SYSC 2100, Fall 2005 Midterm Solutions
Instructor: T. Kunz

Question 1. Big-O Notation (10 marks)

a. 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).

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). **(3 marks)**

b. 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).

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). **(4 marks)**

c. 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).

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

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

Perform an execution-frames trace to 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 (10 marks)

a. State two advantages, and one disadvantage, of using an ArrayList object instead of an array object.

Advantages: automatic resizing; methods to accomplish common tasks such as inserting, removing, and searching; size automatically maintained. **(1 mark)**

Disadvantage: index operator, [ ], not available, and so get / set methods must be invoked to access / modify elements in the ArrayList object. **(1 mark)**

b. Show that, for the task of appending n elements to an ArrayList object, worstTime(n) is linear in n.

To append an element to an ArrayList object, the one-parameter add method is invoked. In any sequence of n calls to the one-parameter add method, there will be at most two expansions of the underlying array. For example, suppose that $n = 500$. If the capacity of the underlying array is 5000, there will be no expansion of the underlying array when n additional elements are appended. If the capacity of the underlying array is 800, there will be one expansion. If the capacity of the underlying array is 600, there will be two expansions (one expansion when $n = 600$, and one expansion when $n = 901$). So for worstTime(n), assume that there will be two expansions of the underlying array. Then there will be $n - 2$ calls to add that do not require expansion, and each of these calls entails the execution of c1 statements, for some constant c1. For the two calls to the add method that require an expansion of the underlying array, each call will entail the execution of $c2n + c3$ statements, for some constants c2 and c3. The total number of statements executed during the n calls to the add method is $(n - 2)\ c1 + 2\ (c2n + c3)$. That is, worstTime($n$) = $(n - 2)\ c1 + 2\ (c2n + c3) = (c1 + 2c2)n + 2c3 - 2c1$, which is linear in n. **(4 marks)**

c. The one-parameter add method in the ArrayList class always returns **true**. Would it make sense to change the return type from **boolean** to **void**? Explain.

Because the ArrayList class implements the Collection interface, the return type for the one-parameter add method must be **boolean**. **(2 marks)**

d. Hypothesize the output from the following code:

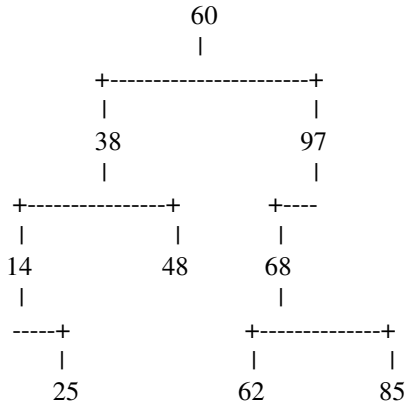
```
ArrayList letters = new ArrayList();

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);
```

The output will be **(2 marks)**

```
[f, r, i, e, z, e]
[f, r, o, z, e]
```

Question 4. Binary Trees (10 marks)

a. For the following binary tree, show the order in which elements would be visited for an inOrder, postOrder, preorder traversal.



inOrder: 14, 25, 38, 48, 60, 62, 68, 85, 97 **(2 marks)**
postOrder: 25, 14, 48, 38, 62, 85, 68, 97, 60 **(2 marks)**
preOrder: 60, 38, 14, 25, 48, 97, 68, 62, 85 **(2 marks)**

b. Show that a binary tree with n elements has $2n + 1$ subtrees (including the entire tree). How many of these subtrees are empty?

For each of the n elements, there are two subtrees, so this accounts for $2n$ subtrees. The entire tree is also a subtree, so the total number of subtrees is $2n + 1$. Each of the n elements is the root of a non-empty subtree, so there are n non-empty subtrees and therefore $n + 1$ empty subtrees. **(4 marks)**