Exam 3 (Review)

1. **Recursion** – Be prepared to explain any of the following in simple terms:
    a. What are the parts that make up a recursive algorithm? What role does each of these play in solving a large complex problem?
    b. What is the difference between an iterative and a recursive solution?

2. **Recursion (code)** – Be able to identify the following for a recursive function, such as the one shown below:
    a. How many recursive calls are made to the function counter() in the example below?
    b. What is the output?
    c. Look at each statement in the function definition and note what each statement's significance is in recursion (you can disregard the "cout" statements, but everything else is needed here.)
    d. (Hint: your answers from #1 should connect to your answers in part c of this problem)

```
#include <iostream>
using namespace std;


void counter(int num) {
   if (num == 0)
      cout << num << " ";
   else {
     cout << num << " ";
      counter(num - 5);
   }
}


int main() {
   counter(45);
   return 0;
}
```

3. **Sorting** – Review the following sorting algorithms we covered in class and lab:
    a. bubble sort
    b. selection sort

    *Each one of these has multiple iterations of a specific task (we can call this a subtask) in order to sort a list of values. Make sure you are able to explain what the subtask accomplishes given a list of values.

For example, what does 1 iteration of selection sort achieve given a list of values?

4. **Search (and complexity)** – review the following search algorithms
    a. Sequential Search
    b. Binary Search

    *What are the restrictions (if any) for using these?
    *How many steps does it take in the worst case to give an answer if I provide you with a list of 1,000 numbers? 2,000?
    *Is one always better than the other? We mentioned in class that one of these can be much much faster than the other – does this mean it is the superior choice in all situations? Be prepared to explain your reasoning.

5. **Trees** - What is the difference between a linked list and a tree-based data structure? How are they similar?

6. **Trees** – Review the tree based data structures that we introduced in class (there were only two so far)

7. **Binary Search Tree (code)** – See the list of values below. Be prepared to complete any of the following with both plain language explanations and code:

{14 8 -10 -20 3 -5 -12 6 -7 -16}

*Draw the resulting Binary Search Tree, and identify things such as the number of leaf nodes, the level of any given node (for example, the level of -5) and the height and size of the tree.

* Given a search value (for example, zero) be prepared to explain in specific steps how to determine whether or not the value is present in the tree (remember that we start with a root node, which can only see its children!)
*If the value is not present, explain the steps needed to insert the node

*You can assume that any code task you are given for your binary search tree will contain Nodes with a format like the one shown below:

```
class Node {
    public:
        int key;
        Node *left;
        Node *right;

        Node(int inKey) {
            key = inKey;
            left = right = nullptr;
        }
};
```