CS 112 Final
May 8, 2008
(Lightly edited for 2011 Practice)
Name: _____
BU ID: _____

Instructions

- This exam is CLOSED book and notes.
- The exam consists of six questions on 11 pages.
- Please answer all questions on this exam sheet.
- Please read through all questions carefully and be sure that you understand the instructions before working on a problem.
- The exam will be scored out of 100 possible points. The questions are not of equal weight, but the weight is indicative of the difficulty.
- You have until 11:00 AM to complete the exam. Please budget your time wisely!
- Please ask if you have any questions.

GOOD LUCK!

1.) _____ (16 points)
2.) _____ (13 points)
3.) _____ (18 points)
4.) _____ (17 points)
5.) _____ (12 points)
6.) _____ (24 points)

Question 1: Short answer (2 pts each).

1. T/F: In hashing, use of Horner's method can avoid arithmetic overflow.
2. T/F: It would be appropriate to have a Giraffe Class inherit from a Mammal Class.
3. T/F: It would be appropriate to have a Bird Class inherit from a Beak Class.
4. T/F: Queue insertion and stack insertion are both O(1) operations.
5. In big-O notation, how large is the sum $1 + 2 + 3 + 4 + \ldots + N$? _____
6. In big-O notation, how many terms are in the sequence $1, 2, 4, \ldots, \frac{N}{4}, \frac{N}{2}, N$? _____
7. Name a data structure used for fast indexing in large databases. _____
8. Which data structure is used by routers to represent Internet graphs. _____

Question 2 (13 pts): Hashing

(a.) (5 pts) Diagram where the following set of input keys will be placed into a standard 7-element array-based hash table when we use the following hash function and we employ standard *linear probing* when a collision occurs.

```
h(key) = key % 7;
Input keys:  4, 10, 11, 24, 20
```

(b.) (3 pts) What is the load factor when this set of keys is inserted into this hash table?

(c.) (5 pts) Assume that in our hash table implementation, each entry either contains a key or is marked as empty (i.e. cells cannot be marked as deleted). For each of the five keys, specify whether it **cannot** be removed (i.e. have its cell marked as empty) since that would result in searches for other valid keys to fail, or whether it **can** safely be removed. Treat each key removal independently from all other removals.

Question 3 (18 pts): Fancy Data Structures.

(a.) (6 pts) Here is a picture of a 3-level B-Tree from our textbook. What is the shortest sequence of insertions needed to cause this B-Tree to break into 4 levels? Write down the insertions in your sequence explicitly.

Note to '11 students: We didn't cover B-Trees this semester, but I could ask a similar question aobout 2-3 trees, say using the picture on p. 330.

(b.) (5 pts) Here is a picture of a skip list from our textbook. Blacken all the edges that are inspected when an unsuccessful search for 46 is performed.

Note to '11 students: Not covered this semester.

(c.) (7 pts) Here is a picture of a Red-Black tree from our textbook. (Assume the final tree on the Figure on p. 346 was provided). Specify a key whose insertion would trigger a double-rotation, and show the subtree after double-rotation is performed.

Question 4 (17 pts): Graphs.

(a.) (4 pts) Provide the adjacency matrix representation for the undirected graph shown above. (Assume a graph like the first seven nodes of tinyG.txt on p. 428 was provided).

(b.) (4 pts) Specify a possible order in which nodes are visited in a *breadth-first search* traversal of the graph starting from node 2.

(c.) (4 pts) Specify a possible order in which nodes are visited in a *depth-first search* traversal of the graph starting from node 6.

(d.) (5 pts) Now consider the same graph with the addition of edge weights as shown. Highlight those edges which are in the minimum spanning tree of this graph (or draw the spanning tree in a separate picture). (Use weights of $i + j$ for edges connecting nodes i and j).

Question 5 (12 pts): Heap Implementation.

Assume that a max-heap (biggest on top) of integers is implemented using the following class.

```
public class Heap
{
    int  MAX_HEAP;     // maximum # of items in the heap
    int  [] items;     // set of items in the heap
    int  size;         // # of items in the heap
};
```

(a.) (3 pts) Sketch a possible layout of the set of keys $\{72, 16, 42, 33, 81, 10, 23\}$ when stored into a Heap named X with `MAX_HEAP = 10`.

(b.) (9 pts) Write the Insert method for the Heap class.

```
void Insert (int newItem)
{









}
```

Question 6 (24 pts): Set Implementations.

A *set* is a collection of distinct objects, often referred to as set *elements*. In this question, we will consider data structures for representing sets of positive integers. Three important operations on sets are union, set difference, and membership. The union of two sets A and B is the set of elements in either A or B (without duplicates). The set difference $A \setminus B$ is the set of elements in A but *not* in B. A membership test asks whether a given element is in a given set. For example, consider the five-element set $A = \{5, 8, 13, 25, 92\}$ and the four-element set $B = \{3, 5, 8, 13\}$. The union of A and B is $\{3, 5, 8, 13, 25, 92\}$, and $A \setminus B = \{25, 92\}$. Element 3 is a member of B, but not of A.

(a.) (3 pts) One easy way to implement sets of positive integers is via a linked list with elements in sorted order. With this implementation, what is the worst-case running time of membership, union, and set difference on sets of size N?

(b.) (10 pts) Now assume you have the following class declarations for sorted linked list implementation of Sets (only the relevant parts for this question are provided).

```
public class ListNode {
    int      item;
    ListNode next;
}

class List {
      ...
    ListNode head;
}

class Set {
     ...
    List l;
}
```

On the following page, complete the skeleton function declaration to compute (and return) the set difference of two Sets. For simplicity, you may assume that this is *not* a method of the class Set, and do not bother using any of the List routines (just access list items by traversing next pointers).

```
Set compute_difference (Set A, Set B)
{
    Set result = new Set();   // assume the constructor correctly initializes
      // the member variable l to an empty list.

    ListNode Aptr = A.l.head;
    ListNode Bptr = B.l.head;

    // your code here









    return result;
}
```

(c.) (3 pts) Often, sets have values over a finite range, like positive integers between 0 and M, inclusive, for a small to moderate value of M. When this is the case, an array-based implementation is a reasonable choice. For example, a *bit array* implementation of a set might use an array A, storing the value 1 in $A[i]$ if i is an element of the set, and 0 otherwise. Using this definition, sketch the bit array used to represent the set $\{0, 2, 5, 7\}$ when M is 8.

(d.) (3 pts) What are the running times of the three set operations using this bit array implementation (as a function of M)?

(e.) (5 pts) Now assume M is much larger than N. Sketch a data structure that achieves the same expected running times as the bit array implementation of a set of N elements, but does so using only $O(N)$ space, not $O(M)$ space. Make sure your data structure can still handle the three set operations.