Personal Information:

Name: Key

Student ID #: _____

-You have 50 minutes to complete this exam
-This exam is closed note. You may not use any computing devices including calculators. Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
-If you write your answer on scratch paper, please clearly write your name on every sheet and write a note on the original sheet directing the grader to the scratch paper. We are not responsible for lost scratch paper or for answers on scratch paper that are not seen by the grader due to poor marking.

Problem	Description	Earned	Max
1	Runtime Analysis		9
2	AVL Trees		9
3	Short Answer		9
4	True False		9
5	Up Trees		8
6	Topological Sort		6
7	Design Decisions		15
8	DFS/BFS		10
TOTAL	**Total Points**		**75**

1) Describe the worst case running time of the following code in "big-Oh" notation in terms of the variable n. You should give the tightest bound possible.

(a)

```
void f1(int n) {
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         for (k = 0; k < n; k++) {
            System.out.println("Alright");
         }
         for (k = 0; k < n; k++) {
            System.out.println("Alright");
         }
      }
   }
}
```

O(n) · O(n) [O(n+n)] = O(n³)

(b)

```
int f2(int n) {
   if (n > 100) {
      return 10;
   } else {
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j = j * 2) {
            System.out.println("Alright");
         }
      }
   }
}
```

for all large n's = O(1)

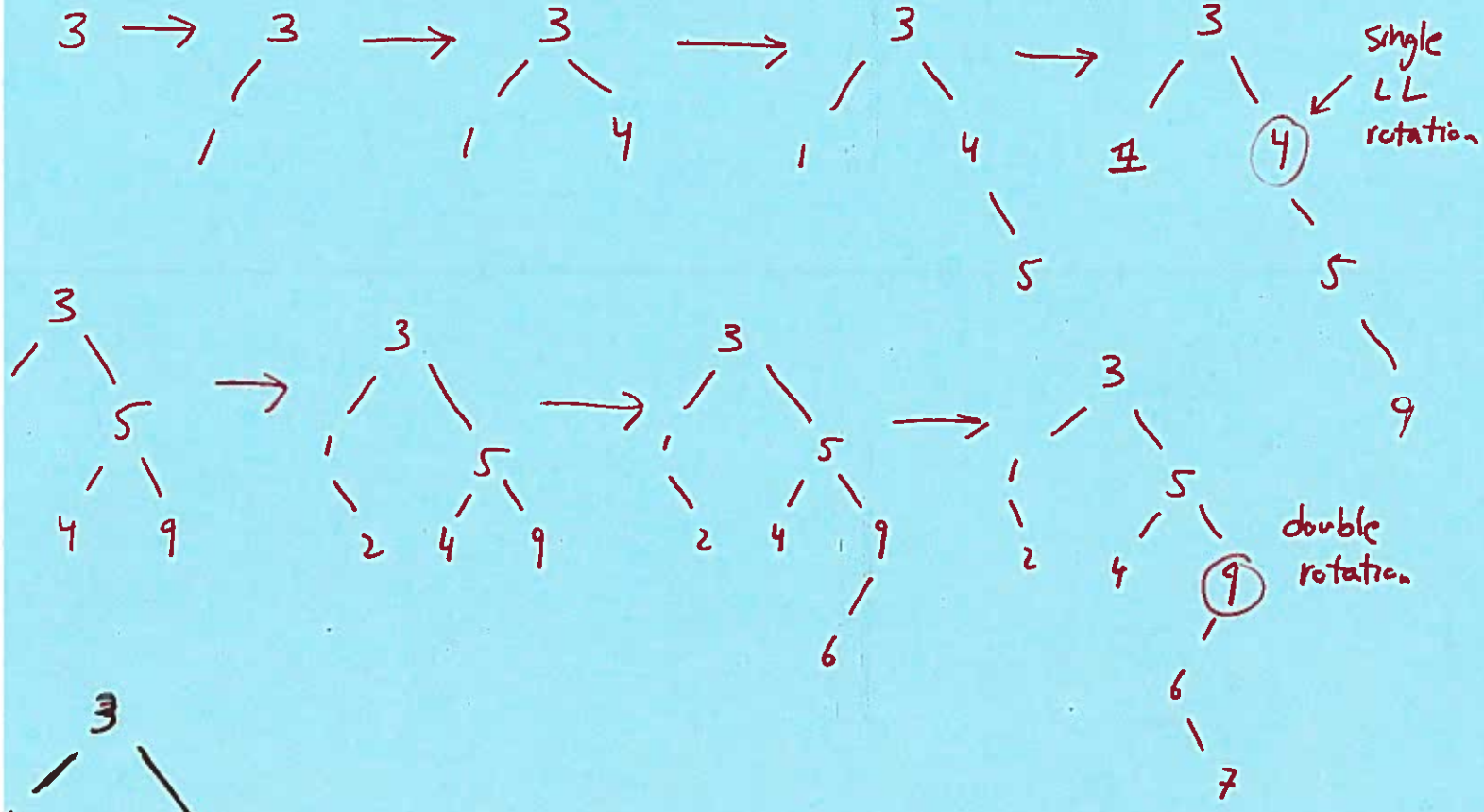
(c)

