Choose from the following data structures (each of which may be used more than once):
Stack, Queue, BST, Heap, Hash Table, Priority Queue

1. Selecting the order of messages to send in a network when some messages require a faster delivery than others. New messages are always coming in.

2. Tracking the orders placed at a counter in a sandwich shop.

3. Tracking the roster of Seahawks players. Individual player records are occasionally requested by the player's name. Requesting the entire list of players sorted by player names is the most common operation.

4. A company has a large amount of data that is not comparable. They want to use a data structure that gives them the fastest possible find operation.

1. (12 points) Assume that the tree in Figure 1 is a binary search tree that contains only integer values and no duplicates. Note that nodes A and B are part of the tree and should be considered as such while answering all the following questions about the tree.

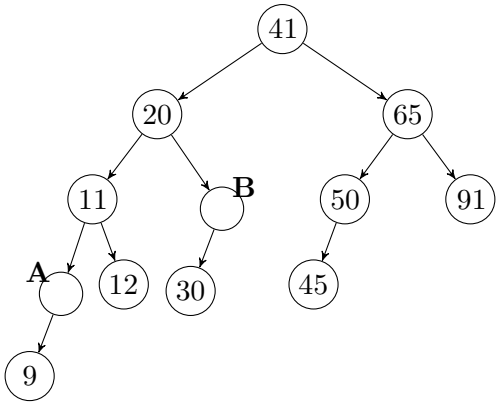

Figure 1: A binary tree.

(a) What could be the value of node A?

(a) _____

(b) What could be the value of node B?

(b) _____

(c) What is the height of the tree?

(c) _____

(d) How many nodes can you insert in this tree without increasing its height?

(d) _____

(e) What is the minimum number of nodes that need to be inserted to make this tree a complete tree?[14] Recall that a complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

(e) _____

(f) This tree is an AVL tree. Name one leaf node, which, if removed, will break the AVL balance property? (There are multiple answers to this. You just need to give one.)

(f) _____

(g) Write the in-order traversal sequence of the tree (include nodes A and B in your traversal)

1. HASHING

(A) Imagine you are given the following set of values to insert into a hash table. Assume this hash table:

- Hashes values using the function $\boldsymbol{h(x) = x}$
- Stores each value at $\boldsymbol{index = h(x)\ \%\ table\_size}$
- Uses **separate chaining** to resolve collisions, adding new values to the **back** of the list
- Has an initial internal capacity of 5
- **Triggers a resizing** of the internal array for the next value **after** reaching λ = 1

Draw the internal state of the hash table as you add the values in the given order. One of these values will trigger a resizing of the array, at this point be sure to transfer the values over so the larger array will represent the final state after adding all values.

values to add: **5, 7, 8, 4, 2, 10, 12, 25, 18**

Figure 1 – Internal array of hash table before resizing

0	1	2	3	4

Figure 2 – Internal array of hash table after resizing

0	1	2	3	4	5	6	7	8	9

(B) Imagine a hash table has an initial capacity of 10. During each resizing, the capacity, or *table_size*, of the internal array is doubled. Integer values are inserted into the table using the hash function $h(x) = 5x$. Then each value will be stored at $index = h(x)\ \%\ table\_size$

(i) Why is this design suboptimal? Explain briefly.

(ii) Describe a change that could improve this design and briefly explain how it would impact the runtime.

4. Heaps

(A) Insert the following sequence of values in the given order, one at a time, into an empty min heap. Fill in *Figure 6* with your final answer, you will not need to fill in all nodes. You may use the space below *Figure 6* to draw your intermediary trees.

