UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

E&CE 250 – ALGORITHMS AND DATA STRUCTURES

Final Examination Instructors: R.E.Seviora and L.Tahvildari 3 hrs, Apr. 11, 2001

Name:					Student ID:	
1.	2.	3.	4.	5.	6.	Total:

Do all problems. The number in brackets denotes the relative weight of the problem (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 (overflow) page. Closed book. No calculators allowed.

PROBLEM 1 [16]

A. Algorithm Analysis

1.Consider the algorithm represented by the following Java program fragment. What value does **bar** compute? Derive a tight, big oh expression for the running time of the method **bar**.

```
static int bar (int x, int n)
{
    int sum = 0;
    for (int i = 1; i <= n; ++i)
        sum = sum + i;
    return  {x + sum};
}
```

2.Consider the algorithm represented by the following Java program fragment. The method **bar1** computes the same function as **bar** above. However, **bar1** is based on a different algorithm and its tight big oh bound is O(n). What value does **foo** compute? Give a tight, big oh expression for the worst case running time of the method **foo**.

```
static int foo (int x, int n)
{
    int sum = 0;
    for (int i = 1; i <= n; ++i)
        sum = sum + bar1 (i, n);
    return  {x + sum};
}
```

B. Solving Recurrences

Solve the following recurrence. You may assume that n is a power of 2. Show all your work.

$$T(n) = \begin{cases} 1 & n = 1 \\ aT(\frac{n}{2}) + n^k & n > 1, k \geq 0, a > 0 \end{cases}$$

PROBLEM 2 [16]

A. Queues

The interface Queue discussed in the lectures contained only the operations **enqueue**, **dequeue** and **getHead**. In some applications, the method **reverse()** is required. This method will reverse the order of items in the queue. For example, if the original queue contained a, b, c, d (in this order), **reverse()** would reorder its contents to d, c, b, a.
In this problem, you are asked to devise an algorithm for **reverse()**, for the case of linked list implementation of the queue. Note that you may use any of the methods of the LinkedList class in your answer.

1. Explain the basic approach to reversing you used in **reverse()** .

2. Given the algorithm for **reverse()** in Java.

```
public class QueueAsLinkedList
   extends AbstractContainer implements Queue
  {
        protected LinkedList list;

        // standard methods of Queue interface
           …
       // reverse () : reverses the order of items on the queue
       public void reverse () {
```

B. Ordered List

1. As discussed in the lectures, the **withdraw** method removes items from a list one at a time. Suppose we are required to provide an additional method, **withdrawAll** which takes one argument and withdraws all items in a list that match the given argument. Devise an O(n) algorithm for this method and write it down in Java. Assume the list is represented using an array. The method is not required to throw an exception if there is no matching item.

```
class ExtendedOrderedListAsArray
        extends OrderedListAsArray
{
        public void withdrawAll (Comparable item)
        {
```

2. Consider an implementation of the **OrderedList** interface that uses a doubly-linked list (i.e., each list element contains a reference to the immediately preceding and immediately following element on the list, if they exist). Show the running times of the **OrderedList** operations for this implementation in the table below.

Method	Ordered List Implementation	
	singly-linked list	doubly-linked list
insert	O(1)	
isMember	O(n)	
find	O(n)	
withdraw	O(n)	
get	O(n)	
findposition	O(n)	
Cursor.getDatum	O(1)	
Cursor.insertAfter	O(1)	
Cursor.insertBefore	O(n)	
Cursor.withdraw	O(n)	

Briefly explain the differences, if any, between the running times for the doubly-linked implementation and the singly-linked one.

PROBLEM 3 [16]

A. Hash/Scatter Table

1.Consider a hash table with separate chaining with ten hash locations. Using the hash function h(x) = x mod 10, insert the keys {33, 54, 69, 74, 18, 19} (in the order given) into the hash table. Draw the resulting hash table. (Note: to keep this example simple, we use table size that is not prime.)

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

2. Consider an open addressing scatter table with ten slots. For the hash function h(x) = x mod 10, insert the keys {33, 54, 69, 74, 18, 19} (in the order given) into the table. Use *linear probing* for collision resolution. Show the result in the table (2) below.

3. Consider an open addressing scatter table with ten slots. For the hash function h(x) = x mod 10, insert the keys {33, 54, 69, 74, 18, 19} (in the order given) into the table. Use *quadratic probing* for collision resolution. Show the result in the table (3) below.

4. Consider an open addressing scatter table with ten slots. For the hash function h(x) = x mod 10, insert the keys {33, 54, 69, 74, 18, 19} (in the order given) into the table. Use the *secondary hash function* $h'(x) = 1 + (x \bmod 9)$ for collision resolution. Show the result in the table (4) below.

