Problem 1: For each statement below, say if it is True or False.

- Dijkstra’s algorithm works on graphs with arbitrary weight functions. **False**
- The Prim algorithm to compute the MST of a graph is a dynamic programming algorithm **False**
- The Floyd-Warshall algorithm to compute all pairs shortest paths in a graph is a greedy algorithm **False**
- Multiplying two polynomials of degree \( n \) can be done with \( O(n \log n) \) scalar multiplications **True**
- If you run Euclid’s algorithm to compute the GCD of two integers \( a, b \), the algorithm will make \( \Theta(\log b) \) recursive calls. **True**

Problem 2: **Amortized Analysis.** A sequence of \( n \) operations is performed on a data structure. The \( i^{th} \) operation costs \( i \) if \( i \) is a perfect square, otherwise it costs 1. Determine the total cost of the sequence of operations, and the amortized cost per operation.

**Solution:** Assume wlog that \( n \) is a perfect square \( n = a^2 \) with \( a = \sqrt{n} \). Between 1 and \( n \) there are exactly \( a \) numbers that are perfect squares: indeed for each \( 1 \leq j \leq a \) we have that \( j^2 \leq n \). So we can compute the total cost by dividing between the perfect squares and the non-squares: \( C = C_{\text{sq}} + C_{\text{nsq}} \). Now if we limit ourselves only to the perfect squares we have

\[
C_{\text{sq}} = \sum_{i=1}^{a=\sqrt{n}} \text{Cost}(i^2) = \sum_{i=1}^{\sqrt{n}} i^2 = O(n^{3/2})
\]

since \( \sum_{i=1}^{k} i^2 = O(k^3) \) (left as exercise). On the other hand

\[
C_{\text{nsq}} = \sum_{i=1, i\neq j^2}^{n} \text{Cost}(i) = n - \sqrt{n}
\]

since in this case the cost of each operation is 1.

So the total cost is \( O(n^{3/2}) \) and the amortized cost per operation is \( O(n^{1/2}) \).

Problem 3: The square of a directed graph \( G = (V, E) \) is the graph \( G^2 = (V, E^2) \) such that \( (u, w) \in E^2 \) if and only if there exists a \( v \in V \) such that \( (u, v) \in E \) and \( (v, w) \in E \). In other words \( G^2 \) contains an edge between \( u \) and \( w \) whenever \( G \) contains a path from \( u \) to \( w \) of length exactly 2.

Show how to compute \( G^2 \) when given \( G \) in either adjacency-list or adjacency-matrix representation.

**Solution:** Let \( n = |V| \) and \( m = |E| \).

For the adjacency list representation the naive algorithm takes \( O(n + m^2) \). Let \( Adj2 \) be the adjacency list representation for \( G^2 \). For each node \( u \in V \) go through the adjacency list of \( u \): if \( v \in Adj[u] \) then go through \( Adj[v] \) and add all the nodes in \( Adj[v] \) to \( Adj2[u] \). However this second loop over \( Adj[v] \) can be easily avoided: just add the entire list \( Adj[v] \) to \( Adj2[u] \). This is a \( O(1) \) operation (just move a pointer). So the total cost is \( O(n + m) \).

For the adjacency matrix representation the algorithm runs in \( O(n^3) \). That’s because you need to produce \( O(n^2) \) values (one for each entry of the matrix) and each value requires a loop over all the nodes: to see if there is an edge between \( (u, v) \) you need to check every \( w \in V \) and see if the edges \( (u, w) \) and \( (w, v) \) are in \( E \).
Problem 4: Prove or disprove: if a directed graph $G$ contains cycles then the DFS-based topological sort algorithm described in class (and in Section 22.4 of the book) produces a vertex ordering that minimizes the number of "bad" edges, i.e. edges that are inconsistent with the ordering produced by the algorithm.

Solution: The statement is false. Consider the graph below: when you start DFS on node 1 you get two bad edges, but when you start DFS on node 5 the resulting topological sort has only 1 bad edge.