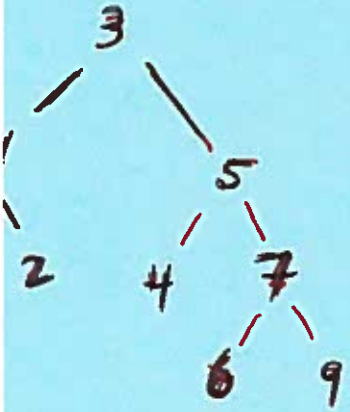
```
int f3(int n) {
   if (n < 10) {
      System.out.println("!");
      return n+3;
   } else {
      return f3(n-1) + 1;
   }
}
```

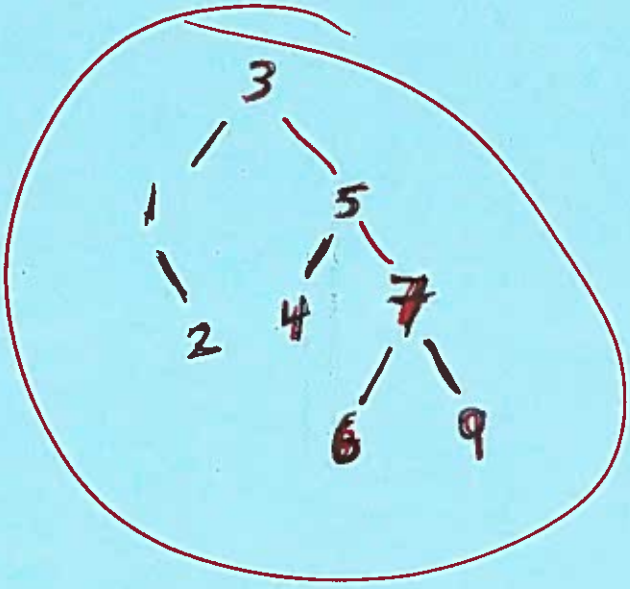
O(n)

2. AVL Trees

Draw the AVL tree that results from inserting the **keys 3, 1, 4, 5, 9, 2, 6, 7** in that order into an initially empty AVL tree. You are only required to show the final tree, although if you draw intermediate trees it may help us award partial credit. **Please circle your final result for ANY credit.**







3) Short Answer

a) List 2 collision resolutions schemes for hash tables other than separate chaining. For each of alternative schemes, state one of its disadvantages. Please be specific, convince me you understand ☺

1) Linear probing. Primary clustering when similar values hash to same location. causes O(n) lookups and adds.

2) Rehasing. Have to maintain multiple hash functions cant have a very large load factors, so resizing has to take place frequently.

b) What the difference between an abstract data type (ADT) and an implementation of an ADT? Provide an example of both.

An ADT is an abstract language independent entity that represents a type of structure. An implementation is a language specific, algorithm specific way of implementing a type of ADT.

ADT: priority Queue, Stack, List Implementation: MinHeap, linked list array, HashTable

4) True or False, circle one

a) Every AVL tree is a complete binary tree. (TRUE or FALSE)

b) Every binary search tree (BST) has a height bounded by log(n), where n is the total number of nodes. (TRUE or FALSE)

c) Adding an element to a AVL tree has a worst-case runtime of O(log(n)) and an amortized runtime of O(1). (TRUE or FALSE)

