Data Structures – Final Review Homework 2

Name _____ Student ID _____

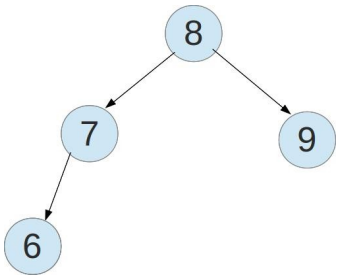
Assigned: **June 8, 2019**
Due: **June 13, 2019**

1) **Insert** the following sequence of numbers, in order, into a **Splay Tree** , and draw the resulting tree: **1, 2, 3, 4, 5, 6, 7**

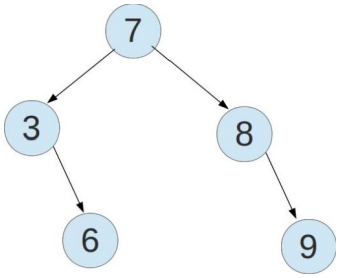
2) **Search** for the **number 3** in the previous Splay Tree. Do not forget, after searching, to perform rotations. Show the final tree.

3) **Search** for the **number 2** in the previous Splay Tree. Do not forget, after searching, to perform rotations. Show the final tree.

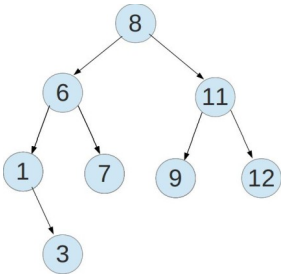
4) **Insert the** number 4 **into each** Splay tree. Do not forget, after inserting, to perform rotations. Draw the final tree.



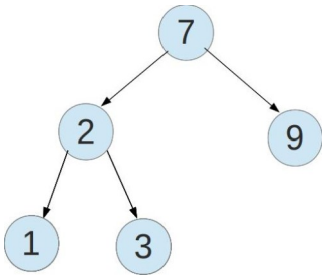
⇒



⇒



⇒



⇒

5) **Insert** the sequence of numbers **7, 20, 2, 16, 9, 34, 31, 28, 13, 25** into an empty **Splay Tree**. Draw the tree before each Splay, and draw the final tree.

6) **Delete** the **number 7** from the previous Splay Tree, and draw the resulting tree

7) Is this tree **always** balanced, or **sometimes** balanced? Is there an **extra variable** stored in a node of this tree, other than **data**, **left**, and **right**? If so, what?

Tree	Balanced?	Extra Variable?
Binary Search Tree		
AVL Tree		
Splay Tree		
Leftist Heap		

8) Let **array1** be an **unsorted** array. When array1 is full its memory **increases by 5**.
Let **array2** be an **unsorted** array. When array2 is full its memory **doubles**.
Let **array3** be a **sorted** array. When array3 is full its memory **doubles**.
Let **tree1** be a Binary Search Tree (**BST**)
Let **tree2** be an **AVL** Tree
Let **tree3** be a **Splay** Tree
Let *n* be the **number of items** in the data structure. What are the **Best Case**, **Average**, **Amortized, and Worst Case** Running Times for each operation?

Consider every possible situation when determining **Best Case** and **Worst Case**.
An Array might have **extra memory** or might **not**.
A Tree might be **balanced** or might be **unbalanced** (if that is possible).

	Best Case	**Average**	**Amortized**	**Worst Case**
insert(**array1**, num)				
search(**array1**, num)				
insert(**array2**, num)				
insert(**array3**, num)				
search(**array3**, num)				
insert(**tree1**, num)				
search(**tree1**, num)				
insert(**tree2**, num)				
search(**tree2**, num)				
insert(**tree3**, num)				
search(**tree3**, num)				

9) Insert the sequence of numbers { **49, 6, 38, 29, 34, 24, 21, 18, 5, 16** } into an initially empty **Binary Heap**. Show the heap after each insertion.

10) Perform **DeleteMin**() on the result from the previous question **twice.** Show the new heap after each DeleteMin().

