Solutions.

1. What does the master template reveal about the recurrence \( T(n) = 2T(n/4) + n\sqrt{n} \)? Why?

Solution. We have \( a = 2, b = 4 \), giving a reference function \( g(n) = n^{\log_b a} = n^{1/2} \). The glue function is \( f(n) = n\sqrt{n} = n^{3/2} \). We observe

\[
\frac{f(n)}{g_+(n)} = \frac{n^{3/2}}{n^{(1/2)+\epsilon}} = n^{1-\epsilon} \to \infty
\]

for \( 0 < \epsilon < 1 \). Also,

\[
a f(n/b) = 2 \left( \frac{n}{4} \right)^{3/2} = \frac{2}{4^{3/2}} n^{3/2} = \frac{1}{4} f(n).
\]

That is, the side condition necessary for case (c) of the master template, \( a f(n/b) \leq c f(n) \) for some \( c \) in the range (0, 1), is satisfied for \( c = 1/4 \).

Case (c) of the master template then reveals that \( T(n) = \Theta(f) = \Theta(n\sqrt{n}) \).

2. Consider the following algorithms, running on input graphs \( G = (V,E) \) for which \( E = \Theta(V) \). For each algorithm in the left list, choose the letter in the right list that corresponds to the tightest asymptotic bound on the worst-case running time.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>B Breadth-First-Search</td>
<td>A. ( O(\lg V) )</td>
</tr>
<tr>
<td>B Depth-First-Search</td>
<td>B. ( O(V) )</td>
</tr>
<tr>
<td>D Bellman-Ford</td>
<td>C. ( O(V \lg V) )</td>
</tr>
<tr>
<td>C Dijkstra</td>
<td>D. ( O(V^2) )</td>
</tr>
<tr>
<td>F Floyd-Warshall</td>
<td>E. ( O(V^2 \lg V) )</td>
</tr>
<tr>
<td>E Johnson</td>
<td>F. ( O(V^3) )</td>
</tr>
<tr>
<td>C Prim</td>
<td>G. ( O(V^3 \lg V) )</td>
</tr>
<tr>
<td>C Kruskal</td>
<td></td>
</tr>
<tr>
<td>B Topological-Sort</td>
<td></td>
</tr>
<tr>
<td>B Strongly-Connected-Components</td>
<td></td>
</tr>
</tbody>
</table>

3. 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.

4. 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.

5. Consider the problem of 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 $v$.

6. For each of the following situations, give an example of a directed graph containing vertices $u$ and $v$ such that the depth-first algorithm behaves as specified. Your example will have to specify the order of edges in the adjacency list representation of the graph, as well as the starting node for the depth-first search.

(a) There is a path from $u$ to $v$; the algorithm discovers $u$ before it discovers $v$; and $v$ is not a $G_\pi$-descendant of $u$.

```
graph
Label 

0
   1

1
   2

2
```

(b) There is a path from $u$ to $v$; the algorithm finishes $u$ before it discovers $v$; and $v$ is not a $G_\pi$-descendant of $u$.

```
graph
Label 

0
   1

1
   2

2
```
7. Consider the following graph during a depth-first-search, starting a vertex \( a \). Write the order in which the vertices are visited. Write the letters clearly and leave some space between them.

Solution: \( aebicjgfnhmd \)

8. For the following graph, consider each of the following edges at the time when the destination vertex is examined on the adjacency list of the source vertex. Next to the edge, write the color of the destination vertex at that time.

- white \((a, e)\)
- white \((a, b)\)
- white \((a, c)\)
- black \((a, d)\)
- black \((b, e)\)
- white \((b, i)\)
- black \((b, c)\)
- white \((b, i)\)
- gray \((n, k)\)
- white \((c, j)\)
- black \((c, h)\)
- white \((c, d)\)
- black \((c, d)\)
- white \((k, f)\)
- black \((m, k)\)
9. For the following graph, draw the $G_\pi$ graph produced by Depth-First-Search.

![Graph Image]

10. The Kruskal and Prim approaches to the minimum spanning tree problem were unified by the concept of safe edges. Specifically, given a connected directed graph, an accumulation of safe edges always results in a minimum spanning tree. The difference between the two approaches lies in the cuts used to identify safe edges.

(a) Explain the cuts employed in the Kruskal algorithm.

Kruskal works through the edge set in order of non-decreasing weight. That is, the lightest edge is first. As each edge is considered, it is either added to the MST set or it is discarded. No edge is considered more than once.

At the outset, the Kruskal algorithm has accrued no safe edges, and each vertex is a distinct tree. These trees merge to form a single minimum spanning tree at the conclusion of the algorithm. At the point where a new edge is added to the growing set of safe edges, Kruskal moves forward in the sorted edge list, finally choosing an edge that connects two trees in the forest. Any interim edges encountered that connect vertices within a tree are discarded because they would introduce a cycle in that tree. Having identified an edge between two trees, Kruskal places a cut around one of the trees. The chosen edge must be a light edge crossing that cut because (1) it runs between the two trees, and (2) of all edges available at this point, it is the lightest one because of the sorted order. Consequently, than edge is safe to add to the growing MST.

(b) Explain the cuts employed in the Prim algorithm.

Prim starts from a source vertex and grows a single tree. All vertices that have not yet been added to the tree are maintained in a minHeap, for which the key is the weight of the lightest edge connecting the vertex to the tree. Vertices that cannot reach the tree with a single link maintain $\infty$ as their keys. Prim extracts the minimal element from the minHeap, which because of the minHeap property, must be the vertex with the lightest link to the tree. Hence Prim’s cut is drawn around the growing tree, and the minHeap property guarantees that the lightest edge crossing that cut will be chosen. That is, the chosen edge is safe to add to the growing MST.