Interpret $\log n$ to any base you find convenient.

1. (12 points) 

Follow the example, $f(n) = n^3$: Left \( C \) Right \( Z \), to locate the following functions in most accurate parts of the left and right diagrams above.

- $f_1(n) = \frac{n^3}{\log n}$, Left \( \_ \) Right \( \_ \)
- $f_2(n) = n^2(\log n)^2$, Left \( \_ \) Right \( \_ \)
- $f_3(n) = \begin{cases} n^3, & \text{if } n \text{ even} \\ n^2 \log n, & \text{if } n \text{ odd} \end{cases}$, Left \( \_ \) Right \( \_ \)
- $f_4(n) = \frac{n^2}{\log n}$, Left \( \_ \) Right \( \_ \)
- $f_5(n) = n^2 \sqrt{n} \log n$, Left \( \_ \) Right \( \_ \)
- $f_6(n) = \begin{cases} n^3, & n \leq 100 \\ n^2 \log n, & \text{otherwise} \end{cases}$, Left \( \_ \) Right \( \_ \)

2. (12 points) What does the master template tell us about the following recurrences?

(a) $T(n) = 2T(\lfloor n/4 \rfloor) + \sqrt{n}$

(b) $T(n) = 3T(\lfloor n/4 \rfloor) + \sqrt{n}$

(c) $T(n) = 2T(\lfloor n/4 \rfloor) + \sqrt{n}/\log(n)$

3. (3 points) Suppose in the recurrence form $T(n) = aT(n/b) + f(n)$, we have $f(n) = \Omega(n^{\log_b a})$. What further condition is necessary to conclude via the master template that $T(n) = \Theta(f(n))$?
4. (8 points) One of the homework problems involved finding the diameter of a tree. The tree is not necessarily binary.

(a) What is the definition of a tree diameter?

(b) What recursive relationship exists between the diameter of a tree and the diameters of root’s subtrees?

5. (12 points) In the depth-first search algorithm of a directed graph $G = (V,E)$, suppose vertex $v$ is examined on the adjacency list of vertex $u$. What are the possible edge types for $(u,v) \in E$ if the color of $v$ is

(a) black

(b) gray

(c) white

6. (3 points) In the depth-first search algorithm on a directed graph, why is the highest finish time assigned to the first vertex discovered in some strongly connected component with no incoming edges from other components?

7. (8 points) One of the homework problems involved counting the number of distinct paths from vertex $u$ to vertex $v$ in a directed acyclic graph.

(a) What recursive relationship exists between the number of distinct paths from $u$ to $v$ and the number of distinct paths to $v$ from the children of $u$?

(b) How can the topological sort algorithm be modified to count the number of distinct paths from $u$ to $v$ in a directed acyclic graph?
8. (12 points) Scanning a graph $G = (V, E)$, the depth-first search algorithm annotates each vertex, say $v$, with a discovery time, $v.d$, and a finish time, $v.f$. These points define a time interval $[v.d, v.f]$.

(a) Why is the overlapping arrangement $v.d < u.d < v.f < u.f$ a violation of the Parenthesis Theorem?

(b) What arrangement of time intervals characterizes $v$ as a descendant of $u$ in a $G_\pi$ tree produced by the algorithm?

(c) Suppose $(u, v) \in E$. In this case, which arrangement of time intervals corresponds to the situation where $(u, v)$ is an edge between two distinct $G_\pi$ trees?

9. (3 points) In a connected weighted undirected graph, $G = (V, E, w : E \rightarrow \mathbb{R}^+)$, suppose we have identified a subset of edges, $A$, such that $A \subseteq T$, where $T$ is a minimal spanning tree. What does it mean to say that an edge $(u, v)$ is safe for $A$?

10. (12 points) Suppose $G = (V, E)$ is a connected, undirected, weighted graph, and suppose $A \subseteq E$ is such that $A$ is contained inside some minimum spanning tree $T$. In the context of the Kruskal and Prim algorithms,

(a) What is “a cut that respects $A$”?

(b) Which of the algorithms, Kruskal or Prim, selects the lightest edge connecting two components in an evolving forest?

(c) The other algorithm, the one not chosen in the previous answer, selects the lightest edge connecting an evolving tree to an isolated vertex. What device is used to track the distance of the isolated vertices to the evolving tree?

(d) On a connected graph, the breadth-first search algorithm produces a $G_\pi$ tree by annotating each vertex with both a $v.\pi$ attribute from which the tree edges may be recovered and a $v.d$ attribute that records the length of the shortest path from the given source vertex. Hence breadth-first search returns a minimum spanning tree. What advantage over breadth-first search is offered by the Kruskal and Prim algorithms?
11. (15 points) The following questions relate to the Single-Source Shortest Path Problem. The notation $\delta(s,v)$ denotes the length of a minimal weight path from source $s$ to vertex $v$.

