NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
EXAMINATION FOR

SEMESTER 2 2007/2008

CS1102S—Data Structures and Algorithms

April 2008 **Time Allowed: 2 Hours**

INSTRUCTIONS TO CANDIDATES

This is an open book examination. Any written and printed material may be used during the examination.

Please observe the following items:

1. Enter your matriculation number here: | | | | | | | | |
2. Enter the answers to the questions in this answer book in the provided spaces.
3. This examination booklet has 16 pages, including this cover page, and contains 8 questions.
4. Answer all questions.
5. The maximal marks achievable is indicated for each question. Overall, the maximal number of marks is 100.

Good Luck !

Do not write below this line

Question	1	2	3	4	5	6	7	8	Total
Maximal Score	26	9	15	9	10	11	8	12	100
Student Score									

Question 1: (26 marks) A vegetable vendor is ordering a software program to keep track of the kinds of vegetables available in his store, including the name of the vegetable, the supplier name, purchase price per kg and sales price per kg.

1. (10 marks) Design an ADT for Vegetable by listing the operations, including parameters and return values, that allow for creating a Vegetable object with a given name, to access and change the supplier name, to access and change the purchase price per kg, and to access and change the sales price per kg.

Answer:

```
createVegetable(in name:string)
// create vegetable with given name
getSupplier():string {query}
// retrieves the supplier
setSupplier(in name:string)
// change supplier to name
getPurchasePrice():integer {query}
// retrieves the purchase price
setPurchasePrice(in pp:integer)
// change purchase price to pp
getSalesPrice():integer {query}
// retrieves the sales price
setSalesPrice(in sp:integer)
// change purchase price to sp
```

2. (10 marks) Write an implementation of the ADT Vegetable using a Java class Vegetable. Make sure you follow the recommendation of keeping data items in private fields. Represent names using String and prices using **int**, denoting Singapore cents.

Answer:

```
class Vegetable {
    private String name, supplier;
    private int salesPrice, purchasePrice;
    public Vegetable(String n) {
       name = n;
    }
    public String getSupplier() {
       return supplier;
    }
    void setSupplier(String s) {
       supplier = s;
    }
    public int getPurchasePrice() {
       return purchasePrice;
    }
    public void setPurchasePrice(int pp) {
       purchasePrice = pp;
    }
    public int getSalesPrice() {
       return salesPrice;
    }
    public void setSalesPrice(int sp) {
       salesPrice = sp;
    }
```

3. (6 marks) Assume the following Java interface:

```
interface ProductWithProfit {
   public int profit();
}
```

Write a Java class VegetableWithProfit that makes use of your class Vegetable of the previous question, but also implements the interface ProductWithProfit. Objects of class VegetableWithProfit have all operations of class Vegetable, and provide an additional non-static method profit with no parameters, that returns the profit per kg, calculated as the difference between sales price per kg and purchase price per kg.

Answer:

```
class VegetableWithProfit extends Vegetable implements ProductWithProfit {
    public int profit() {
       return getSalesPrice() - getPurchasePrice();
    }
}
```

Question 2: (9 marks) Instances of classes that implement the interface

```
interface ProductWithProfit {
   public int profit();
}
```

all have a way to access the sales profit by calling the method `profit`. In this question, you need to provide a method to sort a list of such instances in order of increasing profit. For this purpose, the JCF class `Collections` defines a method

```
static <T> void sort(List<T> list, Comparator<? super T> c)
```

which destructively sorts a given `List<T>` in ascending order. Recall that the interface `Comparator<T>` requires the method

```
public int compare(T o1, T o2)
```

which compares its two parameters for order. The method `compare` is supposed to return a negative integer, zero, or a positive integer as the first parameter is less than, equal to, or greater than the second.
Define a static method `sortProductWithProfitList` that destructively sorts a given `List` of `ProductWithProfit` instances, using the mentioned `sort` method of `Collections`, in order of increasing profit. Give the code of the method, including parameter type declarations and of the additional class that you need to define in order to apply the static method `sort` in the JCF class `Collections`.

