CIS 121
Homework Assignment 4

Given: February 04, 2016
Due: February 11, 2016

Note: The homework is due electronically on the course website on Thursday, February 11 by 10:30 am, the beginning of the class. For late submissions, please refer to the Late Submission Policy on the course webpage: http://www.seas.upenn.edu/~cis121
You must use the hw121.cls template provided on the course website.
Please write concise and clear solutions; you will get only a partial credit for correct solutions that are either unnecessarily long or not clear.
You are allowed to discuss ideas for solving homework problems in groups of up to 3 people but you must write your solutions independently. Also, you must write on your homework the names of the people with whom you discussed.
Finally, you are not allowed to use any material outside of the class notes and the textbook. Any violation of this policy may seriously affect your grade in the class.

Note the following.
1. lg n means log_2 n.
2. You may find Steve Seiden’s Theoretical Computer Science Cheat Sheet (posted on the class page under “Resources”) useful.

[10] 1. Solve the following recurrences using the method of expansion (iteration), giving your answer in Θ notation. For all recurrences, assume that T(n) = 1, for all n ≤ 2 and n is an exact power of 2.
   a. T(n) = T(n/2) + n
   b. T(n) = T(n − 1) + lg n

[10] 2. Solve any two of the following recurrences using the method of expansion (iteration), giving your answer in Θ notation. If you solve more, we will only grade the first two. For all recurrences, assume that T(n) = 1 for n ≤ 10 and n is an exact power of 2.
   a. T(n) = T(n − 2) + n^3
   b. T(n) = 2T(n/2) + log n
   c. T(n) = 7T(n − 2)
   d. T(n) = √nT(√n) + n
3. On most computers, the operations of subtraction, testing the parity (odd or even) of an integer and halving can be performed more quickly than computing remainders. This problem investigates the following algorithm which avoids the remainder computations used in Euclid’s algorithm.

a. Prove that if \( a \) and \( b \) are both even, then \( \gcd(a, b) = 2 \gcd(a/2, b/2) \).

b. Prove that if \( a \) is odd and \( b \) is even, then \( \gcd(a, b) = \gcd(a, b/2) \).

c. Prove that if \( a \) and \( b \) are both odd with \( a \neq b \), then \( \gcd(a, b) = \gcd(|a - b|/2, b) \).

d. Give an efficient divide-and-conquer gcd algorithm for two \( n \)-bit input integers \( a \) and \( b \), where \( a \geq b \).

e. Analyze the running time of your algorithm and show that it runs in \( O(n^2) \) time (assume that subtracting two \( n \)-bit integers takes \( O(n) \) steps, testing parity and halving can be done in unit time).

4. In the well-known coupon collector’s problem, the objective is to find the expected number of cereal boxes that need to be purchased to collect \( m \) distinct coupons, where a cereal box contains exactly one coupon and each coupon is equally likely to be in a cereal box. In this question, we consider the following variation of the coupon collector’s problem: instead of buying one cereal box at a time, one can purchase a pack of \( n \) cereal boxes at a discounted price. As before, each cereal box in the pack will contain exactly one coupon, thus one gets \( n \) coupons from a pack of \( n \) cereal boxes. To encourage people to purchase packs of cereal boxes over individual boxes, the cereal maker has announced a prize for anyone who purchases a pack such that more than \( n/2 \) of the cereal boxes in the pack contain the same coupon. In this variation, our objective is to determine if the purchaser of a pack of cereal boxes has won a prize. Assume that the only feasible operation that one can perform with the coupons is to pick two of the coupons, say coupon \( i \) and coupon \( j \), and determine in constant time whether the coupons \( i \) and \( j \) are the same.

a. Given a pack of \( n \) cereal boxes, give a \( O(n \log n) \) divide-and-conquer algorithm to determine if the purchaser of the pack of cereal boxes has won a prize or not.

b. Design a linear time algorithm that is based on the following approach:

- Pair up the \( n \) coupons arbitrarily, to get \( n/2 \) pairs (each coupon belongs to at most one pair).
- Consider each pair of coupons: if the two coupons are different, discard both of them; if they are the same, keep just one of them.

To prove the correctness of your algorithm, it may help to prove the following: after performing the above procedure, there are at most \( n/2 \) coupons left and if there is a coupon \( c \) that appears in more than \( n/2 \) cereal boxes of the pack, then coupon \( c \) appears in more than half of the remaining coupons after performing the above procedure.
5. After last week’s triumph planning the bus logistics of The Pigeonhole Principle Players of Philadelphia (PPPP), you have been promoted to manage the whole logistics operation. The band is currently vacationing in Florida to celebrate the success of their world tour, but has received multiple requests to play some smaller shows. You have a list of cities that have requested a show, and their distances from the PPPP’s Floridian hideaway. In order to make everyone happy, you decide that you will simply pick a city somewhere in the middle: the city with the median distance to Florida.

However, you’re not sure if all the cities have enough capacity for the enthusiastic fans, so you will need some backup cities. Your task is to design an algorithm which takes in the unsorted list of cities \( C \), their distances to PPPP’s Floridian hideaway, and \( b \), the number of backup locations required, and returns \( b \) cities with distance to Florida closest to the median city’s distance to Florida. For example, if \( C \) contains five cities \( c_1, c_2, c_3, c_4, c_5 \), with distances to Florida given by \( (90, 45, 28, 38, 67) \) and \( b = 2 \), then your algorithm should return \( c_3 \) and \( c_4 \). Your algorithm must run in linear time. Prove the correctness of your algorithm and justify its running time.

6. Indians are known for transporting food items – they carry food items wherever they go and bring food items back from wherever they travel to. In particular, the people from the state of Gujarat (called Gujaratis) like to bring khakhras\(^1\) back whenever they travel to India. Being a Gujarati and a lover of khakhras, I bring back khakhras whenever I travel to India. The problem though is that khakhras being fragile, they break easily during travel, if they are not properly wrapped. Given that I travel to India often and bring back khakhras each time, during my most recent trip to India, I decided to figure out exactly how much wrapping would each packet of khakhras require so that they don’t break during travel and also so that I don’t use more packaging than necessary (due to baggage weight restrictions). So after wrapping them with some bubble wrap, I decided to find out the maximum amount of weight (in integer units) that a wrapped packet of khakhras can withstand without breaking. I know that \( n \) units of weight when placed on the khakhras will surely destroy them.

One natural strategy is to try binary search: try weight of \( n/2 \), see if the khakhras break, then recursively try weight \( n/4 \) or \( 3n/4 \) depending on the outcome. Each time the khakhras break, a new packet of khakhras is used to continue testing until we find the answer. But I was not comfortable with this method as this would mean losing too many khakhra packets to find the answer.

Another strategy that I considered (since I wanted to conserve as many khakhra packets as possible) was to start from the lowest weight of one unit, then two units, and so forth, testing the khakhras by placing them under one larger unit of weight each time until the khakhras break. Using this strategy, I only would lose a single packet of khakhras – the first time the khakhras break, I have my answer. The drawback of this strategy is that in the worst case, I may have to test against \( n \) weights (rather than \( \log n \) as in the binary search solution).

So the trade-off is that we can test against fewer weights, if we are willing to lose more khakhra packets. Suppose I am willing to lose upto two packets of khakhras, design a

\(^1\)https://en.wikipedia.org/wiki/Khakhra
strategy for finding the maximum integral weight that the packaged khakhras can withstand without breaking, by testing them against at most \( f(n) \) different weights. The function \( f(n) \) grows slower than \( n \), i.e., \( \lim_{n \to \infty} \frac{n}{f(n)} = \infty \). Prove your answer.