Change log:

- Problem 1 (version 2): added more clarification.
- Problem 2 (version 2): changed required runtime bound to $\tilde{O}(n^2)$ (from $\tilde{O}(n)$).
- Problem 3 (version 2): changed range of $j$ to $j > i$.
- Problem 4 (version 2): added clarification on ‘shift of $b$ against $a$’.
- Problem 6 (version 3): clarified that it’s an easier version of problem 12.
- Problem 7 (version 3): added that $| \equiv$ means bitwise OR.
- Problem 8 (version 3): clarified that $y = O(n)$.
- Problem 8 (version 3): switched the $k$ out of the subscript into a 2 parameter function ($f_k(x) \rightarrow f(k,x)$), both notations are acceptable for solutions.
- Problem 12 (version 3): added details on the implicit representation of input.

This problem set has a total of 12 written problems, plus 1 coding problem, on 3 pages. Written solutions should be submitted to Canvas by 2pm Wednesday Feb 10, 2020. Coding solutions can be submitted separately.

- If you find the sources of these problems, a citation of the source is sufficient for full points.
- If you choose not to submit a typed write-up, please write neat and legibly.

1. Prove that in a recursive in-place fast Fourier transform on $n$ elements where $n$ is a power of 2, the transformed coefficient $i$ ends up in the index whose bit representation is the reversal of entry $i$.

SOLUTION:

Consider a coefficient at index $i$, represented by $b$. At the first recursive level, the tree with branch based on the least-significant bit of $b$: if $b_0 = 0$, $c_i$ is in $A_{even}$ and if $b_0 = 1$, $c_i$ is in $A_{odd}$. In general, we can see that the $j^{th}$ level of recursion compares the bit at index $b_{j-1}$.

We notice that FFT returns the transformation by returning the even subtree then the odd subtree. Thus, we notice that the order of returns corresponds with
0...0, 0...1, 0...10, 0...11, ..., 1...1

which is the bit-reversal of the index of the original coefficient.

2. Show that convolutions naturally generalize into 2D. That is, given a pair of 2-D arrays \( A \) and \( B \) of dimensions \( n \times n \), we can compute for each shift values \( \delta x \) and \( \delta y \), the value

\[
\sum_{i,j} A[i][j] \cdot B[i+\delta x][j+\delta y]
\]

in \( \tilde{O}(n^2) \) time (over all \( 1 \leq \delta x, \delta y \leq n \)). Out of boundary entries can be treated as 0.

**SOLUTION:**

Let us create \( 2n^2 \) sized arrays \( A' \) and \( B' \) by unstacking the rows of each matrix and joining them with a \( 0 \) vector of length \( n \) in between. For example, for a \( n \times n \) matrix \( X \), we have

\[
X' = X[0] + 0 + X[1] + 0 + \ldots + X[n-1]
\]

where + indicates a vector concatenation. We notice that performing a fast convolution on \( A' \) and \( B' \) results in a vector \( C' \) where

\[
C[i][j] = C'[i \cdot n + j]
\]

As we fast convolve two vectors of size \( 2n^2 \), we have a complexity of \( O(n^2 \log n^2) = \tilde{O}(n^2) \).

3. Show that given a sequence \( A[1] \ldots A[n] \), we can compute for each \( i \), the values

\[
\sum_{j>i} \frac{A[j]}{(i-j)^2} \quad (\text{mod } p)
\]

for all \( 1 \leq i \leq n \), for some in prime \( p \in (10n, n^2) \) with \( p = 2^k \cdot 3 + 1 \) in \( \tilde{O}(n) \) time.

**SOLUTION:**

Let us use fast convolution to calculate the values as

\[
\sum_{j>i} \frac{A[j]}{(i-j)^2} = \sum_{j>i} A[j]B[j-i] \quad (\text{mod } p)
\]

Note that we must then find \( B \) such that \( B[j-i] = \frac{1}{(i-j)^2} \) (mod \( p \)) \( \implies B[i] = (i^2)^{-1} \) (mod \( p \)).

