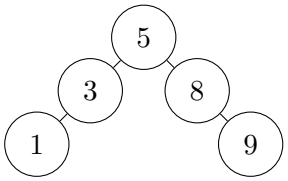
Final Exam Review

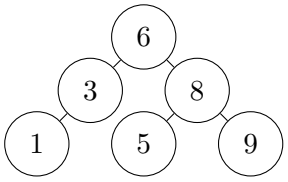
AVL Trees

1. Insert the following numbers, in order from left to right, into a Binary Search Tree and an AVL Tree.
   (a) Tree 1: [10, 15, 18, 5, 7, 21, -3]
   (b) Tree 2: [1, 2, 3, 4, 5, 6]
2. Are the following trees valid AVL trees?

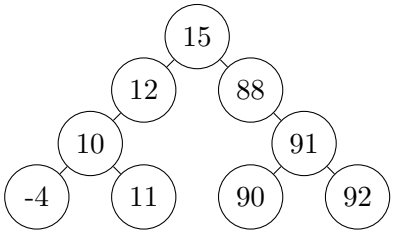
(a)



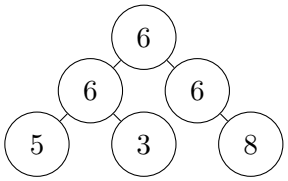
(b)



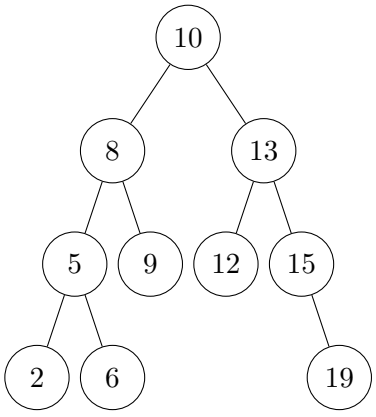
(c)



(d)

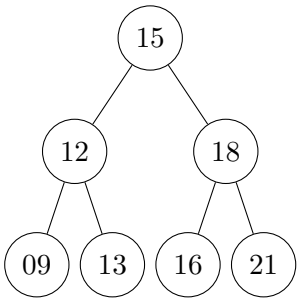


(e)



(f) $\emptyset$

3. Remove from the following AVL Tree in order left to right: [21, 18, 16, 9]



4. Runtime Questions: Fill in the worst-case runtime in big-O notation; then for all the below, describe the tree that gives the worst case runtime.

	BST	AVL Tree
insert		
delete		
find		
findMax		
findMin		

Heaps

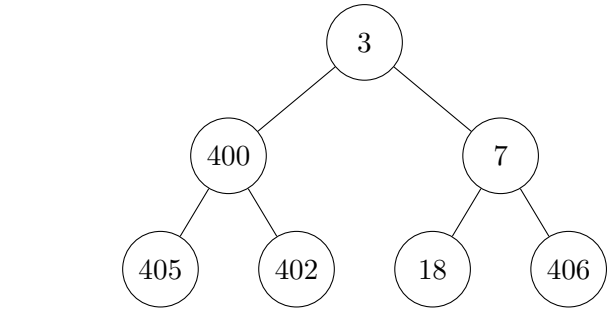
1. Runtime Questions: Fill in the worst-case runtime in big-O notation.

	Min Heap with n elements
remove min	
insert	
build heap	

2. What are the heap invariants?

3. When we store heaps in an array, how do we find the parent of a node at the index `i`? How do we find the children?

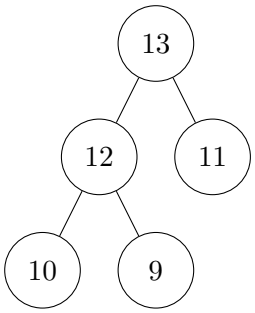
4. Convert the following min heap tree into an array (leave index 0 empty! (why?))



