CSE525 Monsoon 2020 — Ashima Garg (PhD19003)
Homework 4 Problem a — Pramil Panjwani (PhD19008)

You CANNOT consult any other person or online resource for solving the homework problems. You can definitely ask the instructor or TAs for hints and you are encourage to do so (in fact, you will get useful hints if you ask for help at least 1-2 days before the due date). If we find you guilty of academic dishonesty, penalty will be imposed as per institute guidelines.

Recall polynomial division that you learned in school. Given a polynomial $P(x)$ and an integer a, we can write $P(x) = Q(x)(x-a) + R(x)$ where $R(x)$ is the remainder polynomial and $Q(x)$ is the quotient polynomial. (Try to recall the properties of the remainder and the quotient polynomials. Frankly, you do not need to look up anything on the internet or any book — everything required for this question follows from first principles.)

(a) Write a (recursive, if you can) $O(n)$ algorithm to compute both $Q(x)$ and $R(x)$ given $P(x)$ and a. Discuss complexity. To access the coefficient of x^k in some polynomial $p(x)$, you can simply write "coeff(k,p(x))". *Hint: What is the degree of $R(x)$?*

Solution: Degree of $R(x)$ is of is always less than the degree of divisor. Degree of $R(x)$ will be 0, since, the divisor is of degree 1. Therefore, $R(x)$ is constant = r. When $P(x = a)$, $P(a) = Q(a)(a-a) + r$ that implies $r = P(a)$. So, evaluating $P(x)$ at $x = a$ will give the remainder in $O(n)$.

Let n be degree of $P(x)$.
Time Complexity of the below algorithm is: $O(n)$, since a for loop runs for n iterations.

```
def calculate_remainder(P(x), a):
    r = 0, mul_fact = 1
    for (i = 0...n):
        r = r + coeff(i, P(x)) * mul_fact
        mul_fact = mul_fact * a
    return r
```

Let r be the remainder we obtain from *calculate_remainder()* function.
Let $N(x)$ be polynomial we obtain by $P(x) - r$. Now, $Q(x) = N(x)/(x-a)$. Then, degree of $Q(x)$ will be $n-1$. Let $Q(x) = q_{n-1}x^{n-1} + q_{n-2}x^{n-2} + \ldots + q_1x^1 + q_0x^0$. To find the coefficients of $Q(x)$,

Time Complexity of the below algorithm is: $O(n-1)$, since a for loop runs for maximum of n-2 iterations.

```
def calculate_quotient(N(x), a, r):
    q_{n-1} = coeff(n, N(x))
    for (i = n-2...0):
        q_i = coeff(i+1, N(x)) + q_{i+1} * a
```

Total time complexity is $O(n) + O(n-1) = O(n)$ ■

CSE525 Monsoon 2020 — Ashima Garg (PhD19003)
Homework 4 Problem b — Pramil Panjwani (PhD19008)

I tell you that $P(x)$ is some degree-$(n-1)$ polynomial but do not tell you the polynomial. I also tell you that $P(x_i) = a_i$ for $i = 1 \ldots n$ (all those a_i points are distinct). Your task is to obtain $P(x)$ (rather, its coefficients). Lagrange's formula can be used for interpolation which is stated below.

$$P(x) = \sum_{k=1}^{n} \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

(b) Write a $\Theta(n^2)$ algorithm for interpolation that returns the coefficients of $P(x)$. *Hint: First, get familiar with Lagrange formula with the help of an example. Then, to solve the question, compute* $\prod_{j=1}^{n}(x - x_j)$ *and* $\prod_{j=1}^{n}(x_j - x_k)$ *and then divide them by* $(x - x_k)$ *and* $(x_j - x_k)$ *as required. Figure out the rest.*

Solution: Product function returns for 0 i n coefficient individual of P(x) = P'(x)(x-xn-1)
T(n)=T(n-1)+O(n) , Base case T(1)=1 This implies $T(n) = O(n^2)$. Total Time complexity $= O(n^2) + O(n^2) + K = O(n^2)$

Step 1: Find Numerator product $P = \prod_{j=1}^{n}(x - xj)$

Step 2: Divide P depending on value of K by particular $x[k] P1[k] = P/(x[j] - x[k])$

Step 3: Find $Z1 = Y[k] * 1/(\prod_{j=1, j!=k}^{n}(x - xj))$

Step 4: Find $\sum_{k=1}^{n-1}(P1[k] * Z1)$

```
def Product(X[1...n]):
    n=length(x)
    r = []
    if n == 1
        return (-X[0],1)
    P'[n] = Product(X[1, ..., n-1])
    q=X[n-1]
    P[0]= P'[0]*q
    P[n]= P'[n-1]
    for i=1 to n-1
        P[i]=P'[i-1] - P'[i]*q
    return P

def Interpolation(X[1...n], Y[1...n])
    n=length(x)
    P= Product(X,Y)
    r=[]
    for k=2 to n
        Z1=1     P=P[1,...,k-1,k+1...,n]
        for j=1 to n
            Sum = 0
            if (j! = k)
                Z1 = Z1 * Y[k]/(X[j]-X[K])
            Sum+ = P[j] * Z1
        r[k]=Sum
    return r
```

■

CSE525 Monsoon 2020 Ashima Garg (PhD19003)
Homework 4 Problem c Pramil Panjwani (PhD19008)

Let X and Y be two sets of n integers each. You are also given that each integer is in the range 0 to $10n$. Define the Cartesian sum of X and Y as the following multiset (unlike a set, a multiset can contain an element multiple times).

$$C = A \oplus B = \{x + y \ : \ x \in A, y \in B\}$$

Clearly, the elements in C are integers from 0 to $20n$.

