UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 250 ALGORITHMS AND DATA STRUCTURES

Final Examination Instructor: R.E.Seviora 9-12 AM, Dec 14, 2002

Name (last, first)			Student ID			Total
1.	2.	3.	4.	5.	6.	7.

Do all problems. The number in brackets denotes the problem weight (out of 100). If information appears to be missing from a problem, make a reasonable assumption, state it and proceed. If the space to answer a question is not sufficient, use the last page. Closed book. Only numerical display calculators allowed.

PROBLEM 1 [14]

A. Recurrences

1. The running time of a certain algorithm is given by the following recurrence. Obtain a tight big-Oh bound for T(n). Show all your work.

$$T(n) = \begin{cases} O(1) & n = 1 \\ 6T(\left\lceil \frac{n}{2} \right\rceil) + O(n) & n > 1 \end{cases}$$

B. Asymptotic Analysis

Consider two algorithms A and B for problem P. Let $T_A(n)$ be the running time of algorithm A and $T_B(n)$ the running time of algorithm B. You were told that

$$T_A(n) = O(n^3)$$

$$T_B(n) = O(n^2 \log n)$$

State whether the following statements are true or false:

Statement	*true or false?*
algorithm B is always faster than algorithm A	
algorithm B is faster than algorithm A for large n	
algorithm B is sometimes faster than algorithm A	
it is possible for algorithm B to be faster than A for all n=0	
$\lim_{n \to \infty} T_B(n)/T_A(n) = 0$	

PROBLEM 2 [14]

A. Enumeration and Visitors

Enumeration and visitors offer two ways to do the same thing – to visit, one by one, all the objects in a container. Give an implementation for the `accept` method of the `AbstractContainer` class that uses an enumeration. (Note: The implementation is fairly short. It should not exceed 4 or 5 lines.)

```
public class SomeContainer extends AbstractContainer {
  public void accept (Visitor visitor) {
```

B. PolynomialAsArray Running Time Reduction

In Project 2, you were to design and code the class `PolynomialAsArray`. This class had to implement (in the Java sense) the interface `Polynomial`. This interface contained, among others, the methods `int getDegree(), double getCoefficient(i), void setCoefficient (int i, double c),` and `Polynomial plus (Polynomial p)`. Internally, the implementation had to use an array of doubles to record the coefficients of the polynomial. At all times, the length of the array had to be equal to the degree of the polynomial plus one.

You have decided to use the algorithm given below for the method `plus`. When you tested it on really large polynomials, you noted that its performance was extremely slow, even when the actual parameter was an instance of `PolynomialAsArray`.

Show a simple modification which would significantly speed up the running time of this method. Explain why the modified method is significantly faster. In big-Oh terms, what has the modification accomplished?

```
Polynomial plus (Polynomial p) {
  Polynomial sum = new PolynomialAsArray();
  int degreeSum = Math.max (getDegree(), p.getDegree());
  for (int i=0; i <= degreeSum; i++) {
    sum.setCoefficient(i, (getCoefficient(i) + p.getCoefficient(i)) ); }
  return sum;
}
```

PROBLEM 3 [14]

A. Hash Functions

You are to design a hash table which will store information about students enrolled at this University. You have decided to use the student userid as the key (an example student userid is j27smith). You now consider using the following hash function, where `charAt(i)` returns the i-th character in the key, `tableSize` is the size of the hash table:

```
public static int hash (String key, int tableSize) {
  int hashVal = 0;
  for (int i = 0; i < key.length; i++ )
    hashVal += key.charAt(i);
  return hashVal%tableSize;
}
```

Should you use this function? Explain.

B. Hash/Scatter Tables

1.Consider an open addressing scatter table of length 11 and h(x) = x mod 11. You are to insert keys {14, 36, 56, 08, 69, 48, 90, 76} into the tables in the order given, for the two collision resolution strategies listed below. In each case, show the contents of the table after all items have been inserted. If a strategy does not permit such insertion, explain why not.

