1. A graph $G$ is bipartite if its vertices can be partitioned into two sets $X$ and $Y$ such that every edge in $G$ has one end vertex in $X$ and the other in $Y$. Design and analyze an efficient algorithm for determining if an undirected graph $G$ is bipartite (without knowing the sets $X$ and $Y$ in advance).

**Algorithm.**

The algorithm (Algorithm 1) starts by running a specialized version of breadth first search that colors each vertex upon visiting. Let all vertices be initialized to GRAY. The source, level 0 of the BFS, is labeled WHITE. Then, the BFS alternates coloring vertices WHITE and BLACK such that even numbered levels are WHITE and odd numbered levels are BLACK. This 2-coloring of the graph indicates whether or not the graph is bipartite. After computing the coloring, a loop over the edges checks to see if both endpoints of all edges differ in color. If so, the graph is bipartite, false otherwise.

**Algorithm 1 isBipartite(G)**

| Input: Undirected graph $G = (V, E)$ |
| Output: true if $G$ is bipartite, false otherwise |

1. 2ColorWithBFS($G$)
2. for all $e \in E$ do
3.  if Colors of $e$’s endpoints differ then
4.    return false
5.  return true

**Time Complexity.**

**Theorem 1.** Algorithm 1 runs in $O(n + m)$ time on an undirected graph.

*Proof.* Let the coloring data structure be implemented as a resizable hash map. The time complexity of BFS to color the graph is the same as a typical BFS, i.e., $O(n + m)$. The final loop to see if any colors differ will take $O(m)$ time. In total, the algorithm takes $O(n + m)$ time. \(\square\)

**Memory Complexity.**

**Theorem 2.** Algorithm 1 uses $O(n)$ additional memory.

*Proof.* The coloring data structure to store the colors will take $O(n)$ additional memory. The BFS also might have as many as $O(n)$ nodes in a single level. In total, $O(n)$ additional memory. \(\square\)
2. Show that if all the weights in a connected weighted graph $G$ are distinct, then there is exactly one minimum spanning tree for $G$. Provide a statement and proof.

**Solution.**

**Theorem 3.** If all the weights in a connected weighted graph $G$ are distinct, then there is exactly one minimum spanning tree for $G$.

**Proof.** Assume the opposite – $G$ contains more than one minimum spanning tree. Let $T$ and $T'$ be two such minimum spanning trees. Let $e'$ be an edge in $T'$ that does not exist in $T$. If we insert $e'$ into $T$ to get a cycle. Let $e$ be an edge of this cycle in $T$ but not in $T'$. The weight of $e$ does not equal the weight of $e'$. Without loss of generality, let the weight of $e$ be less than that of $e'$. Then, we know we could insert $e$ into $T'$ and remove $e'$ from $T'$ to yield an overall spanning tree of lesser weight. This implies $T'$ is not a true minimum spanning tree and a contradiction is reached, and the original assumption must be false. 

\[ \square \]
3. NASA wants to link \( n \) stations spread over the country using communication channels. Each pair of stations has a different bandwidth available, which is known a priori. NASA wants to select \( n - 1 \) channels (the minimum possible) in such a way that all the stations are linked by the channels and the total bandwidth (defined as the sum of the individual bandwidths of the channels) is maximum. Give an efficient algorithm for this problem and analyze its worse-case performance. Consider the weighted graph \( G = (V, E) \), where \( V \) is the set of stations and \( E \) is the set of channels between the stations. Define the weight \( w(e) \) of an edge \( e \) in \( E \) as the bandwidth of the corresponding channel.

Algorithm.

Essentially use Kruskal’s algorithm but sort edges in decreasing order. This is called a maximum spanning tree. The algorithm is described in Algorithm 2.

**Algorithm 2** Maximum spanning tree

**Input:** Undirected Graph \( G = (V, E) \)

**Output:** Maximum spanning tree \( T \)

1. \( T \leftarrow \text{Kruskal}(G) \) [Use reverse comparator]
2. return \( T \)

**Time Complexity.**

**Theorem 4.** Algorithm 2 runs in \( O(m \log n) \) time on an undirected connected graph.

*Proof.** Kruskal’s runs in this time, and changing the comparator does not change the algorithm. Thus, \( O(m \log n) \) \( \square \)

**Memory Complexity.**

**Theorem 5.** Algorithm 2 uses \( O(n) \) additional memory.

*Proof.** Kruskal’s needs \( O(n) \) memory for the union find data structures. \( \square \)
4. **Bonus.** An old MST method, called Barvka’s algorithm, works as follows on a graph $G$ having $n$ vertices and $m$ edges with distinct weights: Let $T$ be a subgraph of $G$ initially containing just the vertices in $V$.

   while $T$ has fewer than $n - 1$ edges do
     for each connected component $C_i$ of $T$ do
       Find the lowest-weight edge $(v, u)$ in $E$ with $v$ in $C_i$ and $u$ not in $C_i$.
       Add $(v, u)$ to $T$ (unless it is already in $T$).

   return $T$

Argue why this algorithm is correct and why it runs in $O(m \log n)$ time.

**Solution.**

The algorithm is correct because of the partition property of a minimum spanning tree. Essentially, each connected component is a partition, so the edge of minimum weight connecting them must be in the minimum spanning tree. Doing this over and over merges the components like Kruskal’s algorithm. Thus it is correct.

The algorithm takes $O(m \log n)$ time because in each iteration edges must be found of minimum weight (taking $O(m)$ time). Each iteration, the number of components is cut in half. This can be done $O(\log n)$ times. Thus in total the algorithm takes $O(m \log n)$. 