1. [5 marks] Start with an empty heap and insert these priorities in the given order. Use a max-heap, i.e., larger priorities are closer to the root. Hand in both the final binary tree form and the final array form.
   5, 7, 15, 13, 30, 11, 21, 40, 9, 36.

2. [10 marks] Prove that a tree in which every vertex has degree at most 2 is a simple path. More precisely:
   Let $G = (V, E)$ be an undirected tree, with $|V| = n \geq 1$, and assume that every vertex has degree at most 2. Then $V$ can be ordered into a simple path $\langle v_1, \ldots, v_n \rangle$ and it uses all edges in $E$.
   Suggestion: use induction.

3. [12 marks] The spanning tree found by breadth-first search depends on the orders of vertices in adjacency lists. For the following directed graph:

   ![Graph Image]

   Give an adjacency-list representation of the graph that leads breadth-first search to find the spanning tree in the left below. And give an adjacency-list representation that leads to the right tree below.

   ![Graph Image]
4. [6 marks] Give a minimum spanning tree of the following undirected graph:

```
A---B
  |   |
  3   4
  |   |
  2   4
  |   |
  1   1
---C---D
     |
     1
---E---F
     |
     1
     |
     2
---G---H
     |
     2
     |
     1
---I---J
```

5. [12 marks] (Like textbook exercise 22.2-7.) There are two types of professional wrestlers: “babyfaces” (“good guys”) and “heels” (“bad guys”). Between any pair of professional wrestlers, there may or may not be a rivalry. Implement an algorithm that takes a group of wrestlers and a list of pairs of wrestlers for which there are rivalries, and computes whether it is possible to designate some of the wrestlers as babyfaces and the remainder as heels, such that each rivalry is between a babyface and a heel.

For simplicity, if there are $n$ wrestlers, they are identified by the integers from 0 to $n-1$. So your method signature is

```java
public bool bipartable(int numWrestlers, Rival[] rivalries)
```

where the Rival class is just a pair of int's:

```java
public class Rival
{
    public int x, y;
    // there is a rivalry between wrestlers #x and #y

    ...
}
```

Your bipartable method should return true if it is possible to designate some of the wrestlers as babyfaces and the remainder as heels such that each rivalry is between a babyface and a heel. Otherwise, your method should return false.

The starter code and the test program are on Blackboard: Bipart.java, Rival.java, TestBipart.java. You will add your own code to only Bipart.java and submit it.

**Package declaration** All files declare “package BP;”. Please try not to change it. I declare a package name because I suspect that some of you use IDEs that force you to declare a package name. So let’s all stick to the same name.

If you use the command line instead of an IDE (I confess I’m primitive and I still do this), the package declaration means that you need to: put the files under subdirectory BP. Then there are several ways to compile and run. Here is one: outside BP,
javac BP/Rival.java BP/Bipart.java BP/TestBipart.java
java BP.TestBipart

There are other ways. What works for you is good.

**How you can test** If compilation is successful, you can use “java BP.TestBipart” or IDE equivalents to run my test program. This runs all tests sequentially, but aborts at the first failure. Failures are represented by exceptions with error messages.

You can use “java BP.TestBipart 0” to run just test #0, for example. See also the “main” method and its comment in the file.

**Marking scheme** If your code looks like an attempt at a general algorithm (as opposed to an attempt at customizing for my test cases), you get 4 marks to start. In addition:

- If your code compiles: 1 more mark per test passed. But watch this requirement:
  I will give each test case 1 second only on the Mathlab server. Timing out is a failure.
  Each test case is timed separately.
- If your code does not compile: 0 more marks.

If your code does not look like an attempt at a general algorithm, 0 to 3 marks depending on how far off it is.

**How and what to submit** Please submit Bipart.java online. (This time there is no need to print out.)

To submit online:

(a) Upload to your CDF account.
(b) Login to your CDF account.
(c) Use the command: submit -c csc263h -a A2 Bipart.java
(d) You can verify by submit -l -c csc263h -a A2
   You can re-submit by submit -f -c csc263h -a A2 Bipart.java

This time there is no need to print out.