(a) Both the Bellman-Ford and the Dijkstra algorithms employ an operation known as relaxing an edge. What exactly occurs when an edge is relaxed.

(b) Assuming no negative cycle is reachable from the source $s$, what condition on edge relaxation (one of the lemmas associated with the Bellman-Ford proof of correctness) guarantees that all vertices $v$ reachable from the source $s$ will exhibit $v.d = \delta(s, v)$ when the algorithm concludes.

(c) What constraint on the $\delta(s,v)$ values is known as the triangle inequality?

(d) If the input graph is acyclic, the Single-Source Shortest Path problem can be solved in linear time. What observation about acyclic graphs ensures that we need to relax each edge only once.

(e) Although the Dijkstra algorithm achieves a lower asymptotic complexity class than does the Bellman-Ford algorithm, Dijkstra’s algorithm can be used only on graphs that satisfy a certain restriction. What is that restrictive condition?
Solutions.

1. (12 points) \( g(n) = n^3 \)

   Follow the example, \( f(n) = n^3 \): Left C Right Z , to locate the following functions in most accurate parts of the left and right diagrams above.

   \[
   f_1(n) = \frac{n^3}{\log n} \quad \text{Left A Right Z}
   \]
   \[
   f_2(n) = n^2 (\log n)^2 \quad \text{Left A Right Z}
   \]
   \[
   f_3(n) = \begin{cases} 
   n^3, & n \text{ even} \\
   n^2 \log n, & n \text{ odd}
   \end{cases} \quad \text{Left B Right Y}
   \]

2. (12 points) What does the master template tell us about the following recurrences?

   (a) \( T(n) = 2T(\lfloor n/4 \rfloor) + \sqrt{n} \)

   Reference \( g(n) = n \log_2^3 \log n = \sqrt{n} \); glue \( f(n) = \sqrt{n} \). \( f = \Theta(g) \), and the master template assigns \( T(n) = \Theta(\sqrt{n \log n}) \).

   (b) \( T(n) = 3T(\lfloor n/4 \rfloor) + \sqrt{n} \)

   Reference \( g(n) = n \log_3^3 = n^{0.5+\epsilon} \) for some positive \( \epsilon \); glue \( f(n) = \sqrt{n} = n^{0.5} \). Consider the polynomially diminished reference function \( g_-(n) = n^{0.5+\epsilon/3} \). Then \( f(n)/g_-(n) \quad \text{0} \), which implies \( f = o(g_-) \) which in turn implies \( f = O(g_-) \). The master template assigns \( T(n) = \Theta(n \log_4^3) \).

   (c) \( T(n) = 2T(\lfloor n/4 \rfloor) + \sqrt{n}/\log (n) \)

   Reference \( g(n) = n \log_2^2 = \sqrt{n} \); glue \( f(n) = \sqrt{n}/\log (n) \). Hence \( f(n)/g(n) \quad \text{0} \), which implies \( f(n)/g_+(n) \quad \text{0} \) for any polynomially enhanced reference \( g_+ \). Therefore, \( f \neq \Omega(g_+) \) for any polynomially enhanced reference \( g_+ \), which excludes the third case of the master template.

   Also \( f(n)/g(n) \quad \text{0} \) means \( f \neq \Theta(g) \), which excludes the second case of the master template.

   A polynomially diminished reference has the form \( g_-(n) = n^{0.5-\epsilon} \) for some \( \epsilon > 0 \). Then \( f(n)/g_-(n) = n^\epsilon/\log n \quad \text{\( \not\sim \infty \)} \), which means \( f \neq O(g_-) \) for any polynomially diminished reference \( g_- \). So, the first case of the master template is not valid.

   We conclude that the master template provides no information about the asymptotic complexity of \( T(n) \).

3. (3 points) Suppose in the recurrence form \( T(n) = aT(n/b) + f(n) \), we have \( f(n) = \Omega(n^{\log_a^3}) \). What further condition is necessary to conclude via the master template that \( T(n) = \Theta(f(n)) \)?

   We need

   (a) the existence of an \( \epsilon > 0 \) with \( f(n) = \Omega(n^{\log_a^3+\epsilon}) \), and

   (b) the existence of a number \( c \) with \( 0 < c < 1 \) and \( af(n/b) < cf(n) \) for all sufficiently large \( n \).
4. (8 points) One of the homework problems involved finding the diameter of a tree. The tree is not necessarily binary.