Answer:

```
static void sortProductWithProfitList(List<ProductWithProfit> list){
    Collections.sort(list,new ProductWithProfitComparator());
}

class ProductWithProfitComparator
   implements Comparator<ProductWithProfit> {
   public compare(ProductWithProfit p1, ProductWithProfit p2) {
        if (p1.getProfit() < p2.getProfit())
           return -1;
        else if (p1.getProfit() == p2.getProfit())
           return 0;
        else return 1;
   }
}
```

Question 3: (15 marks) Let us say a merchant needs to keep track of Product objects in his store, using a data structure. In order to easily identify the products, the Product class implements the KeyedItem<Integer> interface; a call p.getKey() returns the Integer product number of product p. To illustrate the situation, consider:

```
class Product implements KeyedItem<Integer> {
   ...
}
interface KeyedItem<KT extends Comparable<?super KT>> {
   public KT getKey();
}
```

There are at most 10000 products in the store, and the product numbers range from 0 to 9999. Furthermore, for any two distinct products, the product numbers are distinct.
The merchant will need the following operations:

- Create an empty data structure,
- insert a new Product into the data structure, whose product number is a given Integer *pn*, raising an exception if there is already a product with product number *pn*,
- retrieve the Product with a given product number, returning **null** if there is no such Product, and
- delete the Product with a given product number *pn*, returning **true** if there was a Product with product number *pn*, and **false** otherwise.

1. (2 marks) What is the name of an ADT that provides these operations? (simply give the name, without explanation)

   Answer: Table

2. (6 marks) Consider the following implementation of the ADT:

```
class UnsortedProductArray {
   static private MaxSize = 10000;
   private int currentSize;
   private Product[] products;
   public UnsortedProductArray() {
      currentSize = 0;
      products = new Product[MaxSize];
   }
   public void insert(Product p) throws Exception {
      if (retrieve(p.getKey())!=null)
         throw new Exception("duplicate item");
      else products[currentSize++] = p;
   }
   public Product retrieve(Integer pn) {
      Product foundProduct = null;
      for (int i=0; i < currentSize; i++)
           if (products[i].getKey().equals(pn)) {
              foundProduct = products[i];
              break;
           }
      return foundProduct;
   }
   public boolean delete(Integer pn) {
      boolean foundFlag=false;
      for (int i=0; i < currentSize; i++)
           if (products[i].getKey().equals(pn)) {
              foundFlag = true;
              products[i] = products[currentSize - 1];
              currentSize = currentSize - 1;
              break;
           }
      return foundFlag;
   }
}
```

What is the worst-case complexity of the operations insert, retrieve, and delete in relation to the current number n of items currently stored in the data structure, using the $O(\ldots)$ notation?

Answer:

- insert: $O(n)$
- retrieve: $O(n)$
- delete: $O(n)$

3. (7 marks) Let us say insertion, deletion and retrieval are very frequent operations. Give an implementation that you would recommend for this data structure in pseudo-code or Java, and give the worst-case complexity of the operations insert, retrieve, and delete in relation to the current number n of items currently stored in the data structure, using the $O(\ldots)$ notation!

Important Note

The marks you get for this part depends on the correctness of your program *and* on the worst-case complexity of the insert, retrieve, and delete operations. The lower the complexity, the higher your marks.

(additional space for Question 3)

Answer: Idea: Use an array of size 10000 and use the given product number (ranging from 0 to 9999) as index.

```
class ProductTable {
   private int MaxSize = 10001;
   private Product[] products;
   public ProductTable() {
      products = new Product[MaxSize];
   }
   public void insert(Product p) throws Exception {
      if (products[p.getKey()] == null)
         products[p.getKey()] = p;
      else throw Exception(...);
   }
   public Product retrieve(Integer pn) {
      return products[pn];
   }
   public boolean delete(Integer pn) {
      boolean foundFlag = (products[pn]!=null);
      products[pn] = null;
      return foundFlag;
   }
}
```

All 3 operations have $O(1)$ complexity. Also permissible is a binary search tree with the the usual logarithmic times, but with less marks.

