Algorithms Midterm Solutions

1. (a) Here are two possible solutions:
   - We can compute $S[1]$ in $O(n)$ by simply iterating. Then, for each $i \geq 2$, we have that $S[i] = S[i-1] - A[i-1] + A[n+i-1]$, so we can compute each subsequent $S[i]$ in $O(1)$ time each, giving an $O(n)$ algorithm.
   - We can first compute the array $C[0..2n-1]$ where $C[i] = A[1]+A[2]+\cdots+A[2n-1]$. We have that $C[0] = 0$, and for $i \geq 1$, $C[i] = C[i-1]+A[i]$. Hence, we compute C in $O(n)$. Then $S[i] = C[i+n-1]-C[i-1]$, so we can compute all the S values in $O(n)$.

   (b) First, sort the array (e.g. using MergeSort) in $O(N \log N)$ time. We know that it is no less optimal to cast our net starting from the same position as a fish. Now there are two possible ways of proceeding:
   - For each fish in sorted order (say, at index L), we assume the net is cast starting from its position x. We binary search to find the first fish at position **strictly greater** than $x + W$. Suppose that fish has index R. Then the number of fish we would catch at this position is $R - L$, so we take the maximum among all these values.
     We perform N binary searches, each taking $O(\log N)$ time, so our overall complexity is $O(N \log N)$.
   - We can use a "two pointers" approach: we keep variables L and R, so the fish in positions $[L..R]$ of the array are those in our net.
     Initially $L = 1$, and in $O(N)$ time we find the largest R so that the Rth fish in sorted order is still in our net. We then repeatedly increment L, and respectively repeatedly increment R to include all fish that fit within the net starting at the Lth fish. At each stage, we know we can catch $R - L + 1$ fish, so we take the maximum among these.
     Since L and R are only ever incremented, and they are each incremented at most N times, the algorithm is $O(N)$.

2. (a) The equivalent polynomial is $P(x) = 4x^3 + 2x + 1$, with $P^2(x) = 16x^6 + 16x^4 + 8x^3 + 4x^2 + 4x + 1$. Hence, $s * s = \langle 1, 4, 4, 8, 16, 16 \rangle$.

   (b) The convolution has $n + k - 1$ terms, because it is the sequence of coefficients of the product of two polynomials, one of degree $n - 1$, the other of degree $k - 1$. Thus the product of these two polynomials is a polynomial of degree $n + k - 2$ and such polynomial has $n + k - 1$ many coefficients.

   (c) Yes. Any of the following justifications will do:
   - The definition of convolution is completely symmetric.
   - Showing this algebraically from the definition of convolution.
   - Convolution can be phrased in terms of polynomial multiplication, which is commutative.

(d) We convert each sequence to a polynomial. Specifically, the sequence $\langle a_0, a_1, \dots, a_{n-1}\rangle$ is converted to $\sum_{i=0}^{n-1} a_i x^i$. The coefficients in the product of the polynomials correspond exactly to the linear convolution of the sequences, so it suffices to compute this product.

Using FFT, we evaluate each sequence at the k roots of unity, where k is the smallest power of two no smaller than the sum of the lengths of the sequences. We multiply the evaluations at the corresponding roots of unity, and use the inverse FFT to retrieve the product in coefficient form. This runs in $O(N \log N)$ where N is the sum of the lengths of the sequences.

3. (a) Sort A and B into non-decreasing order. Then, connect the first house in sorted order with the first hub in sorted order, and repeat for all n houses. The time complexity is $O(n \log n)$.

Beware of incorrect solutions: for instance, connecting the closest house / hub pair is typically not correct.

(b) Without loss of generality, suppose A and B are in sorted order. Note that we can express any solution by an array $M[1..N]$, where $M[i] = x$ means the ith house in sorted order was connected to the $M[i]$th hub in sorted order. We seek to show that when $M[i] = i$ for all i, the total length of cable required is no more than for any other solution M'.

Let M' be some solution (a permutation) which is not ours. Then, necessarily, there is some $i \leq N-1$ with $M'[i] > M'[i+1]$. Now if we exchange $M'[i]$ and $M'[i+1]$ we get a solution that is no less optimal. To see, this consider the relative order of $A[i], A[i+1], B[M'[i]]$ and $B[M'[i+1]]$ (draw them on a number line). Then, if the two houses are both either before or after the hubs, or the hubs are between the houses, or the houses are between the hubs, then the solution strictly improves. Otherwise, the solution stays the same.

By repeating this (by Bubble sort), we know that we will obtain our sorted order (the identity) in a finite number of steps regardless of the starting order M'. Hence, our solution is no worse than any other, so it is optimal.

4. We first find $M[N/2]$. We can do this in N questions by simply examining each element in row $N/2$, and finding the largest one. Suppose $M[N/2] = x$. Then, we know for all $i < N/2$, $M[i] \leq x$ and for all $j > N/2$, $M[j] \geq x$. Thus, we can recurse to solve the same problem on the subgrids $A[1..(N/2)-1][1..x]$ and $A[(N/2)+1..N][x..N]$. It remains to show that this approach uses at most $2N\lceil \log N \rceil$ questions.

Consider the recursion tree: it forms a perfect binary tree. The recursive calls at each level of the recursion correspond to some range of columns. Because of the monotonicity of M, these columns are *almost* disjoint: notice that column x is shared by both calls. Thus, the sum of columns is at *most* $2N$: N if they were disjoint, and each column is repeated at most once. Since the number of questions we ask in any call is equal to the number of columns in the call, we use at most $2N$ calls in each level, and there are $\lceil \log N \rceil$ levels. Hence, we use at most $2N\lceil \log N \rceil$ questions overall.