Name: PEIGUO GUAN

zID: z5143964

1.[20 marks] You are given two polynomials, $P_A(x) = A_0 + A_3x^3 + A_6x^6$ and $P_B(x)=B_0 +B_3x^3 +B_6x^6 +B_9x^9$ where all A_i's and all B_j's are large numbers. Multiply these two polynomials using only 6 large number multiplications

Answer:

The degree of $P_A(x)$ is 6, and the $P_B(x)$= is 9, so the degree of $P_A(x)P_B(x)$ is 15 and it needs 16 multiplications.

Let $y = x^3$, and then $P_A(y) = A_0 + A_3y + A_6y^2$, $P_B(y) = B_0 +B_3y +B_6y^2 +B_9y^3$ and $R_A(y) = P_A(y) P_B(y)$, it has degree of 5 and it needs 6 multiplications.

$R_A(y)=P_A(y)P_B(y)=A_0B_0+(A_0B_3+A_3B_0)y+(A_0B_6+A_3B_3+A_6B_0)y^2+(A_0B_9+A_3B_6+A_6B_3)y^3+(A_3B_9+A_6B_6) y^4+ A_6B_9y^5$

2. (a) [5 marks] Multiply two complex numbers (a + ib) and (c + id) (where a, b, c, d are all real numbers) using only 3 real number multiplications.

(a) [5 marks] Find $(a + ib)^2$ using only two multiplications of real numbers.

(b) [10 marks] Find the product $(a + ib)^2(c + id)^2$ using only five real number multiplications.

Answer:

(a) Here (a + ib)(c + id) = ac+i(ad+bc)-bd
 (1) ac
 (2) bd
 (3) (a+b)(c+d) = ac+(ad+bc)+bd, which miners (1) and (2) to get (ad+bc)

So, we just need 3 real number multiplications.

(b) Here $(a + ib)^2 = a^2 – b^2 + 2iab$

(1)$(a+b)(a-b) = a^2 – b^2$
(2)2ab
So, it just need 2 real number multiplications.

(c)Here $(a + ib)^2(c + id)^2=[(a + ib)(c + id)]^2$, and we know from question(a) that, $(a + ib)(c + id) = ac+i(ad+bc)-bd=(ac-bd)+i(ad+bc)$, we let $X = ac-bd$, and $Y = (ad+bc)$, so it can be seen like $X+iY$, and the $[(a + ib)(c + id)]^2$ can be seen as $[X+iY]^2$, we can know from the qu estion(b) that $(a + ib)^2$ need 2 real number multiplications, and from question(a) we can know that problem $(a + ib)(c + id)$ need 3 real number multiplications, so in total, $(a + ib)^2(c + id)^2=[(a + ib)(c + id)]^2$ need 3+2 = 5 multiplications.

3. (a) [2 marks] Revision: Describe how to multiply two n-degree polynomials together in O(nlogn) time, using the Fast Fourier Transform (FFT). You do not need to explain how FFT works – you may treat it as a black box.

(b) In this part we will use the Fast Fourier Transform (FFT) algorithm described in class to multiply multiple polynomials together (not just two). Suppose you have K polynomials $P_1,...,P_K$ so that

$$\text{degree}(P_1) + \cdots + \text{degree}(P_K) = S$$

. (i) [6marks]Show that you can find the product of these K polynomials in O(KSlogS) time. Hint: How many points do you need to uniquely determine an S-degree polynomial?

. (ii) [12marks]Show that you can find the product of these K polynomials in O(SlogSlogK) time. Hint: consider using divide-and-conquer; a tree which you used in the previous assignment might be helpful here as well. Also, remember that if x,y,z are all positive, then $\log(x + y) < \log(x + y + z)$

Answer:

(a): Describe: Let two n degree polynomials as P and Q, PQ has degree at 2n, and it needs 2n+1 distinct points to uniquely determine it.

Using FFT to evaluate P and Q, we spend time complexity $O(NlogN)$to find the value and change the coefficient form to value form. Then, multiply value of P and Q at each point which is the result we need, it takes time complexity of $O(N)$. At last we use IFFT to convert PQ from value form to coefficient form which takes $O(NlogN)$ time.

(b)(i): The total degree is S, so we need S+1 multiplications to determine the S-degree polynomials. $R(x) = P_1(x) \cdot P_2(x) \ldots P_k(x)$. Firstly, we start from k = 1, $Q(k = 1) = P_1(x)$, $Q(k = 2) = Q(1)P_2(x)$ and then until $Q(k+1) = Q(k)P_2(x)$, each part of R(x) is less than degree S. And then using FFT in each multiplication which cost time complexity $O(SlogS)$. We need K such multiplications from 1 to k, so the total time complexity is $O(KSlogS)$.

