Sample Midterm Exam

Instructions: This is a closed book, closed notes exam. Do all **4** problems. Communicate your ideas ***clearly*** and ***succinctly***. Show all work.

NAME:

On problem	you got	out of
1		45
2		15
3		15
4		25
Σ		100

1 Short answers [45 points]

Part A. You have two programs, P and Q which correctly solve some problem Π. Your analysis shows that P and Q have worst-case running times of $\Theta(n)$ and $\Theta(n \lg n)$, respectively. You extensively test P and Q, on a variety of inputs of all sizes your computer can handle, and you find that P consistently runs *slower* than Q. Give a good reason why this might be occurring.

Part B. *Complete the following definition.* We say that C is a **program checker** for the program P, where P is supposed to compute the function π, if

1.

2.

Part C. Consider n distinct keys $k_1, k_2, \cdots, k_n$ taken from a totally ordered universe. Let $T(n)$ be the number of different (valid) binary search trees for $k_1, \cdots, k_n$. Write a recurrence relation for $T(n)$. Do *not* solve your recurrence relation. *Hint: any k_i could be the root. How many trees would you get if k_i is the root? Sum over these.*

$T(n) \ =$

Part D.1 Let $S(n)$ be the worst-case number of comparisons for MERGESORT to sort n numbers. (Assume the version of MERGESORT covered in class.) Write a recurrence for $S(n)$. Be exact: specify the base case, include any "floors" and "ceilings", etc.

$$S(n) \quad =$$

Part D.2 Using your formula, compute $S(n)$ for $n = 1, 2, \ldots, 6$:

n	$S(n)$
1	
2	
3	
4	
5	
6	

Part E.1 Recall the information-theoretical lower bound we did for comparison-based sorting: any comparison-based sorting algorithm uses at least $\boxed{}$ comparisons to sort n numbers. (Give the tightest bound you know.)

Part E.2. Comparing the bound you just wrote with your answers in D.2, what is the least value, N_0, for which the information theoretic bound allows for the possibility of a comparison-based algorithm which sorts N_0 elements in fewer than $S(N_0)$ comparisons?

Part E.3. Is there really a comparison-based sorting algorithm that sorts N_0 elements in fewer than $S(N_0)$ comparisons? Just answer "yes" or "no," and say something which makes it obvious that this is not a guess.

Part F. Solve the following recurrence relations. Give your answer in $\Theta(\cdot)$ notation. Assume $T(n) \in \Theta(1)$ for sufficiently small n.

(F.1) $T(n) = 3T(n/4) + n$

(F.2) $T(n) = 3T(n - 4)$

Part G. Alice has defined some bizarre family of hash functions, $\mathcal{H} = \{h_a\}$, where $a \in \{1, \ldots, 100\}$. Each $h_a : \{1, \ldots, 10\} \to \{1, \ldots, 10\}$. Complete the following program which computes and prints out the smallest value of ϵ such that $\mathcal{H}$ is ϵ-universal-2. That is, your program prints the maximum, over all distinct x, y in the domain of the hash functions, of the probability, given a random hash function h_a in the family, that $h_a(x) = h_a(y)$.)

```
#include <iostream.h>

int h(int a, int x)
{return 1+ (5*a*x*x + 3+a*x - a^x)%10;}      // ignore the specific function

int main()
{















}
```

2 Algorithm Design [15 points]

Let $L = (x_1, \ldots, x_n)$ be a list of n real numbers. The numbers may appear in any order. A number x is said to appear *often* in L if at least $1/3$ of the numbers in the list are equal to x. Describe a deterministic, comparison-based algorithm which identifies all numbers which appear often in L. Your algorithm may use only $\Theta(n)$ comparisons. (Your solution may call, as a subroutine, any deterministic linear-time algorithm we have presented in class.)

3 Divide-and-Conquer [15 points]

You are given a pot of $n \geq 3$ coins, $n - 1$ of which are authentic gold coins, and 1 of which is a counterfeit coin made of some gold-colored alloy. The counterfeit coin cannot be distinguished by any means other than its weight: it is either heavier or lighter than an authentic coin (you don't know which). Your only means of measuring weights is to use a balance scale. This scale has two bowls, a left-bowl and a right-bowl. If you place coins into the left and right bowls, the scale will do one of three things: the left-bowl will rise (indicating that its contents are lighter than the right-bowl's); the right-bowl will rise (indicating that its contents are lighter than the left-bowl's); or the bowls will balance, indicating that they contain the same weight.

Part A. Describe a recursive algorithm which finds the counterfeit coin using $O(\lg n)$ weighings. (*Hint: as your first step, divide the coins roughly into thirds.*)

Part B. Write a recurrence relation whose solution represents the maximum number of weighings your algorithm needs to find the counterfeit coin.

$$W(n) \quad =$$

Part C. Solve the recurrence relation. If you don't get a logarithmic function in n, you've done something wrong.

4 Dynamic Programming [25 points]

You are at the airport in Germany, about to fly back to the U.S., and in your pocket you find C Deutsch marks (DM). There is no foreign exchange facility, and so you want to spend as much of your money as possible at the duty-free shop. The duty-free shop has an infinite number of packages of each of N different types of items. You don't care about which of these you buy. Formally, the Airport Shopping Problem is the following (where "number" means a non-negative integer):

Instance: A number N, numbers $c_1, \ldots, c_N$, and a number C.

Find: Numbers $x_1, \ldots, x_n$ such that $\sum c_i x_i$ is maximized but $\sum c_i x_i \leq C$.

(Of course c_i is the cost of item i, C is the amount of money you have, and x_i is the number of packages of item i you'll buy.) The number $\sum c_i x_i$. is the *cost* of your solution. You want to find a maximal-cost solution whose cost is $\leq C$. For example, if items have costs $(c_1, c_2, c_3) = (8, 15, 6)$ and you have $C = 19$ DM, then the best solution is $(x_1, x_2, x_3) = (0, 0, 3)$: you take three of item 3 and go home with your purchases plus 1 DM, since the cost of your solution is 18.

Part A. Let $A[n, c]$ be the cost of the optimal solution if you have c DM and items have values $c_1, \ldots, c_n$. Write a recurrence relation for $A[n, c]$. Remember to specify the base case.

Part B. Write a fragment of code which calculates the $A[N, C]$-value in a reasonable amount of time. ("Reasonable" means a polynomial in N and C.)

Part C. What is the running time of your algorithm?