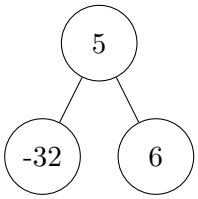
∅									

5. Are the following valid MAX heaps?

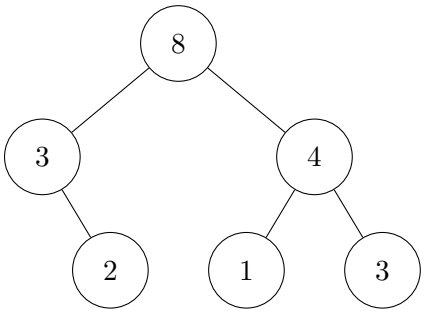
(a)



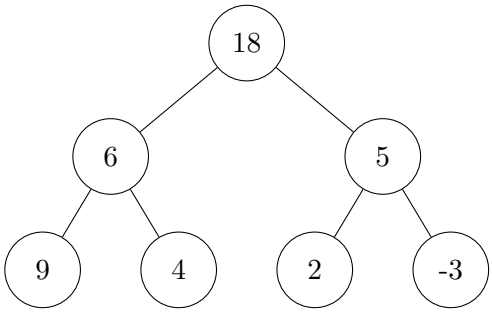
(b)



(c)



(d)



(e)

X	30	15	6	2

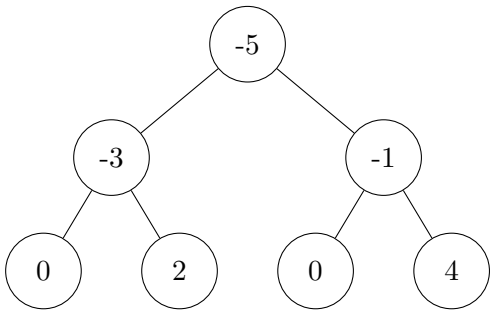
(f)

X	90	100	5

(g)

X	30	15	18	12	3	9	14	13		

6. Consider this min heap:



X	-5	-3	0	2	-1	0	4			

   Say we insert a new element with value -25.

   - Where in the tree/array does this value initially go? What invariant are we fixing?
   - After step 1, what invariant do we need to fix? How do we do that?
   - Try removing the new root. Repeat steps (1-2).
   - What are the runtimes of these operations?

7. Heap sort

   - Describe how to sort a heap stored in an array. What are the time/space complexities?
   - How can you sort an unordered vector *in place* using the heap sort strategy? Complexity of this operation?
   - Why is a heap used in Dijkstra's algorithm?
   - What is another application of a priority queue (heap)?

Hash Tables

1. What is a hash function?
2. What makes a good hash function?

3. Describe the sequence of events from getting a key to insertion into an empty table of size 10.

4. What is the worst case runtime of insertion into a hash table using chaining (linked list)? Describe the table that causes this runtime.

5. What is the worst case runtime of (successful) insertion into a hash table using linear probing (a form of open addressing)? Describe the table that causes this runtime.

6. What is the best case/desired runtime of insertion, deletion, and search? Describe, for both types of collision resolution, what the table looks like, and what the runtime is.

7. When and why should we expand a hash table?

8. Consider the hash function, `h(key) = key`, where `key` is a positive integer. Ex: `key=35; h(key) = 35`.

   Insert the following numbers into a table, in order: [5, 6, 13, 10, 15, 20].

   - Table 1: use chaining. Expand when the load factor `(n/m) > 0.5`, where `n` is the number of elements, and `m` is the number of buckets.

     table 1:

   - Table 2: Use linear probing. Expand when the load factor `n/m > 0.75`

     table 2:

   - Why do we need to re-hash when we expand?

Hash Table Bonus Questions

1. Consider a hash table that uses a more advanced form of chaining. Instead of linked lists, it stores an AVL tree based on the key.

   (a) What is the new worst case runtime of the following operations?

insert(key)	
search(key)	
delete(key)	
deleteTable()	

   (b) Why?

   (c) Why don't we always use this approach?

2. Consider a hash table which uses a modified linear probing collision resolution approach. `h(key, i) = key + 2i;`, where `i` is the attempt number - 1.

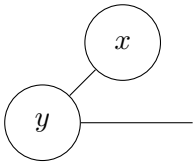
   Given the following table, what order could the numbers have been inserted in?

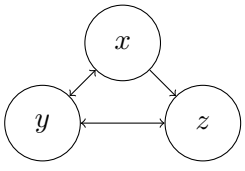
20	24	30		2

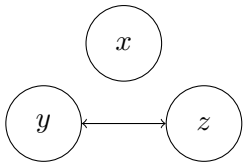
Graphs

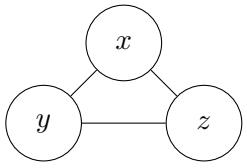
1. Are the following valid graphs?

