Data Structures and Algorithms
Week 2 problem sheet

## Linked Lists

1. Give three points of differences between an array and a linked list.
2. Draw the **stack** and **queue** data structures with linked list implementations for each step in the following sequence:
* add(1), add(2), remove, add(3), add(4),
remove, remove, add(5).
1. Let Link be an object with two member variables:
* public class Link {
 char item;
 Link next;
}
* Assume that the variable first is a reference to a Link object containing 'a', whose next is a link object containing 'b', whose next is null.
* 
* Which of the following code snippets successfully reverses this structure, so as to give the following list?
* 
* Draw a picture of the result of each code snippet to explain your answer.
	1. first.next = null;
	first.next.next = first;
	first = first.next;
	2. first.next.next = first;
	first = first.next;
	first.next.next = null;
	3. Link temp = first;
	first = temp.next;
	first.next = temp;
	temp = null;
	4. Link temp = first.next;
	temp.next = first;
	first.next = null;
1. A Cyclic Linked Implementation of a Queue
* An (unbounded) queue can be implemented cyclically based on a linked representation. In this case, rather than referencing “null”, the successor of the last item in the queue references the beginning of the queue. While this is not necessary to prevent memory erosion, it does mean that rather than having two references to the beginning and the end of the queue, only a *single* reference is needed.
* Write a cyclic linked implementation of a queue called QueueLinked using this approach.
* Your queue ADT must implement the QueueADT interface.
* Fully document your code.
1. A double-ended queue (*deque*) of characters differs from a standard queue in that it allows objects to be added and deleted from both ends of the queue. Contrast this to a standard queue, where objects can only be added to the end of the queue and removed from the front.
* Write a singly linked-list implementation of the deque ADT in Java.
* Your implementation should contain the following methods:
	+ DequeCharCyclic(s): create an empty deque of size s.
	+ isEmpty(): return true iff the deque is empty, false otherwise.
	+ isFull(): return true iff the deque is full, false otherwise.
	+ pushLeft(c): add character c as the left-most character in the deque, or throw an Overflow exception if the deque is full.
	+ pushRight(c): add character c as the right-most character in the deque, or throw an Overflow exception if the deque is full.
	+ peekLeft(): return the left-most character in the deque, or throw an Underflow exception if the deque is empty.
	+ peekRight(): return the right-most character in the deque, or throw an Underflow exception if the deque is empty.
	+ popLeft(): remove and return the left-most character in the deque, or throw an Underflow exception if the deque is empty.
	+ popRight(): remove and return the right-most character in the deque, or throw an Underflow exception if the deque is empty.
1. **Challenge:** See extra questions on dequeues at <http://teaching.csse.uwa.edu.au/units/CITS2200/Tutorials/tutorial04.html>

## Trees



1. (source: Weiss 18.1) For the tree shown above determine:
	1. Which node is the root
	2. Which nodes are leaves
	3. What is the tree’s height
	4. What are the results of traversing the tree using
		1. preorder
		2. postorder
		3. inorder
		4. level order

## Binary trees

1. State whether each of the following statements is TRUE or FALSE. Why?
	1. The height of a binary tree is $O(logn)$
	2. The proper descendants of a node’s ancestors are also descendants of that node.
	3. A binary tree always has more external nodes than internal nodes.
	4. Every tree has a root node.
2. (Source: Weiss 18.9) Write efficient methods (and give their Big-Oh running times) that take a reference to a binary tree root t and compute
	1. The number of leaves in t
	2. The number of nodes in t that contain one non-null child
	3. The number of nodes in t that contain two non-null children
3. (Source: Weiss 18.10) Suppose a binary tree stores ints. Write efficient methods (and give their Big-Oh running times) that take a reference to a binary tree root t and compute
	1. The number of even data items
	2. The sum of all the items in the tree

## Binary search trees

1. (Source: Weiss 19.1) Show the result of inserting 3, 1, 4, 6, 9, 2, 5, and 7 in an initially empty binary search tree. Then show the result of deleting the root.
2. (Source: Weiss 19.2) Draw all binary search trees that can result from inserting permutations of 1, 2, 3, and 4. How many trees are there? What are the probabilities of each tree’s occurring if all permutations are equally likely?
3. (Source: Weiss 19.15) Implement the BinaryTree methods find, findMin, and findMax recursively.