COMPSCI330 Design and Analysis of Algorithms
Midterm Exam

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Timing** Exam starts at 3:05 pm and ends at 4:20 pm.

Name: _____ Duke ID: _____

For grading use only:

Problem	1a	1b	2	3	4	Total
Score						

Problem 1 (Recursions)**.** Please solve the following recursions (write the answer in asymptotic notations $T(n) = \Theta(f(n))$).

If you decide to use the recursion tree method, please draw the tree this time because it's easy to do on a piece of paper. You need to bound the amount of work in each layer, and take the sum over all layers. You do not need to write the induction proof if you are using the recursion tree method.

(a) (10 points)

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + 2T(\frac{n}{8}) + n.$$

(Base case: $T(1) = 0$)

(b) (15 points)

$$T(n) = 2\sqrt{n}T(\sqrt{n}) + n.$$

(Base case: $T(2) = 0$)

Problem 2 (Gas Stations)**.** You are driving to a faraway city. There is only one route (a direct highway connecting two cities) that you can take, and the total distance is m miles. Your car can only hold enough gas for 300 miles, so you will need to stop for gas. There are n gas stations along the highway between the two cities. The i-th gas station has distance x_i to the origin (the x_i's may not be sorted).

(a) (10 points) Suppose you want to minimize the number of stops you need to make. Design a greedy algorithm that finds out which gas stations you should stop at. (You always have a full tank at the origin. Every time you go to a gas station, you will fill up the tank and will be able to go for another 300 miles. The distance between two consecutive gas stations is at most 300). Make your algorithm as fast as possible, and analyze its running time.

(b) (10 points) Prove the correctness of the algorithm you designed in (a).

(c) (5 points) Suppose different gas stations are at different distances to the highway, so it takes a detour of t_i minutes if you use gas station i.

Instead of minimizing the number of stops, we now want to minimize the total detour. The detour is defined to be the sum of t_i's for the gas stations you visited. You can visit as many gas stations as you need as long as the total detour is minimized.

Give an example where the algorithm you gave in (a) is no longer optimal. (Your example should contain at most 4 gas stations.)

(This page is left empty intentionally.)

Problem 3 (Product of Sums)**.** After solving the Sum of Products problem in homework, you are going to solve a more interesting problem - product of sums.

Similar as before, you are given a sequence of real numbers $a[1..n]$. You can now add '+' and '$\times$' signs between these numbers, and your goal is to generate an expression that has the largest value.

What is different from the original problem is that this time we want you to compute the sums first, and then the products. That is, if there are 4 numbers and you added 3 signs $+$, $\times$, $+$, the final result should be $(a[1] + a[2]) \times (a[3] + a[4])$. You should think of this as there is always a bracket for the sums.

As an example, if $a = [-2, 0.3, -10, 2]$, then the optimal expression is $(-2) \times (0.3 + (-10)) \times 2 = 38.8$. This is larger than any other expression (note here by larger we mean the usual meaning of larger, i.e. a positive number is always larger than a negative number, we are not comparing absolute values). You must add either a '+' or a '$\times$' between two consecutive numbers, and sum is always evaluated before products.

Design an algorithm that runs in time $O(n^2)$ and outputs the largest possible value. For this problem you can assume additions, multiplications and comparisons of two real numbers can be done in $O(1)$ time.

(a) (15 points) Define the state (sub-problems), write the transition function, and specify the base cases.

(b) (10 points) Design an algorithm for the Product of Sum problem. (No need to prove correctness for this problem. If your algorithm is slower than $O(n^2)$ you will not receive full credit.)

(This page is left empty intentionally.)

Problem 4 (Balls and Bins)**.** In this problem we consider throwing n balls to m bins. Each ball will land randomly into one of the bins with equal probability. Different balls are completely independent, and they can land into the same bin. After throwing all n balls, we would like to check how many balls are in each bin.

(a) (10 points) Prove that, if $m = n^2$, then the probability that all balls are in different bins is at least $1/2$.

(Hint: Let $X_{i,j}$ be 1 if balls i, j are in the same bin. Try to compute the expected number of "collisions" (two balls in the same bin) using $X_{i,j}$'s.)

(b) (10 points) Suppose $m = 2n$, let n_i (for $i = 1, 2, ..., m$) be the number of balls in i-th bin. Let s be the sum of squares of n_i:

$$s = \sum_{i=1}^{m} n_i^2.$$

Prove that, the probability that s is at most $2n$ is at least $1/2$. ($\Pr[s \leq 2n] \geq 1/2$.)
(Hint: Again, let $X_{i,j}$ be 1 if balls i, j are in the same bin. Try to relate s to $X_{i,j}$'s.)

