Data Structures and Algorithms
Week 3 problem sheet

## Maps

Use the MapDemo.java class from the code bundle to answer the following questions about maps.

1. How do you enter a new pair into a map object?
2. Maps have at most one value per key. What happens if you enter a pair with a key that already exists in the map?
3. How do you find all the keys in a map (the map’s domain)?
4. How do you find all the values in a map (the map’s range)?
5. Describe two ways to remove a pair from a map.

## Collections API

1. Consider the four core interfaces of the Collections API: Set, List, Queue, Map. For each of the four assignments below, specify which of the four core interfaces is best-suited to the problem, and explain how to use an implementation of it to implement the assignment. You can complete the code for this in CollectionsDemo.java.
	1. Whimsical Toys Inc (WTI) needs to record the names of all its employees. Every month, an employee will be chosen at random from these records to receive a free toy.
	2. WTI has decided that each new product will be named after an employee but only first names will be used, and each name will be used only once. Prepare a list of unique first names.
	3. WTI decides that it only wants to use the most popular names for its toys. Count up the number of employees who have each first name.
	4. WTI acquires season tickets for the local lacrosse team, to be shared by employees. Create a waiting list for this popular sport.

## Hash tables

1. What is a perfect hash function? Explain why it is almost always impossible to have a perfect hash function.

## Graph algorithms

Use the AdjacencyMatrixGraph.java from the code for week 3 to answer the following problems on graphs.

1. Use the graph generation methods to generate some random graphs of different densities and see how graphs are represented using an adjacency matrix.
2. Describe (in words) an algorithm to count the number of edges in a **directed** graph using the adjacency matrix.
3. Describe how to count the number of edges in an **undirected** graph using the adjacency matrix.
4. Describe how to count the number of edges in an **undirected** graph using an **adjacency list** representation of a graph.
5. Describe how to generate the list of neighbours for a given node using an **adjacency matrix**.
6. Describe how to generate the list of neighbours for a given node using an **adjacency list**.
7. On paper, show the steps followed by Prim’s method for calculating a **minimum spanning tree** for graph in Figure 14.1 (Weiss) below. Start your tree from vertex V0. You could write some code using PrimMST.java to check your answer.
* 
1. PrimMST can be implemented using either an **adjacency matrix** or an **adjacency list** to represent a weighted graph. What difference does the choice of implementation make to the big O complexity of the algorithm? Why?
2. Test your answer to the previous question by implementing an adjacency list version of the algorithm. You can use PrimMST.java as the basis of your second solution. Run some experiments (as in the week 1 homework) to test the run times of these two algorithms. What do you notice?