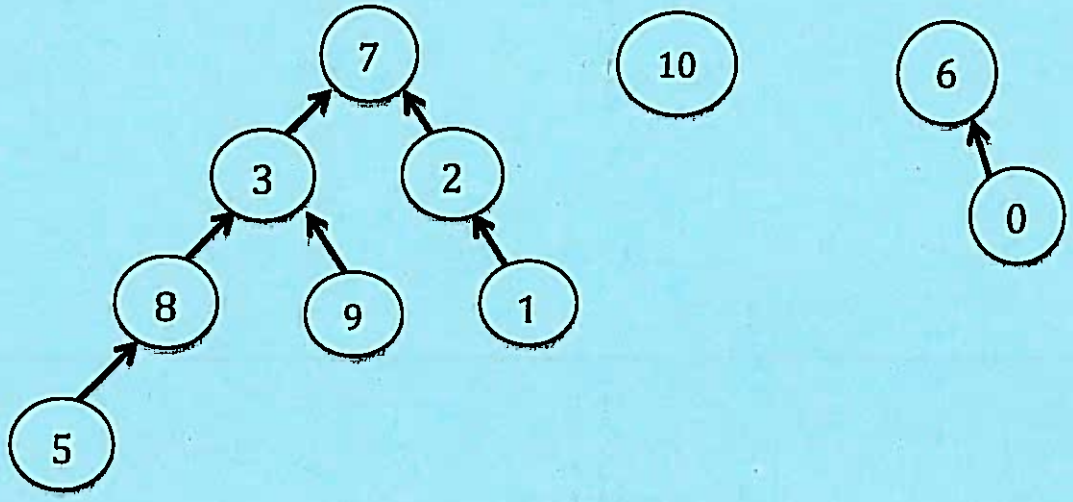
d) The number of nodes in each sub-tree of an AVL tree differ by at most one. (TRUE or FALSE)

e) Running breadth first search from a given node of an unweighted graph will be guaranteed find the shortest path between that node and all other nodes. (TRUE or FALSE)

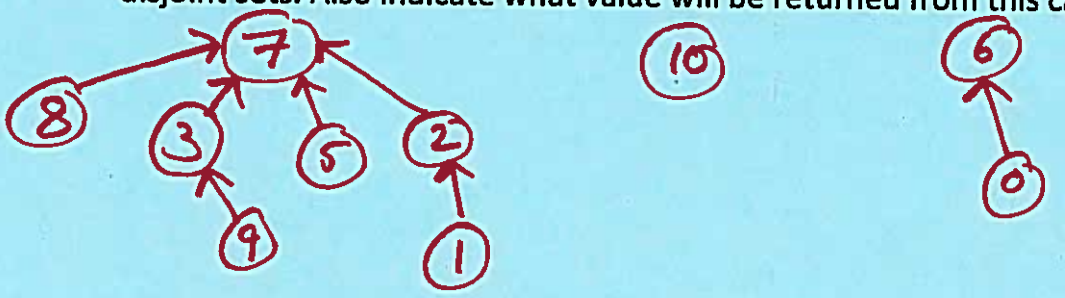
f) Running depth first search from a given node of an unweighted graph will be guaranteed to find the shortest path between that node and all other nodes. (TRUE or FALSE)

5) Up Trees

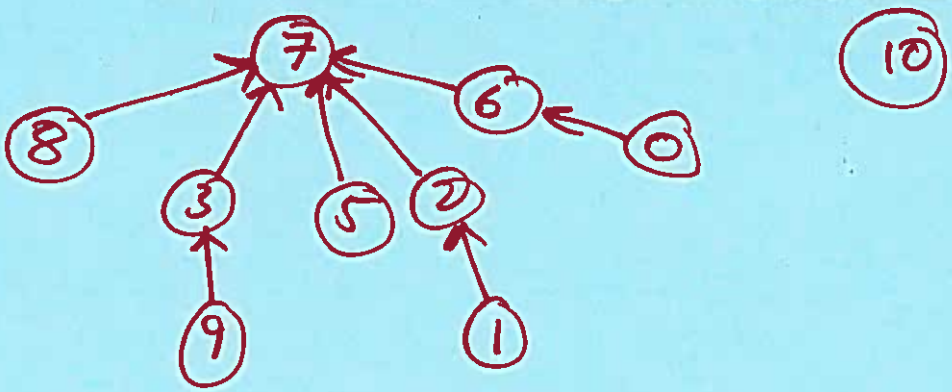
Given the following up trees from our implementation of disjoint sets seen in lecture, answer the following questions: (assuming both weighted-unions and path compression)



a) Draw the resulting compressed tree(s) after calling find(5) on the above disjoint sets. Also indicate what value will be returned from this call.