(c) Write a $O(n \log n)$ algorithm to determine the distinct elements of C and the number of times each such element appears in C. *Hint: Represent X and Y as polynomials of degree $10n$. You can use the fact that polynomial multiplication can be solved in $O(n \log n)$ using the FFT algorithm.*

Solution: Let X be a polynomial represented as $X(x) = a_{10n}x^{10n} + a_{10n-1}x^{10n-1} + a_{10n-2}x^{10n-2} + \ldots + a_1x^1 + a_0x^0$. and Y be a polynomial represented as $Y(x) = b_{10n}x^{10n} + b_{10n-1}x^{10n-1} + b_{10n-2}x^{10n-2} + \ldots + b_1x^1 + b_0x^0$.
Let the coefficient of x^i be the number of times element i appears in the multiset. Thus, for an element 10 appearing 2 times in a set is represented by $2x^{10}$. *X_poly*, *Y_poly* are the arrays of length $(10n-1)$ where element at index i represent the count of that element in X and Y respectively
Now, using the FFT algorithm, we can multiply these two polynomials in $O(20nlog(20n))$ time or to be precise it will be $O(MlogM)$ where M is the largest absolute value of an element of $X \cup Y$. *FFTMultiply* in the sub routine that calls *FFT* to multiply the two polynomials and uses the subroutine *InverseFFT* to obtain the final result.
Total Time Complexity is O(20nlog(20n))

def convert_to_polynomial($X[], Y[], n$):
⟨⟨*Coefficients represent the count of the element corresponding to that power.*⟩⟩
  X_poly ← $[0] * 10n$⟨⟨*X_poly contains coefficients $a_{10}, a_{10n-1} \ldots a_1, a_0$ to represent $X(x)$*⟩⟩
  Y_poly ← $[0] * 10n$⟨⟨*Y_poly contains coefficients $b_{10}, b_{10n-1} \ldots b_1, b_0$ to represent $Y(x)$*⟩⟩
  for($i = 1 \ldots n$): ⟨⟨*Time complexity of this loop is $O(n)$*⟩⟩
    $X\_poly[i] \leftarrow X\_poly[X[i]] + 1$
    $Y\_poly[i] \leftarrow Y\_poly[Y[i]] + 1$
  return X_poly, Y_poly

def InverseFFT($result'[0 \ldots size-1]$): ⟨⟨*size will be 20n*⟩⟩
  $result[0 \ldots size-1] \leftarrow FFT$(result')⟨⟨*Time complexity is $O(20nlog(20n))$*⟩⟩
  for ($i = 0 \ldots size-1$): ⟨⟨*Time Complexity is $O(n)$*⟩⟩
    result*[i] ←$\overline{result'[i]}/(size)$
  return $result[0 \ldots size-1]$

def FFTmultiply($X\_poly[0 \ldots 10n-1], Y\_poly[0 \ldots 10n-1]$):
  $X\_poly' \leftarrow$ FFT(X_poly) ⟨⟨*Not implemented FFT. Time complexity of FFT is $O(20nlog(20n))$*⟩⟩
  $Y\_poly' \leftarrow$ FFT(Y_poly) ⟨⟨*Not implemented FFT. Time complexity of FFT is $O(20nlog(20n))$*⟩⟩
  for($j = 0 \ldots 20n-1$): ⟨⟨*Time Complexity is $O(20n)$*⟩⟩
    $result' \leftarrow X\_poly' * Y\_poly'$
  return InverseFFT($result'$) ⟨⟨*Time Complexity is $(20nlog(20n))$*⟩⟩

def main($X[], Y[], n$):
  $X\_poly, Y\_poly \leftarrow$ convert_to_polynomial($X[], Y[], n$) ⟨⟨*O(n)*⟩⟩
  $result \leftarrow$FFTmultiply($X\_poly[0 \ldots 10n-1], Y\_poly[0 \ldots 10n-1]$) ⟨⟨*O(20nlog(20n))*⟩⟩
  $final\_elements \leftarrow []$
  for($i = 0 \ldots (20n-1)$): ⟨⟨*O(20n)*⟩⟩
    if $result[i] != 0$:
      add element i to final_elements array.
      $result[i]$ represents the count of that element in the resultant array.

■