Problem 1 Analyzing a Minimum Spanning Forest Algorithm

Algorithm 1 Build-MSF-By-Add-And-Fix (G,w)

```
F = {}
for each edge $e \in E$ taken in arbitrary order
    F = $F \cup \{e\}$
    if f has a cycle c
        let e' be a maximum weight edge on c
        F = F - $\{e'\}$
return F
```

(a) **Complete the following proof that this algorithm constructs an MSF**

- **Base Step k = 1**
  After one edge has been inspected then it is only the edge that creates a cycle therefore as a result is the MST.
- **Inductive Step**
- Case 1: Suppose that adding the edge does not result a cycle.
  Suppose that adding the (k+1)th edge e does not create a cycle in graph G. Then this implies that the edge e connects two separate sub-trees and it is still a forest. Furthermore, the edge that is added is the only one that connects the subtrees, therefore, the forest's weight is minimum over the edges.
- Case 2: Suppose that adding the edge does result a cycle.
  Suppose that adding the (k+1)th edge e does create a cycle. Then from the inductive hypothesis we know that the graph has a MSF on the first k edges.
  Therefore, if adding the (k+1)th edge e creates a cycle then from the brief explanation of Safe Edge Theorem (Removes the maximum weight in a single cycle) then removal of the heaviest weight will result in a MSF.

(b) **Describe an efficient algorithm for finding the maximum weight edge on a cycle, to use in lines 4-5.**
We can modify the DFS search for detecting cycles to also keep track of the max weight edge traversal in the active recursion tree.
Then returning the edge as soon as the back edge is found.

(c) **Analyze the time complexity of Build-MSF-By-Add-And-Fix, assuming you use the algorithm in (b)**
The overall run time for this is O(V+E) as it traverses through the edges and verticies of the graph.

Problem 2 MST on Restricted Range of Integer Weights

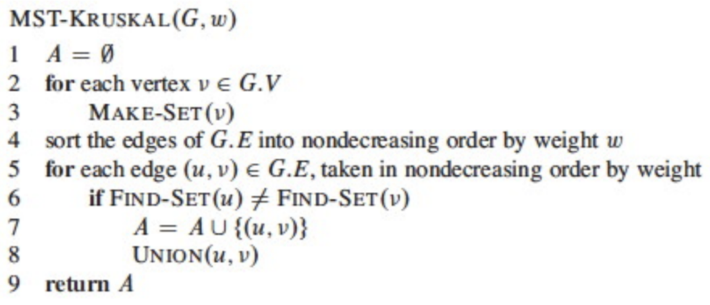
(a) **All edge weights are integers in the range from 1 to C for some constant C.**
The costs for Kruskal's algorithms are the following:

- Initialization: O(1)
- The For Loop when it enters for the first time: $|V|$
- Sorting E: O(E lg E)
- The For loop with Find-Set and Union: O(E)
- For a disjoint-sets O($E\alpha(V)$)
- The total running time is O(E lg V) if the graph is connected then can be simplified further to O(E lg E) as stated in the CLRS book

The modification one would make to the Kruskal's algorithm is to one can do a Counting-Sort to sort the edges with the range of 1 to C such that $C \in \mathbb{R}$.
Recall that from our notes that Counting-Sort has a running time of O(n+k). In this case n = $|E|$ and k = C, therefore, the sorting itself will take O(E+C) or O(E) since C is a constant.
Then the total runtime with the Counting Sort mentioned in the previous paragraph, the overall is O(V + E + E α(V)) which then simplifies further to O(E α(V)).
From the CLRS book page 633 section 23.2 α is a slow growing function then the result will be faster than O(E lg V).

(b) **All edge weights are integers in the range from 1 to $|V|$.**
Similar to the previous problem let us sort using Counting Sort. Then recall that Counting Sort has an overall run time of O(n+k). In this case n = $|E|$ and k = $|V|$, therefore, it follows that O($|E| + |V|$). However, the difference between this problem and the previous is that since V is not a constant it cannot be cancelled out.
Since the assumption is that the graph is connected then O(E) = V then the overall run time is O(E).
Then it follows that the overall expression is O(E α(V) + E) = O(Eα(V)).