   (a) 

   (b) 

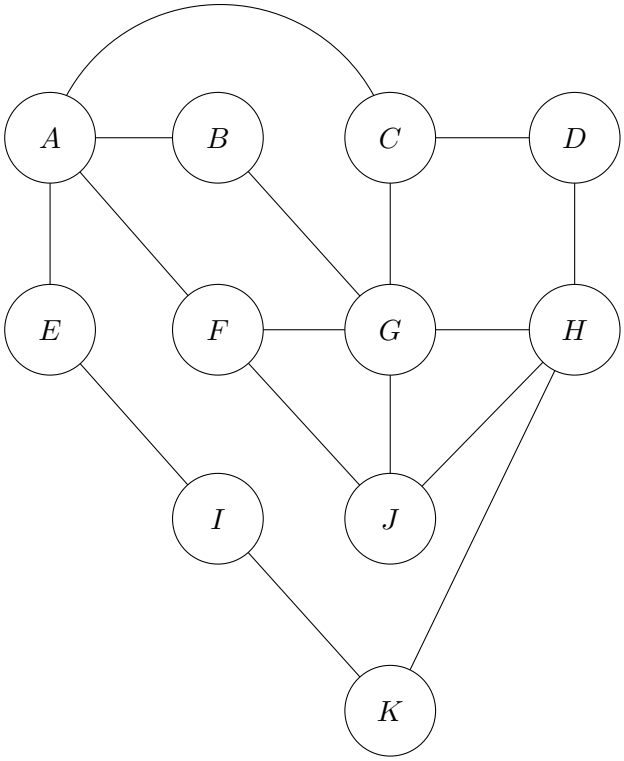
   (c) 

   (d) 

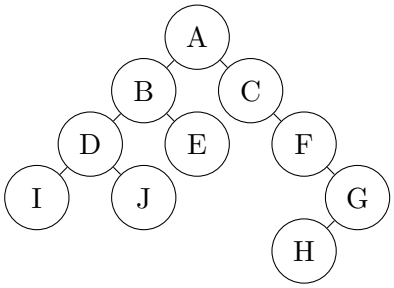
   (e) 

   (f) No nodes

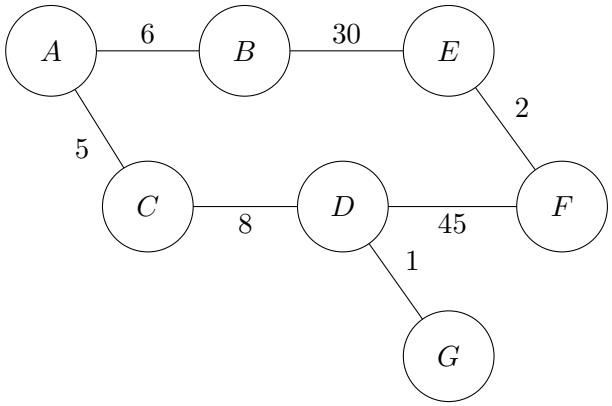
2. BFS - Find the shortest path from S to A, report the distance and path.



3. DFS - Run DFS on the graph. Report the shortest path you see.
4. What is the runtime of the following and why?
   (a) BFS
   (b) DFS
   (c) Dijkstra's Algorithm
5. Run an in-order traversal on the following tree.



6. Is the tree above a graph?
7. Is the search we did BFS, DFS, or Dijkstra's algorithm?
8. Explain how to run both BFS and DFS on the tree?
9. Can a BFS report the best (shortest path) from the source to a vertex when
   (a) Graph is not connected
   (b) Graph has weighted edges (and is connected)
   (c) Graph has unweighted edges (and is connected)
   (d) Graph has cycles
   (e) Graph has n nodes and n/2 edges
10. Complete Dijkstra's algorithm on...



   (a) Fill in the complete table from lecture
   (b) What is the fastest path from A to F?

Sorting

	Best Runtime	Worst Runtime	Example input for best and worse case
Quick Sort			
Merge Sort			
Insertion Sort			
Selection Sort			
Counting Sort			
Radix Sort			
Heap Sort			