```
1a. 12

1b. 8

1c. Name is Susie and age is 3

1d. Error condition!

1e. Wonder Woman

1f. No equality!
```

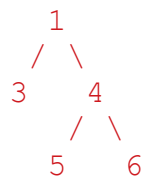
Rubric:
+5 each, all or nothing
Do not deduct points for the following:

- extra use of (or failure to use) newlines or spaces
- capitalization differences
- minor punctuation differences
- minor spelling errors
- use of quotation marks
- forgetting the word “and”
- spelling out numbers (ex. “three” instead of “3”)

2a. Because of the way an AVL tree is structured, when we conduct our search, we can throw away half of the remaining tree at each step without searching through it first. This gives us a runtime of O(log n).

Rubric:
-3 if all the student mentions is that an AVL is a balanced BST
-2 if the student speaks to the idea that there are some parts of the AVL that are not explored but does not go into additional detail
+5 if the student demonstrates an understanding that we throw away half the tree at each step without searching it. Do not deduct points for any additional information provided.

2b.



Rubric:
+5 The result is a valid heap with only the numbers shown in the example on the left. The heap shown here is only one of many possible solutions.
No penalty if the student draws a max heap instead of a min heap.

2c. If all of the values are stored in a single location, then we have to search through the entire LinkedList every time we want to find an item. This is no different than searching through a LinkedList without a hash map, and we lose the constant runtime for accessing data that hash maps give us.

Rubric:
+5 Student demonstrates an understanding that we lose the constant runtime advantages of hash maps.

2d. 18 16 21 15 17 19

Rubric:
-3 if the student gives the correct depth-first search traversal (18 16 15 17 21 19)
-1 if the student forgets the 18 or 19 (-2 for both)
+5 All or nothing otherwise
No penalty if the student instead provides a correct explanation of the trace or if the list of numbers includes the nodes we backtrack to during the graph traversal.

2e. The load factor is 2, since we have an average of 2 items in each hash map location.

Rubric:
+2 Correct load factor
+3 Correct explanation

2f. As we iterate over the LinkedList, we reach a point where i = 2. Therefore, the program tries to access the values in the LinkedList at indices 2 and 3, but since the LinkedList only has 3 items, there is no value at index 3. The program throws the aforementioned exception as a result.

Rubric:
-2 Student identifies the error in the code but does not provide an adequate explanation
+5 Student demonstrates an understanding that we try to access a value at an index that doesn't exist.

3.

Key:
3a modifications are in red
3b modifications are in blue
3c modifications are in green

```
class NotFreshRecordHashMap {

	private HashMap<String, LinkedList<GroceryItem>> notFreshMap;

	public NotFreshRecordHashMap() {
		notFreshMap = new HashMap<String, LinkedList<GroceryItem>>();
	}

	public void addGroceryItem(String name, GroceryItem item)
	{
		LinkedList<GroceryItem> itemsList;
		try {
			itemsList = this.getGroceryItem(name);
		}
		catch (ItemNotFoundException e) {
			itemsList = new LinkedList<GroceryItem>();
		}
		itemsList.add(item);
		notFreshMap.put(name, itemsList);
	}

	public LinkedList<GroceryItem> getGroceryItem(String name)
		throws ItemNotFoundException {

		if(!notFreshMap.containsKey(name)) return null;
			throw new ItemNotFoundException(name + " not found");

		return notFreshMap.get(name);
	}
}

class ItemNotFoundException extends Exception {
	private String err;

	public ItemNotFoundException(String err) {
		this.err = err;
	}

	public String getErr() {
		return err;
	}
}
```

Rubric:

3a. Access modifiers (+10 points, +2 points each)

__ `notFreshMap` has `private` access modifier
__ `addGroceryItem()` has `public` access modifier
__ `getGroceryItem()` has `public` access modifier
__ `err` has `private` access modifier
__ `getErr()` method has `public` access modifier

3b. Exception handling (+10 points)

__ (+1) `return null` is crossed out in `getGroceryItem()`
__ (+2) `throw` statement put under `if` statements
__ (+2) `throw` statement instantiates `ItemNotFoundException`
__ (+1) instantiation of `ItemNotFoundException` is correct (message doesn't matter)
__ (+4) `getGroceryItem()` method signature modified to indicate it throws an `ItemNotFoundException`

NOTES:

- No penalty for mixing up `throw` and `throws`.
- No penalty for missing the word `new` in the instantiation of `ItemNotFoundException`
- -2 for `throws Exception` or similar instead of `throws ItemNotFoundException` in method header

3c. Collision detection (+10 points)

__ (+3 points, +1 each) Each instance of `GroceryItem` replaced with `LinkedList<GroceryItem>`
__ (+7 points, +1 each) In `addGroceryItem()`,
    __ `LinkedList` is retrieved from hash map
    __ `LinkedList` retrieval uses `getGroceryItem()`
    __ Call to `getGroceryItem()` is in `try/catch` block
    __ `ItemNotFoundException` is caught
    __ In catch block, new `LinkedList` is initialized
    __ `GroceryItem` is added to `LinkedList`
    __ `LinkedList` is put back in to hash map

NOTES:

- -2 if return type of `addGroceryItem()` is changed, but no return statement provided
- -1 if second parameter of `addGroceryItem()` is modified

3d. We can have both NotFreshRecordHashMap and NotFreshRecordSet implement a common interface, and then the code that uses these classes can do so by using the interface as our data type. The interface probably looks something like this:

```
public interface INotFreshRecord {
      public void addGroceryItem(String name, GroceryItem item);
      public LinkedList<GroceryItem> getGroceryItem(String name)
            throws ItemNotFoundException;
}
```

Rubric (+10 points):

__ (+2) Explanation indicates that both classes implement a common interface

__ (+2) Explanation indicates that we use the interface as the data type

__ (+3) Method header for `addGroceryItem()` that matches the method header in the student's modified `NotFreshRecordHashMap` class.

__ (+3) Method header for `getGroceryItem()` that matches the method header in the student's modified `NotFreshRecordHashMap` class.

NOTES:

- Only one point for each method header if it does not match the method header in the modified code from parts a – c.
- -2 for forgetting the interface declaration
- No penalty for communicating the idea of a common interface through additional code instead of a written explanation
- No penalty if the student further modifies the code in parts a – c, although this should not be considered part of the solution for part d.