1. The running times of Bellman-Ford (O(|V| · |E|)) and Floyd Warshall (O(|V|^3)) will not change due to integer weights. For Dijkstra’s algorithm, the running time is determined by the type of priority queue we use. When all the weights are integers between 1 and n, all the shortest path lengths will be between 1 and n^2. The data structure can thus be an array of size n^2, each element i containing a list of vertices v satisfying dist(s, v) = i. In such a queue, insert, deletemin and decreasekey are all O(1) operations, thus making the running time of Dijkstra’s algorithm O(|V| + |E|).

2. (a) The graph H will be a directed acyclic graph, since otherwise all the vertices in the components in a cycle in H, taken together, form a strongly connected component in G. Since H is acyclic, it must have a source component and a sink component. If we do a search in a sink component, the discovered vertices will all belong to a sink component. Thus we only need to identify vertices in sink components. This can be achieved by noting that the vertex with the highest post number in a DFS on G has to belong to a source component. Thus, if we do a DFS on GR (the graph with all edge directions reversed), the vertex with the highest post number will be in a sink component. Once we have identified a component, we can repeat the search from the vertex with the next highest post number in the remaining graph. Time taken is O(|V| + |E|).

(b) A biconnected graph is a connected graph with no articulation vertex (a vertex whose removal causes the graph to be disconnected). We could assume for our purposes that G is undirected. The graph H in this case will be a tree, since a vertex in a component in a cycle of H can safely be deleted without disconnecting G. Also note that a vertex in G could be in more than one biconnected component.

In a DFS traversal of G, if we identify an articulation vertex, then all the edges in the subtree rooted at any child of this vertex would be in a biconnected component. These edges can be retrieved by keeping them on a stack while performing DFS. To find out if a vertex u is an articulation point, note that it cannot have back edges from a descendant to a node higher than u in the DFS tree. To check for this, we can maintain depth information on each node in the DFS tree, that is, each node keeps track of the lowest depth of its children and propagates it up the tree when it is done. This procedure will take O(|V| + |E|) time.

3. An odd cycle in an undirected graph can be detected easily by maintaining parity/2-color information while doing a DFS search. Every new node gets the opposite color of its predecessor. Then any non-tree edge with both endpoints the same color would prove the existence of an odd cycle.

Detecting odd cycles in directed graphs is almost the same, except that there is more than one type of non-tree edge in a DFS traversal. The only monochromatic non-tree edges that would give us odd cycles are the back edges (which are easy to identify: the post numbers of a back edge (u, v) satisfy post(u) < post(v)). DFS takes O(|V| + |E|) time.

4. We will have a path variable for each node, with path(s) = 1 and all other path variables initialized to 0. Do a BFS traversal starting at s. Then each node’s path counter is updated with the addition of the path counter value of its predecessor (even if the node is visited before). Finally, the path counter of t will give us the number of paths from s to t. A BFS takes O(|V| + |E|) time.