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

(2)

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

(3)

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

(4)

5. Consider your answer to (3) above. What happens when you attempt to insert the key 94 into the final hash table obtained after inserting the keys given in (3). Explain.

B. Primary Clustering

Figure 1 illustrates an open addressing scatter table with linear probing, as it is being filled, at increments of 10 percent of table capacity. The black bands in this figure represent clusters. A cluster is a set of adjacent occupied entries in the table. Figure 2 illustrates the table being filled under the same scenario. However, in this case, double hashing is used.

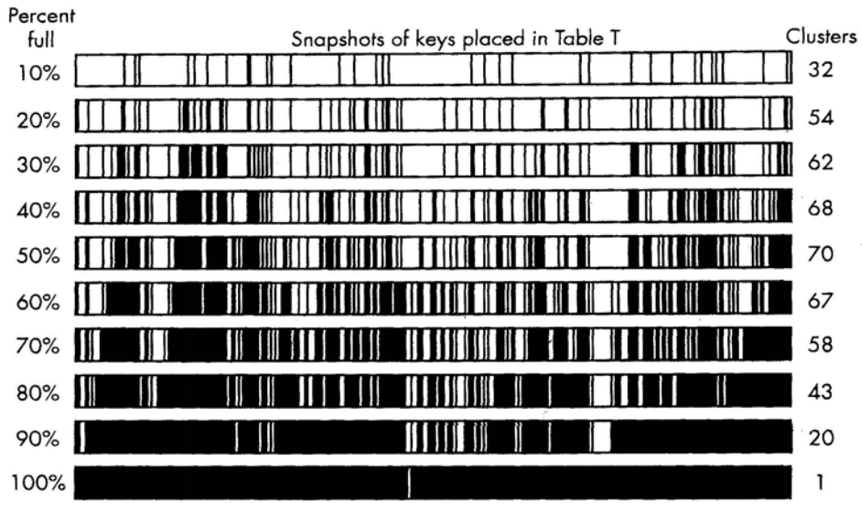


Figure 1 – Linear Probing Clusters

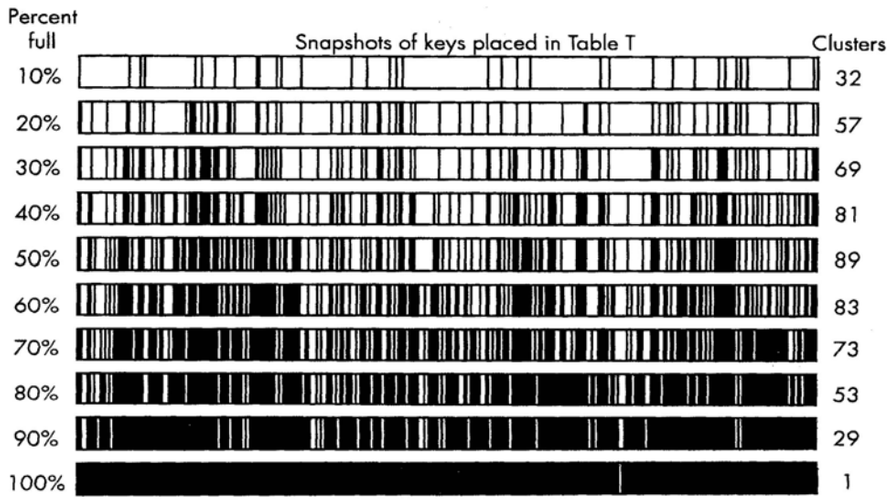


Figure 2 – Double Hashing Clusters

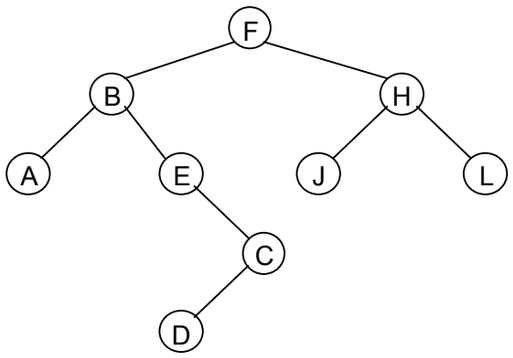
1. State what the 'primary clustering' phenomenon is.

2. Various open addressing schemes differ in their susceptibility to primary clustering. How is this reflected in the two figures above?

