Problem Solving: Homework 3.9

Name: Chen Shaoyuan Student ID: 161240004

November 16, 2017

1 [TC] Problem 30.1-2

Let $A(x) = \sum_{i=0}^{n-1} a_i x^i$, $q(x) = \sum_{i=0}^{n-2} b_i x^i$, then we have

$$\begin{aligned} A(x) &= q(x)(x - x_0) + r \\ &= -x_0 b_0 + r + \sum_{i=1}^{n-2} (b_{i-1} - x_0 b_i) x^i + b_{n-2} x^{n-1} \\ &= a_0 + \sum_{i=1}^{n-2} a_i x^i + a_{n-1} x^{n-1} \end{aligned}$$

By comparing the coefficients, we get the equations for computing the remainder r and the coefficients of $q(x)$:

$$\begin{aligned} b_{n-2} &= a_{n-1} \\ b_{n-3} &= x_0 b_{n-2} + a_{n-2} \\ &\cdots\cdots \\ b_1 &= x_0 b_2 + a_2 \\ b_0 &= x_0 b_1 + a_1 \\ r &= x_0 b_0 + a_0 \end{aligned}$$

2 [TC] Problem 30.1-4

We can get one linear equation by substituting one point into the polynomial, and we can get less than n linear equations in total. However, a polynomial of degree-bound n has n coefficients, and by linear algebra we know the equations have more than one solution. Hence, the polynomial can not be uniquely determined, which means n distinct point-value pairs are necessary.

3 [TC] Problem 30.1-5

We can precalculate $P(x) = \prod_j (x - x_j)$ in $O(n^2)$ time, since multiplying each term can be regarded as adding two polynomials, which takes $O(n)$ time, and there are n terms in total. Then, we compute each term of the summation. For the numerator, $\prod_{j \neq k} (x - x_j) = P(x)/(x - x_k)$, which can be calculated in $O(n)$ time by using the method mentioned in Problem 30.1-2. The denominator is just the product of n numbers, which can be easily calculated in $O(n)$ time. Hence all these n terms can be calculated in $O(n^2)$ time. Finally, we sum up all the terms in the summation. Each term is a polynomial of degree-bound n and there are n terms in all, the summation can be done in $O(n^2)$ time. Therefore, the interpolation can be done in $O(n^2)$ time using equation (30.5).

4 [TC] Problem 30.2-1

By cancellation lemma,

$$\omega_n^{n/2} = \omega_{2n}^n = \omega_2^1 = \omega_2 = -1$$

5 [TC] Problem 30.2-4

IFFT(a)
1 $n = a.length$
2 $y =$ RECURSIVE-IFFT(a)
3 **for** $i = 0$ **to** $n - 1$
4 $\quad y_i = y_i / n$
5 **return** y

RECURSIVE-IFFT(a)
$\cdots\cdots$
4 $\omega_n = e^{-2\pi i/n}$
$\cdots\cdots$
8 $y^{[0]} =$ RECURSIVE-IFFT$(a^{[0]})$
9 $y^{[1]} =$ RECURSIVE-IFFT$(a^{[1]})$
$\cdots\cdots$

The omitted parts of the procedure are exactly the same as those in RECURSIVE-FFT.

6 [TC] Problem 30.2-5

For polynomial $A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$, we define these three new polynomials:

$$\begin{aligned} A^{[0]}(x) &= a_0 + a_3 x + a_6 x^2 + \cdots + a_{n-3} x^{n/3-1} \\ A^{[1]}(x) &= a_1 + a_4 x + a_7 x^2 + \cdots + a_{n-2} x^{n/3-1} \\ A^{[2]}(x) &= a_2 + a_5 x + a_8 x^2 + \cdots + a_{n-1} x^{n/3-1} \end{aligned}$$

and

$$A(x) = A^{[0]}(x^3) + xA^{[1]}(x^3) + x^2A^{[2]}(x^3)$$

Similarly to the halving lemma, we have $(\omega_n^{k+n/3})^3 = (\omega_n^{k+2n/3})^3 = (\omega_n^k)^3$, so we divide the problem of size n to three subproblems of size $n/3$. The recurrence is

$$f(n) = 3f(n/3) + O(n)$$

By the master theorem, $f(n) = n\log n$.

7 [TC] Problem 30.2-7

The polynomial can be written as

$$P(x) = \prod_{i=0}^{n-1}(x - z_i)$$

We can compute these multiplications by divide and conquer: we divide the multiplications into to parts, and calculate the two parts recursively, then multiply the two parts by FFT. The recurrence is

$$f(n) = 2f(n/2) + O(n\log n)$$

We claim that $f(n) = O(n\log^2 n)$, i.e. $f(n) \le kn\log^2 n$ for all n, and we are going to prove this by mathematical induction.