(b)(ii): We can recursively divide this problem by 2(just like binary tree), and until $2^m = K$ leaves, then $m = \log K$. However, when it $2^m < K$ and we need to built a perfect binary tree, so we need to add 2 nodes to K-2^m nodes in each of the leftmost leaves. By this way we have exactly K leaves in this tree. In each level the node of their children have two degree d1 and d2(from 1 to logK level the sum of their degree are all equal to S, and the node of the degree is d1+d2). We can recursively use FFT to evaluate node of two children to get the node value which cost(O(d1+d2)log(d1+d2)). Since we have logK levels and in each level the sum of the degree is S add every nodes up to get upper node, so it cost (d1+d2)log(d1+d2) = SlogS totally in each level, so the time complexity of the whole algorithm is SlogS*logK = SlogSlogK.

4. Let us define the Fibonacci numbers as $F_0 = 0$, $F_1 =1$and $F_n = F_{n-1}+F_{n-2}$ for all n≥2. Thus, the Fibonacci sequence looks as follows: 0, 1, 1, 2, 3, 5, 8, 13, 21, . . .

   (a) [5 marks] Show, by induction or otherwise, that

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

   for all integers n ≥ 1.

   (b) [15 marks] Hence or otherwise, give an algorithm that finds F_n in O(log n) time.

Answer:

(a): n>=1, so the base case if n =1, then:

$$\begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1, F_0 = 0, F_1 = 1, F_2 = 1$$

let n = k: $\begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k$

let n= k+1: $\begin{pmatrix} F_{k+2} & F_{k+1} \\ F_{k+1} & F_k \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k+1} = \begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix} *$
$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_{k+1} + F_k & F_{k+1} \\ F_k + F_{k-1} & F_k \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k * \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k+1}$

$$F_{k+2} = F_{k+1} + F_k, F_{k+1} = F_{k+1}, F_{k+1} = F_k + F_{k-1}, F_k = F_k$$

(b): From the above, we assume K=$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, we can use divide and conquer to solve it.

Firstly, we can check the number of n, and we need to get K^n: if n is even, then we can recursively compute $K^{\frac{n}{2}}$ and square it. If n is odd then we recursively compute $K^{\frac{n-1}{2}}$, square it and multiply by one more K.

And we can compute every steps in O(1).

Since there are O(log n) steps of the recursions, so the time complexity of algorithm is O(logn)

5. Your army consists of a line of N giants, each with a certain height. You must designate precisely L ≤ N of them to be leaders. Leaders must be spaced out across the line; specifically, every pair of leaders must have at least K ≥ 0 giants standing in between them. Given N,L,K and the heights H[1..N] of the giants in the order that they stand in the line as input, find the maximum height of the shortest leader among all valid choices of L leaders. We call this the optimisation version of the problem.

   For instance, suppose N = 10, L = 3,K = 2 and H = [1,10,4,2,3,7,12,8,7,2]. Then among the 10 giants, you must choose 3 leaders so that each pair of leaders has at least 2 giants standing in between them. The best choice of leaders has heights 10, 7 and 7, with the shortest leader having height 7. This is the best possible for this case.

   (a) [8 marks] In the decision version of this problem, we are given an additional integer T as input. Our task is to decide if there exists some valid choice of leaders satisfying the constraints whose shortest leader has height no less than T. Give an algorithm that solves the decision version of this problem in O(N) time.

   (b) [12 marks] Hence, show that you can solve the optimisation version of this problem in O(N logN) time.

   Answer:

   (a): Assume the height of the giant that no less than T called *valid giant*.

   In this problem, we can just look through the giants list from the first element and find the first valid giant as root number, and then skip to the next K giants to find the second valid giant and go on(for example: if we find the first valid giant, then skip to next K giant, from here we look for the first valid giant and repeat). By this way, if find the L number of valid giant, it means it has valid choices, otherwise, there is not this valid choice. We can see that if the process of all the value that we have been through is equal or larger than T then we satisfy the question and find the valid choice, otherwise there is not such choice. It is obviously to see that the algorithm is O(N).

(b):In this problem, we can solve this problem as find the max T in the giant list.

Firstly, we can sort out the height number list by merge sort in O(nlogn). We need to use the method that we use in question(a), if there is valid choice with input T then return true, otherwise return false. Then we can use binary search to find the max T in the sorted giant height number list as input, if the result is true try to input higher number and deciding whether to go higher or lower base on the problem. It takes O(logN) in binary search, and the method in question(a) to check the T cost O(N) then the algorithm we use cost O(NlogN) in total.