University of Waterloo
David R. Cheriton School of Computer Science
Final CS 240 Fall 2007

Course:	CS 240
Title:	Data Structures and Data Management
Section(s):	1 & 2
Instructors:	Therese Biedl and Matthew Nichols

Student Name:

Student ID Number:

Date of Exam:	December 13, 2007
Time Period	Start time: 9:00 am End time: 11:30 am
Duration of Exam:	2 hour 30 minutes
Number of Exam Pages:	15 (including cover sheet)
Exam Type:	Special Material
Additional Materials Allowed:	One 8.5" x 11" double-sided help sheet **No Calculators. No Other Additional Materials**

Instructions

- Please sign this exam below these instructions.
- Please ensure that the information recorded on the exam label is correct.
- Cheating is an academic offense. Your signature on this page indicates that you understand and agree to the University's policies regarding cheating on exams.
- If you are unclear about a definition, please clearly state your assumption and proceed to answer the question. Your assumption must not trivialize the question.

Your Signature:

Problem	Your Mark	Out of	Marker
1		26	
2		8	
3		7	
4		5	
5		6	
6		10	
7		8	
8		6	
9		10	
10		6	
11		8	
Bonus			
Total		100	

Problem 1 (26 marks)
For each question below, indicate in the box provided whether the statement is **True** or **False**. No explanation is required. Please do not simply write T or F as these as easily misread. You will receive 2 marks for a correct answer, 0 for a blank answer, and -0.5 for a wrong answer. Your minimum total mark is 0.

a. ☐ If a problem has a lower bound of $\Omega(n^2)$, then no algorithm exists that solves it in $o(n^3)$.

b. ☐ LSD radix sort uses $O(1)$ auxiliary space to sort n base-10 numbers that have 32 digits each.

c. ☐ Any AVL-tree is a B-tree of order 4.

d. ☐ Any heap is balanced like an AVL-tree, i.e., for every node the height of the left and right subtree differs by at most 1.

e. ☐ Any *kd*-tree is balanced like an AVL-tree.

f. ☐ Implementing a dictionary with hashing may lead to $\Theta(n)$ runtime for insertion and searching.

g. ☐ The expected amount of space for a skip list is $O(n)$, but in the worst case we may use $\omega(n)$ space.

h. ☐ Interpolation search always performs at least as well as binary search.

i. ☐ A quad tree that stores n 2-dimensional points has at most $2n$ nodes.

j. ☐ A range tree that stores n 2-dimensional points has at most $2n$ nodes.

k. ☐ A compressed trie that stores n words has at most $2n$ nodes.

l. ☐ If we applied the Boyer-Moore algorithm without the looking glass heuristic, then we would never skip characters of the text (except if we found the pattern, or after trying all shifts.)

m. ☐ The Knuth-Morris-Pratt (KMP) algorithm is a good choice if we want to search for the same pattern repeatedly in many different texts.

Problem 2 (8 marks)
Assume we execute heapsort on an array that is sorted in decreasing order (largest value first).

a. During `heapify` in this case, how often will we exchange elements in the array, i.e., how often will we execute line (*) in the pseudo-code below? Briefly justify your answer.

b. Show that during the rest of the heapsort in this case, we will execute line (*) $\Theta(n \log n)$ times.

Hint: Running the algorithm on a small example, say $n = 10$, should be helpful. However, your answer must be for arbitrary size n.

For your reference, the following is the code of the heapsort-algorithm as we had it in class.

```
void heapsort(int[] A, int n) {
    // post: A[1...n] is sorted
    heapify(A, n);
    for (int i = n down to 1) {
        swap A[i] and A[1]
        sink(A, 1, i-1);
    }
}

void heapify(int[] A, int n) {
    // post: A[1..n] is a max-heap
    for (int i = floor(n/2) down to 1) {
        sink(A, i, n);
    }
}
```

```
void sink(int[] A, int k, int size) {
    // considers only elements in A[1..size],
    // sinks A[k] down
    while (2*k < size) {
        int largerChild = 2*k;
        if (2*k+1 < size
            and A[2*k+1] > A[2*k]) {
            largerChild = 2*k+1;
        }
        if (A[largerChild] > A[k]) {
            swap A[k] and A[largerChild]      (*)
            k = largerChild;
        }
    }
}
```

Problem 3 (7 marks)
Consider a dictionary that stores integers implemented as a unordered array $A[1..n]$.

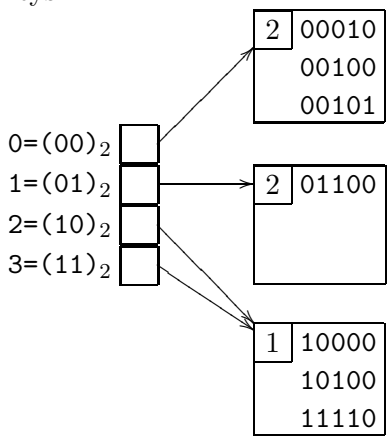
We have seen in class the Move-to-Front (MTF) heuristic, which should make successful searches faster than if we simply made no change to the order of the array at all.

