**Algorithm 1** Prim's algorithm with edge priority queue

**Input:** Graph $G = (V, E)$
**Output:** Minimum spanning tree of $G$

// Initialization
Select an arbitrary vertex $s \in V$
Construct a tree *mst* that contains only the vertex $s$
Construct an empty priority queue $Q$ that will contain edges ordered by their weights // We will maintain the invariant that for every edge $(u, v) \in Q$, at least one of $u$ or $v$ is in *mst*
**for** $v \in$ neighbors($s$) **do**
  Add edge $(v, s)$ to $Q$
**end for**

// Construct *mst*
**while** $Q$ is not empty **do**
  Let edge $(u, v) = Q$.findMin()
  $Q$.removeMin()
  **if** node $u \in mst$ and node $v \in mst$ **then**
    // TODO: What goes here?
  **else**
    // TODO: What about here?
  **end if**
**end while**
**return** *mst*

**Algorithm 2** Prim's algorithm with node priority queue

**Input:** Graph $G = (V, E)$
**Output:** Minimum spanning tree of $G$

// Initialization
Select an arbitrary vertex $s \in V$
Construct an empty tree *mst*
Construct an empty priority queue $Q$ that will contain nodes ordered by their "distance" from *mst* // If $v \notin mst$, then the distance of $v$ is defined as the weight of the minimum cost edge $(u, v)$ such that $u \in mst$
Insert $s$ into $Q$ with priority 0

// Construct *mst*
**while** there exists a vertex $v$ s.t. $v \in V$ and $v \notin mst$ **do**
  Let $v = Q$.findMin()
  $Q$.removeMin()
  **for** vertex $u \in$ neighbors($v$) **do**
    **if** $v \notin mst$ **then**
      **if** weight$(u, v) <$ Q.getPriority($u$) **then**
        //TODO: What goes here?
      **end if**
    **end if**
  **end for**
**end while**
**return** *mst*

**Algorithm 3** Kruskal's algorithm: high level

**Input:** Graph $G = (V, E)$
**Output:** Minimum spanning forest of $G$
Create a forest $msf$, initialized so that every vertex $v \in V$ is a singleton tree in $msf$
Create a set $remainingedges$ containing all the edges in the graph
**while** $remainingedges \neq \emptyset$ and $msf$ is not spanning **do**
  Remove an edge $(u, v)$ with minimum weight from $remainingedges$
  **if** $(u, v)$ connects two different trees in $msf$ **then**
    Add $(u, v)$ to $msf$, combining the two trees
  **end if**
**end while**

**Algorithm 4** Kruskal's algorithm: with priority queue over edges

**Input:** Graph $G = (V, E)$
**Output:** Minimum spanning forest of $G$

// initialize forest
Create an empty map $msf$ whose keys are nodes and values are trees
**for** vertex $v \in V$ **do**
  $msf[v] \leftarrow$ tree with $v$ as only node
**end for**

// initialize priority queue
Create an empty priority queue $Q$ whose elements are edges and priorities are weights
**for** edge $(u, v) \in E$ **do**
  Insert $(u, v)$ into $Q$
**end for**

// main loop
**while** $|Q| > 0$ and $msf$ is not spanning **do**
  $(u, v) \leftarrow$ Q.findMin()
  Q.removeMin()
  **if** $msf[u] \neq msf[v]$ **then**
    // TODO: What goes here?
  **end if**
**end while**
**return** $msf$