For the base step, we can take some k large enough such that $f(2) \le 2k\log^2 2$. For the induction step, assume that $f(n_0) \le kn_0\log^2 n_0$ for all $n_0 < n$, then we have

$$\begin{aligned} f(n) &\le 2kn/2\log^2(n/2) + O(n\log n) \\ &= kn(\log n - 1)^2 + O(n\log n) \\ &\le kn\log^2 n \end{aligned}$$

The last inequality holds only if we take some k large enough. By mathematical induction, we get that $f(n) = O(n\log^2 n)$, and thus we can find the coefficients in $O(n\log^2 n)$ time.

8 [TC] Problem 30.2-8

We have two ways to compute the inverse FFT. The first one is to replace ω_n by ω_n^{-1}, then divide each term of the result by n. The second one is to run ITERATIVE-FFT reversely. DFT can be viewed as the inverse of the inverse DFT, and we use the two different ways to compute the two inverses. The pseudocode is shown below.

```
ITERATIVE-FFT'(a)
 1  n = a.length
 2  for s = log_2 n downto 1
 3      m = 2^s
 4      ω_m = e^{-2πi/m}
 5      for k = n - 1 to 0 by -m
 6          ω = 1
 7          for j = m/2 - 1 downto 0
 8              ω = ω/ω_m
 9              u = (a_{k+j} + a_{k+j+m/2})/2
10              v = (a_{k+j} - a_{k+j+m/2})/2
11              a_{k+j} = u
12              a_{k+j+m/2} = t/ω
13  BIT-REVERSE-COPY(a, A)
14  return A
```

9 [TC] Problem 30-1

a. Let $u = ac$, $v = bd$, $w = (a+b)(c+d)$ be three multiplications. Then $(ax+b)(cx+d) = acx^2 + (ad+bc)x + bd = ux^2 + (w-u-v)x + v$.

b. Assume n is some power of 2. In the first algorithm, for polynomial A, define

$$A_L = a_0 + a_1x + a_2x^2 + \cdots + a_{n/2-1}x^{n/2}$$
$$A_H = a_{n/2} + a_{n/2+1}x + a_{n/2+2}x^2 + \cdots + a_{n-1}x^{n/2}$$

then we have

$$A(x)B(x) = (A_L + A_Hx^{n/2})(B_L + B_Hx^{n/2})$$

Let $U = A_HB_H$, $V = A_LB_L$, $W(x) = (A_L + A_H)(B_L + B_H)$, we have

$$A(x)B(x) = Ux^n + (W - U - V)x^{n/2} + V$$

By applying the equation recursively, we can compute $A(x)B(x)$. The recurrence of running time is

$$f(n) = 3f(n/2) + O(n)$$

By the master theorem, $f(n) = O(n^{\log_2 3})$. In the second algorithm, for polynomial A, define

$$A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}$$
$$A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}$$

then we have

$$A(x)B(x) = (A^{[0]}(x^2) + xA^{[1]}(x^2))(B^{[0]}(x^2) + xB^{[1]}(x^2))$$

Let $U = A^{[0]}(x^2)B^{[0]}(x^2)$, $V = A^{[1]}(x^2)B^{[1]}(x^2)$, $W = (A^{[0]}(x^2) + A^{[1]}(x^2))(B^{[0]}(x^2) + B^{[1]}(x^2))$, we have

$$A(x)B(x) = Ux^2 + (W - U - V)x + V$$

Similarly, we can compute $A(x)B(x)$ by recursively applying this equation, and the running time is still $O(n^{\log_2 3})$.

c. Two n-bit integers can be added or subtracted in $O(n)$ time, using add with carry (or subtract with borrow) from lower bit to higher bit. To avoid discussion of trivial details, let n be some power of 2, by padding with zeros. Let A, B be two n-bit integers, and A_L, B_L (or A_H, B_H) be their lower halves (or higher halves). Let $U = A_L B_L$, $V = A_H B_H$, $W = (A_L + A_H)(B_L + B_H)$, then

$$AB = U2^n + (W - U - V)2^{n/2} + V$$

The product consists three parts, corresponding to 1 to $n/2-1$, $n/2$ to $n-1$, n to $2n-1$ bits, respectively, yet some parts may consist more than $n/2$ bits. We only have to add the bits higher than $n/2$-th bit to the next part, which takes only $O(n)$ time. The total running time of the divide-and-conquer algorithm is $O(n^{\log_2 3})$.