Lecture 23: Course Review

High Level

- What is a problem? What is an algorithm? (R01)
- Analyzing running time: **How to count?**
  - **Asymptotics** (R01)
  - **Recurrences** (R02)
  - **Model of computation**: Word-RAM (R01), Comparison (R07)
- How to solve an algorithms problem
  - Reduce to a problem you know how to solve
    - Use a data structure you know (e.g. **search**)
    - Use an algorithm you know (e.g. **sort**)
  - Design a new algorithm (harder, mostly in 6.046)
    - Brute Force
    - Decrease & Conquer
    - Divide & Conquer
    - Dynamic Programming
    - (Greedy/Incremental)
  - Some problems not solvable efficiently! (Complexity)

Data Structure

Reduce your problem to using a data structure storing a set of items, supporting certain search and dynamic operations efficiently. You should know **how** each of these data structures implement the operations they support, as well as be able to **choose** the right data structure for a given task. (R04-R08)

Sequence Interface Data Structure Implementation	Operation, Worst Case $O(\cdot)$					Space
	Static		Dynamic			
	`at(i)` `set(i,x)`	`left()` `right()`	`insert_at(i,x)` `delete_at(i)`	`insert_left(x)` `delete_left()`	`insert_right(x)` `delete_right()`	$\times n$
Array	1	1	n	n	n	~ 1
Linked List	n	1	n	1	1	~ 3
Dynamic Array	1	1	n	n	$1_{(a)}$	~ 4

Set Interface	Operation, Worst Case $O(\cdot)$						Space
	Static	Dynamic (D)		Order (O)		D + O	
Data Structure Implementation	`find(k)`	`insert(x)`	`delete(k)`	`find_next(k)`	`find_max()`	`delete_max()`	$\sim \times n$
Unsorted Array	n	n	n	n	n	n	1
Linked List	n	1	n	n	n	n	3
Dynamic Array	n	$1_{(a)}$	n	n	n	n	4
Sorted Array	$\lg n$	n	n	$\lg n$	1	n	1
Max-Heap	n	$\lg n_{(a)}$	n	n	1	$\lg n$	1
Balanced BST (AVL)	$\lg n$	$\lg n$	$\lg n$	$\lg n$	(1)	$\lg n$	5
Direct Access	1	1	1	u	u	u	u/n
Hash Table	$1_{(e)}$	$1_{(e,a)}$	$1_{(e,a)}$	n	n	n	4

Algorithm

Reduce your problem to a classic problem you already know how to solve using known algorithms. You should know **how** each of these algorithms can be implemented to solve each problem, as well as be able to **choose** the right algorithm for a given task.

- Sort n integers (R02-R08)

Algorithm	Time $O(\cdot)$	In-place?	Stable?	Comments
Insertion Sort	n^2	Y	Y	$O(nk)$ for k-proximate
Selection Sort	n^2	Y	N	$O(n)$ swaps
Merge Sort	$n \lg n$	N	Y	stable, optimal comparison
Heap Sort	$n \lg n$	Y	N	low space, optimal comparison
AVL Sort	$n \lg n$	N	Y	good if also need dynamic
Counting Sort	n	N	Y	$u = \Theta(n)$ is domain of possible keys
Radix Sort	cn	N	Y	$u = \Theta(n^c)$ is domain of possible keys

- Graph exploration, count connected components
- Topological sort, Cycle detection, negative weight cycle detection
- Single Source Shortest Paths (SSSP), Relaxation framework
- All Pairs Shortest Paths (APSP), SSSP $|V|$ times, or Johnson's

Restrictions		SSSP Algorithm		
Graph	Weights	Name	Running Time $O(\cdot)$	How it works
DAG	Any	DAG Relaxation	$\lvert V\rvert + \lvert E\rvert$	Relax in topological order
General	Unweighted	BFS	$\lvert V\rvert + \lvert E\rvert$	Relax level by level
General	Nonnegative	Dijkstra	$\lvert V\rvert \log \lvert V\rvert + \lvert E\rvert$	Relax in priority order
General	Any	Bellman-Ford	$\lvert V\rvert \lvert E\rvert$	Relax in $\lvert V\rvert$ rounds

Recursive Algorithm Paradigms

Class	Subproblem Dependency Graph
Brute Force	Star
Decrease & Conquer	Chain
Divide & Conquer	Tree
Dynamic Programming	DAG (Overlapping subproblems)

Dynamic Programming Steps (SR. BST)

1. Define **Subproblems** subproblem $x \in X$
    - Describe the meaning of a subproblem **in words**, in terms of parameters
    - Often subsets of input: prefixes, suffixes, contiguous subsequences
    - Often record partial state: add subproblems by incrementing some auxiliary variables
2. **Relate** Subproblems $x(i) = f(x(j), \ldots)$ for one or more $j < i$
    - State topological order to argue relations are acyclic and form a DAG
3. Identify **Base** Cases
    - State solutions for all reachable independent subproblems
4. Compute **Solution** from Subproblems
    - Compute subproblems via top-down memoized recursion or bottom-up
    - State how to compute solution from subproblems (possibly via parent pointers)
5. Analyze Running **Time**
    - $\sum_{x \in X} \text{work}(x)$, or if $\text{work}(x) = W$ for all $x \in X$, then $|X| \times W$