IDENTIFICATION

University of Waterloo
Final Preparation Examination
Spring 2008

Course Number:	CS 240
Course Title:	Data Structures and Data Management
Sections:	001, 002
Instructors:	Amir H. Chinaei and Eric Y. Chen

Date of Exam:	—
Time Period:	—
Duration:	2.5 hours
Exam Type:	Closed Book

Prob	Mark	Max	Init.
1		6	
2		8	
3		9	
4		6	
5		5	
6		5	
7		8	
8		6	
9		9	
10		2	
Total		69	

Instructions

- Calculators are not allowed.
- Do not open the examination until the start of the exam is announced.
- Do not separate the pages of the examination.
- If you need additional space, use the back of the previous page, and indicate clearly that you have done so. If you make a mistake be sure to cross it out clearly so we are sure which answer to mark.
- In the interests of fairness and to treat all students equally, we cannot answer any questions about the examination. If you believe there to be an error in the examination paper, you may bring it to our attention. If we determine that there is an error, we will inform the entire class.
- Please sign your initials in the space at the bottom-right corner of each page.

1. [/ 6] True/False

Write either True or False in the box, and justify your answer briefly.

(a) A radix sort algorithm always take $O(n)$ time to sort n keys.

[ ]

(b) Using the LZW algorithm, we compress two pieces of text (ASCII coded) into two strings of bytes (8-bits per byte). If two substrings, one from each compressed text are the same, then the two substrings in the original text corresponding to these substrings are the same.

[ ]

(c) Given a Huffman tree and the correct compressed text, we consider two sub strings of bits s_1 and s_2, in the compressed text. If s_1 and s_2 are the same, then the original text represented by s_1 and s_2 are also the same.

[ ]

2. / 8 Running-Time Analysis

Analyze the running time (in terms of n) for each of the following code fragments, using Θ-notation, and justify your answer briefly.

(a) / 4

```
int one_count(n)
{
    // base case: 1
    if n == 0
        return 0
    // base case: 2
    if n == 1
        return 1
    m ← ⌈log n⌉
    n_l ← n >> ⌈m/2⌉ << ⌈m/2⌉
    n_r ← n − n_l
    n_l ← n_l >> ⌈m/2⌉
    return one_count(n_l) + one_count(n_r)
}
```

Assume n is a non-negative integer, and it can fit in one machine word. Shifting operations can be done in $O(1)$ time. Bit 0 will be shifted in from both the left and right ends.

Hint: consider the length of the bit representation of n

(b) / **4**

```
int A(n)
{
    sum ← 0
    for i from 1 to n
        for j from 1 to 3^i
            sum++
    return sum
}
```

3. / **9** Hash Tables Operations

Given input 4070, 1020, 6070, 4090, 4372, 9690, 1983 and a hash function h(x) = x mod 10, show the resulting hash table (of size 10) if we resolve collisions with

(a) / **3** Chaining

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

(b) / **3** Open addressing with linear probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

(c) / **3** Open addressing with double hashing where the second hash function is $h'(x) = (x \bmod 3) + 1$

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

4. [/ 6] # Huffman Trees

Using the Huffman compression algorithm explained in class, build a Huffman tree and encode the following text: NW2N1W2N12NW2W1S5W2SE2

(a) [/ 4] Build the Hufffman tree, and show the content of the priority queue for intermediate steps.

(b) [/ 2] Compress the given text with the Huffman tree built in the first step.

5. [/ 5] LZW Compression

Using ASCII code (0 to 127) as the original dictionary, compress/decompress the following text into string of bytes (8 bits per byte)

Decompress the following text: "97, 110, 95, 128, 116, 95, 99, 128, 130, 110, 116, 105, 95, 101, 108, 101, 112, 108, 131"

6. [/ 5] Burrows-Wheeler Transformation

Perform Burrows-Wheeler Transformation on "ACBCCACB" and show the results. Show strings in the cyclic order and the sorted order, as the intermediate step.

7. / 8 Short-Answer Questions

Among sorted array, hash-table, B-trees, and suffix-tries, which data structures should we choose in each of the following cases. **Briefly** justify your answer.

(a) / 2 Given a piece of English text, we need to build an indexing structure, so that for a query key, we can know whether this key is contained as a **whole** word in the text. As an additional request, we are more interested in the average query time, and construction time.

(b) / 2 We are making a website for selling used cars. Our customers either post the advertisement for selling their cars or send a request to find all listed cars in a specified price range. After a car is sold, that car and its price should also be removed from our list.

(c) / 2 We are selling a database which is stored on a fast-speed read-only memory chip. We should be able to search for a record using a key. The price of this type of memory chip is proportional to the amount of memory used, and we would like to minimize our cost.

(d) / 2 We need a data structure to store a DNA sequence, so we can quickly find whether a given query string is contained within the sequence.

8. [/ 6] Extendible Hash Tables

(a) [/ 2] Extendible hashing "guarantees" that one can discover whether a key value is present or not in a small number of disk accesses. Let n be the number if keys in the structure and m be the number of key/pointer pairs that will fit on a page of disk. How many disk accesses does extendible hashing require to determine whether a key is in the table. (Assume the dictionary can be stored in memory)

(b) [/ 2] This behavior is better than a B-tree, when n is very large. Give a query that can be answered by a B-tree but not extendible hashing?

(c) [/ 2] What possible (though very unlikely) difficulty could cause the index used in extendible hashing to become very large?

9. / **9** Suffix Trees and Tries

In the suffix trees in this question, we assume:

- internal nodes store a substring for each branch;
- each leaf node stores an entire suffix.

(a) / **3** Using only the suffix tree T, denoted as S_T, output all suffixes of T in lexicographically increasing order.

(b) / **3** Suffix trees can also help us to match patterns with typos. Given a suffix tree S_T and a pattern P. Determine whether P can be found in the text T with at most 2 mismatched characters.

(c) / **3** Given two suffix trees S_{T1} and S_{T2}, report the longest common string between $T1$ and $T2$.

10. [/ 2] Algorithm Improvement

Consider the Boyer-Moore algorithm (from lecture 19). When we build the last-occurrence function. The loop is from 1 to m. Can that loop go from 1 to $m - 1$? Why?