Data Structures II

Theoretical Homework 1

Question 1

After 0 unions, the largest set has 1 element. In order to get the largest possible set with any one union, the largest set and the second largest set will have to be joined. So the first union will make one set with 2 elements and the rest will still have 1 element, and all subsequent unions will take a set with 1 element and add it to the largest set.

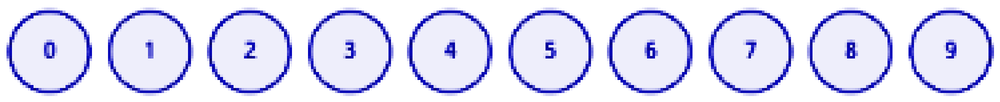
So if at any point, after *u* unions, there are *u*+1 elements in the largest set, then after *u*+1 unions, after adding a set with one element to the largest set, there will be *u*+2 elements in the largest set. That is a proof by induction.

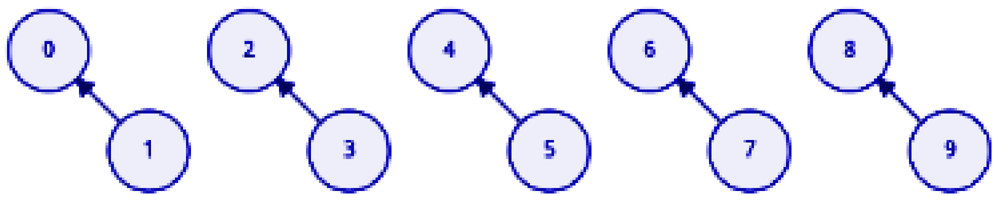
Question 2

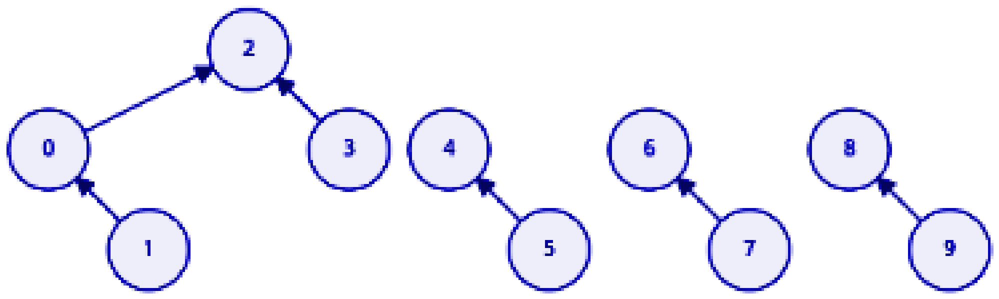
The question makes no sense. It asks to prove that there exists an algorithm that takes O(m + n lg(n)), but the question does not specify what said algorithm is supposed to accomplish. Here is my an algorithm which takes O(m + n lg(n)) time, but does not accomplish anything.

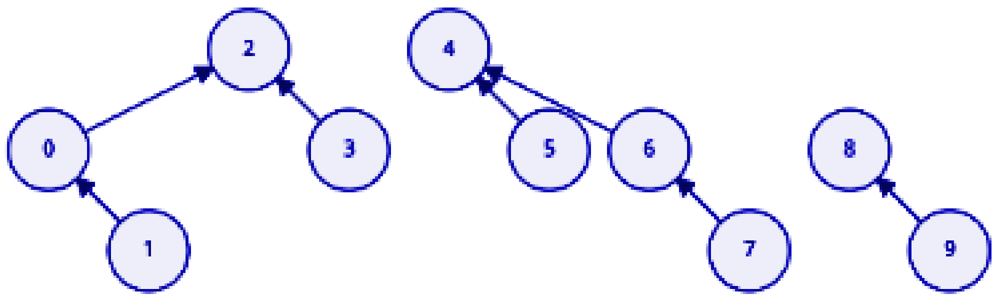
```
int n, m;
cin >> m >> n;
for(int i = 0; i < m; ++i) {}
for(int i = 0; i < n; ++i) {
      makeset();
      for(int j = 0; j < lg(n); ++j) {}
}
return 0;
```

