Name ______________________________

Quiz - Module 4: Graphs (BFS / DFS)

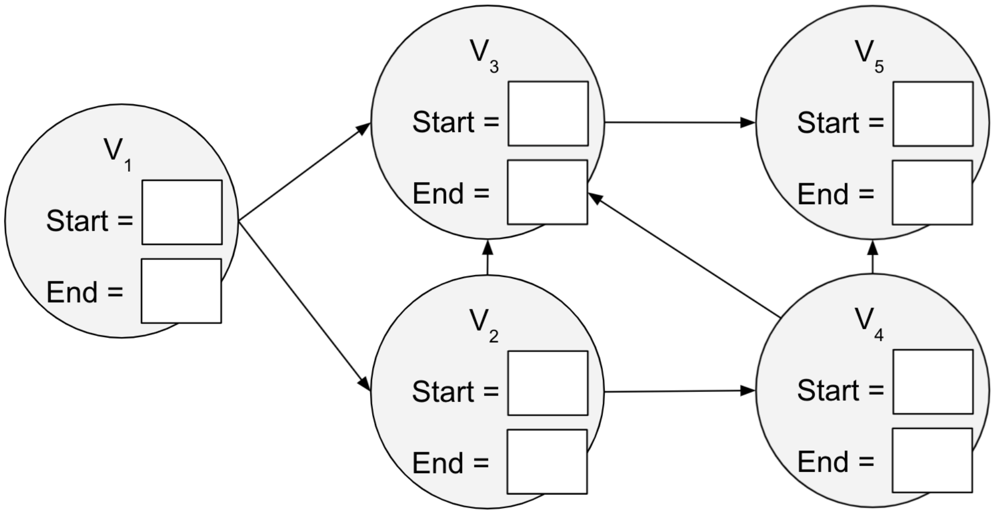
1. [8 points] Answer the following True/False questions regarding *topics from module 4.*

When running *BFS*, a node has color *gray* while that node is waiting on the Queue.	**True**	**False**
The runtime of *BFS* is $\Theta(V * E)$	**True**	**False**
To change the *BFS* code into *DFS*, it is sufficient to only change the internal Queue into a Stack.	**True**	**False**
DFS, run from one single arbitrary start node, is guaranteed to visit every node on a **directed** graph (assume the graph is weakly-connected, meaning the underlying undirected graph is connected).	**True**	**False**
DFS, run from one single arbitrary start node, is guaranteed to visit every node on a connected, **undirected** graph.	**True**	**False**
DFS finds a back-edge in a directed graph iff an outgoing edge while traversing leads to a *black node*.	**True**	**False**
DFS can produce a topological ordering of nodes in a directed graph simply by sorting the nodes in increasing finish times.	**True**	**False**
The runtime of *DFS* is asymptotically better than *BFS*.	**True**	**False**

2. [2 points] Which of the following are true statements about strongly connected components in a digraph G? In the questions below, S is a strongly connected component of G. *Circle all that apply. You only have to circe the little dot.*

- If a node v in S (and its incident edges) is removed from G, there must still be a path connecting any pair of the remaining nodes in S.
- If we start a recursive DFS search on any node in S, its guaranteed that it will visit every node in S.
- If a call to recursive DFS is started on a node v in S, it will only visit nodes in S and not visit any nodes in another connected component.

3. [5 points] Run DFS on the following graph. List start and finish times (beginning at $t = 1$) in the boxes in each node. Use V_1 as your start node. *When multiple nodes can be searched, always search neighboring nodes in increasing order (e.g., if V_2 and V_3 are both adjacent to the current node, search V_2 first).*



This question will ask you to think about the execution of *BFS*. Suppose the graph $G = (V, E)$ (assume undirected, connected) contains an edge $e = (u, v) \in E$ between two nodes u and v. This question will ask you to think about the relative values $u.d$ and $v.d$ (the distances *BFS* calculated to these two nodes). Assume *BFS* is implemented correctly, and answer the following questions:

4. [2 points] Is it possible for $u.d == v.d$? Explain in one sentence why or why not OR draw a simple graph to illustrate.

