CIS 338 (Fall 2011)

Final, 12/20/11

Name (sign)

Name (print)

email

Question	Score
1	
2	
3	
4	
5	
6	
7	
8	
9	
A	
B	
C	
D	
E	
F	

Question 1

Provide, for each of the following, **a)** its worst-case performance and **b)** its expected (average/amortized) case performance. You may use the following abbreviations:

A $\sim c$	**C** $\sim c\ N$	**E** $\sim c\ N^2$
B $\sim c \lg N$	**D** $\sim c\ N \lg N$	**F** $\sim c\ N^3$

Enqueue on a Queue (linked list implementation).

Insertion sort.

Merge sort.

Three-way string quicksort.

Priority queue **`deleteMin()`**.

Binary search tree (BST) **`put()`**.

Two/three tree **`select()`**.

Left-leaning Red/Black search tree **`ceiling()`**.

Left-leaning Red/Black search tree **`keys()`**.

Hash table **`get()`**.

Question 2

Complete the method below to find the index in a *sorted* array **a** of an **Item** having **key** equal to **k**. Return -1, if there is no **Item** with **key** equal to **k** in **a**. (Assume that every **Item** has a **key** field of type **Key** and that **Key** objects are **Comparable**.)

```
int find(Key k, Item[] a) {
    return find(k, a, 0, a.length-1);
}
int find(Key k, Item[] a, int lo, int hi) {
```

Question 3

Complete the three methods below to implement a stack class.

```
public class Stack {
  private Item[] data = new Item[1];
  private int next = 0;
  public void push(Item I) {









  }
  public Item pop() {









  }
  private void resize(int max) {
    assert next < max;









  }
}
```

Question 4

Complete the method below to implement quicksort. (Assume that every **`Item`** has a **`key`** field of type **`Key`** and that **`Key`** objects are **`Comparable`**.)

```
public void sort(Item[] a) {
  sort(a, 0, a.length-1);
}
private void sort(Item[a], int lo, int hi) {
```

Question 5

The class **`PhoneNumber`** has a method **`digit()`** that returns the numeric value of the i-th digit of a 10 digit phone number. Use it to complete the method below to sort an array of phone numbers. (*Hint: use the LSD (least significant digit) sorting algorithm for fixed length strings.*)

```
private int digit(int i) {…}
public static sort(PhoneNumber[] pn) {
```

Question 6

Complete the methods below to override **`hashCode()`** in a way that is consistent with equals() (equal parts should have the same hash code; unequal parts should mostly have different hash codes) and to implement **`hash()`** to map **`Part`**s to the range [0. .257).

```
public class Part {
  private String name;
  private double weight;
  private Part[] subparts;
  public Part(String n, double w, Part[] sp) {
    name = n; weight=w; subparts = sp;
  }
  public boolean Equals(Object o) {
    if (null == o) return false;
    if (this == o) return true;
    if (this.getClass() != o.getClass()) return false;
    Part p = (Part) o;
    if (!name.equals(p.name)) return false;
    if (weight != p.weight)) return false);
    if (subparts.length != p.subparts.length) return false;
    for (int i=0; i<subparts.length; i++) {
      if (!subparts[i].equals(p.subparts[i]) return false;
    }
    return true;
  }
  public int hashCode() {









  }
  public int hash() {
```

Question 7

Below is a fragment of a BST (binary search tree) class implementation. Complete its **`rank()`** method.

```
public class BST {
  private class Node {
    Key key;
    Value val; // might be null as a result of a delete()
    Node left, right;
    int N; // number of entries in the subtree rooted here
    Node(Key k, Value v) {
      key = k, val = v; N = null == v ? 0 : 1;
    }
  }
  Node root; ...
  public int rank (Key k) {
    return rank(root, k);
  }
  private int rank(Node n, Key k) {
```

Question 8

Below is a fragment of a left-leaning Red/Black search tree class implementation. Complete its **`floor()`** method.

```
public class RedBlack {
  private class Node {
    Key key;
    Value val; // might be null as a result of a delete()
    Node left, right;
    int N; // number of entries in the subtree rooted here
    Node(Key k, Value v) {
      key = k, val = v; N = null == v ? 0 : 1;
    }
  }
  Node root; ...
  public Key floor (Key k) {
    Node n = floor(root, k);
    if (null == n) return null;
    return n.key;
  }
  private Node floor(Node n, Key k) {
```

Question 9

Below is a fragment of a Trie class implementation. Complete its **`get()`** method.

```
public class Trie {
  static private int R = 264;
  private class Node {
    Value val;
    Node[] next = new Node[R];
    Node(Value v) {val = v;}
  }
  private Node root; ...
  public Value get(String key) {
    node n = get(root, key, 0);
    if (null == n) return null;
    return n.val;
  }
  private Node get(Node n, String key, int index) {
    assert index <= key.length();
```

Question A

Below is a fragment of a TST (ternary search trie) class implementing an ordered search table whose keys are Strings. Complete the **`min()`** method for this class. (*Hint: remember that the* ***`val`*** *of the left-most leaf might be* ***`null`*** *following a* ***`delete()`***.)

```
public class TST {
  private class Node() {
    Char ch;
    Value val;
    Node left, mid, right;
    N; // number of (non-null) values in this subtree
    Node(Char c, Value v) {
      ch = c;
      val = v;
      N = null == v ? 0 : 1;
    }
  }
  Node root; …
  public Value min() {
    Node n = min(root);
    if (null == n) return null;
    return n.val;
  }
  private Node min(Node n) {
```

Question B

Complete the following method to determine if the directed graph, **`g`**, contains a path from **`v`** to **`w`** (**`g`** maps each vertex **`x`** to the set of all vertices **`y`** such that **`<x, y>`** is an edge of the graph.) Extra credit: write a method, **`getPath()`**, which takes the same parameters and returns a path (as a **`List<Vertex>`**) from **`v`** to **`w`** if one exists (otherwise it should return **`null`**).

```
boolean isPath(Vertex v, Vertex w, Map<Vertex, Set<Vertex>> g){
```

Question C

Using the probabilities in the table below, construct the Huffman coding tree for the symbols and fill in the code box for each symbol. Using your codes, encode the message “DEADBEEFCAFE”.

Symbol	Probability	Code
A	15.00%	
B	10.00%	
C	10.00%	
D	20.00%	
E	30.00%	
F	15.00%	

Question D

You are given the three methods that determine the costs of deleting a given input character, inserting a given output character, and changing a given input character into a different output character. Use dynamic programming to complete the distance method below to determine the minimum cost of converting a given input String (**`src`**) into a given output String (**`dst`**).

```
static double delete(char in) {…}
static double insert(char out) {…}
static double change(char in, char out) {…}
static double distance(String src, String dst) {
```

Question E

What does it mean for a problem to be in P?

What does it mean for a problem to be in NP?

What does it mean for a problem to be NP-complete?

What would be the implications of finding an algorithm for discovering a minimal coloring for any undirected graph that ran in ~ c V^3 time (where V is the number of vertices in the graph)?

Question F (Extra Credit)

An ***articulation point*** in an undirected graph is a vertex that, if removed (along with any incident edges), would separate two (or more) otherwise connected vertices. Complete the following method to compute the articulation points of an undirected graph. *(Hint: perform a depth-first search on the graph. On the way in, assign a number to each vertex as you visit it. On the way out, report the smallest numbered vertex reachable from the visited vertex. If that number is not as small as the visiting node then the visited node is an articulation point. The first node visited in a connected component is a special case. How can you tell if it is an articulation point?)*

```
Set<Vertex> articulation(Map<Vertex, Set<Vertex>> g){
```