13, 42,11, 35, 3, 22, 8, 9

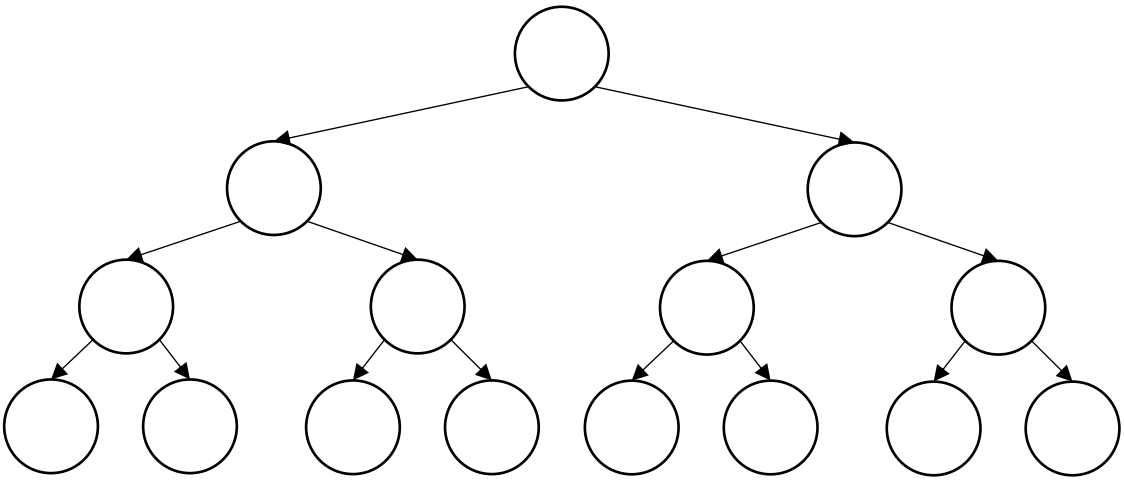

Figure 6: Fill in this tree with your **final** answer. Leave unused nodes empty.

(B) Given the array in *Figure 7* representing the current state of a min heap, fill in the empty array in *Figure 8* with the state of the heap after performing a single `removeMin()`. You may use the space below *Figure 8* to show your work.

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
8	**15**	**13**	**32**	**24**	**41**	**29**	**54**	**82**	

Figure 7

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

Figure 8

$$T(n) = \begin{cases} 4 \; when \; n = 1 \\ 8T\left(\frac{n}{2}\right) + n^2 \; otherwise \end{cases}$$

(A) Find an exact closed form of this recurrence using the **tree method**.
As in class, consider the root level *i = 0*. This means at *i* = 0 the input is n and the number of nodes is 1.

(i)	What is the total number of nodes at level *i*?	
(ii)	What is the size of the **input** to each node at level *i*?	
(iii)	What is the total **work** done across the *i*-th *recursive* level?	
(iv)	What is the value of *i* on the last level of the tree (the level of the leaf nodes)?	
(v)	What is a mathematical expression in terms of *i* for the total work done across all the *recursive* levels?	
(vi)	How many leaf nodes are on the last level of the tree?	
(vii)	What is the total **work** done across the last level of the tree?	
(viii)	What is an expression in terms of *i* for the total **work** done by *T(n)?*	

(ix) Give the closed form of the total work done by *T(n)*. You must show your work to receive credit.

(x) What is the simplified tight big-O bound for *T(n)*? _____

(B) Use Master Theorem to validate your answer above. Write out the appropriate inequality and the resulting Θ to show how you applied the theorem.

4. BIG-O DEFINITIONS

For each entry below give the simplified tight big-O bound and then answer the corresponding true or false question about that function.

Function	Simplified Tight Big-O	True or False? (Circle)	
$a(n) = 4\log_2 n$		$a(n)\ is\ in\ O(n)$	**T** or **F**
$b(n) = 4n + 3$		$b(n)\ is\ in\ \Omega(n \log n)$	**T** or **F**
$c(n) = n^2 - 3n$		$c(n)\ is\ in\ O(n^2)$	**T** or **F**
$d(n) = n^3 + 3^n$		$d(n)\ is\ in\ \Omega(n^2)$	**T** or **F**
$e(n) = 10C_1 + C_2$		$e(n)\ is\ in\ O(n)$	**T** or **F**
$f(n) = \frac{n}{2} + \log_2(2n)$		$f(n)\ is\ in\ \Omega(\log n)$	**T** or **F**

5. CODE MODELING

(A) Consider the following code for the `sortGrades` method:

(c) (4 points) Give the recurrence $T(n)$ for the runtime of the following snippet of the code. Use variables appropriately for constants (e.g. c_1, c_2) in your recurrence (you do not need to attempt to count the exact number of operations). **YOU DO NOT NEED TO SOLVE** this recurrence, just give the base case and a recurrence case.

```
public int bar(int n) {
    if (n <= 10) {
        return 0;
    }
    int r = 0;
    for (int i = 0; i < n; i++) {
        r++;
    }
    return bar(n/3) + bar(n-1);
}
```

$$T(n) = \begin{cases} & \text{if } n \_\_\_\_\_ \\ & \text{otherwise} \end{cases}$$

(d) (4 points) Similar to how you solved the above question (c), give the recurrence $T(n)$ for the following snippet of the code

```
public int foo(int n) {
    if (n < 10) {
        return 0;
    } else if (n < 1000) {
        return foo(n - 1);
    }
    return foo(n - 1) + foo(n - 2);
}
```

$$T(n) = \begin{cases} & \text{if } n \_\_\_\_\_ \\ & \text{otherwise} \end{cases}$$

May the K-D trees be with you

1. When would you want to use a K-D tree instead of the K dimensional version of a quad tree?

2. What operation is enhanced by using a K-D tree over a BST?

3. What input to the creation of a K-D tree will result in the worst case behavior of the data structure?

4. What input to the creation of a K-D tree will result in the best case behavior of the data structure?

5. Can we represent a K-D tree using an array like we can with heaps? Why or why not?