5. [2 points] Is it possible for $|u.d - v.d| == 1$ (i.e., $u.d$ and $v.d$ differ by exactly one)? Explain in one sentence why or why not OR draw a simple graph to illustrate.

6. [2 points] Is it possible for $|u.d - v.d| == 2$ (i.e., $u.d$ and $v.d$ differ by exactly two)? Explain in one sentence why or why not OR draw a simple graph to illustrate.

Name ____

Quiz - Module 5: Kruskal's and Find-Union

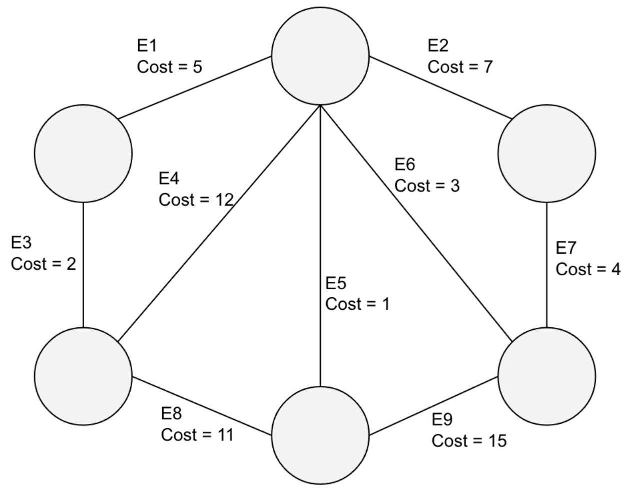
1. [6 points] Answer the following True/False questions regarding *Minimum Spanning Trees and Kruskal's Algorithm*

All undirected, connected graphs have at least one *spanning tree*.	**True**	**False**
When implementing *union(i,j)* in an array-based find-union, it does not matter whether i takes the label of j or the other way around (for correctness).	**True**	**False**
If we run Kruskal's algorithm and the graph G has a unique edge at minimum weight, then that edge is guaranteed to be in the minimum-spanning tree.	**True**	**False**
Kruskal's algorithm works by adding nodes of G to a priority queue and removing one per loop iteration.	**True**	**False**
The Find-Union's *find()* method is worst-case linear-time because we may to trace a linear amount of "steps" to find the label.	**True**	**False**
The Find-Union's *union()* method is worst-case linear time if given the labels of the two sets as the two parameters (rather than two arbitrary items that belong to the set).	**True**	**False**

2. [4 points] Answer the following short questions which ask about runtimes, sizes of graphs, etc.

Given a connected, undirected graph $G = (V, E)$, how many edges does any *spanning tree* of G contain (in terms of $\|V\|$ and/or $\|E\|$)?	
Suppose I have a **complete**, undirected graph $G = (V, E)$ with exactly four nodes. How many unique *spanning trees* does G have (we want the exact number)?	
What is the runtime of Kruskal's algorithm if find() and union() are $\Theta(V)$ time (as in the naive implementation)	
What is the runtime (expected) of Kruskal's algorithm when using the optimal implementation (path compression plus union-by-rank) of the find-union data structure?	

3. [3 points] Run Kruskal's algorithm on the given graph. List the order in which the edges are added to the MST. List an edge by its provided label.



Consider the following recursive implementation of *find(x)* on an array-based find-union data structure.

```
/*The internal array*/
int[] sets;

/*returns the integer label of the set that int x belongs to*/
int find(int x){
        if(sets[x] == x)                //line 1
                return x;               //line 2

        return find(x+1);               //line 3
}
```

4. [1 points] One of the three lines listed above contains a bug. Which one is it?

5. [2 points] Rewrite this one line to fix the bug.

6. [2 points] Now rewrite one of the three lines to add the *path compression* optimization to the code. Clearly list which line you are changing AND the new line of code. If you want to split the line into two lines of code, that is fine but you may not write three or more lines.

NOTHING BELOW THIS POINT WILL BE GRADED. USE THE SPACE BELOW FOR SCRATCH WORK