Thus, we can use the Extended Euclidean Algorithm to find the multiplicative inverse of \( i^2 \) (mod \( p \)) in \( \tilde{O}(n \log^2 p) \), giving us a total running time of \( \tilde{O}(n \log^2 p + n \log n) = \tilde{O}(n) \).
4. Show that given binary strings $a$ and $b$ with total length $n$, we can compute the number of mismatches of each shift of $b$ against $a$ in total time $\tilde{O}(n \log n)$. That is, for $b$ with length $|b|$, we want to compute for each location $i$, the number of mismatches between $b$ and $a[i \ldots i + |b| - 1]$. Out of bounds entries in $a$ can be treated as mismatches.

**SOLUTION:**
We notice that the convolution of $a$ and $\overline{b}$, where $\overline{b}$ indicates the inverse of string $b$, counts the number of mismatches of the form $a_i = 1, b_{i-j} = 0$ for some index $i$ and some shift of $b j$.

Similarly, the convolution of $\overline{a}$ and $b$ indicates the number of mismatches of the form $a_i = 0, b_{i-j} = 1$. Using FFT for fast convolution twice and summing the results gives us the solution.

5. An integer is palindromic if it reads the same forward and backward. Show that the number of palindromic integers between 1 and $10^U$ that have residue $k$ modulo $p$ can be computed in $\tilde{O}(p^2 \log U)$ time. (**note:** this only needs generating functions, no fast convolution)

**SOLUTION:**
Consider integers between 0 and $10^{p-1}$. We construct a generating function

$$f(x) = \sum_{i \geq 1} c_i x^i$$

such that $c_i$ is the number of integers $n$ between 0 and $10^{p-1}$ such that $n + n' \equiv i \pmod{p}$ ($n'$ is the reverse of integer $n$).

We note that if $p = 2$ or $p = 5$, we only need to check the least-significant digit of an integer to take mod($p$).

Otherwise, $\gcd(p, 10) = 1 \implies 10^{p-1} \equiv 1 \pmod{p}$. Thus, we can calculate mod($p$) of any integer by splitting it into groups of $p - 1$ digits and summing each group mod($p$). We then have our overall generating function

$$g(x) = f(x) \frac{U}{p}$$

where the coefficients $c_i$ of $g(x)$ are the number of palindromes $n$ where $n \equiv i \pmod{p}$. Note that the polynomials do not get larger with every multiplication because we can sum $c_i$ and $c_{i+p}$.

6. Show that given a collection of elements, each with a positive integer size, size is at most $n$, we can compute, for each $x = 0 \ldots n$ and some modulus $M \leq O(n)$, the number of subsets of these elements whose total size is $x$, in $\tilde{O}(n \log^2 n)$ time.

**note:** this is an easier version of 12, with the extra condition that the total sizes of elements is $O(n)$. 
**SOLUTION:**
Letting the $i^{th}$ element have weight $t_i$, we have the generating function

$$A(x) = (1 + x^{t_1})(1 + x^{t_2}) \ldots (1 + x^{t_k}) \equiv \sum_{i \geq 0} c_i x^i \pmod{x^{n+1}}$$

where we only need to find the first $n$ terms. We then need to calculate its coefficients in an efficient way.

Consider a divide and conquer algorithm on the two halves of the product of $k$ polynomials. We use FFT to combine the two halves through polynomial multiplication on the first $n$ terms. Notice that we then have a recurrence relation

$$T(k) = 2T\left(\frac{k}{2}\right) + \tilde{O}(n \log n)$$

and as we are given a collection of elements, $k \leq n$. We then can solve this recurrence by counting the number of operations $\tilde{O}(n \log n) + 2\tilde{O}(\frac{n \log n}{2}) + \ldots + 2^{O(\log n)}\tilde{O}(\frac{n \log n}{2^{O(\log n)}}) = \log n\tilde{O}(n \log n) = \tilde{O}(n \log^2 n)$.

7. Define the prefix-or of a sequence $A[1 \ldots n]$ to be the sequence with the $i^{th}$ entry given by


where $\mid$ denotes bitwise OR of the numbers.

Show that the number of length $n$ sequences with all entries in the range $1 \ldots 2^k - 1$ (non-zero $k$-bit numbers) that have strictly increasing prefix-or values can be computed in $\tilde{O}(\log n \cdot k)$ time.

