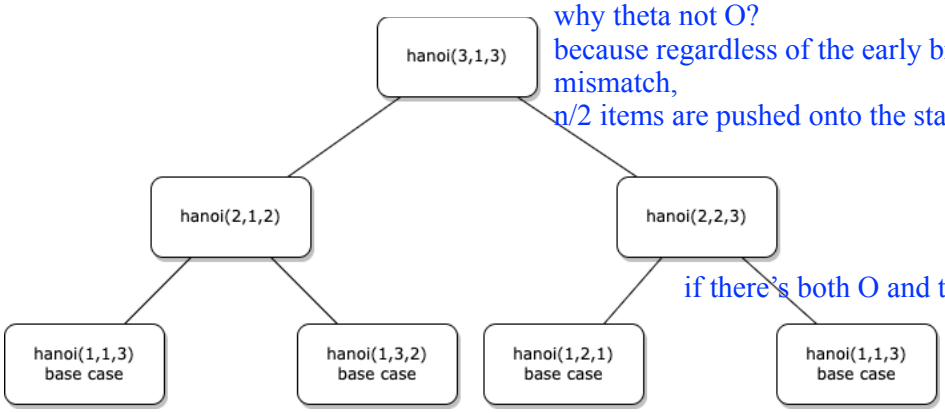
Name & UNI: Xintong Qi(Amy)_____ Date: Oct 14 _____

Point values are assigned for each question. Points earned: ____ / 100, = ____ %

1. Draw the recursion tree generated when calling `hanoi(3, 1, 3)`. The first parameter is `numDisks`, the second is the number of the `fromPeg`, and the last is the `toPeg`. Each node in the tree should include the function name and three parameters described above. `hanoi(3, 1, 3)` is the root node in the drawing. (18 points)



why theta not O?
because regardless of the early breaking when there is a mismatch,
n/2 items are pushed onto the stack

if there's both O and theta, it'll probably be theta

2. How you can use a stack to efficiently check if a String of length `n` is a palindrome?
What is the time complexity of your algorithm? (3 points) _**theta(n)**_____
Complete the Java method below. Make use of all variables – `stack`, `i`, `length`, and `mid`. (20 points)

This one should be O(n),
because you added an early breaking

```
/**
 * Returns whether or not String s is a palindrome. An empty string is considered
 * to be a palindrome. String s is guaranteed to not be null.
 * @param s  the String to check
 * @return   true if String s is a palindrome; false otherwise
 */
public static boolean isPalindrome(String s) {
    Stack<Character> stack = new Stack<>();
    int i = 0, length = s.length(), mid = length / 2;

   if(length <= 1){ //empty string or single char, return true
         return true;
   }


    for(i = 0; i < mid; i++){
         stack.push(s.charAt(i));
    }

    for(i = mid; i < length; i++){
         if(i == mid && length % 2 != 0){ // length is odd, skip mid
               continue;
         }
```

```
            if(stack.pop() != s.charAt(i)){ // fail to match up, return false
                  return false;
            }
      }

   return true;
}
```

3. Suppose + and - have HIGHER PRECEDENCE than * and /, and that you have the following infix expression: 3 * 5 + 2 - (6 + 3 / 3)
   Convert the expression from infix to postfix. Fill out the table, showing each step you take. List the action of each step and show the contents of the stack and postfix expression after completing the action. The first step is done for you. Popping an element off the stack and appending it to the postfix expression is one step. (17 points)

Action	Stack	Postfix
Add 3 to postfix.	(empty)	3
Push * to stack	*	3
Add 5 to postfix	*	3 5
Push + to stack	* +	3 5
Add 2 to postfix	* +	3 5 2
Pop + from stack	*	3 5 2 +
Push – to stack	* -	3 5 2 +
Push (to stack	* - (	3 5 2 +
Add 6 to postfix	* - (	3 5 2 + 6
Push + to stack	* - (+	3 5 2 + 6
Add 3 to postfix	* - (+	3 5 2 + 6 3
Pop + from stack	* - (+	3 5 2 + 6 3 +
Push / to stack	* - (/	3 5 2 + 6 3 +
Add 3 to postfix	* - (/	3 5 2 + 6 3 + 3
Pop / from stack	* - (	3 5 2 + 6 3 + 3 /
Pop (from stack	* -	3 5 2 + 6 3 + 3 /
Pop – from stack	*	3 5 2 + 6 3 + 3 / -
Pop * from stack		3 5 2 + 6 3 + 3 / - *

   Evaluate the postfix expression to produce the final integer result. (3 points) **_12____**

4. Suppose our MyLinkedList<E> class represented a doubly linked list, where each Node is implemented as follows:

```
private class Node {
    Node prev, next;
    E element;

    public Node(E element) {
        this.element = element;
    }
```