(a) What is the definition of a tree diameter?
   Given two tree vertices, u and v, there is a unique path from u to v containing the fewest number of links. Let \( \delta(u, v) \) be the number of links in the shortest path from u to v. Then the tree diameter is
   \[
   D = \max \{ \delta(u, v) : u, v \text{ are vertices in the tree} \}.
   \]

(b) What recursive relationship exists between the diameter of a tree and the diameters of root’s subtrees?
   Let \( D(r) \) be the diameter of a tree rooted at vertex r. Then \( D(r) \) is the largest of the following quantities:
   - the largest diameter of a subtree rooted at a child of r
   - 1 + the height of the tallest subtree rooted at a child of r (equals the height of the tree rooted at r)
   - 2 + the heights of the two tallest subtree rooted at two children of r.

5. (12 points) In the depth-first search algorithm of a directed graph \( G = (V, E) \), suppose vertex v is examined on the adjacency list of vertex u. What are the possible edge types for \( (u, v) \in E \) if the color of v is
   (a) black: \( (u, v) \) is a forward or cross edge; forward if \( v.d > u.d \), otherwise cross.
   (b) gray: \( (u, v) \) is a backward edge.
   (c) white: \( (u, v) \) is a tree edge.

6. (3 points) In the depth-first search algorithm on a directed graph, why is the highest finish time assigned to the first vertex discovered in some strongly connected component with no incoming edges from other components? Consider a component A that has an incoming edge from component B. If the algorithm enters B while A is still undiscovered (all white), then it will find the edge to A before finishing B. In this case, B vertices are still on the stack when the algorithm starts stacking A vertices. Because A is strongly connected, all of its vertices will be stacked and unstacked before control returns to a stack frame associated with a B vertex. Consequently, every A vertex will have earlier finish times than any B vertices. So, the highest finish time will not be assigned to an A vertex.

The remaining possibility is that the algorithm enters A while B is still undiscovered. In that case, it cannot discover any B vertex before finishing A because that would imply a path from A to B, which together with the assumed path from B to A would produce a cycle between the two separate components – a contradiction.

So, A finishes before B is discovered and therefore all B vertices have larger finish times that any vertices of A. Again, the highest finish time is not assigned to an A vertex.

We conclude that the highest finish time must be assigned to a component with no incoming edges.

7. (8 points) One of the homework problems involved counting the number of distinct paths from vertex u to vertex v in a directed acyclic graph.

(a) What recursive relationship exists between the number of distinct paths from u to v and the number of distinct paths to v from the children of u?
   Let \( N(w) \) be the number of distinct paths from w to v. Then
   \[
   N(u) = \sum \{ N(w) : w \text{ is a child of } u \}.
   \]

(b) How can the topological sort algorithm be modified to count the number of distinct paths from u to v in a directed acyclic graph?
   We place an attribute \( c \) in all vertices before running the topological sort algorithm. Initially, \( v.c = 0 \) for all vertices v. If there are paths from u to v, then v must finish before u in the depth-first search, because all nodes in the topological sort point from old finish times toward early finish times. When v finishes, we set \( v.c = 1 \). At each subsequent finish, say w.f, we set \( w.c \) equal to the sum of the \( c \)-values of its children. Since the children of a node w must finish before w in accordance with edges pointing from old finish times toward early finish times, their \( c \) values will be already established. When u finally finishes, \( u.c \) will contain the number of distinct paths from u to v. Note that this algorithm will also deliver the correct answer, zero, when there are no paths from u to v by virtue of u finishing before v.
8. (12 points) Scanning a graph \( G = (V,E) \), the depth-first search algorithm annotates each vertex, say \( v \), with a discovery time, \( v.d \), and a finish time, \( v.f \). These points define a time interval \([v.d,v.f]\).

(a) Why is the overlapping arrangement \( v.d < u.d < v.f < u.f \) a violation of the Parenthesis Theorem?
In this arrangement, \( u \) is discovered while \( v \) is still gray. Therefore the stack frame controlling the \( u \) visitation lies above the stack frame controlling the \( v \) visitation. In order for the \( v \) stack frame to conclude, which happens just after \( v.f \) is assigned, all stack frames above it must have closed. Specifically, the higher stack frame associated with the \( u \) visitation must have closed, forcing \( u.f < v.f \) — a contradiction of the given order.

(b) What arrangement of time intervals characterizes \( v \) as a descendant of \( u \) in a \( G_\pi \) tree produced by the algorithm?

\[ u.d < v.d < v.f < u.f \]

(c) Suppose \((u,v) \in E\). In this case, which arrangement of time intervals corresponds to the situation where \((u,v)\) is a edge between two distinct \( G_\pi \) trees?

\[ v.d < v.f < u.d < u.f \]

corresponds to the described situation. It also corresponds to an edge between nodes in the same \( G_\pi \) tree that have no ancestor-descendant relationship.

9. (3 points) In a connected weighted undirected graph, \( G = (V,E,w: E \rightarrow \mathbb{R}^+) \), suppose we have identified a subset of edges, \( A \), such that \( A \subseteq T \), where \( T \) is a minimal spanning tree. What does it mean to say that an edge \((u,v)\) is safe for \( A \)?