Problem 3 Suppose we change the representation of edges from adjacency lists to matrices



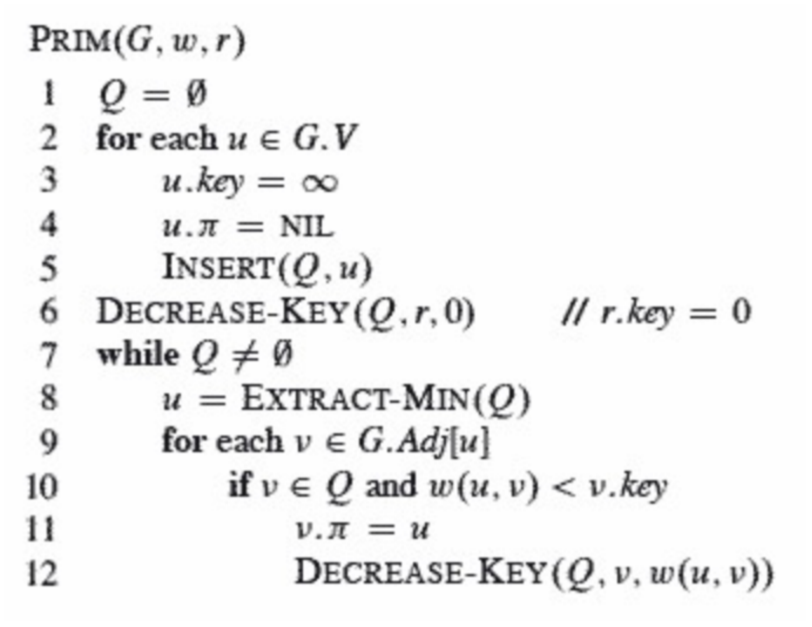
```
MST-KRUSKAL($G, w$)
1  $A = \emptyset$
2  for each vertex $v \in G.V$
3      MAKE-SET($v$)
4  sort the edges of $G.E$ into nondecreasing order by weight $w$
5  for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6      if FIND-SET($u$) $\neq$ FIND-SET($v$)
7          $A = A \cup \{(u, v)\}$
8          UNION($u, v$)
9  return $A$
```

(a) **What line(s) would have to change in the CLRS version of Kruskal's algorithm (shown), and how?**
Based off of the algorithm found in the CLRS book line 1 to 3 does not need modification as it deals with the vertex and not with the edge.
For line 4 since we are scanning through a matrix we would have to modify it as the current algorithm scans through a list. For the adjacency matrix, one could use the objects in the list cells to sort and then scan through a matrix to make an edge objects to be sorted.
Otherwise, lines 5-9 would not need any modifications since line 5 just sorts in nondecreasing order and line 6-8 is the check of duplicates.

(b) **What would be the resulting asymptotic runtime of Kruskal's algorithm on both dense (E = O(V^2)) and sparse (E = O(V)) graphs, and why?**
The total cost to go through the matrix for the non-zero entries is O(V^2). To do this let us break this into two cases.

- **Case: Both Sparse**
  Suppose that we are working with a sparse graph, then it follows that $E = O(V)$. Furthermore, sorting trough the edge is O(E lg E) and from a mathematical standpoint V^2 is greater than E lg E.
  Thus, the run time is O(V^2).
- **Case: Both Dense**
  Suppose that we are working with a dense graph. Then it follows that E = O(V^2) which would not bring any asymptotic change in the overall runtime as O(E lg E) = O(V^2 lg E).