11) Show the **Leftist Heap** that results from performing the following operations. Show the heap after each call to Insert(), or after each recursive call to Merge():

a. **Insert** the numbers **9, 8, 7, 6** in order into an empty **Leftist Heap**

b. **Insert** the numbers **1, 2, 3, 4, 5, 10, 11, 12** into an empty **Leftist Heap**

c. **Merge** these two Leftist Heaps.

12) Draw a **Binomial Queue** containing 22 elements. Do not write any data, but draw every node.

13) Begin with an empty **Binomial Queue**

a. Insert the numbers { 49, 6, 28, 39, 34, 14, 21, 18, 7, 4 } into an empty Binomial Queue in order. Show the Binomial Queue after each insertion.

b. Perform **DeleteMin()** on the result of the previous question **three** times. Show the Binomial Queue after each deletion.

14) Show the **Fibonacci Heap** that results from performing the following operations. Show the heap after each call to Insert().

a. **Insert** the numbers **9, 8, 7, 6** in order into an empty **Fibonacci Heap**

b. **Insert** the numbers **1, 2, 3, 4, 5, 10, 11, 12** into an empty **Fibonacci Heap**

c. **Merge** these two Fibonacci Heaps.

d. Perform **DeleteMin** on the above Fibonacci Heap

15) What are the **advantages** of the **Fibonacci Heap** over the Binomial Queue?

16) What are the **advantages** of the **Binomial Queue** over the Fibonacci Heap?

17) What is the **best case**, **average**, **amortized**, and **worst case** running time of **Insert()**, **DeleteMin()**, and **Merge()** in each Priority Queue?

	Binary Heap			
	Best Case	Average	Amortized	Worst Case
Insert()				
DeleteMin()				
Merge()				
	Leftist Heap			
Insert()				
DeleteMin()				
Merge()				
	Binomial Queue			
Insert()				
DeleteMin()				
Merge()				
	Fibonacci Heap			
Insert()				
DeleteMin()				
Merge()				

18) For each data structure, does **Decrease Key** use **Percolate Up**, or does Decrease Key **Detach** the subtree and **Merge** it with the original Heap?

Priority Queue	**Decrease Key** or **Detach and Merge**
Binary Heap	
Leftist Heap	
Binomial Queue	
Fibonacci Heap	

19) For each data structure, what is found in the C definition of a node?

a. typedef struct AVLTREENODE {

} AVLTreeNode

b. typedef struct LEFTISTHEAPNODE {

} LeftistHeapNode

c. typedef struct BINOMIALTREENODE {

} BinomialTreeNode

d. typedef struct FIBONACCIHEAPNODE {

} FibonacciHeapNode

20) A single operation has the following running times:

Single Operation			
Best Case	Average Case	Amortized	Worst Case
Θ(1)	Θ(logn)	Θ(logn)	Θ(n)

The operation is **repeated n times**. What is the **total time** for all **n** operations:

Total Time for n operations		
Best Case	Average Case	Worst Case
Θ(_____)	Θ(_____)	Θ(_____)

21) An operation has the indicated running times. If the operation is performed **n** times, the **worst case** can only happen **at most the given number of times**. What is the Amortized time?