(i) Open addressing with *quadratic probing*.

(ii) Open addressing with *secondary hashing*, with the secondary hash function h'(x) = 1 + (x mod 10).

x	h(x)	h'(x)
14	3	
36	3	
56	1	
08	8	
69	3	
48	4	
90	2	
76	10	

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

(i)

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

(ii)

2. Consider an open addressing scatter table with linear probing and 'lazy' withdrawal policy (i.e. a delete state). Its initial contents are below. The entries in the table hash to the values shown at the left.

(i) Give the contents of the table after the withdrawal of key 92.

(ii) Give the contents of the table after the insertion of key 82 into the table obtained in (i).

x	h(x)
34	03
31	09
14	03
70	04
92	04
59	04
27	05
09	09

(ordered by x)

0	null	empty
1	null	empty
2	null	empty
3	34	occup
4	14	occup
5	70	occup
6	92	occup
7	59	occup
8	27	occup
9	31	occup
10	09	occup

(initial)

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

(after withdrawal of 92)

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

(after insertion of 82)

C. Uniform Hashing Model

Because of the complex nature of hashing functions, it is difficult to analyze the running times of hash/scatter tables. Various simplified models are used to make the analysis more tractable. One such model is the 'uniform hashing model'. This model can be used to obtain the average running times for operations on scatter tables with open addressing.

1. State the simplifications (assumptions) of this model.

2. For this model, give a formula for the average number of probes in an unsuccessful search. Show how the formula was obtained.

PROBLEM 4 [14]

A. Tree Traversals

1. Consider a binary tree whose nodes contain single-character keys. When the keys are output in **in-order**, the output is `{ G W N T E P Z A }`. When the keys are output in **post-order**, the output is `{ G N W E T Z A P }`.

(i) Devise an algorithm for the construction of the tree from these traversals and describe its steps (in English text; Java code not required). Hint: consider divide & conquer.

(ii) Draw the binary tree whose in- and post-order traversals would result in the sequences given above.

B. Space Requirements of Trees

1. Derive an expression for the *total* space needed to represent a binary tree with n internal nodes. The expression should include the space requirement of the tree only, and exclude the space needed to store the objects referred to by the nodes of the tree.

2. Assuming 4-byte integer and object references, give the total space for a tree with 100 internal nodes.

PROBLEM 5 [14]

A. Search Trees and AVL Trees

1. What is the minimum and the maximum number of internal nodes in an AVL tree of height three? Draw a concrete example of an AVL tree which illustrates the *minimum* case.

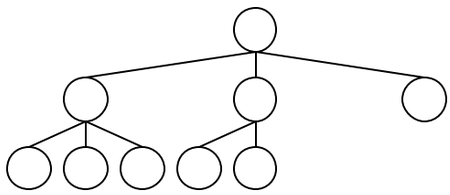
2. Consider the set of keys `{4,2,1,7,6,5,3}`.

(i) Draw the *binary search tree* obtained when the keys are inserted one-by-one in the order given into an initially empty tree.

(ii) Draw the *sequence* of AVL trees obtained when these keys are inserted one-by-one in the order given into an initially empty AVL tree. Identify the rotations involved, if any.

B. Priority Queues and Heaps

1. Consider the *array* representation of a complete ternary tree. An example ternary tree is below. Assume that the root of the tree is at the position (index) one in the array.



(i) For a node at position i > 1, give the formula for the position of its parent in the array.

(ii) Give the formulas for the positions of the children of the node at the position i in the array.

2. Consider a priority queue implemented as a heap-ordered, complete *ternary* tree (min-heap). Draw the sequence of trees obtained when the keys `{9,8,7,6,5,4,3,2}` are inserted in the order given into an initially empty priority queue.

