

1. (5 points) What would happen if this code were executed?

```
NaryTree[] kids = new NaryTree[5];

for (int i=0; i<5; i++)
    System.out.println(kids[i].toString());
```

2. (5 points) What is the running time of this pseudocode? Why?

```
for each element, i, of theList
    for each element, j, of theList
        if (i != j){
            for each element of theList that is smaller than the ith {
                increment the ith's count
            } // counting for
        } //if
    } // inner for
} // outer for
```

3. (5 points) When is a binary search tree degenerate?
4. (5 points) What is the definition of a heap (assume it contains ints).

5. (5 points) When is a heap degenerate?

6. (10 points) Write a `Heap` class, including a default constructor, plus, `boolean isEmpty()` and `void insert(int)`.

7. (10 points) Complete the constructor and `myClone()`. Don't forget to set `whoseTurn`!

```
public class Board {
    static final boolean DEBUG = true;
    static final int X = 1;
    static final int O = -X;
    static final int EMPTY = 0;
    static final char displayX = 'X';
    static final char displayO = 'O';
    static final char displayEMPTY = '-';

    int n;           // how big the board is
    int whoseTurn;
    int[][] sqs;     // the array!

    Board(int n) {

    }

    Board myClone() {

    }
}
```

8. (10 points) ...still in `Board`, assume it has `MoveList legalMoves()` (an `ArrayList` of the legal moves on this Board), and `void playMove(Move)` (which plays a `Move`). Complete:

```
BoardList generateNextBoards() {
```

9. (10 points) Given this `BinaryTree` class (assume standard accessors), finish `ht()` (which returns the depth of the deepest leaf); hint: an empty tree has `ht=0`.

```
class BinaryTree {
    Integer root;
    BinaryTree right, left;

    boolean isEmpty() {
        return root == null;
    }

    int ht() {
```

10. (20 points) Write a complete `NaryTree` class for use as a game-tree. It should have a `Board` at the root, and a `KidList` of `NaryTree` children. Include a constructor which is passed a `Board` and builds the complete game-tree under it. Assume `Board` has a `BoardList generateNextBoards()` method.

11. (15 points) Use an array of `Object`s to write a complete `Stack` class with methods `void push(Object)`, `Object pop()` and, `boolean isEmpty()`. Assume there will never be more than 100 `Object`s in the `Stack` (read the next question before beginning!).

12. (10 points extra credit!!!!) Still using an array, make the `Stack` in the previous question expandable (i.e. start with room for 100, but don't crash when the 101st (or 1000001st) `Object` is pushed -- instead, double the size of the array). But, you must make it fit on this page somewhere!