Problem 5: When using the complex root of unity to compute the FFT, one might encounter problems related to precision errors. When dealing with polynomials with only integer coefficients, it might be desirable to use different interpolation points that are not subject to precision errors. One possible approach uses modular arithmetic. Let $p$ be a prime such that $p = kn + 1$ where $n$ is the length of the FFT input vector and $k$ is an arbitrary constant (chosen as small as possible). Assume that such prime can be found easily.

As we saw in class then for any element $a \in Z_p^n$ we have that $a^{p-1} = a^{kn} = 1 \mod p$. Consider an element $g \in Z_p^n$ such that $g^j \neq 1 \mod p$ for all $1 \leq j < kn$ (i.e. $g^j = 1 \mod p$ only if $j = kn$ - such elements are called generators of $Z_p^n$ and can be found efficiently).

Let $\omega = g^k \mod p$. Note that $\omega \neq 1 \mod p$ but $\omega^n = 1 \mod p$, so $\omega$ is an $n$-root of unity in $Z_p^n$.

- Let $p = 17$. Check that 3 is a generator for $Z_{17}^*$, and then compute the 8-root of unity in $Z_{17}^*$.
  - For general $p$, argue that you can still compute FFT in $Z_p^*$ using the divide-and-conquer algorithm we saw in class.

Solution:

- One way to check that 3 is a generator is to check that for all $j = 1, \ldots, 16$, $3^j \neq 1 \mod 17$. This is easy but a bit tedious. Another way is to check that $3^7 \neq 1 \mod 17$ only for the integers $j$ that divide $p - 1 = 16$. This is because for general $g, p$, the subset $G = \{g, g^j\} = \{a = g^j\}$ is a subgroup of $Z_p^*$ and therefore its order must divide $p - 1$. Either $G = Z_p^*$ in which case $g$ is a generator, or $G \subset Z_p^*$ but in that case $g^j = 1 \mod p$ for $j = |G|$ which must be a divisor of $p - 1$. So in our case we can limit our check to $j = 2, 4, 8$. $3^2 = 9 \neq 1 \mod 17; 3^4 = 9^2 = 81 = 13 \neq 1 \mod 17; 3^8 = 13^2 = (-4)^2 = 16 \neq 1 \mod 17$. So 3 is a generator.
  
  Note that we can write $16 = 8 \times 2$ so above $n = 8$ and $k = 2$. So $\omega = 9 \mod 17$. Then we have $\omega_2 = \omega^2 = 13 \mod 17, \omega_3 = 13 \cdot 9 = (-4) \cdot 9 = -36 = -2 = 15 \mod 17, \omega_4 = 16 = -1 \mod 17, \omega_5 = -9 = 8 \mod 17; \omega_6 = 9 \cdot 8 \mod 17 = 4 \mod 17; \omega_7 = 36 = 2 \mod 17 and \omega_8 = 2 \cdot 9 = 18 = 1 \mod 17$.

- Recall that the FFT invokes itself recursively on two problems of size $n/2$ over the values $\omega^2_i \mod p$. If $\omega$ is the $n$-root of unity, then obviously like in the complex case, the values $\omega^2_i$ are the $n/2$-root of unity in $Z_p^*$. That's because the halving lemma (Lemma 30.5 in the textbook) applies in this case too:

$$\omega_i^{n/2} = \omega_i \mod p$$ since $\omega_i^{n/2} = \omega_i^{(i+n/2)/2} = \omega^{2i} \cdot \omega^n = (\omega^i)^2 = \omega_i^2 \mod p$ since $\omega^n = 1$. 


Problem 6: Let $n$ be an integer, and $a, b \in \mathbb{Z}_n^*$. If $GCD(a, n) = 1$ how many solution does the equation $ax = b$ have in $\mathbb{Z}_n^*$? Justify your answer.

Solution: Since $GCD(a, n) = 1$ we know that there exists a multiplicative inverse of $a$, i.e. an unique value $\alpha \in \mathbb{Z}_n^*$ such that $a\alpha = 1 \mod n$. So there is an unique solution $x = \alpha b \mod n$. 