**SOLUTION:**

8. Define a sequence of functions from three coefficients $a$, $b$, $c$, as

$$f(0, x) = 1$$

$$f(k, x) = \sum_{i=0}^{x} (ai^2 + bi + c) f(k-1, i)$$

Show that for some $n$, some $y = O(n)$, and some modulus $M = \Theta(n)$, we can compute

$$f(j, y) \quad \forall \ 0 \leq j \leq n$$
in $\tilde{O}(n)$ time.

**SOLUTION:**
Define the generating function

$$G(\cdot, y)(z) = \sum_{i=0}^{\infty} f(i, y) z^i.$$  

Decomposing the summation into its first $k - 1$ terms and the last term gives

$$f(k, x) = f(k, x - 1) + (ax^2 + bx + c) f(k - 1, x)$$

which in generating equation form gives

$$G(\cdot, y)(z) = G(\cdot, y - 1)(z) + (ay^2 + by + c) z G(\cdot, y)(z).$$

Solving gives

$$G(\cdot, y)(z) = \frac{G(\cdot, y - 1)(z)}{1 - (ay^2 + by + c) z},$$

which upon telescoping gives

$$G(\cdot, y)(z) = \prod_{i=1}^{y} \frac{1}{1 - (ai^2 + bi + c) z}.$$

The denominator product takes $\tilde{O}(n)$, upon which inverting gives the required coefficients.

**note:** a runtime of $\tilde{O}(n \log y)$ is also possible, but bound on $y$ was not specified in the initial version, so was not required. Will ask this in the next problem set.

9. Show that for $n$ and prime $p \in (10n, n^2)$ with $p = 2^k \cdot 3 + 1$, and distinct integers $A[1] \ldots A[n] \in [0, p)$, we can compute

$$\sum_{j \neq i} \frac{1}{A[i] - A[j]} \pmod{p}$$

for all $1 \leq i \leq n$ in $\tilde{O}(n)$ time total.

**Note:** although this reads like a finite field analog of fast multipole method (FMM), the solution is actually quite different.

**SOLUTION:**
Divide-and-conquer combined with multi-point evaluation of polynomials.
Consider the function \( g(x) \)

\[
f(x) = \prod_{j \geq 0} (x - A[j])
\]

\[
g(x) = f'(x) = (x - A[2])(x - A[3]) \ldots x(x - A[n]) + \ldots + (x - A[1])(x - A[2]) \ldots x(x - A[n - 1])
\]

We note that every term \( x - A[i] \) appears in \( g(x) \) \( n - 1 \) times and we wish to

(a) Evaluate \( x - A[i] \)
(b) Take the modular inverse mod(\( p \))
(c) Take \( e \) to the power of the result

Note that we can do all these steps in \( O(\log p) = O(\log n^2) \) using the Extended Euclidean Algorithm to find the modular inverse.

Thus evaluating the \( g(x) \) at \( n \) terms \( (A[1], \ldots, A[n]) \) with FFT in this way, we end up with \( e^{y_i} \), where \( y_i = \sum_{j \neq i} \frac{1}{A[i] - A[j]} \) (mod \( p \)). Note that while there are \( O(n^2) \) terms in \( g(x) \), each term repeats \( O(n) \) times. Taking the ln of the results, we are then left with a complexity of \( O(\frac{n^2 \log n^2 \log n^2}{n}) = \tilde{O}(n) \).

10. The next three problems concern the exponential and logarithms of generating functions. For a generating function \( A(x) \), define its exponential as

\[
\exp (A (x)) = \sum_{i \geq 0} \frac{A(x)^i}{i!}
\]

Show that given a \( n \) term generating function \( A(x) \) with integer coefficients and some modulus \( p \), the first \( n \) coefficients of \( \exp(A(x)) \), mod \( p \), can be computed in \( \tilde{O}(n) \) time.

**SOLUTION:**
Define the logarithm

\[
\ln (B (x)) = \sum_{i \geq 1} -\frac{(1 - B(x))^i}{i}
\]

We want to find \( B(x) \) such that

\[
\ln (B (x)) = A(x),
\]

or if we take derivative,

\[
\frac{B'(x)}{B(x)} = A(x),
\]
which rearranges to
\[ B'(x) = A(x)B(x). \]

Suppose we already have the first \( t \) terms of this, then solving for coefficients \( t + 1 \ldots 2t \) gives
\[ B'(x)_{t+1\ldots 2t} = (A(x)B(x)_{1\ldots t})_{t+1\ldots 2t} + A(x)_{1\ldots t}B(x)_{t+1\ldots 2t}. \]

The first term is fixed, so by recursion, we get the first \( 2t \) terms by solving two problems of size \( t \).

