SYSC 2100, Sample Solution to Winter 2014 Midterm
February 27, 2014
Duration: 70 minutes
Instructor: T. Kunz

Question 1. Big-O Notation (10 marks)

1. Suppose we have a method whose worstTime(n) is linear in n. Determine the effect of tripling n on the estimate of worst time.That is, estimate worstTime($3n$) in terms of worstTime(n).

   Answer (3 marks):
   For all $n >$ some constant K, worstTime(n) $\approx Cn$, for some constant $C > 0$.
   Then worstTime($3n$) $\approx C\,(3n) = 3\,Cn \approx 3$ worstTime(n).

2. Suppose we have a method whose worstTime(n) is quadratic in n. Determine the effect of tripling n on the estimate of worst time. That is, estimate worstTime($3n$) in terms of worstTime(n).

   Answer (4 marks):
   For all $n >$ some constant K, worstTime(n) $\approx Cn^2$, for some constant $C > 0$.
   Then worstTime($3n$) $\approx C\,((3n)^2) = 9\,Cn^2 \approx 9$ worstTime(n).

3. Suppose we have a method whose worstTime(n) is constant. Determine the effect of tripling n on the estimate of worst time. That is, estimate worstTime($3n$) in terms of worstTime(n).

   Answer (3 marks):
   For all $n >$ some constant K, worstTime(n) $< C$, for some constant $C > 0$.
   Then worstTime($3n$) < C, so worstTime($3n$) = worstTime(n).

Question 2: Recursion (10 marks)

A ***permutation*** is an arrangement of elements in a linear order. For example, if the elements are the letters 'A', 'B', 'C' and 'D', we can generate the following 24 permutations:

ABCD BACD CABD DABC ABDC BADC CADB DACB ACBD BCAD CBAD DBAC
ACDB BCDA CBDA DBCA ADBC BDAC CDAB DCAB ADCB BDCA CDBA DCBA

Determine the output from the following ***(potentially incorrect)*** version of the recPermute method after an initial call to permute ("ABC") invokes recPermute (['A', 'B', 'C'], 0);

```
/**
* Finds all permutations of a subarray from a given position to the end
of
* the array.
*
* @param c an array of characters
* @param k the starting position in c of the subarray to be permuted.
*
* @return a String representation of all the permutations.
*
*/
public static String recPermute (char[ ] c, int k)
{
      if (k == c.length - 1)
            return String.valueOf (c) + "\n";
      else
      {
            String allPermutations = new String();
            char temp;
            for (int i = k; i < c.length; i++)
            {
                  allPermutations += recPermute (String.valueOf
            (c).toCharArray(), k+1);
                  temp = c [i];
                  c [i] = c [k];
                  c [k] = temp;
            } // for
            return allPermutations;
      } // else
} // method recPermute
```

Answer (10 marks):

ABC
ABC
ABC
ABC
BAC
BAC

Question 3. ADT List (12 marks)

1. Hypothesize the output from the following code:

```
ArrayList letters = new ArrayList();

letters.add ("f");
letters.add (1, "i");
letters.add ("e");
letters.add (1, "r");
letters.add ("e");
letters.add (4, "z");
System.out.println (letters);

letters.remove ("i");
int index = letters.indexOf ("e");
letters.remove (index);
letters.add (2, "o");
System.out.println (letters);
```

Answer (2 marks):
[f, r, i, e, z, e]
[f, r, o, z, e]

2. In the Java Collections Framework, the LinkedList class is designed as a circular, doubly-linked list with a dummy entry (pointed to by the header field). What is the main advantage of this approach over a circular, doubly-linked list with head and tail fields?

Answer (2 marks):
In the LinkedList class, the main advantage of having a dummy entry is that every entry, even the first or last entry, has a predecessor and a successor, so there is no need for a special case to insert or delete at the front or back of a LinkedList object.

3. Briefly sketch the code for the following task (task1): for each of *n* indexes, randomly generated, retrieve the list element at that index.

Answer (2 marks):

```
public void task1 (List myList)
{
  Random rand = new Random();

  for (int i=0;i<myList.size();i++)
    myList.get(Math.abs(rand.nextInt())%myList.size());
} // method task1
```

4. Briefly sketch the code for the following task (task2): repeatedly remove the first element (at index 0) from a list until the list is empty

Answer (2 marks):

```
public void task2 (List myList)
{
  while (!myList.isEmpty())
    myList.remove(0);
} // method task2
```

5. Hypothesize the runtime complexity of each piece of code when using ArrayList and when using LinkedList.

Answer (4 marks):

```
task1 & ArrayList: O(n)
task1 & LinkedList: O(n^2)

task2 & ArrayList: O(n^2)
task2 & LinkedList: O(n)
```

Question 4. ADT Stack (10 marks)

1. Consider the usual algorithm to convert an infix expression to a postfix expression. Suppose that you have read 10 input characters during a conversion and that the stack now contains these symbols:

   Now, suppose that you read and process the 11th symbol of the input. Draw the stack for the case where the 11th symbol is:
   a) A number:
   b) A left parenthesis:
   c) A right parenthesis:
   d) A minus sign:
   e) A division sign:

   Answer (5 marks):
   Here are the correct stacks with top on the RIGHT:
   a) A: * (+
   b) B: * (+ (
   c) C: *
   d) D: * (-
   e) E: * (+ /

2. Here is an INCORRECT pseudocode for the algorithm which is supposed to determine whether a sequence of parentheses is balanced:

```
create an initially empty character stack
while ( more input is available)
{
   read a character
   if ( the character is a '(' )
      push it on the stack
   else if ( the character is a ')' and the stack is not empty )
      pop a character off the stack
   else
      print "unbalanced" and exit
}
print "balanced"
```

   Which of these unbalanced sequences does the above code think is balanced?
   a) ((()
   b) ())(()
   c) (()()))
   d) (()))()

   Answer (5 marks):
   The first example, a).