Fundamental Algorithms, Assignment 10
Solutions

1. Suppose we are given the Minimal Spanning Tree $T$ of a graph $G$. Now we take an edge $\{x, y\}$ of $G$ which is not in $T$ and reduce its weight $w(x, y)$ to a new value $w$. Suppose the path from $x$ to $y$ in the Minimal Spanning Tree contains an edge whose weight is bigger than $w$. Prove that the old Minimal Spanning Tree is no longer the Minimal Spanning Tree.

Solution: We can replace the edge whose weight is bigger than $w$ with the edge $\{x, y\}$ resulting in a lower weight spanning tree.

2. Suppose we ran Kruskal’s algorithm on a graph $G$ with $n$ vertices and $m$ edges, no two costs equal. Suppose the the $n - 1$ edges of minimal cost form a tree $T$.

(a) Argue that $T$ will be the minimal cost tree.

Solution: From Kruskal’s Algorithm we will accept all the edges of $T$. Then we have a spanning tree so no more edges are accepted.

(b) How much time will Kruskal’s Algorithm take. Assume that the edges are given to you an array in increasing order of weight. Further, assume the Algorithm stops when it finds the MST. Note that the total number $m$ of edges is irrelevant as the algorithm will only look at the first $n - 1$ of them.

Solution: We do $n$ operations UNION[$x, y$], each takes time $O(\ln n)$ so the total time is $O(n \ln n)$.

(c) We define Dumb Kruskal. It is Kruskal without the SIZE function. For UNION[$u, v$] we follow $u, v$ down to their roots $x, y$ as with regular Kruskal but now, if $x \neq y$, we simply reset $\pi[y] = x$. We have the same assumptions on $G$ as above. How long could Dumb Kruskal take. Describe an example where it takes that long. (You can imagine that when the edge $u, v$ is given an adversary puts them in the worst possible order to slow down your algorithm.)

Solution: As UNION[$x, y$] must take time $O(n)$ (as there are only $n$ vertices) the whole algorithm will take time $O(n^2)$. This can happen. Suppose the edges were, in order, $\{2, 1\}, \{3, 1\}, \{4, 1\}, \ldots, \{n, 1\}$. For the first edge we make $\pi[1] = 2$. The second edge we follow 1 down to root 2 and set $\pi[2] = 3$. Now for the third
edge we follow 1 to 2 to root 3 and set \( \pi[3] = 4 \). On the \( i \)-th step we are taking time \( \sim i \) so it is a \( \Theta(n^2) \) running time. tem

(d) Consider Kruskal’s Algorithm for MST on a graph with vertex set \( \{1, \ldots, n\} \). Assume that the order of the weights of the edges begins \( \{1, 2\}, \{2, 3\}, \{3, 4\}, \ldots, \{n - 1, n\} \). Assume that when in Kruskal’s Algorithm we have a tie \( \text{SIZE}[x] = \text{SIZE}[y] \) we set the smaller of \( x, y \) to be the parent of the largest.

i. Show the pattern as the edges are processed. In particular, let \( n = 100 \) and stop the program when the edge \( \{1, 73\} \) has been processed. Give the values of \( \text{SIZE}[x] \) and \( \pi[x] \) for all vertices \( x \).

**Solution:**
First we set \( \pi[2] = 1 \) and \( \text{SIZE}[1] = 2 \). Now for \( i = 3, 4, \ldots \) when we process \( 1, i \) we have \( \pi[i] = i \) and \( \pi[i - 1] = 1 \). (In a formal mathematical sense this would be by induction, but its OK just to see the pattern.) So the WHILE loop sends \( i - 1 \) to \( 1 \) with \( \text{SIZE}[1] = i - 1 \) and \( i \) to itself with \( \text{SIZE}[i] = 1 \) so we set \( \pi[i] = 1 \) and reset \( \text{SIZE}[1] = i - 1 + 1 = i \). (That is, the \( \text{SIZE}[1] \) goes up by one for each iteration.) With \( n = 100 \) after \( \{1, 73\} \) is processed we have \( \pi[i] = 1 \) for all \( 1 \leq i \leq 73 \) and \( \text{SIZE}[1] = 73 \) and \( \text{SIZE}[i] = 1 \) for \( 2 \leq i \leq 73 \). For the yet untouched \( i \) from 74 to 100 we still have the initial values \( \text{SIZE}[i] = 1, \pi[i] = i \).

ii. Now let \( n \) be large and stop the program after \( \{1, n\} \) has been processed. Assume the ordering of the weights of the edges was given to you, so it took zero time. How long, as an asymptotic function of \( n \), would this program take. (Reasons, please!)