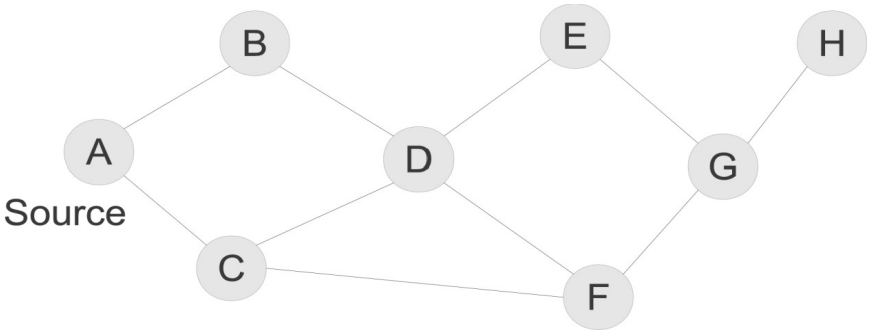
Best Case	**Average Case**	**Worst Case**	**Frequency of Worst Case**	**Amortized**
Θ(1)	Θ(logn)	Θ(n)	One time	Θ(_____)
Θ(1)	Θ(logn)	Θ(n)	100 times	Θ(_____)
Θ(1)	Θ(logn)	Θ(n)	Log n times	Θ(_____)
Θ(1)	Θ(logn)	Θ(n)	n/2 times possible	Θ(_____)
Θ(1)	Θ(logn)	Θ(n)	Always possible, but unlikely	Θ(_____)

22) Give **four** examples of each kind of data structure:

Searchable Data Structure	**Priority Queue**

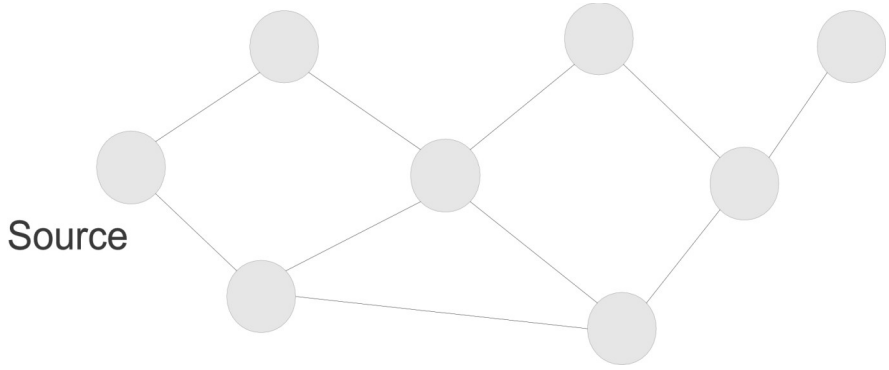
23) Run Breadth First Search (BFS) on this graph.

a. Show the frontier at each step. The first two rows have been filled in:



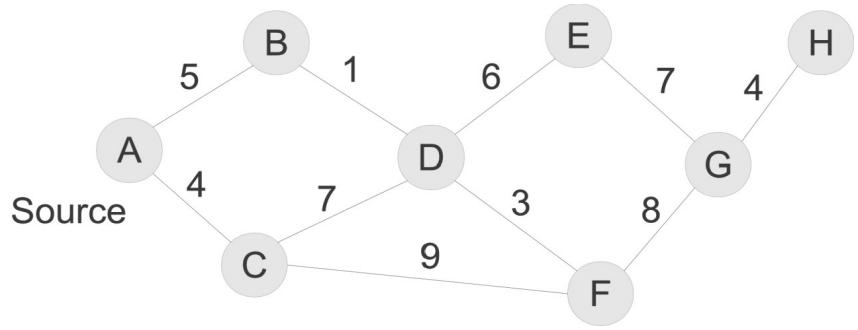
Node Removed	Distance to Node Removed	Frontier
- A	- 0	A-0 B-1, C-1

b. Fill in the **final distances** to each node



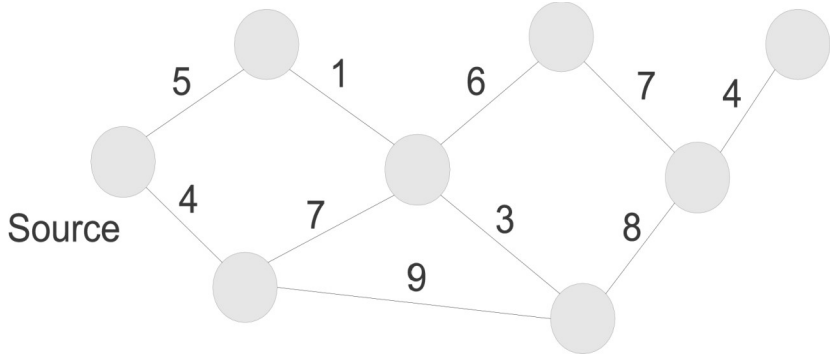
24) Run **Dijkstra's Algorithm** on this graph.

