CS 122A Winter 2013. HW 2 Due by 5pm Friday January 25, 2013

1. Problem 2.4 in the book
2. Problem 2.5 a) through e)
3. Problem 2.12
4. Problem 2.17

Answer: Here is a solution that is similar to binary search. Look at the middle element of the array A, say at position k. Check if $A[k] = k$, or $A[k] > k$ or $A[k] < k$. In the first case, you are done. In the second case, the desired position i, if it exists, can only be less than k because $A[i] > i$ for every $i > k$, since i increases by exactly one for each new position, and $A[i]$ increases by at least one for each new position. To be more explicit, lets look at position $k+d$ for some positive d. Then $A[k+d] \geq A[k] + d > k + d$, since $A[k] > k$.

Similarly, by the same logic in reverse, in the third case, the desired position i, if it exists, can only be greater than k.

So the algorithm is able to reject half of the remaining positions with each query and hence runs in $O(\log n)$ time.

5. Problem 2.23 a)

Answer: We can solve the problem with a recursive divide and conquer method call $M(L)$ which takes in a set L and either returns the majority element in L or returns that there is no majority element in L.

The key idea is this: If there is a majority element in a list L, say element X, then X must be the majority element in one or both of the two halves of L, call them A and B. This follows from arithmetic. Remember that majority means strictly more than half of L. So, procedure $M(L)$ divides L into two equal size lists A and B and calls $M(A)$ and $M(B)$. Again, if X is the majority element in L, then one or both of these calls must return X.

So, if the recursive calls to $M(A)$ and $M(B)$ find that neither A or B has a majority element, then L has no majority and the algorithm terminates. Otherwise, when the recursive calls $M(A)$ and $M(B)$ return, we finish the call $M(L)$ by doing the following: the candidate majority element returned from $M(A)$, if one was found, is compared to all the elements in L to see if it is actualy majority in L. Similarly, if there is a majority element returned from $M(B)$, then it is compared to all the elements in L to see if it is actually the majority in L. These comparisons are essential (do you see why?), and the time needed for it is $O(|L|)$. Only one of the two returned candidates (assuming there are two candidates) can be the majority element of L, so $M(L)$ can determine it and return it, or return that there is no majority element in L.

The total time for the algorithm is then given by $T(n) = 2T(n/2) + O(n)$, which we have seen solves to $O(n \log n)$.

6. Recall Problem 1 in HW 1.

Let $R(1), ..., R(n)$ and $C(1), ..., C(m)$ be any non-negative integers labeling rows and columns respectively, such that $\sum_{i=1}^{n} R(i) = \sum_{j=1}^{m} C(j)$. In HW 1 we learned that one can find non-negative values for the table entries to make each row i sum to $R(i)$ and each column j sum to $C(j)$. Such a set of values is called a *legal solution* for the table. There may be many different legal solutions.

Obviously, in every legal solution, the value in cell (i, j) is less than or equal to the Minimum of $R(i)$ and $C(j)$. Show that for any (i, j) there is a legal solution where the value of cell (i, j) is exactly the Minimum of $R(i)$ and $C(j)$. Note, that there might not be (and usually will not be) a legal solution where *each* cell (i, j) is set to that value. Rather, the nm legal solutions (one for each cell in the table) could all be different.

Answer: There are many possible solutions. Here is one.

In the solutions to HW 1 problem 1, the method presented to find a legal solution stated with cell (1,1) and gave it value $\min[C(1), R(1)]$, which proves what we want for cell (1,1). But the problem calls for proving this for any cell (i, j). To prove this, suppose we take the given empty table T and move column j to the first position, i.e. make it the new column 1. Next, suppose we move row i to the first position, i.e., make it the new row 1. Call this new table T', and note that cell (i, j) of T is now cell $(1, 1)$ of T'. Note that the *set* of row sums in T and T' are the same, and that the *set* of column sums in T and T' are the same. Next, find a legal solution in T' where cell $(1, 1)$ is given value equal to the minimum of column 1 sum and row 1 sum in T'. As mentioned above, we know from HW 1 that this is possible. Next take that legal solution to T' and move the first row back to position i and then move first column back to position j. The result is a filled-in table T, which is a legal solution for T (since the set of row and column sums are the same in T and T'), where cell (i, j) of T has value equal to $\min[C(i), R(j)]$.