Complexity Proofs

1. Insertion into a hash table that uses chaining to resolve collisions takes $O(1)$ time.
2. Deletion in a hash table that uses chaining to resolve collisions takes $O(1)$ time.
3. Lookup in a hash table that uses chaining to resolve collisions takes $O(1)$ time.
4. Insertion into a hash table that uses open addressing takes $O(1)$ time.
5. Deletion in a hash table that uses open addressing takes $O(1)$ time.
6. Lookup in a hash table that uses open addressing takes $O(1)$ time.
7. Inserting a node x into a binary search tree takes $O(\log n)$ time.
8. Deleting a node x from a binary search tree takes $O(\log n)$ time.
9. Searching for a node x in a binary search tree takes $O(\log n)$ time.
10. Inserting a node x into a heap takes $O(\log n)$ time.
11. Deleting a node x from a heap takes $O(\log n)$ time.
12. Searching for a node x in a heap takes $O(\log n)$ time.
13. Inserting a node x into an AVL tree takes $O(\log n)$ time.
14. Deleting a node x from an AVL tree takes $O(\log n)$ time.
15. Searching for a node x in an AVL tree takes $O(\log n)$ time.
16. Inserting a node x into a red-black tree takes $O(\log n)$ time.
17. Deleting a node x from a red-black tree takes $O(\log n)$ time.
18. Searching for a node x in a red-black tree takes $O(\log n)$ time.
19. Solving the knapsack problem with dynamic programming takes time $O(n^2)$.
20. Solving the subset sum problem with dynamic programming takes time $O(n \cdot s)$ time and space where s is the sum.
21. Solving the $2-$SAT problem with dynamic programming takes times $O(n + m)$.
22. Breadth-first search takes time $O(V + E)$ where V is the number of vertices and E is the number of edges.
23. Depth-first search takes time $O(V + E)$ where V is the number of vertices and E is the number of edges.
24. $2-$coloring of a graph times time $O(V + E)$ where V is the number of vertices and E is the number of edges.
25. Adding two sparse matrices takes time $O(k)$ where k is the greater number of non-zero entries among the two matrices.
26. Multiplying two sparse matrices takes time $O(c_1 m + c_2 n)$ where c_1 and c_2 are the number of columns and n and m are the number of non-zero entries of the two respective matrices.

Correctness Proofs

These claims are taken from the course textbooks in the corresponding sections. You may use the textbooks as references when providing proofs.

1. Finding the Closest Pair of Points [KT 225]: If there exists $q \in Q$ and $r \in R$ for which $d(q, r) < \delta$ then each of q and r lies within a distance of δ of L.

2. Sequence Alignment in Linear Space [KT 284]: The length of the shortest corner-to-corner path in G_{XY} that passes through (i, j) is $f(i, j) + g(i, j)$.

3. Sequence Alignment in Linear Space [KT 284]: Let k be any number in $\{0, \ldots, n\}$ and let q be an index that minimizes the quantity $f(q, k) + g(q, k)$. Then there is a corner-to-corner path of minimum length that passes through the node (q, k).

4. Shortest Paths in a Graph [KT 290]: If G has no negative cycles, then there is a shortest path from s to t that is simple (i.e., does not repeat nodes), and hence has at most $n - 1$ edges.

5. Negative Cycles in a Graph [KT 301]: There is no negative cycle with a path to t if and only if $\text{OPT}(n, v) = \text{OPT}(n-1, v)$ for all nodes v.

6. The Heapsort Algorithm [CLRS 159]: Argue the correctness of HEAPSORT using the following loop invariant: At the start of each iteration of the for loop of the subarray $A[1..i]$ is a max-heap containing the i smallest elements of $A[1..i]$, and the subarray $A[i + 1..n]$ contains the $n - i$ largest elements of $A[1..n]$, sorted.

7. Perfect Hashing [CLRS 277]: Suppose that we store n keys in a hash table of size $m = n$ using a hash function h randomly chosen from a universal class of hash functions. Then, we have

$$\mathbb{E}\left[\sum_{j=0}^{m-1} n_j^2\right] < 2n$$

   Where n_j is the number of keys hashing to slot j.

8. Red-Black Trees [CLRS 308]: A red-black tree with n internal nodes has height at most $2\log(n + 1)$.

9. Red-Black Trees [CLRS 308]: Let f be an attribute that augments a red-black tree T of n nodes, and suppose that the value of f for each node x depends on only the information in nodes x, $x.left$, and $x.right$, possibly including $x.left.f$ and $x.right.f$. Then, we can maintain the values of f in all nodes of T during insertion and deletion without asymptotically affecting the $O(\log n)$ performance of these operations.

10. Interval Trees [CLRS 348]: Any execution of INTERVAL-SEARCH(T, i) either returns a node whose interval overlaps i, or it returns $T.nil$ and the tree T contains no node whose interval overlaps i.

11. Huffman Codes [CLRS 428]: Let C be an alphabet in which each character $c \in C$ has frequency $c.freq$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

12. Huffman Codes [CLRS 428]: Let C be a given alphabet with frequency $c.freq$ defined for each character $c \in C$. Let x and y be two characters in C with minimum frequency. Let C' be the alphabet C with the characters x and y removed and a new character added, so that $C' = C - \{x, y\} \cup \{z\}$. Define f for C' as for C, except that $z.freq = x.freq + y.freq$. Let T' be any tree representing an optimal prefix code for the alphabet C' . Then the tree T , obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix code for the alphabet C.

13. Matroids [CLRS 437]: If $M = (S, I)$ is a weighted matroid with weight function w, then GREEDY(M, w) returns an optimal subset.

14. Matroids [CLRS 437]: If S is a set of unit-time tasks with deadlines, and I is the set of all independent sets of tasks, then the corresponding system (S, I) is matroid.

15. Breadth-first Search [CLRS 594]: Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

16. Breadth-first Search [CLRS 594]: Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$ where v_1 is the head of Q and v_r is the tail. Then, $v_r.d \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}.d$ for $i = 1, 2, ..., r - 1$.

17. Depth-first Search [CLRS 603]: In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

18. Topological Sort [CLRS 612]: A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

19. Topological Sort [CLRS 612]: TOPOLOGICAL-SORT produces a topological sort of the directed acyclic graph provided as its input.

20. Strongly Connected Components [CLRS 615]: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \to u'$. Then G cannot also contain a path $v \to v'$.

21. Strongly Connected Components [CLRS 615]: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$. Suppose there is an edge $(u, v) \in E$, where $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.

22. Strongly Connected Components [CLRS 615]: The STRONGLY-CONNECTED-COMPONENTS procedure correctly computes the strongly connected components of the directed graph G provided as its input.

23. Minimum Spanning Trees [CLRS 624]: Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for A.

24. The Bellman-Ford Algorithm [CLRS 651]: Let BELLMAN-FORD be run on a weighted, directed graph $G = (V, E)$ with source s and weight function $w : E \to \mathbb{R}$. If G contains no negative-weight cycles that are reachable from s then the algorithm returns TRUE, we have $v.d = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest-paths tree rooted at s. If G does contain a negative-weight cycle reachable from s, then the algorithm returns FALSE.

25. Dijkstra's Algorithm [CLRS 658]: Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with non-negative weight function w and source s, terminates with $u.d = \delta(s, u)$ for all vertices $u \in V$.