a. Show the frontier at each step. The first two rows have been filled in:



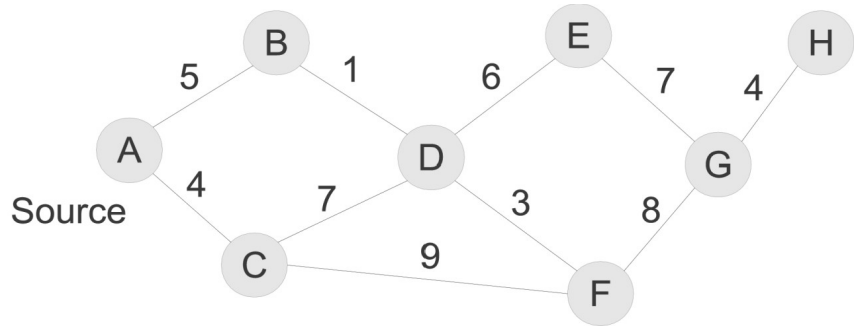
Node Removed	Distance to Node Removed	Frontier
- A	- 0	A-0 C-4, B-5

b. Fill in the **final distances** to each node



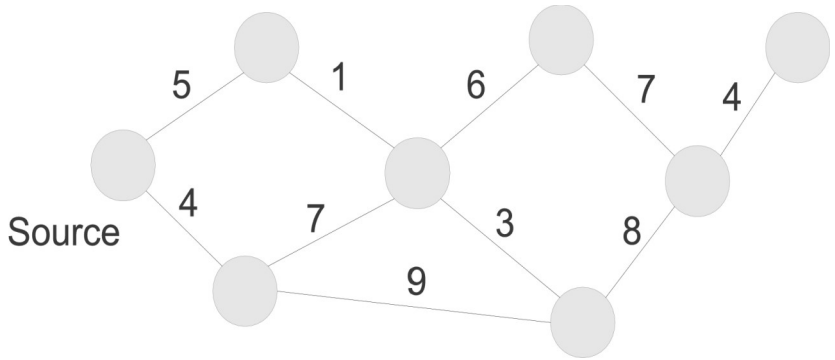
25) Run **Prim's Algorithm** on this graph.

a. Show the frontier at each step. The first two rows have been filled in:



Edge Removed	Node Visited	Frontier
- AC	A C	AC-4, AB-5 AB-5, CD-7, CF-9

b. **Circle the edges** in the Minimum Spanning Tree (MST)



26) Let n be the number of nodes, and m be the number of edges in a graph:

While running **Breadth First Search (BFS)**,

a. How many times (at most) is **Enqueue** performed?

b. How many times (at most) is **Dequeue** performed?

c. How many **neighbors** are tested (at most), to check if Enqueue is necessary?

While running **Dijkstra's Algorithm**,

d. How many times (at most) is **Insert** performed?

e. How many times (at most) is **DeleteMin** performed?

f. How many **DecreaseKey** operations (at most) are performed?

While running **Prim's Algorithm**,

g. How many times (at most) is **Insert** performed?

h. How many times (at most) is **DeleteMin** performed?

i. How many **neighbors** are tested (at most), to check if Insert is necessary?

27) What is the **total time** to perform the operation with the given data structure:

Operation	**Frontier Data Structure**				
	Queue	Binary Heap			Fibonacci Heap
		Best Case	Average	Worst Case	
Insert All Nodes					
Delete All Nodes					
Insert All Edges					
Delete All Edges					
Dijkstra's Algorithm	XXXXXX				
Prim's Algorithm	XXXXXX				