- **These must be completed and shown to your lab TA either by the end of this lab, or by the start of your next lab.**
- **If you work with a partner, you must both be present to be marked for the lab.**
- **The bonus question is worth an extra 0.25 points ($1/8^{\text{th}}$ of a lab), but only if you successfully finish the rest of the lab. You cannot get bonus points until you finish the lab.**

1. Download `binaryHeap.zip` from the course web page under Lab 5. Compile the code with `make`. You need to complete the `printHeap` function so that it prints the contents of the heap in a "tree-like" fashion. For example, if your heap (as an array) is `[0,2,1,4,3,9,5,7,6,8]`, then `printHeap` should output:

```
      5
   1
      9
0
      3
         8
   2
         6
      4
         7
```

   If you rotate this output by 90 degrees, you can see the tree.

   Hint: You may find it easier if you try the following first: First print the current element, then its left subtree, and then its right subtree. Preface each element with a number of asterisks equal to its depth in the heap. For the above heap, you should see the following output:

```
0
*2
**4
***7
***6
**3
***8
*1
**9
**5
```

   Can you then modify this code to produce the original format?

The remaining questions are designed to emphasize the fact that often the **simplest** algorithm is best.

2. Implement the following function:

```
/**
 * PRE:  heap points to an array representing a heap
 *       key is the value to be removed from the heap
 *       size is the number of elements in the heap
 * POST: all elements with key value = key have been removed from
```

```
 *  the heap and size is the new heap size.
 */
void remove(int* heap, int key, int & size);
```

What is the asymptotic running time of your solution (as a function of `size`)? Be prepared to explain how your code works.

3. Implement the following function:

```
/**
 * PRE:  heap1 and heap2 contain size1 and size2 elements respectively.
 * POST: output a new heap (whose size is size1+size2) containing all
 *  the elements in heap1 and heap2 (including duplicates).
 */
int* mergeHeap(int* heap1, int* heap2, int size1, int size2);
```

What is the asymptotic running time of your solution (as a function of `size1` and `size2`)? Be prepared to explain how your code works.

4. **(Bonus)** Implement the `insert` and `swapUp` functions in `bonus.cpp`. (You can test the bonus program with `make bonus`.)

   What is the asymptotic running time of both of these (in terms of `size`)?