Question 4: (9 marks) Let us say you create a repository of all known animal species. There are millions of species and for each species, you want to store possibly large amounts of data, including photos and videos. Overall, the required memory will by far exceed the main memory of your computer. On the other hand, you have access to a large external storage in form of random access files, organized in blocks, which can easily hold all data.
You will have to perform many retrieve and insert operations, but never any deletion or traversal. For the retrieve and insert operations, the `String` that represents the name of the species is used as key.

1. (2 marks) The module covered two alternative implementations of data structures that provide these operations under the mentioned memory requirements. What are the alternatives?

   Answer:

   - External hash table
   - External B-Tree

2. (2 mark) Which implementation of the data structure would you recommend? (simply name the implementation)

   Answer: External hash table

3. (5 marks) Give a careful argument for your recommendation, compared to the alternative, using at most three English sentences.

   Answer: External hash table is best because hashing using strings works well; will be faster than accessing B-tree (log n time)

Question 5: (10 marks) Assume that you represent binary trees, whose nodes hold Integer values, using instances of the following class.

```
public class BinaryTree {
   private Integer value;
   private BinaryTree leftChild;
   private BinaryTree rightChild;
   public BinaryTree(Integer v, BinaryTree left, BinaryTree right) {
      value = v;
      leftChild = left;
      rightChild = right;
   }
   public Integer getValue()    { return value;      }
   public BinaryTree getRight() { return rightChild; }
   public BinaryTree getLeft()  { return leftChild;  }
   public List<Integer> breadthFirst() {
      // to be provided
   }
}
```

Note that the constructor may be called using **null** as a subtree. Therefore,

```
new BinaryTree(new Integer(4),null,
               new BinaryTree(new Integer(5),null,null))
```

represents a binary tree whose root with value 4, and which only has one child, the right child, which is a leaf with value 5.
Give a complete implementation of the method breadthFirst(), which should return a List<Integer> that contains all items, listed in breadth-first order. For the resulting list, subsequent calls of poll should return items of nodes at a given level l_1 before items of nodes at a level l_2 if $l_1 < l_2$.

Note

You may use a private helper function.

Hints

- You may use a LinkedList implementation of List<E>, which implements the following methods:

  ```
  void addFirst(E e)  // inserts element e at beginning of this list
  void addLast(E e)   // inserts element e at end of this list
  E poll()            // retrieves and removes element from beginning of this list
  ```

- You may need an additional position-oriented ADT...

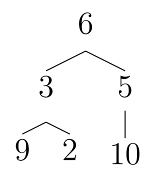
Use the next page for your solution to this question!

(space for Question 5)

Answer:

```
public List<Integer> breadthFirst() {
    List<BinaryTree> toBeHandled = new List<BinaryTree>();
    List<Integer> values = new List<Integer>();
    toBeHandled.addLast(this);
    while (toBeHandled.peek() != null) {
        BinaryTree nextTree = toBeHandled.poll();
        values.addFirst(nextTree.getValue());
        if (nextTree.getLeft() != null)
            toBeHandled.addLast(nextTree.getLeft());
        if (nextTree.getRight() != null)
            toBeHandled.addLast(nextTree.getRight());
    }
    return values;
}
```

