CS 6505 - Homework 3

Caitlin Beecham (Worked with Qiaomei Li and Shenghua Xiang)

1

Our algorithm is as follows:

1. Take the array A and make a polynomial A(x) whose terms run from $x^0(=1)$ to $x^{max\{A_j|j\in[n]\}}$ where the coefficient of x^k is l_k, where l_k is the number of times k appears in A (the coefficient can be 0 if k does not appear in A). This step runs in O(n) time since for each element in A we are adding a term to the polynomial (this may mean updating the coefficient of a term if this element of A has appeared earlier).

2. Cube this polynomial using the Fast Fourier Transform. What is the runtime of this step? Well squaring it takes O(alog(a)) = O(nlogn) time where $a \leq n$ is the maximum element of A. Then multiplying by A(x) again to get the cube runs in O(2a*log(2a)) time, which is equal to $O(2a*(log(a)+1)) = O(2a*log(a)+2a) = O(a*log(a)) = O(nlog(n))$ since $a \leq n$. So, in total this step take $O(nlogn) + O(nlogn) = O(nlogn)$ time.

3. Finally, we check the coefficient of x^t in $A^3(x)$. If this is non-zero, there is such a triple, since the coefficient of x^t counts the number of ordered triples $(i,j,k) \in [n]^3$ such that $x^{A[i]} * x^{A[j]} * x^{A[k]} = x^{A[i]+A[j]+A[k]} = x^t$. (If $A[i_1] = A[i_2]$ for some $i_1, i_2 \in [n]$ this properly counts both triples of the form (i_1, j, k) and triples of the form (i_2, j, k) as separate, which they are, because the coefficient, l_{i_1} of $x^{A[i_1]}$ in A(x), properly accounts for this). This step is done in $O(1)$ time, since we simply access the coefficient of x^t.

Thus, our algorithm in total takes $O(n) + O(nlogn) + O(1) = O(nlogn)$ time.

2

Our algorithm is as follows:

1. Take the array A and make a polynomial A(x) whose terms run from $x^0 (= 1)$ to $x^{max\{A_j | j \in [n]\}}$ where the coefficient of x^k is l_k, where l_k is the number of times k appears in A (the coefficient can be 0 if k does not appear in A). This step runs in O(n) time since for each element in A we are adding a term to the polynomial (this may mean updating the coefficient of a term if this element of A has appeared earlier).

2. Cube this polynomial using the Fast Fourier Transform. What is the runtime of this step? Well squaring it takes O(alog(a)) = O(nlogn) time where $a \leq n$ is the maximum element of A. Then multiplying by A(x) again to get the cube runs in O(2a*log(2a)) time, which is equal to $O(2a * (log(a) + 1)) = O(2a * log(a) + 2a) = O(a * log(a)) = O(nlog(n))$ since $a \leq n$. So, in total this step take $O(nlogn) + O(nlogn) = O(nlogn)$ time.

3. Finally, we check the coefficients of x^r where $r \leq t$ in $A^3(x)$. We do this in O(n) time. We then add these t+1 coefficients in O(n) time. This number is the number of ordered triples for which the sum $A[i] + A[j] + A[k] \leq t$. Why? Because the coefficient of x^r counts the number of ordered triples $(i, j, k) \in [n]^3$ such that $x^{A[i]} * x^{A[j]} * x^{A[k]} = x^{A[i]+A[j]+A[k]} = x^r$. (If $A[i_1] = A[i_2]$ for some $i_1, i_2 \in [n]$ this properly counts both triples of the form (i_1, j, k) and triples of the form (i_2, j, k) as separate, which they are, because the coefficient, l_{i_1} of $x^{A[i_1]}$ in A(x), properly accounts for this). In total this step takes O(n) + O(n) = O(n) time.

Thus, in total our algorithm takes $O(n) + O(nlogn) + O(n) = O(nlogn)$ time.