Prim's Minimum Weight Spanning Tree Algorithm

Definition:
Suppose weights are associated with each edge of a connected undirected graph $G$ of $n$ vertices and $m$ edges (since $G$ is connected, $m \geq n-1$).

The \textit{weight} of a spanning tree $T$ for $G$ is the sum of the weights of its edges. $T$ is a \textit{minimum spanning tree} for $G$ if there is no other spanning tree for $G$ of lower weight.

Prim's Algorithm:

$T :=$ an arbitrary vertex of $G$

\textbf{while} $T$ has less than $n-1$ edges \textbf{do}

\hspace{1em} Add to $T$ the lowest cost edge $(v,w)$ such that $v$ is in $T$ and $w$ is not in $T$. 
Proof of Correctness of Prim's Algorithm

**Lemma:** Let $e$ be a minimum cost edge that connects $T$ to a vertex not in $T$. If there exists a minimum spanning tree that contains $T$, then there exists a minimum spanning tree that contains both $T$ and $e$.

**Proof:** Suppose the contrary; that is, that there exists a minimum spanning tree $U$ that contains $T$ but there does not exist a minimum spanning tree that contains both $T$ and $e$:

- The edges of $T$ are in $U$ (since $U$ contains $T$), and hence there is an edge $f$ in $U$ that connects $T$ to the remainder of $U$ (i.e. to a vertex not in $T$).
- By definition of $e$, $\text{cost}(e) \leq \text{cost}(f)$, and we can replace $f$ by $e$ in $U$ to obtain a new spanning tree $V$.
- We now have a contradiction: Either $V$ is minimal (and it contains both $F$ and $e$) or it is not, and $U$ is not either (since $\text{cost}(e) \leq \text{cost}(f)$ implies $\text{cost}(V) \leq \text{cost}(U)$).

Correctness now follows by induction on the number of iterations of the *while* loop (the current number of edges $i$ in the spanning tree being built):

- For $i=0$, $T$ is a single vertex and is contained by any minimal weight spanning tree.
- For $0 < i \leq m$, assume the $i-1$ edges are contained by a minimal weight spanning tree. Then so must the $i$ edges, since the edge added by the $i^{th}$ iteration satisfies the lemma.
(minimum weight spanning trees continued)

**Implementation of Prim's Algorithm**

\[ T := \text{an arbitrary vertex} \ r \text{ of} \ G \]
\[ S := \{r\} \]
\[ H := \text{a heap containing all edges incident to} \ r \]

**while** \( S \) contains less than \( n \) vertices and \( H \) is not empty **do begin**

\( (u,v) := \text{DELETEMIN}(H) \)

Assume \( u \) is in \( S \) (otherwise exchange the names of \( u \) and \( v \)).

**if** \( v \) is not in \( S \) **then begin**

Add \( (u,v) \) to \( T \).

Add \( v \) to \( S \).

**for** each edge \( (v,w) \) such that \( w \) is not in \( S \) **do** \( \text{INSERT}((v,w),H) \)

**end**

**end**

**if** \( S \) contains less than \( n \) vertices **then** ERROR — \( G \) is not connected

Output \( T \).
(implementation of Prim's algorithm continued)

**Time:** $O(m \log(n))$

- A standard LCHILD-RSIB representation can be used for $T$ (so edges can be added in $O(1)$ time).
- $S$ can be represented with a bit-vector $S[1]...S[n]$ where $S[i]$ is 1 if the $i^{th}$ vertex is in $S$ and 0 otherwise.
- So that the *while* loop can easily test if $S$ contains $n$ vertices, a counter can be initialized to 1 and then incremented each time a vertex is added to $S$.
- If we exclude the time spent to insert items into $H$, $O(\log(m))$ time is used by each iteration of the *while* loop to perform a DELETEMIN and $O(1)$ additional work, for a total of $O(m \log(m))$ time.
- $H$ can be implemented as a balanced tree (e.g., a self-adjusting BST) and initialized in $O(m \log(m))$ time by simply inserting the items one at a time (or $H$ could use a standard heap data structure and initialized in $O(m)$ time).
- The total time spent by all executions of the *for* loop is $O(m \log(m))$, since each edge can be inserted into $H$ at most two times (an edge $(v,w)$ appears once on the adjacency list for $v$ and once on the adjacency list for $w$).
- Since $\log(m) \leq \log(n^2) = 2\log(n) = O(\log(n))$ the time is $O(m \log(n))$. 
Space: $O(m)$

$S$ uses $O(n)$ space, $T$ uses $O(m)$ space, and $H$ uses $O(m)$ space, for a total of $O(n+m)$ space, which is $O(m)$ if the graph is connected.