Problem 1

a. First, let’s prove that any node of height k has at least $2^k$ nodes. To do so, let’s prove it through induction.

Base Case: $k = 0$ In this instance, the height of the tree is 0, and this means that there’s exactly 1 element in the tree (height is measured as edges from the top-most node to the bottom layer). Thus, trivially this case holds.

I.H. Assume that for some $k$, the tree has at least $2^k$ nodes.

I.S. Now we want to show that this is the case when the height is $k + 1$.

Let us consider the union between trees a and b that created this new tree of height $k + 1$. There are three scenarios. There are three situations for this case:

Case 1: a’s height < b’s height: In this case the tree’s height would not have increased, therefore this would not be a valid case for the union that created this tree of height $k + 1$;

Case 2: a’s height > b’s height: Symmetric case as above.

Case 3: a’s height is the same as b’s height: In this situation, if their height is both the same and unioning them increased to $k + 1$, this means that each of their heights had to be $k$. Thus, using the I.H. we know that each of them have at least $2^k$ nodes. Therefore, the unioned tree must have $\geq 2^k + 2^k = 2^{k+1}$ nodes which completes the induction.

Now that we’ve proved this, let us assume for contradiction that the height of the tree was greater than $\log n$. If so, then this means that the number of nodes in the tree was $> 2^{\log n} = n$, which is clearly a contradiction. Therefore the height of the tree is upper bounded by $\log n$.

Problem 2

1) Give the contents of the $id[]$ for the Quick Union algorithm discussed in class.

2) Give the contents of the $id[]$ for the Quick Find algorithm.

Solution.

1) $id[] = [2, 8, 9, 3, 0, 9, 1, 9, 8, 9]$

2) There are many correct solutions to this. In Quick Find, all of the elements in the same connected component will have the same connected component will have the same ID as each other, and a different ID than elements outside of their component. The connected components are:

1 = \{9, 2, 7, 5, 0, 4\}
2 = \{8, 1, 6\}
3 = \{3\}

The Quick Find algorithm that we discussed in class will assign an ID to each connected component based on one of the elements within it, so a valid solution would be:

$id[] = [9, 8, 9, 3, 9, 8, 9, 8, 9, 9, 8, 9, 8, 9], \text{but not}$

$id[] = [1, 2, 1, 3, 1, 1, 2, 1, 2, 1], \text{since 1 is not an element of connected component 1, and 2 is not an element of connected component 2.}$
Problem 3

1. Worst case analysis provides a running time bound that holds for every input of length N. TRUE! Since worse case analysis looks at the input that is the most difficult for the algorithm, it stands to reason that any inputs that are easier than that difficult input should take no more time.

2. Worst case analysis is usually easier to establish than average case analysis. TRUE! The proofs for average case analysis require a good model of what average input looks like, which can be difficult, or can rely on a probabilistic guarantee (like in quicksort) which can be difficult to work through.

3. We retain lower order terms in asymptotic analysis. FALSE! We ignore lower order terms, since as N grows very large, they have little impact when trying to estimate the running time.

4. Constant factors can depend on system architecture, choice of compiler or programming language. TRUE! Our justification for dropping constant factors in asymptotic analysis is that the often depend on things that are external to the algorithm, like system architecture.

5. To establish the bounds on the class of algorithms that solve a problem, we typically implement an algorithm to establish the lower bound, and rely on a proof to establish the upper bound. FALSE! We can establish an upper bound on a problem/class of algorithms by implementing an algorithm, but for lower bounds we have to show that all imaginable algorithms must do at least that much work. The lower bound requires a proof rather than implementation.

6. Big Oh provides a good estimate of the average running time for an algorithm. FALSE! This is a common misinterpretation of Big Oh notation. A counter example is quicksort which is $O(N^2)$ but which has a much better average running time.

7. Asymptotic analysis is concerned with large values of N and can be inaccurate for small N. TRUE! Asymptotic analysis discards lower order terms, which are more dominant for small values of N than with large values. Typically we establish a threshold N0 after which our asymptotic analysis holds, but that would be a large value.

8. If an algorithm has a running time of $\Theta(N \log N)$ then:
   (a) It is $O(N \log N)$. TRUE. The definition of $\Theta$ is $O = \Omega$.
   (b) It is $\Omega(N \log N)$. TRUE. The definition of $\Theta$ is $O = \Omega$.
   (c) It is optimal. FALSE. Just because we have established a tight bound on the running time of an algorithm does not mean it is optimal. For example, selection sort is $O(N^2) = \Omega(N^2) = \Theta(N^2)$ but it is clearly not an optimal sorting algorithm because others like merge sort do better than it does.

9. If the lower bound on the class of algorithms that solve a problem is an algorithm $\Omega(N^2)$, and an algorithm in that class is $O(N^2)$ then:
   (a) The algorithm is $\Theta(N^2)$. TRUE. It must have an $\Omega(N^2)$ (as described below) so therefore it is $\Theta(N^2)$.
   (b) The algorithm is $\Omega(N^2)$. TRUE. If there is a lower bound on the class of algorithms and this algorithm is in that class, then it $\Omega$ is the same as for the class.
   (c) The algorithm is optimal. TRUE. Its worse case time is equal to the lower bound for the class of algorithms, so that is the best that is possible, and it is optimal.

10. If two algorithms are equivalent in terms of Big Oh, then they will be equivalent in Tilde notation. FALSE. Tilde notation keeps the leading constants, so we can differentiate between algorithms with the same BigOh. For example, two algorithm might be $O(N)$ but one could be $1000N$ and the other could be $2N^2$.
Problem 4

Evaluating the Claims of A Startup Company

Facebook has hired you as a special advisor to Mark Zuckerberg. Congrats - taking CIS 121 really paid off. Mark considers acquiring a startup in stealth mode. The company claims to have invented a new algorithm that will use polynomial time to solve a problem previously thought to be solvable in exponential time. The company refuses to release its code because it is worried that it will be stolen. They will allow you to do black box testing by sending whatever inputs you want to their server. You get back the output, and you can time how long it takes to run. Mark asks you to test it on the double, so you sketch out this chart.

<table>
<thead>
<tr>
<th>Input size</th>
<th>Response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>2.9</td>
</tr>
<tr>
<td>128</td>
<td>24</td>
</tr>
<tr>
<td>256</td>
<td>188</td>
</tr>
<tr>
<td>512</td>
<td>1503</td>
</tr>
<tr>
<td>1024</td>
<td>12026</td>
</tr>
</tbody>
</table>

a. What is your estimate of the order of growth of the company’s algorithm?

b. How did you arrive at this estimate?

Solution.

a. The table of ratios should look like this.

<table>
<thead>
<tr>
<th>Input size</th>
<th>Response time</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>2.9</td>
<td>-</td>
</tr>
<tr>
<td>128</td>
<td>24</td>
<td>8.27</td>
</tr>
<tr>
<td>256</td>
<td>188</td>
<td>7.883</td>
</tr>
<tr>
<td>512</td>
<td>1503</td>
<td>7.994</td>
</tr>
<tr>
<td>1024</td>
<td>12026</td>
<td>7.999</td>
</tr>
</tbody>
</table>

b. Cubic.

c. We can use the doubling rule to get an estimate of the exponent for the order of growth by taking the \( \log_2 \) of the ratio of times on doubled inputs. The ratio seems to approach 8 in our experiments, so the order of growth is \( \log_2(8) = 3 \), which means cubic order of growth.

Students should type out a table that looks like this to show the ratio by the running time.