Topics covered by Test 1 (CSCE 221)

You should be familiar with the following topics.

1. Analysis of iterative algorithms.
   (a) Understand behavior of an algorithm using a small input data
   (b) Find a running time function for this algorithm
   (c) Use Big-O asymptotic notation to classify this algorithm
   (d) Find an input such that the algorithm takes the minimum number of operations (the best case)
   (e) Find an input such that the algorithm takes the maximum number of operations (the worst case)
   (f) Find the average case for this algorithm

2. Practice problems.
   (a) Homework questions
   (b) Assignment 1: `Collection class` implementation, Assignment 1 (three parts)
   (c) Find a running time function and use the Big-O notation to classify the algorithms below

```
//-Algorithm 1-----------------------------------------------------
   int i = n;
   while(i > 0) {
      i /= 10;
   }
//-Algorithm 2-----------------------------------------------------
   int i = n; int s = 0;
   while(i > 0) {
      for(j = 1; j <= i; ++j) s++;
      i /= 2;
   }
//-Algorithm 3-----------------------------------------------------
   int i = n;
   while(i > 0) {
      for(j = 1; j <= n; ++j) s++;
      i /= 2;
   }
//-Algorithm 4------------------------------------------------------
   int i = n;
   for(j = 1; j <= n; ++j)
      while(i > 0) i /= 2;
//-Algorithm 5------------------------------------------------------
   for(j = 1; j <= n; ++j) {
      int i = n;
      while(i > 0) i /= 2;
   }
```

3. Provide the best algorithm for all the problems below. Write their running time functions using Big-O notation.
   (a) Find the inner product of two vectors `v1` and `v2`, each of size `n`.
   (b) Find the number of occurrences of an element `t` in a vector/array `v`, of size `n`.
   (c) Concatenate two strings, each of `n` characters long.
   (d) Remove the `i-th` element from a vector/array
   (e) Find in a vector/array the sum of only those elements which are less than 43.
   (f) Add an element to the beginning/end of a vector/array `v`.
4. List the sorting algorithms implemented based on comparisons. Illustrate their behavior using a small input data. Provide their best, worst and average cases.
5. List the sorting algorithms which are non-comparison based. Provide assumptions on the input data that ensure linear running time. Provide conditions on the input sequence that allow you to make decisions about the best choice of a non-comparison based algorithm to sort input data in linear time.
6. Be familiar with material covered in class, on slides and in the course textbook.
   (a) Analysis of algorithms - Chapter 2
   (b) Sorting algorithms and their analysis - Chapter 7
      i. insertion sort, pp. 292
      ii. bubble sort
      iii. selection sort, p. 335
      iv. counting sort – see slides
      v. radix sort, p. 331
      vi. bucket sort, p. 331
   (c) definition of the stable algorithm
   (d) the lower bound theorem and its consequences, pp. 323.
7. Stack and Queue ADTs and the run times using:
   (a) the worst case analysis
   (b) the amortized analysis (lecture notes)
8. The practice questions:
   (a) What is the amortized analysis and when can it be used?
   (b) Assume that the initial size of a vector is 1 and you want to insert 32 items in the vector. If there is no enough memory then a resize function is called and it doubles the amount of the memory assigned to the vector.
   (c) How many times should the resize function be called to place all the items into the vector and keep extra space for a few more elements?
   (d) How many additional copy operations are performed to place all the items in the vector?
   (e) What is the amortized (average cost) of one insert (`push_back`) operation?
   (f) Provide answers to all the questions above when the input size of the vector is n.

(g) Assume that the initial size of a vector is 1 and you want to insert 32 items in the vector. If there is no enough memory then a resize function is called and it increments the amount of memory by adding memory for 10 more items to the vector.

(h) How many times should the resize function be called to place all the items in the vector and keep extra space for more elements?

(i) How many additional copy operations are performed to place all the items into the vector?

(j) What is the amortized (average cost) of one insert (`push_back`) operation?

(k) Provide answers to all the questions above when the input size of the vector is n.

(l) Provide the stack and queue ADT and their running times using big-O notation based on the array\vector implementation.