CS 315 Week 1 (Jan 28 and 30) summary and review questions

SUMMARY:

- Data structures are how collections of (logically related) data are stored internally on a computer in order to make software design simpler and/or more efficient. Primary resources that we want to use efficiently are time and storage memory.
- Data structures play a key role in the design of virtually every type of software.
- Data structures can be classified into abstract structures and concrete ones.
- Abstract data structures are a specification for a collection of data and the operations to manipulate the data. Such specifications are found to have wide applicability.
- Examples of abstract data structures are:
    - Stack
    - Queue
    - Priority queue
    - Dictionary etc.
- Stack is a container on which we can perform insert and delete. Deletion will follow last-in first-out. Queue is similar except that the deletion will follow the first-in-first-out sequence. Priority queue is also similar except that deletion will follow the highest-priority-in-first-out.
- Dictionary is a container that supports search, insert and delete.
- Concrete data structure denotes how the data is physically stored in the memory of a computer. Examples are linked list array, binary trees, hash table etc.
- It is possible to implement a given abstract data structure using more than one concrete data structure but usually one of them is better than the rest.
- Arrays are best for providing constant time access to the key in the k-th position independent of how big k is or n (the size of the array) is. Lists provide greater flexibility to insert or delete at a specified position, but accessing the k-th item takes k steps.
- Recursive programs involve calls to themselves.

- To successfully implement a recursive program some rules should be followed. First, make sure to provide exit from recursion via base cases. Second make sure that a recursive call makes progress towards base case(s). Once a recursive call is made you need not know any details about how many further calls are made. Just take the return value from the call and proceed. Whenever possible, make sure that the same call is not made more than once. Such redundant calls can significantly degrade the performance. Some examples are directly coding the recursive formulas for computing the binomial coefficient C(n, m) or the n-th Fibonacci number.
- Tail recursion is a particularly simple form of recursion in which there is only one self-call appearing as the last instruction of the code.

- Sometimes, a recursive algorithm can be faster than an iterative algorithm. An example is the computation of x for a given x and n. (A fast recursive algorithm was discussed in class and is presented in Chapter 2 of the text.)

Review questions

1) If the following operations are performed on a priority queue, what is the sequence of outputs produced? (Note : not all operations produce an output.)

   Insert(9); insert(12); insert(8); insert(11); delete(); delete(); insert(14); delete()

2) Answer (1) for a stack and a queue.

3) Write a recursive program to insert a key k at position j of a linked list s.

4) In lecture 2, we looked at a recursive algorithm for computing x^n. Show the successive recursive calls made when computing x^{35}.

5) Read the following recursive program and answer the questions below:

```
int f(int k, int m) {
    if (k >= m) return 0;
    else if (m%k == 0) return 1;
```

```
    else return f(k+1, m);
  }
```

(a) What is the output on input k = 5 and m = 9? On input k = 2 and m = 17? And on input k = 6 and m = 27?

(b) Let m > 1. Argue that f(2, m) will return 0 if m is prime and 1 if m is composite.

Hint: recall that m is a prime if m has no divisor j in the range 2 <= j <= m – 1. Show that f(k, m) returns 0 if m has no divisor j such that k <= j <= m – 1.