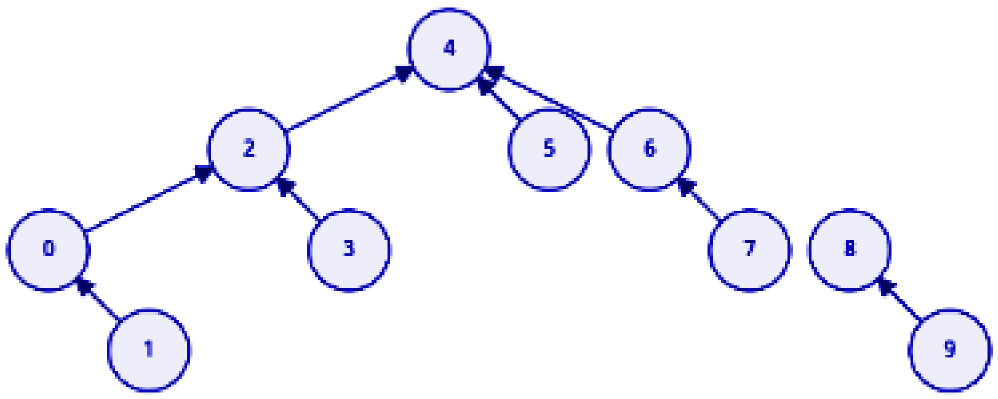
Question 3

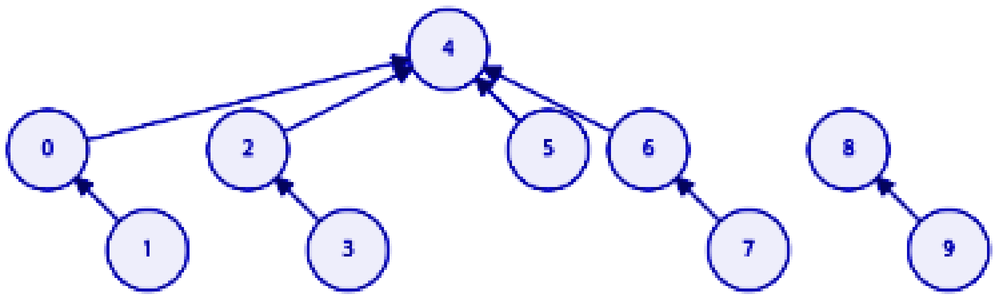












Question 4

Question 4a

Using the linked list implementation of disjoint sets, in order to make a disjoint set from a graph of nodes and connections, we must perform n make-set operations, where n is the number of elements in the graph. The make-set operation can be performed in constant time. So far we have O(n).

Then we must perform n union operations. Each union operation costs O(n/2) time. So the entire Connected-Component function takes O(n + (n/2)*n), which is equal to O(n^2).

Question 4b

Using the tree implementation of disjoint sets, in order to make a disjoint set from a graph of n nodes and connections, we must make a parent array of size n, initialising each element in said array to its index. So far we have O(n).

Then we must perform at most n union operations, each taking O(1) if union takes two set representatives, or O(lg(n)) if union takes any 2 elements. However with path compression and weighted union, this complexity is reduced to the inverse ackermann function, which is almost constant. Therefore in total, we have O(n).

Question 5

```
Template <class T>
void make_set(T n) {
	set_list<T>* set = new set_list<T>;
	set->first = new set_list_node<T>;
	set->first->value = n;
	set->first->head = set;
}

Template <class T>
set_list<T>* find_set(set_list_node<T>* n) {
	return n->head;
}

Template <class T>
void union(set_list_node<T>* n1, set_list_node<T>* n2) {
	set_list<T>* set1 = find_set(n1);
	set_list<T>* set2 = find_set(n2);
	set_list<T> *large_set, *small_set;
	if (set1->size >= set2->size) {
		large_set = set1;
		small_set = set2;
	} else {
		large_set = set2;
		small_set = set1;
	}
	large_set->last->next = small_set->first;
	large_set->last = small_set->last;
	for (auto i = small_set->begin(); i != small_set->end(); ++i) {
		(*i)->head = large_set;
	}
	delete small_set;
}
```