There are many other possible heuristics; one of the them is Move-To-Middle (MTM). This is similar to the MTF heuristic, except that once an element is found at $A[i]$, it is moved only halfway to the front, i.e., it is moved to $A[\lfloor i/2 \rfloor]$. All elements between $A[\lfloor i/2 \rfloor]$ and $A[i]$ are then moved over by one position.

Let $A[1..7]$ be the dictionary, with $A[i] = i$ for $1 \leq i \leq 7$ initially. For each of the following three heuristics: DoNothing (where the order is not changed at all), MTF and MTM, show the array after searching for the following keys in the given order: $4; 2; 7; 7$. Some entries have been filled in already. For each heuristic, also give the total number of (key) comparisons required to execute all of these searches, i.e., compute the sum of the indices where these elements were found.

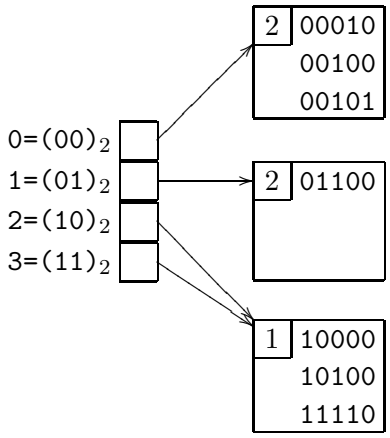
Search	DoNothing	MTF	MTM
Start	1 2 3 4 5 6 7	1 2 3 4 5 6 7	1 2 3 4 5 6 7
4	i= 4 1 2 3 4 5 6 7	i= 4	i= 4 1 4 2 3 5 6 7
2	i= 2 1 2 3 4 5 6 7	i=	i= 3
7	i= 7	i=	i=
7	i=	i=	i=
Total comparisons			

Problem 4 (5 marks)
Let H be the following extendible hash table with global depth 2 for which leaf pages can hold at most 3 keys.

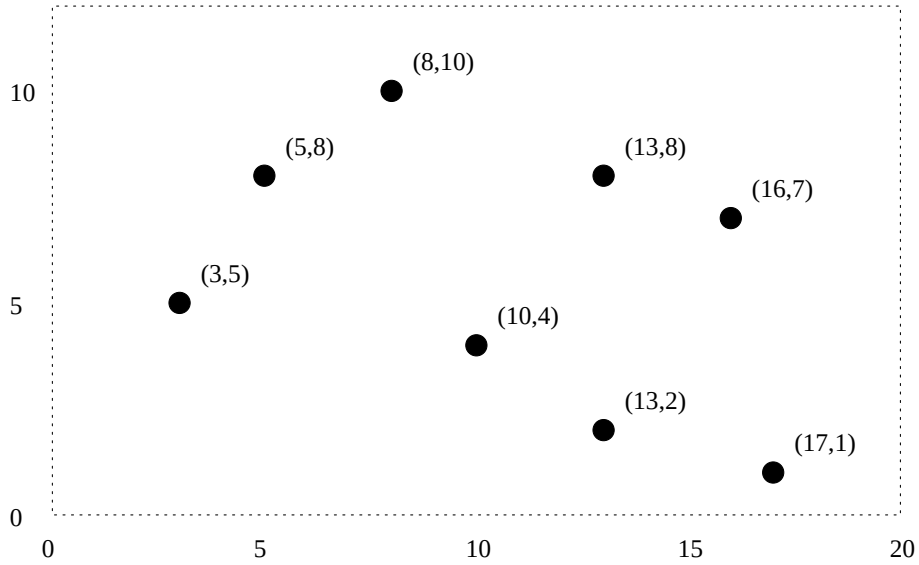


a. Insert the key 01010 into H and draw the resulting extendible hash table. Make sure you indicate the local depth for each leaf page. The order of items within a leaf page is not important.

b. Insert the key 00001 into H (not into your answer to part a!) and draw the resulting extendible hash table. Make sure you indicate the local depth for each leaf page. The order of items within a leaf page is not important. Here is H again:



Problem 5 (6 marks)
Consider the following 8 points:

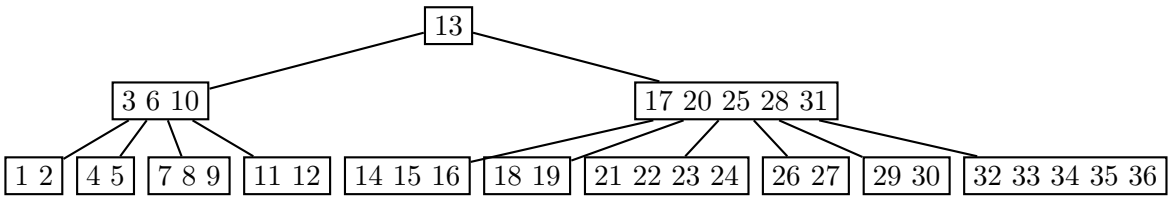


Show the kd-tree that corresponds to this set of points. Be sure to split by x-coordinate at the root level, and to add points that are on a splitting line to the left/bottom part.

