 5 30 (L) | 35 50 45 (R)

[33] 10 20 11 25 5 30 | 35 50 45
L R

Swap 33 with the Rightmost item in the Left partition

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

[20] 10 | 30 11 25 5
L R

10 11 | 30 25 5
L R

[20] 10 11 5 | 25 30
L R

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

[10] 5 | 11
L R

[35] 50 45
R

45 [50]

5 [10] 11 [20] 25 30

[25] 30
R

5 [10] 11 [20] [25] 30 [33] [35] 45 [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

compares $O(N * \log_2 N)$

moves $O(N * \log_2 N)$

Quicksort

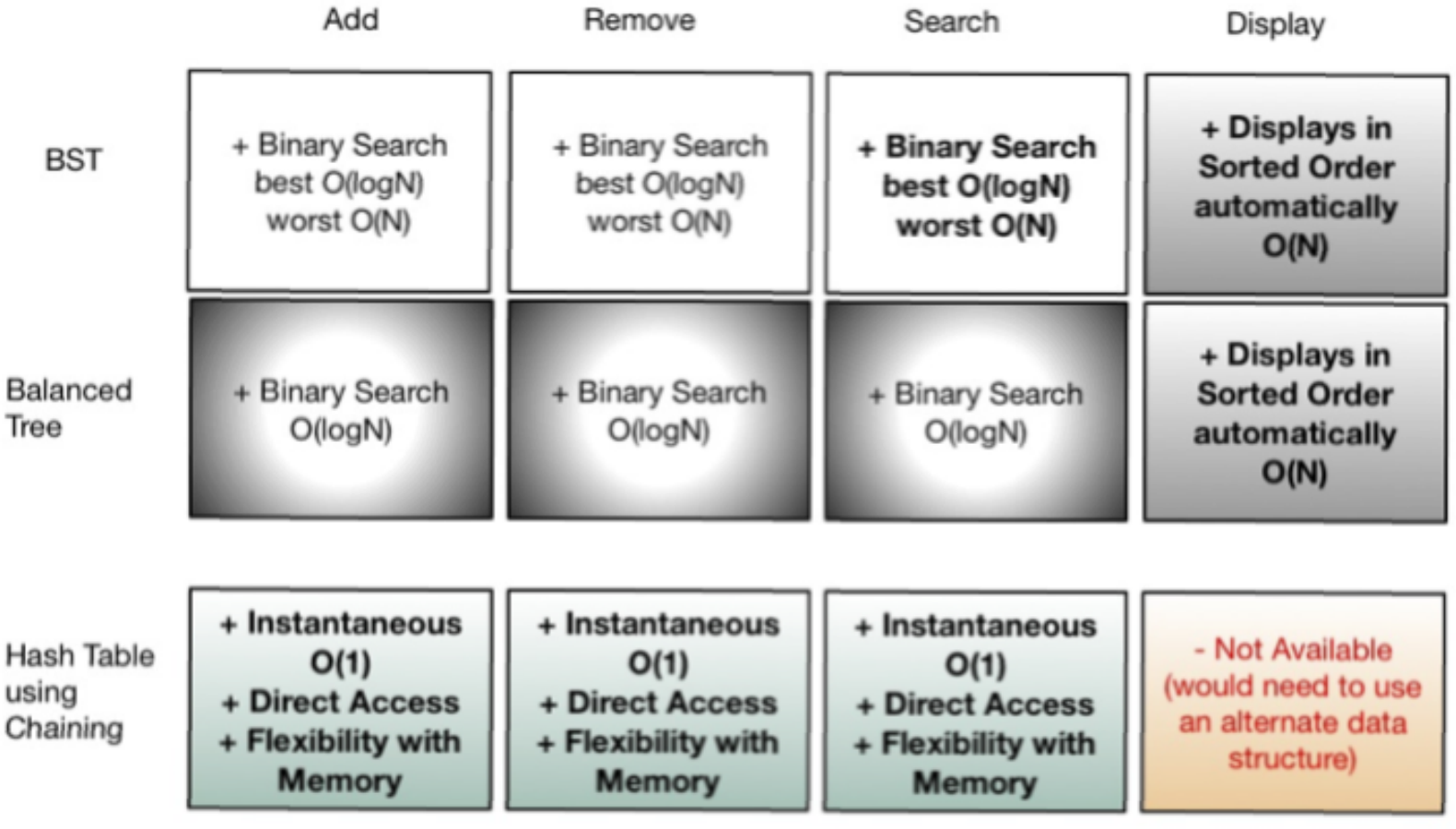
compares $O(N * \log_2 N) \longleftrightarrow$ (w) $O(N^2)$

moves $\emptyset \rightarrow O(N * \log_2 N)$

Understanding Efficiency for 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)