Data Structures and Algorithms
Week 6 problem sheet

## A. ADTs

1. Which of the following are ADTs and which are concrete data types?
	* array
	* linked list
	* graph
	* binary search tree
	* queue
	* priority queue
	* hash table
*
* **Solution**
* The following are concrete data types: array, linked list, binary search tree, hash table.
* The following are ADTs: graph, queue, priority queue.
*
1. Suppose you are asked to create an ADT intended to represent a *lock*. A lock can be locked and unlocked, and should have accessor methods so the lock’s state can be observed. Show the Java code you would use for this ADT.
* Since this is an ADT, not a concrete data type, we would use a Java interface to create it rather than a Java class. From the problem description, we should have “lock” and “unlock” operations, and an accessor method “isLocked” so we can observe the state.
* Possible Java code would be:
* interface Lock {
 void lock();
 void unlock();
 bool isLocked();
 }

## B. Stacks and queues

1. Suppose we have a some integers, and perform the following operations with them on a stack or queue object:
	* add 3, add 2, add 5, remove, remove, add 9
* What would the object contain, after each operation, if we used a …
	+ … array-based stack, with capacity 5?
	+ … linked list–based queue?
	+ … priority queue?
*
* **Solution**
* For an array based stack, we show the contents of the array as a list, and the value of the “topOfStack” instance variable as “top”. The contents would be:
	+ initially: (top: -1, [0,0,0,0])
	+ add 3: (top: 0, [3,0,0,0])
	+ add 2: (top: 1, [3,2,0,0])
	+ add 5: (top: 2, [3,2,5,0])
	+ remove: (top: 1, [3,2,5,0])
	+ remove: (top: 0, [3,2,5,0])
	+ add 9: (top: 1, [3,9,5,0])
* For a linked list–based queue, we just show the contents as a list, with the FRONT of the queue to the right and the BACK of the queue at the left.
* The contents would be:
	+ initially: empty.
	+ add 3: [3]
	+ add 2: [2, 3]
	+ add 5: [5, 2, 3]
	+ remove: [5, 2]
	+ remove: [5]
	+ add 9: [9, 5]
* If the contents were a priority queue, then we either need to know the priority for each element added, or need to make the assumption that elements can be added using some default priority (e.g. 0).
* If we assume elements will be added with a default priority, then the contents will be exactly the same as for the linked list–based queue; since for items with the *same* priority they’ll simply follow the same “first in, first out” behaviour of a normal queue.
*

## C. Linked lists

1. Suppose we want to write a Node class for use in a linked list that stores Strings.
* Write code for this class.
* **Solution**
* class Node {
 String value;
 Node next;
 public Node(String value, Node next) {
 this.value = value;
 this.next = next;
 }
 }
1. How would the class in question 1 change if we wanted to use a generic class?
* **Solution**
* class Node<T> {
 T value;
 Node<T> next;
 public Node(T value, Node<T> next) {
 this.value = value;
 this.next = next;
 }
 }

## D. Big “O” notation

1. Suppose we have one algorithm *A* that runs in $O\left(logn\right)$ time, and another algorithm *B* that runs in $O\left(n\right)$ time.
* We write a program in which we run algorithm *A*, then algorithm *B*. What is the big “O” complexity of our program?
* **Solution**
* Assuming no other operations are needed, then the complexity of our program will be $O\left(logn+n\right)$ which is $O\left(n\right)$, as $O\left(n\right)$ is bigger than $O\left(logn\right)$.

# E. Sorting and search

1. Suppose we want to sort an array, but it turns out (unbeknownst to us) that the array is already sorted.
* Of the sorting algorithms we have seen (insertion sort, merge sort and quick sort), which is likely to perform *worst*? Why?
* **Solution**
* QuickSort is likely to perform worst. As noted in the lecture on QuickSort, if we apply it to an array which is already sorted, and choose position 0 for our fencepost (as the code in the code samples does), then we end up with very poor performance of $O\left(n^{2}\right)$. (However, if we use a variation of QuickSort which uses a “median of 3” approach to choosing the fencepost, we can avoid this.)
*
1. Suppose we want to perform binary search on an un-sorted list of numbers. Can we do so? If so, how? If not, why not?
* **Solution**
* We can’t perform binary search on the list unless we sort it first. It is a requirement of binary search that the array be sorted – if it is not, the algorithm simply won’t give correct results.
* This is because the algorithm assumes that we can repeatedly narrow down the range of places where the value is, by looking to the left or right of the current position.

# F. Trees



1. Is the tree shown above a binary tree?
* Is it also a binary search tree?
* Is it also a *balanced* binary search tree?
* Explain why or why not.
* **Solution**
* The tree is indeed a binary tree, as the nodes always have at most 2 children. It is also a search tree, as for every node, all the values in its left subtree are less than the node’s value, and all the values in its right subtree greater than the node’s value.
* However, it is not a balanced BST: it has 7 nodes, and thus *could* be fitted into 4 layers, but instead has 5 layers. Hence it is not balanced, as a balanced BST always has the minimum possible number of layers.
1. Suppose we wanted to use binary search to see if the number 8 was in the tree. What steps would be involved?
* **Solution**
* The steps would be:
* - inspect 3, go down the right subtree
- inspect 4, go down the right subtree
- inspect 6, go down the right subtree
- inspect 9, do down the left subtree
- inspect 7, do down the right subtree
- encounter an empty node; this means 8 is not in the tree.
1. Can we use a binary search tree to implement the Set ADT? How would we implement the “add” operation?
* **Solution**
* We can. If we use the BinarySearchTree class included in the sample code, we can use this to implement the Set ADT. BinarySearchTree is a generic class, so we could make our Set class generic as well. Our class, together with its “add” operation (we omit other methods) would look like this:
* class BSTSet<T extends Comparable<? super T>> {
 BinarySearchTree<T> bst;

 void add(T value) {
 bst.insert(value);
 }
 }

# G. Graphs

1. Suppose we have a very dense weighted graph, and will be using Prim’s algorithm to obtain a minimum spanning tree from it.
* Would it be better to use an adjacency list implementation, or an adjacency matrix implementation? Explain why.
* **Solution**
* For a dense graph, it will normally be better to use an adjacency matrix implementation; little space will be wasted in the matrix for unused elements, and looking up elements in an array is likely to be faster than traversing linked lists.
1. Are all trees also graphs? Explain why or why not.
* **Solution**
* Yes, all trees are also graphs (if we consider that there is an edge between any node and its child nodes). Specifically, a tree is a graph in which there is exactly one route between any two vertices in the graph.