Problem 6 (10 marks)
Describe (in English, not pseudo-code) an algorithm to do a range-query in a B-tree T of order M that stores n integer keys. Thus, given two integers x', x'', the algorithm must output all keys x that satisfy $x' \leq x \leq x''$. Each node stores its keys and references to children in sorted arrays of size M. The whole tree fits into internal memory.

Your algorithm should have worst-case run time $O(\log n) + O(\#$ keys in output), but partial credit may be given for slower algorithms. Briefly analyze the running time of your algorithm.

Hint: We recommend that you test your algorithm on the B-tree of order 6 below. A range-query for $[5, 15]$ should output the keys $5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$ (but they need not be in this order).



Problem 7 (8 marks)
Build a suffix tree for the text "`ittibitti$`". If you build any intermediate trees, then clearly indicate which tree is your final tree. Draw the suffix tree where references and nodes are labelled with strings, not the one that uses indices.

Problem 8 (6 marks)
Show how to search for pattern $P =$ `dacbdc` in the text $T =$ `eacbdecbdcdabbccacbdc` using the Boyer-Moore algorithm. First fill in the "last"-values for each character. Then indicate in the table which characters of P were compared with which characters of T. (Any method of indicating clearly which comparisons happened is acceptable; we recommend you place each character of P in the column of the compared-to character of T and circle it if an actual comparison happened.) You may not need all space in the table.

	a	b	c	d	...
last					

e	a	c	b	d	e	c	b	d	c	d	a	b	b	c	c	a	c	b	d	c

Please indicate here the total number of comparisons that happened: [ ]

Problem 9 (10 marks)

a. Encode the text "`TIN␣TIPTIP`" using Lempel-Ziv encoding, by filling in the table provided. (Showing which new items are added to the dictionary is optional.) The first few steps have been completed already. You may not require all space in the tables.

Dictionary

s	#(s)
␣	32
...	...
I	73
...	...
N	78
...	...
P	80
...	...
S	83
T	84
...	...
TI	128
IN	129

s	*c*	*output*	*sc*	#(*sc*)
T	I	84	TI	128
I	N	73	IN	129
N	␣	78	N␣	130

b. Decode the sequence of numbers [65 65 66 129 67 132 131] which has been obtained from Lempel-Ziv encoding, by filling in the tables provided. (Showing which new items are added to the dictionary is optional.) The first few steps have been completed already. You may not require all space in the tables.

Dictionary

s	#(s)
A	65
B	66
C	67
D	68
...	...
AA	128
AB	129

n	s_{new}	c	$s_{old} + c$	$\#assigned$
65	A	-	-	-
65	A	A	AA	128
66	B	B	AB	129

Problem 10 (6 marks)
Decode the following text that has been encoded using the Burrows-Wheeler transform. The answer will be on the last line of the table.

											L
											L
											D
											O
											O
											A
											A
											D
											B
											B
											I
											A

You don't need to fill-in the whole table, but show enough work to make it clear how you obtained the answer.

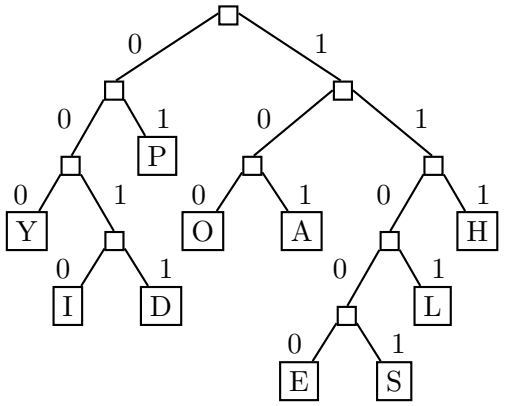
Problem 11 (8 marks)

a. Build the Huffman-tree for the following set of frequencies. Always put the subtree with the smaller frequency to the left.

char c	T	H	E	N	D	!
$f(c)$	6	5	9	4	2	1

Also, please indicate here the total cost of your tree, i.e., the total length of an encoding of these letters with these frequencies. ☐

b. Decode the following bitstring, which has been encoded using the prefix-code given below.
Trie of prefix-code:



Bitstring:
1111010101000111100110100100011101000110 01

Answer:

Problem 12 (Bonus)
You want to do pattern matching for pattern P in string T (both have ASCII characters.) However, T has been compressed. Can we compress P and use this to do pattern matching?

a. T has been encoded using Huffman-encoding, resulting in the bitstring T'. You now encode P with the same codes, resulting in bitstring P'.

   - If P' occurs in T', does this imply that P occurs in T? Why or why not?

   - If P' doesn't occur in T', does this imply that P doesn't occur in T? Why or why not?

b. T has been encoded using Lempel-Ziv-encoding, resulting in the sequence of numbers T'. You now encode P with the same dictionary (you do not add to it during this encoding), resulting in a sequence of numbers P'.

   - If P' occurs in T', does this imply that P occurs in T? Why or why not?

   - If P' doesn't occur in T', does this imply that P doesn't occur in T? Why or why not?