Melissa's Responses to Week 8 Skills Assessment

Runtime

1. When calculating the Big O notation for a particular algorithm, it's necessary to consider the length of time it takes for the algorithm to run as the algorithm's workload approaches the size of infinity. What is the **workload** of a function that takes in a list of integers and returns a new list of the even integers?

```
Array = [1,2,3,4,5,6,.....100]
tempArray=[]  #drops out of consideration because this is a constant

For item in array: #touching each item of the array O(n)
        If item%2 ==0: #touching each item of the array O(n)
                tempArray.append[item] # constant time so this drops out
```

O(2n) and the 2 drops out giving us O(n)

2. Order the following runtimes in ascending order by efficiency as **n** approaches infinity:
** I included some extra statements here that are more for me than you.

    - O(1) - constant time: no matter how big the input is, it always takes the same amount of time to compute things.
    - O(log n): Logarithmic Time (Divide and conquer, Binary Tree) each time the amount of 'stuff' to sift through is halved.
    - O(n) - Linear: growth is slow and dependent on number of items
    - O(n log n): Logarithmic Time: It is log of n * n which is still smaller than n^2
    - O(n^2) - Quadratic: for every item in the list you have to do n more operations (two nested for loops are an example)
    - O(2^n): Exponential Time: recursive function

Stacks and Queues

1. In the following cases, would a stack or queue be an appropriate data structure?
    1. The process of loading and unloading pallets onto a flatbed truck
    When you load a truck and then unload the last items are the first to come out therefore you are working with a Stack. (LIFO)

2. Putting bottle caps on bottles of beer as they roll down an assembly line
   You put a bottle cap on as a bottle rolls by therefore it is a queue because the first bottle to roll past gets the first bottle cap. (FIFO)

3. Calculating the solution to this mathematical expression: 2 + (7 * 4) - (3 / 2)

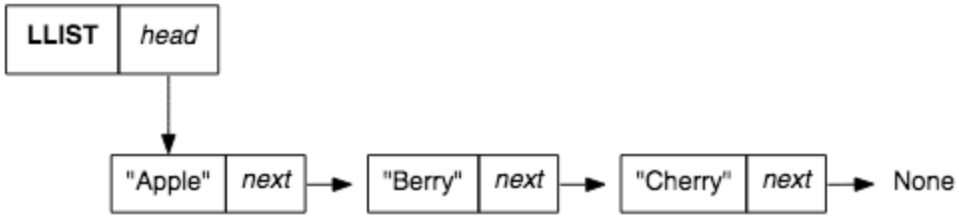
   A stack would be used for evaluating what is inside each set of parentheses based on order of operations (PEMDAS) but then a queue would be used to evaluate the expression.
   Stack: (3/2), (7*4), 2
   Queue: 2, 2 +28, 30-1.5

2. Describe 2 more examples of when a queue would be an appropriate data structure.
   When answering questions for my kids. I 'try' to queue them up and answer the first question asked and then move to the next (FIFO)
   When you multi-thread a process you do so through a queue'ing process. First task comes in and is added to a queue, next task comes in and is added behind the first. Each will be dealt with in the order that it entered the queue. (FIFO)
   A line at disneyland (so not the happiest place on earth). You stand in line and first rider is served first and new riders are added to the end.

3. Describe 2 more examples of when a stack would be an appropriate data structure.
   Putting away dishes is a stack. You place new dishes on top and remove dishes from the top. (LIFO)
   The undo function on sublime or any program for that matter. Allows user to replay the last event that occurred and then last event before that, so on and so forth in a drill down manner. (LIFO)

Linked Lists

1. Given the below linked list, which are the nodes? What is the data for each node? Where is the head? Where is the tail? (Please be as specific as possible — exactly which parts of the diagram correspond to each part? Arrows? Boxes? Text?)



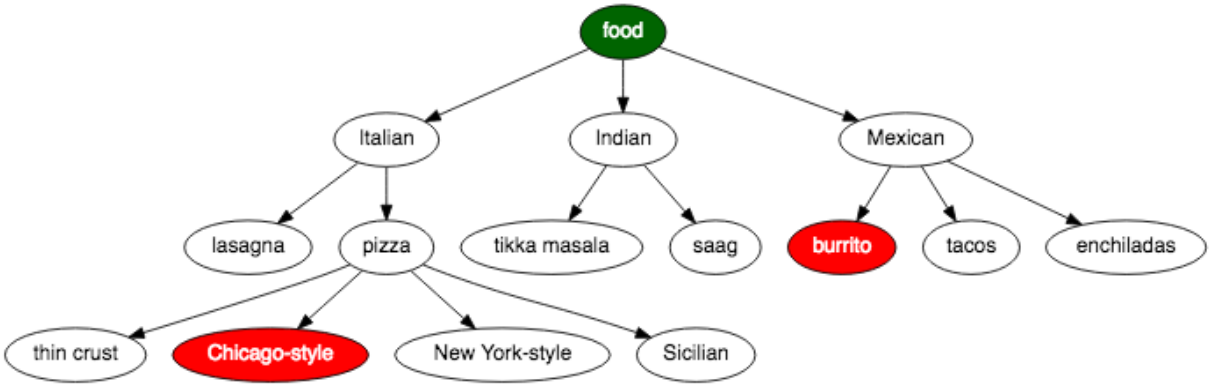
An instance of the linkedlist class is bound to the name LLIST and it has an attribute 'head' to hold the pointer to a node.

The nodes are ‘Apple’, ‘Berry’, ‘Cherry’ and represented as boxes
Each node has two attributes: data and next
The arrows represent the pointers to each of the next nodes.
Apple.next->Berry
Berry.next ->Cherry
Cherry.next ->None
There is no attribute ‘tail’ in this model there is only a linked list that has to be traversed element by element asking for <node>.next == None. This means the search time and append time of this list is slower because the more elements you add the longer it will take to search to find the end node.

1. What’s the difference between a doubly and singly linked list?
   You can only walk down a singly linked list because each node only points to the next. Whereas, a doubly linked list can be traversed both forward and back because the nodes hold pointers to both the node before them as well as the node after themselves.

2. Why is it faster to append to a linked list if we keep track of the **tail** as an attribute?
   It is faster to append to a linked list when the tail is tracked because we know where the end of the list is and the node can be directly accessed (O(1)) as opposed to having to traverse the entire list (O(n)) looking for the end element.

Trees



1. Given the above tree, in what order would a Breadth First Search (BFS) algorithm visit each node until finding **burrito** (starting at **food**)? 9th node searched
   Food ->italian->indian->mexican->lasagna->pizza->tikka masala->saag->burrito!

2. Given the above tree, in what order would a Depth First Search algorithm visit each node until finding **Chicago-style** (starting at **food**)? 6th node searched
   Food -> Italian->lasagna -> pizza -> thin crust -> chicago-style

3. How is a binary search tree different from other trees?
   A binary tree should approximately halve the amount of items to search each time it runs. O(log n)