CSE 101 Homework 3

Winter 2015

This homework is due Friday February 13th at the start of class. Remember to justify your work even if the problem does not explicitly say so. Writing your solutions in \LaTeX is recommend though not required.

**Question 1** (Deterministic Order Statistics, 40 points). Although the algorithm that we saw in class for computing order statistics requires randomization, it turns out that this is not required to find an efficient algorithm. Consider the following algorithm for order statistics:

```
DeterministicOrderStatistics(L,k) \ finds the kth largest element in list L

Split the elements of L into groups of 5 elements each
Compute the medians of each group of 5 (using some constant time algorithm for each one)
Put these medians together in a list M
Let x = DeterministicOrderStatistics(M,|M|/2) \ x is the median of M
Compare all elements of L to x and sort them into lists S (the elements smaller than x) and B (the elements larger than x)
If |B| > k-1 \ use x as a pivot as before to reduce to a problem of smaller size
    Return DeterministicOrderStatistics(B,k)
If |B| = k-1
    Return x
If |B| < k-1
    Return DeterministicOrderStatistics(S,k-1-|B|)
```

(a) Show that if \(|L| = n\) that this algorithm makes one recursive call on a list size at most \(n/5 + O(1)\) and one on a list of size at most \(7n/10 + O(1)\). \([15\text{ points}]\)

(b) Write a recurrence relation for the runtime of this algorithm on lists of size \(n\). \([10\text{ points}]\)

(c) Prove by induction that the runtime of this algorithm is \(O(|L|)\). Note that you will need to be careful to show that constant in your big-O does not change between inductive steps (recall the last problem of Homework 0). \([15\text{ points}]\)

**Solution 1.**

(a) The size of the list \(M\) is \(n/5 + O(1)\) since it selects the median out of each group of 5 elements. The extra \(O(1)\) term is present because \(n\) might not be a multiple of 5. For the next recursive call, consider the elements in \(L\) that are guaranteed to be less than \(x\). Since, \(x\) is the median of medians, it is greater than at least \(n/10+O(1)\) of the medians. Each of these medians is greater than two numbers in their respective 5 element lists. Thus, there are at least \(n/10 - O(1) + 2(n/10 - O(1)) = 3n/10 - O(1)\) elements in \(L\) that are smaller than \(x\). Consequently, there are at most \(7n/10 + O(1)\) numbers in \(L\) that are greater than \(x\). By following the same argument for numbers greater than \(x\) in \(L\), we can say that there are at most \(7n/10 + O(1)\) elements smaller than \(x\) in \(L\). The size of either of the lists \(S\) and \(B\) is at most \(7n/10 + O(1)\).

(b) The time to split and compute median for groups of 5 is a linear term since there are \(n/5 + O(1)\) groups which require constant time each for finding median. Also, the time to split \(L\) into lists \(S\) and \(B\) is
linear. Let $T(n)$ be runtime of this algorithm for a list of size $n$. Since there are at max two recursive calls as seen from Part (a), we have

$$T(n) \leq T(n/5 + O(1)) + T(7n/10 + O(1)) + O(n)$$

(c) $T(1) = O(1)$ is trivially true. Suppose $T(n) \leq cn$ for all $n \geq n_0$ for some constants $c$ and $n_0$. Suppose the linear term in the above recurrence is $\leq dn$ for all $n \geq n_1$ for constants $d$ and $n_1$. For the induction step, we show $T(n + 1) = O(n + 1)$. We have

$$T(n + 1) \leq T((n + 1)/5 + O(1)) + T(7(n + 1)/10 + O(1)) + O(n + 1)$$

$$\leq c(n + 1)/5 + O(1) + c(7(n + 1)/10 + O(1)) + d(n + 1) \quad \forall n \geq \max(n_0, n_1)$$

$$= 9(n + 1)/10 + d(n + 1) + O(1)$$

$$= (n + 1)(9c/10 + d) + O(1)$$

$$\leq c(n + 1)$$

This is true for $n \geq n_2$ where $n_2$ is sufficiently large and $d < c/10$.

**Question 2** (Echoes and Convolutions, 35 points). David is exploring a cave. He finds that when he shouts at volume $V$ that he hears an echo of it at volume $a_i \cdot V$ exactly $i$ seconds later for each $1 \leq i \leq n$ (though perhaps some of the $a_i$ are 0, meaning that there is no echo at that time). In fact, if he considers the initial shout to be a trivial echo with $a_0 = 1$, all of the noise produced in this way can be thought of as ‘echoes’. David also discovered that if he shouts several times that the volumes of the various echoes of his shouts add together.