(c) (5 points) The reason we had questions (a) and (b) is to discuss the *perfect hashing algorithm*. The goal of this new algorithm is to design a data-structure that supports the "Find" operation in $O(1)$ time even in the worst-case. (Note that usual hash tables will only guarantee $O(1)$ time *in expectation*).

The input of the algorithm is n pairs of numbers $(x_0, y_0), (x_2, y_2), ..., (x_{n-1}, y_{n-1})$ where $x_i \neq x_j$ for all pairs $i \neq j$. The "Find(x)" operation should return y_i if $x = x_i$, and NULL if x is not equal to any of the x_i's.

The algorithm will store these pairs in a two-layer table structure. See Figure 1. Each pair (x_i, y_i) is put into location $g(x_i)$ in table 1, where g is a random hash function. Of course, there might be collisions in the first layer, where n_t x_i's are mapped to the same cell $g(x_i) = t$. These pairs are then stored in a second-level hash table (instead of a linked list) of size n_t^2. Each second-level hash table will have a different hash function $f_t(\cdot)$, and the algorithm will make sure there are no collisions in the second layer. See the pseudo-code below.

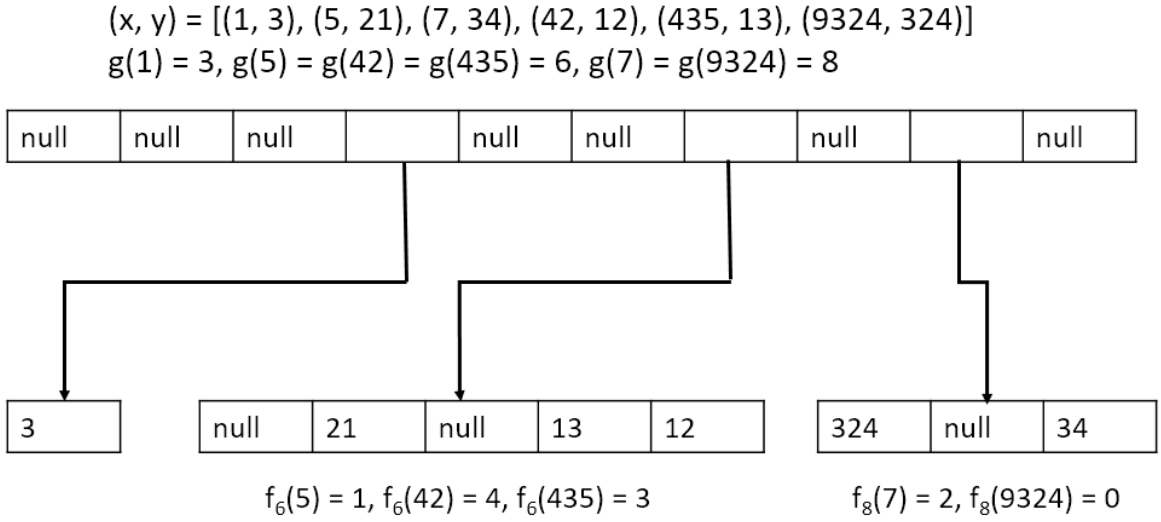

Figure 1: A two-layer hash table (size of the table is just for illustration and do not agree with what's specified in algorithm)

Algorithm 1 ConstructSubTable($x[0..n-1], y[0..n-1]$)

1: Pick $z = n^2$, allocate table of size z (with index $0..z-1$).
2: **repeat**
3: Pick a random hash function f that maps each $x[i]$ to an independent random number in $\{0, 1, 2, ..., z-1\}$.
4: **until** all $x[i]$'s are mapped to different entries.
5: Store the Hash function, and put the value of $y[i]$ to entry $f(x[i])$.

Prove that the ConstructHashTable algorithm is expected to run in $O(n)$ time. (Here we assume randomly picking a Hash function, and evaluating a hash function at a given position both take $O(1)$ time.)

Algorithm 2 ConstructHashTable($x[0..n-1], y[0..n-1]$)

Pick $m = 2n$, allocate a first-level table A of size m (with index $0..m-1$).
repeat
  Pick a random hash function g that maps each x_i to an independent random number in $\{0, 1, 2, ..., m-1\}$.
  Let n_i be the number of elements such that $f(x_j) = i$.
  Let $s = \sum_{i=0}^{m-1} n_i^2$.
until $s \leq 2n$.
Store the Hash function.
for $i = 0$ **to** $m - 1$ **do**
  Let P_i be the set of all pairs (x_j, y_j) such that $f(x_j) = i$
  if P_i is empty **then**
    ConstructSubTable(P_i)
    Link $A[i]$ to the table constructed.
  else
    Mark $A[i] = NULL$
  end if
end for