PROBLEM 6 [16]

A. Divide-and-Conquer Algorithms

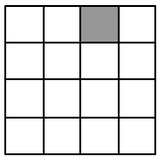
A *defective chessboard* is a $2^k \times 2^k$ board of squares with exactly one defective square. In this problem, you are required to tile a defective chessboard with triominoes. (A triomino is an L-shaped piece which covers three squares.) In the tiling, two triominoes may not overlap. The triominoes must **not** cover the defective square and must cover all other squares. You have a sufficiently large supply of triomino pieces. (A simple calculation shows that the number of triominoes needed is $(2^{2k}-1)/3$, which is always an integer.)

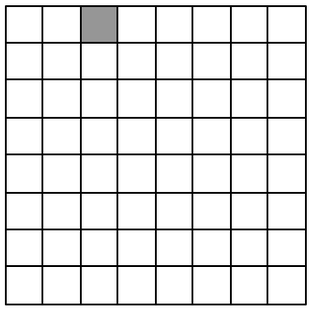
Devise a divide-and-conquer algorithm for this defective chessboard tiling problem.

(i) Briefly describe the steps of the algorithm

(ii) Illustrate the steps of your algorithm on the 4×4 defective chessboard given below left. Show how the problem is divided into subproblems and the solution for the subproblems.

(iii) Show the tiling resulting from your algorithm for the 8×8 board below right.





B. Greedy Problem Solving Strategy

Consider the *inverse change counting* problem. You have a set of coins, $P=\{p_1, p_2, .. p_n\}$. Let d_i be the denomination (value) of coin p_i. You are required to count out an amount A of money. You would like to use as many coins as possible (for example, to lighten your pocket when it is too full).

1. Give a mathematical formulation of this problem, i.e. specify the objective function, the constraints...

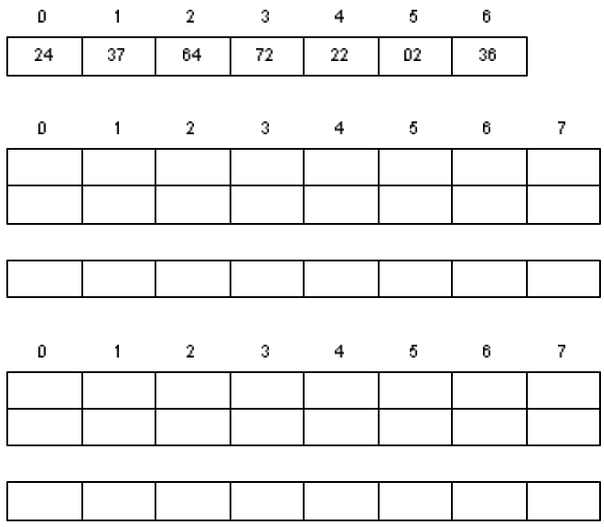
2. Describe a greedy algorithm for solving this problem. To illustrate your algorithm, consider the set P = {p,p,p,p,p,n,n,d,q} and show the coins handed out for A=10 cents. Here, p is a penny (1 cent), n is a nickel (5c), d is a dime (10c), and q is a quarter (25c).

3. Would this algorithm ever fail? Illustrate using the set P of the preceding subquestion (2).

PROBLEM 7 [14]

A. Radix Sort

At the top of the figure below is an array of octal numbers. Show the steps involved in sorting this array using the radix sort.



B. Sorting

1. Consider the following sorters discussed in the class - the (straight) insertion sorter, the bubble sorter, and the selection sorter. State the big Oh bound for the running time these sorters need to sort an *already sorted* array of n items.

Sorter	*O() for an already sorted array*
Insertion	
Bubble	
Selection	

2. This problem is concerned with the bubble sorting algorithm discussed in the class (see below). A simple modification can significantly reduce its running time when the array is sorted, or could become sorted partway through its execution. Show the modification. Explain how the reduction is achieved.

```
public class BubbleSorter extends AbstractSorter {
  protected void sort () {
    for (int i = n; i > 1; --i)
      for (int j = 0; j< i-1; ++j)
        if (array[j].isGT(array[j+1]))
          swap (j, j+1);
  }
}
```

OVERFLOW SHEET [Please identify the question(s) being answered.]