COS 485 Crib Notes

I. FORMULA YOU SHOULD KNOW ALREADY

$$\sum_{x=0}^{n} x = 1 + 2 + ... + n - 1 + n = (n+1)\left(\frac{n}{2}\right)$$

$$\sum_{x=0}^{\infty} \left(\frac{1}{2}\right)^x = \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}.... = 2$$

$$\sum_{i=0}^{\lceil \lg n \rceil} \frac{n}{2^i} = \frac{n}{1} + \frac{n}{2} + \frac{n}{4} + \frac{n}{8}.... = 2n - 1$$

$$\sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}$$

$$\frac{n(n+1)(2n+1)}{6} = 1^2 + 2^2 + 3^3 + ... + (n-1)^2 + n^2$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\lg x = k, \text{ where } 2^k = x$$

$$\lg(a^b + = b \lg a$$

$$a^{\lg b} = b^{\lg a}$$

II. TIME COMPLEXITY

Just a quick refresher. Featured below is an analysis for Selection Sort.

Inner loop iterations:								Total
0	1	2	3	4	5	...	n-1	n-1
	1	2	3	4	5	...	n-1	n-2
		2	3	4	5	...	n-1	n-3
			3	4	5	...	n-1	n-4
				4	5	...	n-1	n-5
					5	...	n-1	n-6
						...	n-1	.
						...	n-1	.
						...	n-1	.
						...	n-1	3
						...	n-1	2
						...	n-1	1

This is is $T(n) = \frac{n^2}{2} - \frac{n}{2}$, or $O(n^2)$.

III. ORDER NOTATION

Definition III.1. For a given complexity function $f(n)$, $O(f(n))$ is the set of complexity functions $g(n)$ for which there exists some real positive constant c and some non-negative integer N s.t. $\forall n \geq N$, $g(n) \leq c \times f(n)$
This means, for example, $\{n^2, 5n + 2, n \lg n ...\} \in O(n^2)$

Example III.1. Show that $n \in O(n^2)$.
By definition, $n \leq c \cdot n^2$. This is true starting at $c = 1$, $N = 1$: $1 \leq 1 \cdot n$.

Definition III.2. $\Omega(f(n))$ is the set of complexity functions $g(n)$ for which there exists some real positive constant c and some non-negative integer N s.t. $\forall n \geq N$, $g(n) \geq c \times f(n)$
This means, for example, $\{n^2, n^3, 2^n ...\} \in \Omega(n^2)$

Definition III.3. $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$, meaning it is the set of complexity functions $g(n)$ where there exists some real positive constants c, d and some non-negative integer N s.t. $\forall n \geq N$, $c \times f(n) \leq g(n) \leq d \times f(n)$
This means, for example, $\{n^2, 4n^2, n^2 + n \lg n ...\} \in \Theta(n^2)$

Definition III.4. $o(f(n))$ is the set of complexity functions $g(n)$ satisfying the following: $\forall c \in \mathbb{R}^+, \exists N \in Z^+$ s.t. $\forall n \geq N$, $g(n) \leq c \times f(n)$.
This means, for example, $\{n, 5n + 2, n \lg n ...\} \in O(n^2)$.

Example III.2. Show that $n \in o(n^2)$.
By definition, $n \leq c \cdot n^2$. To find a working expression that conforms to our definition, we note $\frac{1}{c} \leq n$, meaning this holds for any $N \geq \frac{1}{c}$.

Example III.3. Show that $n \notin o(5n)$.
Use a proof by contradiction. Let $c = \frac{1}{6}$. If $n \in o(5n)$, there must exist some N such that, for $n \geq N$, $n \leq \frac{1}{6}5n$. This reduces to $1 \leq \frac{5}{6}$, which cannot be true.

IV. DIVIDE AND CONQUER

Binary Search Recurrence equation: We halve our problem every step, and make one comparison on each level. This is $W(n) = W\left(\frac{n}{2}\right) + 1$ for $n > 1$, n a power of 2. $W(1) = 1$.
The solution is $\lg n + 1$ if n is a power of 2, or $\lfloor \lg n \rfloor + 1$ otherwise.

Mergesort Recurrence equation: We split our array in half, mergesort both halves, and then merge the results together. $W(n) = 1 + 2W\left(\frac{n}{2}\right) + n - 1$. $W(1) = 1$
This is $(\lceil \lg n \rceil + 1)n$, or $\Theta(n \lg n)$.

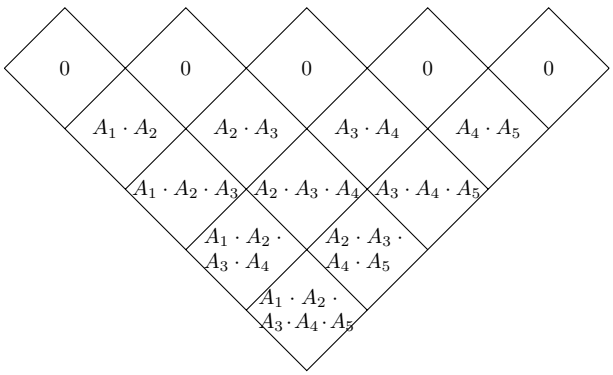
Quicksort: *(Best-case)*
Recurrence equation: We partition our array on a pivot, quicksort both partitions, and we're done. In the best case, our partition is separates the array such that it is divided in half. $B(n) = n + 2B\left(\frac{n}{2}\right) + 0$. $B(1) = 1$
(Worst-case)
Array is sorted in nondecreasing order. We sort the left subarray, then the right, then partition with a cost of: $W(n) = W(0) + W(n-1) + n - 1 = W(n-1) + n - 1$. $W(1) = 1$
As we keep reducing our problem size by one from $n-1$ to $n-2$, $n-3$..., this is $\frac{n(n-1)}{2}$.