PROBLEM 4 [16]

A. Trees

1.Consider the binary tree shown below. For each of the traversals listed, give the order in which the nodes are visited.



preorder									
inorder									
postorder									
breadth-first									

2. As discussed in the class, **preorder, postorder** and **inorder** traversals are special cases of the more general **depthfirst** traversal. Rather than implementing individual traversals separately, we made use of a design pattern, called *adapter*, which allowed a single depthFirstTraversal method to provide most of the needed traversal functionality. This point is illustrated in the following for the postorder case:

```
Visitor v = new PrintingVisitor ( );
Tree t = new SomeTree ( );
// …
t.depthFirstTraversal (new PostOrder (v));
```

Consider an expression tree. Complete the following PostOrder adapter class and the visit method of the PrintingVisitor. The PrintingVisitor prints the contents of the tree in the *postfix* notation. It is permissible that the postfix format such as ab/cd-e*+ be printed with one operator/variable per line.

```
public class PostOrder
        extends AbstractPrePostVisitor
{
        protected Visitor visitor;
        public PostOrder (Visitor visitor)
        {
        .
        .
        .
        }
        public postVisit (Object object)
        {
             .
             .
             .
        }
        public Boolean isDone ( )
        {
             .
             .
             .
        }
}

 public class PrintingVisitor
         implements Visitor
{
         public void visit (Object object)
         {
              .
              .
              .
         }
          // etc
 }
```

B. Space Requirements of Trees

Derive an expression for the total space needed to represent a tree of *n* internal nodes using each of the following classes. Assume four-byte integers and references. The expression should include the space requirement of the tree only, and not include the space needed to store the objects referred to by the nodes of the tree.

1. **GeneralTree**

```
public class GeneralTree
    extends AbstractTree
{
    protected Object key;
    protected int degree;
    protected LinkedList list;

    // …
}
```

2.**NaryTree**

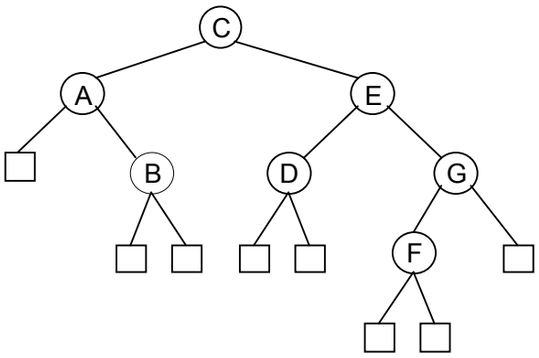
```
public class NaryTree
    extends AbstractTree
{
    protected Object key;
    protected int degree;
    protected NaryTree[] subtree;

    // …
}
```

PROBLEM 5 [20]

A. Search Trees

1. For each node shown in the binary tree below, show its depth, height, and AVL balance factor. Write your answers in the following table.



node	depth	height	balance factor
A			
B			
C			
D			
E			
F			
G			

2. Draw the sequence of AVL trees obtained when the following keys are inserted one-by-one, in the order given into an initially empty AVL search tree: {F, E, A, B, D, C, G}. Identified rotations, if there is any.

3. What is the main advantage of an **AVL Tree** over a **Binary Search Tree** (BST)?

B. Binary Trees and Heaps

1. The method **isHeap** determines whether a particular binary tree is a heap. The implementation this method given below contains a few bugs. Find these bugs and correct them.

```
public class BinaryTree
      extends AbstractTree
{
      public Boolean isHeap ()
      {
            if (isEmpty ())
                return true;
            if (!getLeft ().isEmpty () || getLeft ().getKey ().isLT (getKey ()))
                return false;
            if (!getRight ().isEmpty () || getRight ().getLey ().isLT (getKey ()))
                retuen false;
            return getLeft ().isHeap () && getRight ().isHeap ();
      }
 }
```

2. What is the big Oh bound on the running time of this algorithm after modification?

3. (a) How many internal nodes are there in a **perfect binary tree** of height $h \geq 0$?

(b) What is the height of a **perfect binary tree** with *n* internal nodes?

PROBLEM 6 [16]

A. Algorithmic Patterns

1. Consider the following problem: There are two computers C_1, C_2, and a set of *n* programs, $\{P_1, P_2, ..., P_n\}$ to be run. Let T_i be the time required to run P_i. Assume that each computer can only run one program at a time. You are to assign the programs to the computers so that the time from the start of execution of the first program until the completion of the last program is minimized. Give a mathematical formulation of this problem that includes defining the binary decision variables, specifying the objective function and the constraints.

2. Consider the 0/1-knapsack problem with the capacity $C = 18$. Solve the problem using the greedy by profit, greedy by weight, and greedy by profit density alternatives solution. Give your answers in the table below.

i	w_i	p_i	p_i / w_i	Greedy by			optimal solution
				profit	*weight*	*density*	
1	10	10					
2	6	6					
3	3	4					
4	8	9					
5	1	3					
			total weight				
			total profit				

B. Sorting Algorithms

1. Illustrate the operation of **straight insertion sort** by completing the left table below. In successive rows of the table, show the array contents after each pass of the algorithm.

2. Illustrate the operation of **bubble sort** by completing the right table below. In successive rows of the table, show the array contents after each pass of the algorithm.

9	8	6	7	5	0

9	8	6	7	5	0

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