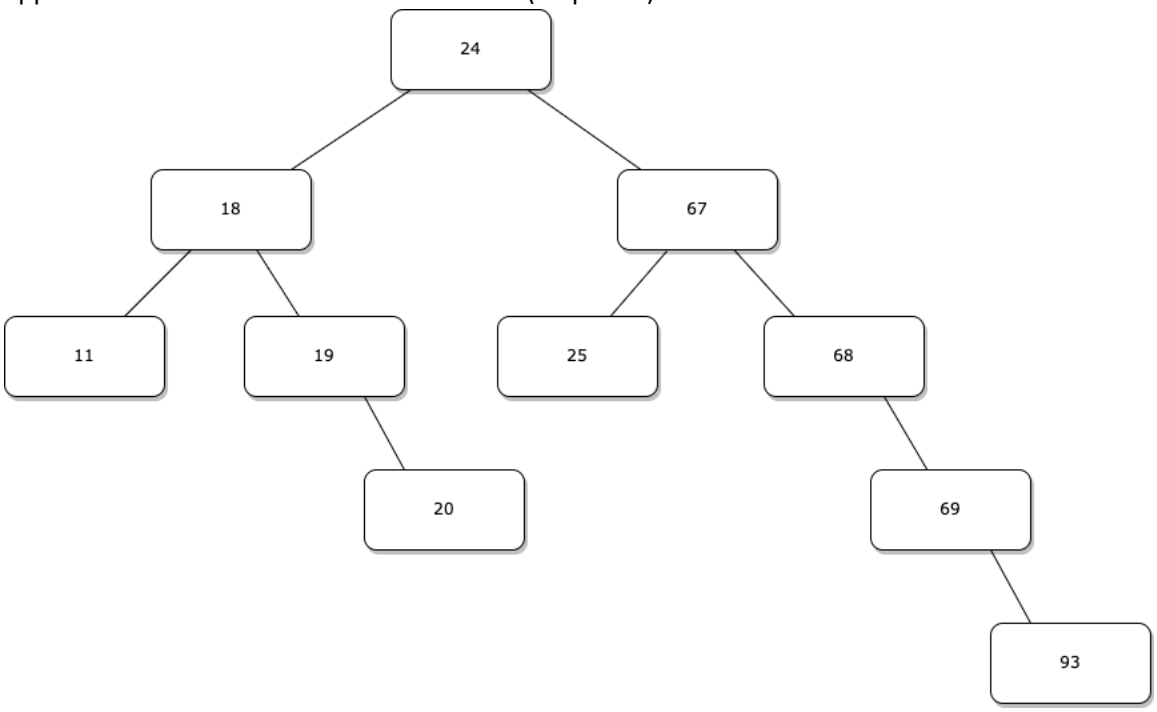
```
}
```

Each Node has a reference to the previous Node as well as to the next Node in the list. MINIMALLY modify the `add(int index, E element)` method below to work properly with the doubly linked Nodes, and highlight or underline your changes. (10 points)

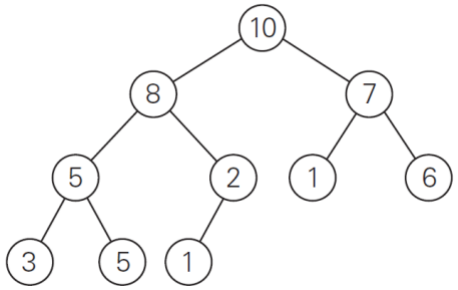
```
/**
 * Inserts the specified element at the specified position in this list.
 * Shifts the element currently at that position (if any) and any subsequent
 * elements to the right (adds one to their indices).
 * @param index    index at which the specified element is to be inserted
 * @param element  element to be inserted
 * @throws IndexOutOfBoundsException if the index is out of range
 *         (index < 0 || index > size())
 * The exception message must be:
 * "Index: " + index + ", list size: " + size
 */
public void add(int index, E element) {
   if (index < 0 || index > size) {
      throw new IndexOutOfBoundsException(
              "Index: " + index + ", list size: " + size);
   }
   Node n = new Node(element), p = head, prev = null;
   for (int i = 0; i < index; i++, prev = p, p = p.next) ;

   if (p != null && prev == null) { // add at index 0
          n.next = p;
          p.prev = n; // implement the double link
          head = n;
   } else if (p != null) { // add at other indices
          prev.next = n;
          n.prev = prev; // implement the double link
          n.next = p;
          p.prev = n; // implement the double link
   }

   if (p == null) { // add to an empty list
          // I think it is better to move this block before the previous if else
          // statement, for readability and code efficiency
          tail = n;
          head = n;
   }

   size++;
}
```

so the answer combined this case and the first case, and there is no head=n in this block

5. Draw the final binary search tree that results after inserting the following keys in the order they appear here: 24 18 67 68 69 25 19 20 11 93 (10 points)



6. Consider the following binary tree. (19 points)



a) List the nodes visited with a preorder traversal. (3 points)
b) List the nodes visited with an inorder traversal. (3 points)
c) List the nodes visited with a postorder traversal. (3 points)
d) How many internal nodes are there? (2 points)
e) How many leaves are there? (2 points)
f) What is the maximum width of the tree? (2 points)
g) What is the height of the tree? (2 points)
h) What is the diameter of the tree? (2 points)

a) 10, 8, 5, 3, 5, 2, 1, 7, 1, 6

b) 3, 5, 5, 8, 1, 2, 10, 1, 7, 6

c) 3, 5, 5, 1, 2, 8, 1, 6, 7, 10

d) 5

e) 5

f) 4

g) 3

h) 6