V. DYNAMIC PROGRAMMING

Floyd's Algorithm: I know this one by heart, (un)fortunately. This uses the matrices $D^(0)...D^(n)$ to provide n^2 answers in n^3 time.

Matrix Chain Multiplication: Base form: $A_1[d_0 \times d_1] \cdot A_2[d_1 \times d_2] => R[d_0 \times d_2]$.
Not all orders of multiplication are equivalent. Consider the following:
$A_1[50\times10]\cdot A_2[10\times30]\cdot A_2[30\times20]$
$(A_1 \cdot A_2)A_3 = 50\times10\times30 + 50\times30\times20 = 15000 + 30000$
$A_1(A_2 \cdot A_3) = 10\times30\times20 + 50\times10\times20 = 6000 + 10000$

We can use dynamic programming in an n^3 algorithm to find the minimum number of multiplications for any given number of matrices. Shown is the setup for matrices $A_1...A_5$.



Sequence Alignment: You know how this works. Pull from the bottom diagonal. Assume a penalty of 1 for a mismatch and a penalty of 2 for a gap, use the dynamic programming algorithm to find an optimal alignment of the following sequences.
As an example from the last homework:

C C G G G T T A C C A
G G A G T T C A

Show both the table and the optimal alignment.

	j	0	1	2	3	4	5	6	7	8	9	10	11
i		C	C	G	G	G	T	T	A	C	C	A	_
0	G	**8**	**6**	5	4	6	7	8	9	11	12	14	16
1	G	9	7	**5**	5	4	6	6	7	9	10	12	14
2	A	11	9	**7**	5	5	4	5	5	7	8	10	12
3	G	12	10	8	**6**	**4**	4	3	4	5	6	8	10
4	T	14	12	10	8	6	**4**	3	2	3	4	6	8
5	T	16	14	12	10	8	6	**4**	3	1	2	4	6
6	C	18	16	14	12	10	8	6	**4**	**2**	**0**	2	4
7	A	20	18	16	14	12	10	8	6	4	2	**0**	2
8	_	22	20	18	16	14	12	10	8	6	4	2	**0**

VI. MINIMUM SPANNING TREES AND GREEDY ALGORITHMS

Spanning Tree For a graph G, it's spanning tree is a connected subgraph that contains all of G's verticies and is a tree.

Minimum Spanning Tree For a graph G, it is a spanning tree of minimum weight.

Prim's Algorithm Builds a minimum spanning tree vertex by vertex using an arbitrary starting vertex. Each step it will add the cheapest possible connection from the current vertex to the next. The algorithm is $\Theta(n^2)$.

Kruskal's Algorithm Builds a minimum spanning tree edge by edge, taking the cheapest edge available unless it causes a cycle. The steps to implement are as follows:

1. Sort the edges in time $\Theta(E \lg(E))$.
2. Loop through the list of edges for a candidate edge. In worst case, it is $\Theta(E)$.

In the worst case, though, every vertex is connected to every other vertex. This means $E \in \Theta(V^2)$. Thus, the worst case complexity is $\Theta((V^2 \lg(E))$.

From the above descriptions, we can quickly conclude that Kruskal's Algorithm is better for sparse graphs, and Prim's Algorithm is better for dense graphs.

Dijkstra's Algorithm Builds a shortest path from a vertex v_0 to a destination v_n vertex by vertex using an arbitrary starting vertex. Each step it will add the cheapest possible connection from the tree to another vertex, *using the available connections already made in order to reduce possible costs*. The algorithm is $\Theta(n^2)$.

VII. BACKTRACKING

Note that the book explicitly states that backtracking is a modification of DFS. We just DFS a lot.

DFS DFS is a $\Theta(V + E)$ traversal of any graph. Pseudocode is as follows.

Backtracking Create a state space tree and traverse it. After determining that a node can only lead to death and destruction, go back to the parent and try again. The only way we can determine this is either by running out of options to come up with a solution, or using an upper bound to state a solution will get no worse. These algorithms are pretty terrible; the $n - queens$ problem uses a $\Theta(n^n)$ solution. The sum-of-subsets problem uses an upper bound and is a $T_n \approx 2^{n+1} - 1$ algorithm

VIII. BRANCH-AND-BOUND

Note that the book explicitly states that backtracking is a modification of *breadth*-first search. We just BFS a lot.

BFS BFS is also a $\Theta(V + E)$ traversal of any graph. Pseudocode is as follows.

Branch-and-Bound Just like Backtracking, with the modification that we visit the children of the current node only if its bound (an estimate guessing what will follow the current node) indicates that that path will lead to a better solution than the current best solution. That's the only effective difference. This is a **best**-first search.