b) Draw the final result of calling `union(2,6)` on the result of part a

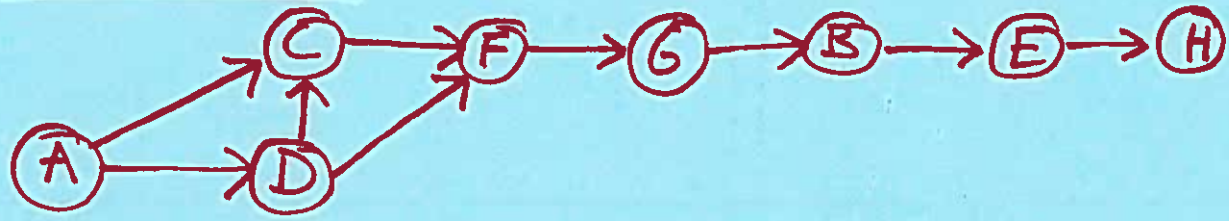


6) Topological Sort

Draw a directed graph with the following requirements:

1) Topological ordering: **A, D, C, F, G, B, E, H**
2) At least two of the vertices in this graph must have an **in-degree > 1**
3) At least two of the vertices in this graph must have an **out-degree > 1**

One example:



7) Design Decisions

For each of the following tasks, choose the most efficient structure in terms of time complexity. You may choose from the following: **sorted array, sorted linked list, binary search tree, AVL tree, min heap, up tree, stack implemented with a linked list, queue implemented with an array.** Explain your decision.

a) Your task is to store a directory of students who go to UW. Important operations include the ability to add a student to the directory, determine whether a student goes to UW (based on name), and ability to print all the students in sorted order. You may assume that no two students have the same name.

AVL tree. AVL trees allow for fast sorted insert O(log(n)) as well as fast lookups O(log(n)). In addition AVL trees can be printed in sorted order in O(n). A sorted array or LL would require O(n) insert.

b) Your task is to keep track of 50 servers, each with an ID from 1 through 50. Important operations include being able to turn a server on and off based on ID.

Sorted Array. Since we have a 1 to 1 correspondence of servers to indices, sorted array gives us O(1) lookup.

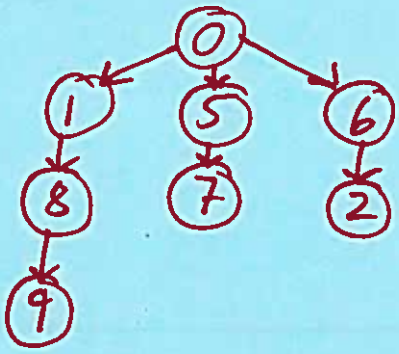
c) Your task is to keep track of the 100 best scores in an online video game. The only operations that must run fast are adding a new score and the retrieval of the n'th best score. For instance, if the following scores are stored: 100, 70, 50, 65, 20, 90 and I ask for the 2nd best score, 90 would be returned.

Sorted Array. Index directly to nth largest by going to arr[n + 1]. Adding new score requires O(n) shift, but every other operation constant time.

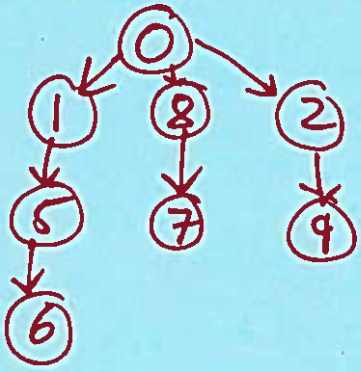
AVL would require very long runtime to find nth largest since there is no way to directly find nth largest in a tree.

8) DFS/BFS

a) Draw a **directed acyclic graph** that would produce the following traversal with the given restrictions:
Breadth first search traversal: **0, 1, 5, 6, 8, 7, 2, 9**
Longest path in graph must be **3**



b) Draw a **directed acyclic graph** that would produce the following traversal with the given restrictions:
Depth first search traversal: **0, 1, 5, 6, 8, 7, 2, 9**
Longest path in graph must be **3**



EXTRA CREDIT: Draw a **directed acyclic graph** that would produce BOTH of the following traversals:
Breadth First Search: 0, 1, 5, 6, 8, 7, 2, 9
Depth First Search: 0, 1, 6, 8, 9, 7, 5, 2