Sample Homoework writeups

1 Find the missing Int [25 points]

Problem 4-2 (Cormen, Leserson and Rivest), find the missing int: We are given an array A with $n = 2^k - 1$ k bit numbers. Thus at least one k-bit number is missing (in the range $0 \ldots 2^k - 1$

We can access the numbers in A one bit at a time (e.g. we have an operation Read(A, i, j) that returns the ith bit of $A[j]$. Give an algorithm that finds a missing integer using at $O(n)$ Read operations (that is, bit access to the array A).

Solution
First note that the "bit-access" restriction only applies to the array A. For other operations we can use normal operations which access an entire number in one step and we can use some extra space (in particular we will use an extra array B to record positions in A which are still of interest in our approach below).

If n is a k bit binary number we can view the array A as a k by n two dimensional array such that $A[i, j]$ is the ith most significant bit of the jth element of A. Examining an elment of this 2D matrix has unit cost.

To begin with assume $n = 2^k - 1$ so there are $n + 1 = 2^k$ potential numbers. The basic idea is as follows: half the potential numbers have a zero in their most significant bit position and half have a 1. Thus we count the number of 0's and store this in Z. If Z equals 2^{k-1} then we know there are only $(n - 2^{k-1}) = 2^{k-1} - 1$ numbers starting with 1, so the missing number starts with a 1 and we can ignore all the numbers which start with a zero. We now have a new problem of exactly the same form as the original: Find the missing number among the $2^{k-1} - 1$ numbers in our list which start with a 1 out of the 2^{k-1} possible numbers which start with a 1. Array B records each index of a number in A which start with a 1.

Similarly, If $Z = 2^{k-1} - 1$ we know the missing value starts with a zero, and we focus on the locations in A whose first bit is zero.

Thus we can do exactly the same process on the next most significant bit, considering only those numbers which start with a 1. We will again find whether our missing number has a zero or one in its second most significant bit position, and will thus be able to eliminate half the "live" locations in A.

If there are m live positions in A we can eliminate half the live positions in $O(m)$ time since we just need to check the current bit in each live position and record the index of each live number in our new list. Thus we get a recurrence of $T(n) = cn + T(n/2)$ which has solution of $T(n) = \Theta(n)$ (using the master method or just expanding it).

When $n < 2^k - 1$ we do not have an exactly even split at each step. However, we can easily calculate the ideal number of 0's and 1's at each step (e.g. if n is a k-bit number, there will be exactly 2^{k-1} possible k-bit numbers which start with a bit of zero, and $(n + 1) - 2^{k-1}$ possible k-bit numbers which start with a bit value of 1 (since $n + 1$ total values in the range $0 \ldots n$.)) So we can tell which bit value is mising at each step. Also, if we fix the value of the first b bits, the number of possible values is always at most 2^{k-b} , so this upper bound does half at each step .

2 Randomized Algorithms [25 points]

Suppose that you have two lists of integers all between 1 and 100. $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$.

You know that Y was created after X and either Y is an exact copy of X (including the order of the items), or Y is a list of random integers drawn uniformly between 1 and 10.

Part A. Give a Monte Carlo algorithm which tests if the two lists are identical and makes an error with probability at most $1/10^9$ for any input list X. Design an algorithm which is as fast as possible. (thus your algorithm should output either "copy" or "random" to describe list Y).

Note that this is for the worst possible input list X, which would have all numbers at most 10.

We simply look at the first 9 elements of each list. If any x_i is not equal to y_i output RANDOM (no error), otherwise (all 9 match) ouput COPY.

Part B. Justify the error bound for your algorithm.

We can only make an error if Y is RANDOM and we call it a copy. This happens if the random choices of y_i for $i = 1, ...9$ equal x_i. If x_i is between 1 and 10 (the worst case), this will happen with probability 1/10. Thus all 9 will match with probability 10^{-9}.

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

the algorithm always takes at most 9 steps (and will always do so when Y is a copy). Thus expected O(1) time.

3 Lower Bound [20 points]

Suppose you are given two sorted lists of n real numbers, $X = x_1, < x_2, < \ldots, < x_n$ and $Y = y_1, < y_2, < \ldots, < y_n$.

We want to check if the two lists are identical. Prove that **any** comparison based algorithm for checking if the two lists are the same must use n comparisons on its worst input.

1) Suppose for contradiction some program P can solve this with n-1 compares.

2) Feed P two lists consisting of $x_i = y_i = 2i$. Output = equal

3) since there are only $n-1$ compares there is some element not used in any compare. Call this x_j (same if only uncompared is in y).

4) change x_j to $x_j + 1$.

5) List is still sorted

6) When feed P the new lists it gets the same feedback from each compare, so P continues as in step 2 outputting equal which is wrong.

thus no program can solve this with less than n compares.