**Solution:** It would be linear \( \Theta(n) \) time. At each iteration the WHILE loop is applied zero times for 1 and one time for \( i \) so it takes constant time – and we have to run the program through the \( n - 1 \) edges. **Remark:** This is quite special – in most cases the WHILE loops get long.

(e) Do NOT hand in -- but give it a try! In Kruskal, a student asked about using \( \text{DEPTH} \) rather than \( \text{SIZE} \). Here we show this works. When \( z \) is a root we want \( \text{DEPTH}[z] \) to be the largest \( l \) so that there is a “path” \( x_0, x_1, \ldots, x_l \) with \( x_{j+1} = \pi(x_j) \) for \( 0 \leq j < l \) and \( x_l = z \). (That is, the longest “slide down the bannister” to \( z \).) Initially all \( \text{DEPTH}[z] = 0 \). The FOR loop starts as before
\[ x \leftarrow x[i]; \ y \leftarrow y[i] \]

WHILE \( \pi(x) \neq x \)
\[ x \leftarrow \pi(x) \ (\text{\textit{*going down the stairs*}}) \]

WHILE \( \pi(y) \neq y \)
\[ y \leftarrow \pi(y) \ (\text{\textit{*going down the stairs*}}) \]

Now we use DEPTH. Flip if necessary so that \( DEPTH[x] \leq DEPTH[y] \). Then
\[ \pi(x) \leftarrow y \ (\text{\textit{redirect to bigger depth*}}) \]

IF \( DEPTH(x) = DEPTH(y) \) THEN \( DEPTH(y)++ \)

i. Show that the new value of \( DEPTH(y) \) is correct with the changed \( \pi(x) \). This has two parts.

A. If \( DEPTH(x), DEPTH(y) \) were equal, the new longest slide down the bannister to \( y \) is one more than it was.

B. If \( DEPTH(x), DEPTH(y) \) were unequal, the new longest slide down the bannister to \( y \) is the same as what it was.

Solution: All paths that had gone to \( y \) still go to \( y \). But the paths that had gone to \( x \) are now one longer. When \( DEPTH(x), DEPTH(y) \) were equal, the extension of the longest path to \( x \) now has length one more than \( DEPTH(x) \), so \( DEPTH(y)+1 \). But when \( DEPTH(x) \) was strictly less than \( DEPTH(y) \) extending these paths by one cannot make a path longer than \( DEPTH(y) \).

ii. Show (use induction on \( t \)) that if \( z \) is a root and \( DEPTH[z] = t \) then the cluster containing \( z \) (that is, the set of all \( x \), including \( z \) itself, that slide down the bannister to \( z \)) has at least \( 2^t \) vertices.

Solution: For \( t = 0 \) it is immediate. Suppose by induction that when \( DEPTH[y] = t \) its component has at least \( 2^t \) vertices. The only way \( DEPTH[y] \) can be incremented is when \( DEPTH[x] = DEPTH[y] = t \) and we reset \( \pi(x) = y \). By induction the components with \( x, y \) both has at least \( 2^t \) vertices. The new component is the union (or “merge”) of the two old components and so its size is at least twice \( 2^t \) or \( 2^{t+1} \).

iii. Deduce that the WHILE loop will have at most \( \lg(V) \) steps, \( V \) being the total number of vertices.

Solution: As SIZE is tautologically at most \( V \) we must always have \( 2^{DEPTH[z]} \leq V \) so \( DEPTH[z] \leq \lg(V) \).

(f) Consider Prim’s Algorithm for MST on the complete graph with vertex set \( \{1, \ldots, n\} \). Assume that edge \( \{i, j\} \) has weight \( (j-i)^2 \).
Let the root vertex \( r = 1 \). Show the pattern as Prim’s Algorithm is applied. In particular, Let \( n = 100 \) and consider the situation when the tree created has 73 elements and \( \pi \) and \( key \) have been updated.

i. What are these 73 elements.
   Solution: Each time the nearest element is the next element and \( S = \{1, \ldots, 73\} \).

ii. What are \( \pi[84] \) and \( key[84] \).
   Solution: The nearest element to 84 that is currently in \( S \) is 73 so \( \pi[84] = 73 \) and \( key[84] = (84 - 73)^2 = 121 \)