An edge \((u,v)\) is safe for \( A \) if \( A \cup \{(u,v)\} \subseteq T' \), where \( T' \) is also a minimum spanning tree. \( T' \) may be \( T \), but that is not necessary for a safe edge.

10. (12 points) Suppose \( G = (V,E) \) is a connected, undirected, weighted graph, and suppose \( A \subseteq E \) is such that \( A \) is contained inside some minimum spanning tree \( T \). In the context of the Kruskal and Prim algorithms,

(a) What is “a cut that respects \( A \)”?
A cut is a partition \((S,V \setminus S)\) that divides the vertices into two sets. A cut respects the edge set \( A \) if no edge in \( A \) have one vertex in \( S \) and the other in \( V \setminus S \).

(b) Which of the algorithms, Kruskal or Prim, selects the lightest edge connecting two components in an evolving forest?
Kruskal’s Algorithm.

(c) The other algorithm, the one not chosen in the previous answer, selects the lightest edge connecting an evolving tree to an isolated vertex. What device is used to track the distance of the isolated vertices to the evolving tree?
Prim’s algorithm use a minHeap, keyed on the \( v.d \) attributes of the vertex, to maintain a collection of vertices that have not yet been incorporated into the evolving minimum spanning tree.

(d) On a connected graph, the breadth-first search algorithm produces a \( G_\pi \) tree by annotating each vertex with both a \( v.\pi \) attribute from which the tree edges may be recovered and a \( v.d \) attribute that records the length of the shortest path from the given source vertex. Hence breadth-first search returns a minimum spanning tree. What advantage over breadth-first search is offered by the Kruskal and Prim algorithms? The breadth-first returns a shortest-path tree in which path length is measured by the number of edges between two vertices. Breadth-first search cannot return a minimum spanning tree when the edge weights vary across the edges, whereas either the Kruskal or Prim algorithm can produce a minimum spanning tree associated with arbitrary edge weights.

11. (15 points) The following questions relate to the Single-Source Shortest Path Problem. The notation \( \delta(s,v) \) denotes the length of a minimal weight path from source \( s \) to vertex \( v \).

(a) Both the Bellman-Ford and the Dijkstra algorithms employ an operation known as relaxing an edge. What exactly occurs when an edge is relaxed.
We relax an edge \((u,v)\) by comparing the current \( v.d \) with \( u.d + w(u,v) \), treating the latter as an approximation for a competitor to the minimum possible value of \( v.d \). If \( v.d \) is larger, we replace \( v.d \) with the smaller expression. Specifically,
Relax\(u, v, w\)
\[
\begin{align*}
&\text{if } v.d > u.d + w(u, v) \{ \\
&\quad v.d = u.d + w(u, v) \\
&\quad v.pi = u \\
&\} 
\end{align*}
\]

(b) Assuming no negative cycle is reachable from the source \(s\), what condition on edge relaxation (one of the lemmas associated with the Bellman-Ford proof of correctness) guarantees that all vertices \(v\) reachable from the source \(s\) will exhibit \(v.d = \delta(s, v)\) when the algorithm concludes.

Path-relaxation property: if \((s = v_0, v_1, \ldots, v_k)\) is a shortest path from \(s\) to \(v\), then relaxing edges in the order \((v_0, v_1), \ldots, (v_{k-1}, v_k)\) guarantees that \(v_k.d = \delta(s, v_k)\), which is the desired final result.

(c) What constraint on the \(\delta(s, v)\) values is known as the triangle inequality?

If \((u, v) \in E\), then the constraint \(\delta(s, v) \leq \delta(s, u) + w(u, v)\) (the triangle inequality) must hold. It is simply the observation that the right-hand-side constitutes a competitor for \(\delta(s, v) = \min\{w(p) : p \text{ is a path from } s \text{ to } v\}\).

(d) If the input graph is acyclic, the Single-Source Shortest Path problem can be solved in linear time. What observation about acyclic graphs ensures that we need to relax each edge only once.

If the input is acyclic, we can obtain a topological sort of its vertices in linear time. We process the vertices in this order, relaxing all edges connecting a vertex to its immediate neighbors. Because all edges from a given vertex point forward in the topological sort, we will relax edges associated with the shortest path from a given vertex in the order necessary to achieve convergence of all the \(v.d\) values. See (b) above. An immediate consequence is that we relax each edge exactly once, as opposed to \(V - 1\) times in the Bellman-Ford algorithm.

(e) Although the Dijkstra algorithm achieves a lower asymptotic complexity class than does the Bellman-Ford algorithm, Dijkstra’s algorithm can be used only on graphs that satisfy a certain restriction. What is that restrictive condition?

Dijkstra’s algorithm requires all edge weights to be nonnegative.