Question 6: (11 marks) A heap of integers is a data structure that allows for efficient insertion of arbitrary integers, and deletion of the largest integer inserted so far. Internally, the heap keeps a complete binary tree, represented by an array that contains the nodes in breadth-first order. Thus, the following heap



is represented by the array `{6,3,5,9,2,10}`. Consider the following textbook implementation of an integer heap, where ArrayList<Integer> items is **protected**.

```
public class IntegerHeap {
  protected ArrayList<Integer> items;
  public Heap() {
    items = new ArrayList<Integer>();
  }
  public boolean heapIsEmpty() {
    return items.size()==0;
  }
  public void heapInsert(Integer newItem)
         throws HeapException, ClassCastException {
    // details not shown here
  }
  public T heapDelete() {
    Integer rootItem = null;
    int loc;
    if (!heapIsEmpty()) {
      rootItem = items.get(0);
      loc = items.size()-1;
      items.set(0, items.get(loc));
      items.remove(loc);
      heapRebuild(0);
    }
    return rootItem;
  }
  protected void heapRebuild(int root) {
    // turns semi-heap rooted at ''root'' into heap
    // details not show here
  }
}
```

Let us assume that occasionally, you need to remove an integer from the heap that is *not* the largest inserted so far.

1. (5 marks) Add a method **void** cancel(Integer i), that removes the integer i from the heap, if it exists, and does nothing otherwise. Do not change the implementation of the existing heap operations.

```
public class IntegerHeapWithCancel extends IntegerHeap {
  public void cancel(Integer i) {
    // to be provided
  }
}
```

(space for answer to Part 1)

Answer:

```
public void cancel(Integer i) {
   // look for entry
   int i = 0;
   for (;  i < items.size(); i++) {
     if (items.get(i)) break;
   }
   if (i != items.size()) {
      int lastIndex = items.size()-1;
      items.set(i,items.get(lastIndex));
      items.remove(lastIndex);
      heapRebuild(i);
   }
}
```

2. (2 marks) What is the worst-case complexity of your cancel in IntegerHeapWithCancel with respect to the current heap size n?

   Answer: $O(n)$

3. (4 marks) Could the worst-case complexity of cancel in IntegerHeapWithCancel be improved without overriding the methods heapInsert and heapDelete? Explain your answer with at most three sentences.

Answer: Cannot be improved, because the items in the heap are not sorted. In the worst case, the entire heap needs to be searched in order to find the item to be deleted.

Question 7: (8 marks) Consider an empty 2-3 tree of integers, on which the following operations are performed in the given order:

1. insert 30,
2. insert 50,
3. insert 100,
4. insert 40,
5. insert 20,
6. insert 90,
7. insert 80,
8. insert 70,
9. delete 80.

Draw the resulting 2-3 tree.

Question 8: (12 marks) Assume that you want to represent all living Singaporean citizens and the "parent-of" relationship between them as a graph. That means that there is an edge from Citizen a to Citizen b if and only if a is a parent of b (father or mother).

- (3 marks) Will this graph be connected? Give an explanation in one complete sentence.

  Answer: The graph will not be connected, because I know two living Singaporeans (S.R. Nathan and Lee Kuan Yew) that are not related via living persons.

- (3 marks) Is it best to use a directed graph or an undirected graph? Give an explanation in one complete sentence.

  Answer: A directed graph is best because the "parent-of" relation is not symmetric; we can express the relationship properly only using a directed graph.

- (3 marks) Estimate the length of the longest path in the graph. Give an explanation in one complete sentence.

  Answer: I estimate the longest path to have a length of 5, although it is well possible to have a length of 6, with the youngest just born, followed by his/her parent (age 20), grand-parent (age 40), great grand parent (age 60), great-great grand parent (age 80), and great-great-great grand parent (age 100).

- (3 marks) For an implementation of this graph, you have the choice between an adjacency list and an adjacency matrix. Which alternative is best in this case? Give an explanation in one complete sentence.

  Answer: The graph will contain more than 3 million vertices, but each vertex has very few (estimated maximum about 13) neighbors. Therefore, an adjacency list will be much more space-efficient than an adjacency matrix implementation.

END OF QUESTIONS