Prim's Algorithm



```
PRIM($G, w, r$)
1  $Q = \emptyset$
2  for each $u \in G.V$
3      $u.key = \infty$
4      $u.\pi$ = NIL
5      INSERT($Q, u$)
6  DECREASE-KEY($Q, r, 0$)        // $r.key = 0$
7  while $Q \neq \emptyset$
8      $u$ = EXTRACT-MIN($Q$)
9      for each $v \in G.Adj[u]$
10         if $v \in Q$ and $w(u, v) < v.key$
11             $v.\pi = u$
12             DECREASE-KEY($Q, v, w(u, v)$)
```

(a) **What line(s) would have to change in the CLRS version of Prim's algorithm (shown), and how?**
At line 9 we have the words G.Adj[u] which would need to be modified as we are working with matrix. For instance, one can replace it with scan through the matrix of row from u.
Then at the next line or line 10 one must have to access or take a look at the weights stroed in the cells of the matrix which takes a constant time. Then store the π into a separate array.

(b) **What would be the resulting asymptotic runtime of Prim's algorithm on both dense (E = O(V^2)) and sparse (E = O(V)) graphs, and why?**
In the original Prim's algorithm at line of the while loop this is executed O(V) and since it is scanning through a matrix then it would take O(V^2).
From the CLRS book the runtime of decrease min is O(lg V) and since it is going through the edges at most one times then the run time is O(E lg V).
Thus the overall runtime is O(V^2 + E lg V).
Similar to the previous problem, let us break this into case statements:

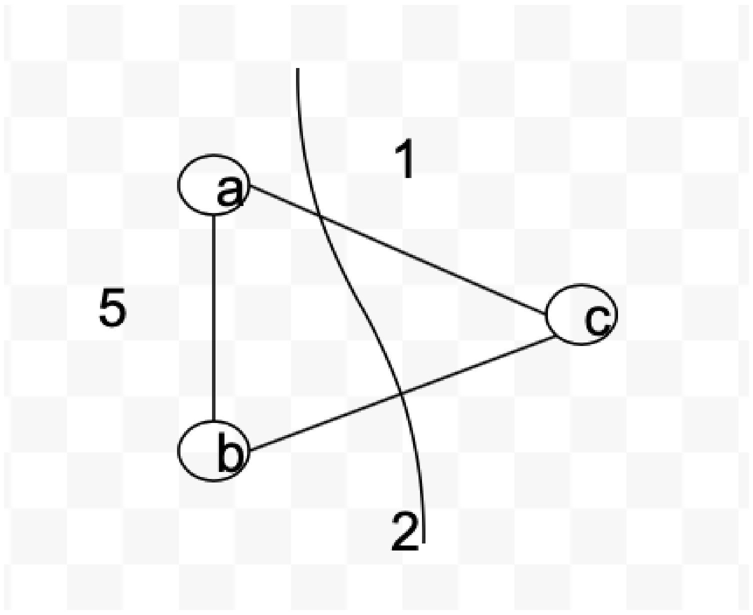
- **Both Dense**
  If they are both dense then E = $O(V^2)$. Then the overall run time would be O($V^2 + V^2 \text{lg} V$) and since from a mathematical standpoint $V^2 \text{lg} V > V^2$ hence the runtime in dense is O($V^2 \text{lg} V$).
- **Both Sparse**
  Suppose that the graph is sparse. Then E = O(V) and the runtime will be fixed as O($V^2 + V \text{lg} V$)

Problem 4 Divid and Conquer MST

Prove that this algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

> Given G = (V, E), partition V into V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve the MST problem on each of the two subgraphs G_1 = (V_1, E_1) and G_2 = (V_2, E_2). Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2) and use this edge to join the resulting two minimum spanning trees into a single spanning tree.

To prove this provide a counterexample.



When once cutting the partition from this are $V_1 = \{a, b\}$ and $V_2 = \{c\}$, and then the recursive calls would be $G_1 = (V_1, \{a, b\})$ and $G_2 = (V_2, \{\})$.
Then the divide and conquer algorithm would add an edge (a,b) to the MST when doing a recursive call on G_1. However, this does not result a MST as the whole graph with the edges $\{(a, c), (b, c)\}$ will have a lighter MST with a total weight of (1+2) = 3.