(a) Suppose that at some starting time, $t$, David shouts at volume $V_t$ exactly $i$ seconds after $t$ for each $1 \leq i \leq n$. Give a formula to express the total volume $R_i$ of all echoes that David hears $i$ seconds after the start time. [10 points]

(b) Give an $O(n \log(n))$ algorithm to compute the values of the $R_i$ given the $a_i$ and the $V_i$. [10 points]

(c) Suppose that Eve is travelling with David. By analyzing the caves, she is able to determine the values of $a_i$, and by listening to echoes can determine the values of the $R_i$. Give an $O(n \log(n))$ time algorithm to show how she could reconstruct the values of the $V_i$ from this data. Hint: if convolution corresponds to polynomial multiplication, how would we undo this operation? [15 points]

**Solution 2.** (a) The total volume at time $i$ is Dave’s shout volume $V_i$ plus the volume at time $i - j$ multiplied by the echo coefficient $a_j$.

$$R_i = V_i + \sum_{j=1}^{i} a_j \cdot V_{i-j}$$

Since $a_0 = 1$, we can also write this more simply as:

$$R_i = \sum_{j=0}^{i} a_j \cdot V_{i-j}$$

(b) The expression derived from part (a) is exactly the same expression which is used to evaluate the product of two polynomials. Thus, calculating the $R_i$ is analogous to calculating the coefficients of the polynomial $R(x) = A(x) \cdot V(x)$, where $a(x) = a_0 + a_1 x + a_2 x^2 \ldots a_n x^n$, and $V(x) = V_0 + V_1 x + \ldots V_n x^n$.

Multiplying the polynomials can be done efficiently if the polynomials are first transformed into characteristic value sets via the Fast Fourier Transform. If we know the values of $A(x)$ and $V(x)$ for $2n$ values of $x$, where $n$ is one more than the degree of $A$ and $V$, we can calculate the values of $R$ for
these values of \( x \), since \( R(x) = A(x) \cdot V(x) \). When we find these values of \( R(x) \), we can invoke the inverse Fast Fourier Transform to get the coefficients of \( R \), which is the solution to our problem.

So our algorithm is:

```python
FindResonance(A_c, V_c, n):
    // A_c is the array of echo coefficients, a_0 ... a_n
    // V_c is the array of shout volumes, V_0 ... V_n
    // n is the size (minus 1) of the arrays A_c and V_c
    R_v = empty array of size 2n
    // Treating A_c as the coefficients of a polynomial,
    // evaluate A at the 2nth roots of unity
    A_v = FFT(A_c, 2n)
    V_v = FFT(V_c, 2n)
    for 0 <= i < 2n:
        R_v[i] = A_v[i] * V_v[i]
    R_c = FFT-inverse(R_v, 2n)
    return R_c
```

(c) This problem is very similar to part (b). Since we know that \( R(x) = a(x) \cdot V(x) \), we can rearrange this equation to get \( V(x) = R(x)/a(x) \). Thus, we can use the same strategy for computing \( V(x) \) as we did to compute \( R(x) \) in the last problem, but using division instead of multiplication: apply the Fast Fourier transform to \( R_i \) and \( a_i \) values; compute the representative values of the polynomial \( V \); and then apply the inverse Fast Fourier transform to the representative values.

```python
FindVolumes(A_c, R_c, n):
    // A_c is the array of echo coefficients, a_0 ... a_n
    // R_c is the array of measured resonance volumes, R_0 ... R_2n
    V_v = empty array of size n+1
    A_v = FFT(A_c, n)
    R_v = FFT(R_c, n)
    for 0 <= i < n:
        V_v[i] = R_v[i] / A_v[i]
    V_c = FFT-inverse(V_v, n)
    return V_c
```

**Question 3** (Binary Search On Unimodal Lists, 25 points). Call a list \( L \) of numbers unimodal if the elements of the list increase for a while and then decrease. In particular, a list \( L \) of length \( n \) is unimodal if 

\[
\]

for some index \( k \).

(a) Give an \( O(\log(n)) \) time algorithm to find the index of the largest element of a unimodal list of length \( n \) (so in the example above, your algorithm should return \( k \)). [10 points]
(b) Give an $O(\log(n))$ time algorithm to search a unimodal list for a given element. In particular, given a unimodal list $L$ of length $n$ and a number $x$ your algorithm should determine whether or not $x$ is an element of $L$. [15 points]

Solution 3.

(a) Assume that a lookup into the list at a given index is a constant-time operation. We can perform a binary search on the list to find the index of the peak element, since we can figure out whether a given index is to the left of the peak element or to its right. If, at a given index $i$, $L[i-1] < L[i]$ and $L[i] < L[i+1]$, then $i$ is to the left of the peak, and we need to search to the right of $i$ in the next step. If $L[i-1] > L[i]$ and $L[i] > L[i+1]$, $i$ is to the right of the peak, and we need to search to the left in the next step. Eventually, we either end up at the beginning or end of the list, or we end up at index $k$. As with an ordinary binary search, the decision to search to the left or to the right is a constant time operation, so the algorithm has complexity $O(\log n)$.

(b) First, search for the peak element. If you find $x$ along the way, return true. Otherwise, once you have found the peak element, the elements to its left form a list sorted in ascending order, and the elements to its right form a list sorted in descending order. One can thus perform binary searches on both of these sublists to find $x$. This requires at most three binary searches, so the time complexity is still $O(\log n)$.

Question 4 (Extra credit, 1 point). Approximately how much time did you spend working on this homework?