A fancier way that only tail-recurses is to do Taylor expansion on
\[ \ln(B_0(x)) \equiv A(x) \pmod{x^t} \]
to obtain
\[ B(x) \equiv B_0(x)(1 - \ln(B_0(x)) + A(x)) \pmod{x^{2t}}. \]

11. Consider the unordered knapsack problem, where there are \( A[i] \) types of item with weight \( i \), each can be used unlimited times. Show that the number of ways of picking a number of these items (unordered) to make total weight \( n \) is given by
\[ \left[ \#x^n \right] \exp \left( \sum_{j \geq 1} \frac{1}{j} A(x^j) \right). \]

where \( A(x) \) is the generating function with coefficients \( A[i] \).

**SOLUTION:**
Choosing only elements of \( i \), we obtain the generating function
\[ (1 + x^i + x^{2i} + x^{3i} + \ldots)^{A[i]} = \left( \frac{1}{1 - x^i} \right)^{A[i]} \]
where there are \( A[i] \) unique elements, all of which can be selected an unlimited number of times.

Thus, our overall generating function is
\[ \prod_{i \geq 0} \left( \frac{1}{1 - x^i} \right)^{A[i]} \]
as we have \( A[i] \) unique elements for each \( i \). Notice that
\[
\prod_{i \geq 0} \left( \frac{1}{1 - x^i} \right)^{A[i]} = \exp \left( \sum_{i \geq 0} \ln \left( \frac{1}{1 - x^i} \right)^{A[i]} \right)
= \exp \left( \sum_{i \geq 0} A[i] \ln \left( \frac{1}{1 - x^i} \right) \right)
= \exp \left( \sum_{i \geq 0} A[i] \sum_{j \geq 1} \frac{(x^i)^j}{j} \right)
= \exp \left( \sum_{j \geq 1} \frac{1}{j} \sum_{i \geq 0} A[i](x^i)^j \right)
= \exp \left( \sum_{j \geq 1} \frac{1}{j} A(x^j) \right)
\]

Notice that the third step follows from the Taylor expansion
\[
\ln \left( \frac{1}{1 - x} \right) = -\ln(1 - x) = \int_0^x \frac{dz}{1 - z} = \sum_{j \geq 1} \int_0^x z^j dz = \sum_{j \geq 0} \frac{x^{j+1}}{j + 1} = \sum_{j \geq 0} \frac{x^j}{j}
\]
where \( \ln \left( \frac{1}{1 - x} \right) = \sum_{j \geq 0} \frac{(x^j)^j}{j} \). Thus we have proven that the two generating functions are equivalent.

12. Let \( S \) be a set of positive integers between 1 and \( n \), given implicitly via its frequency vector. Aka. for element \( i \), there are \( a[i] \leq n \) copies. Show that we can compute modulo some prime \( p = \Theta(n) \) with \( p = 2^k \cdot 3 + 1 \), for each \( x \) between 1 and \( n \), the number of subsets of \( S \) that sum to \( x \), in \( \tilde{O}(n) \) time.

**SOLUTION:**

We want to take product of \( (1 + x^i) \). Taking logs gives
\[
\ln \left( 1 + x^i \right) = x^i - \frac{x^{2i}}{2} + \frac{x^{3i}}{3} \ldots
\]
the coefficients of these up to \( x^n \) can be aggregated in \( O(n \log n) \) time, because exponent \( i \) only affects \( n/i \) other terms.

Once we have this aggregated (with multipliers by \( a[i] \)), we compute its exponential in \( \tilde{O}(n) \) time to get the final set of coefficients.
CODE **Separate submission on Canvas, 5 points.** Code an algorithm that evaluates $n! \pmod{p}$ for some $p \leq n^2$ in $\tilde{O}(n^{2/3})$ time (or better). Autojudge: [https://dmoj.ca/problem/factorial3](https://dmoj.ca/problem/factorial3).

**SOLUTION:**
[https://codeforces.com/blog/entry/63491](https://codeforces.com/blog/entry/63491)