

## CSE 361 Complexity of Sequential and Parallel Algorithms

Spring 2008; Homework3 Solutions

1. The coefficients of the polynomial are given by  $\binom{n}{i}$ ,  $i = 0, 1, \dots, n$ .

Since  $\binom{n}{i} = \binom{n}{i-1} (n-i+1)/i$ , the coefficients can be computed in time  $O(n)$ .

FFT can be used to multiply two  $n$ th degree polynomials in  $O(n \log n)$  time. We can compute the coefficients of  $(1+x)^n$  by multiplying  $(1+x)^{n/2}$  and  $(1+x)^{n/2}$ . If  $T(n)$  is the time needed to compute  $(1+x)^n$ , then,  $T(n) = 2T(n/2) + O(n \log n)$ , which solves to  $O(n \log^2 n)$ .

2. Let  $A$  be a *Toeplitz* matrix and  $B$  be an  $n \times 1$  vector. Let's consider the multiplication of the lower triangular part of  $A$  (including the main diagonal elements) with  $B$ .

Let the elements of  $A$  be the following:

$$a_{n,n} = a_{n-1,n-1} = a_{n-2,n-2} = \dots = a_{2,2} = a_{1,1} = a_1$$

$$a_{n,n-1} = a_{n-1,n-2} = a_{n-2,n-3} = \dots = a_{2,1} = a_2$$

$$a_{n,n-2} = a_{n-1,n-3} = a_{n-2,n-4} = \dots = a_{3,1} = a_3$$

$\vdots$

$$a_{n,1} = a_n$$

Let the elements of  $B$  be the following:

$$b_{1,1} = b_1, b_{2,1} = b_2, \dots, b_{n,1} = b_n$$

Multiplication of the lower triangular part of  $A$  with  $B$  gives the following:

$$\begin{bmatrix} a_1 b_1 \\ a_2 b_1 + a_1 b_2 \\ a_3 b_1 + a_2 b_2 + a_1 b_3 \\ \vdots \end{bmatrix}$$

We can notice that the above is nothing but the multiplication of two polynomials  $(a_1 + a_2x + a_3x^2 + \dots)$  and  $(b_1 + b_2x + b_3x^2 + \dots)$ .

Since the polynomials can be multiplied in  $O(n \log n)$  time, the matrices can also be multiplied in  $O(n \log n)$  time. Multiplication of the upper triangular elements of  $A$  with  $B$  is symmetrical to the above and it would not affect the asymptotic complexity.

3. Using Taylor series expansion for  $f(\cdot)$ ,

$$f(a+x) = f(a) + x f^1(a) + \frac{x^2}{2!} f^2(a) + \dots + \frac{x^n}{n!} f^n(a)$$

where  $f^i(x)$  stands for the  $i$ th derivative of  $f(x)$ . Let  $F(x)$  denote  $f(a+x)$ . Evaluate  $F(x)$  at the  $n$ th roots of unity. This can be done in  $O(n \log^2 n)$  time as was mentioned in class (see Section 9.5 in the text – An  $n$ th degree polynomial can be evaluated at  $n$  arbitrary points in  $O(n \log^2 n)$  time). Then, use inverse FFT to compute the coefficients of  $F(x)$ . This can be done in  $O(n \log n)$  time. Once the coefficients of  $F(x)$  are known, it is easy to determine the derivatives.

Run time =  $O(n \log^2 n) + O(n \log n) + O(n) = O(n \log^2 n)$ .

**Another solution:** Let  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ . Then,  $f'(x) = n a_n x^{n-1} +$

$(n-1)a_{n-1}x^{n-2} + \dots + 2a_2x + a_1$ ;  $f''(x) = n(n-1)a_nx^{n-2} + (n-1)(n-2)a_{n-1}x^{n-3} + \dots + 2a_2$ ;  $f'''(x) = n(n-1)(n-2)a_nx^{n-3} + (n-1)(n-2)(n-3)a_{n-1}x^{n-4} + \dots + 6a_3$ ; and so on. Now consider the following two polynomials:  $A(x) = (n!a_n)x^{n-1} + ((n-1)!a_{n-1})x^{n-2} + \dots + (2!)a_2x + a_1$  and  $B(x) = \frac{a^{n-1}}{(n-1)!} + \frac{a^{n-2}}{(n-2)!}x + \dots + \frac{a^2}{2!}x^{n-3} + ax^{n-2} + x^{n-1}$ . Compute the product of  $A(x)$  and  $B(x)$  in  $O(n \log n)$  time. We can obtain the required derivatives from the coefficients of this product. The total run time is  $O(n \log n)$ . However note that the coefficients of  $A(x)$  could become very large creating practical difficulties.

4. Let  $\text{KnapD}(P)$  be an algorithm for solving the knapsack decision problem,  $P$  being the target profit. In the optimization version of the problem, we are required to find the maximum profit. We know that the maximum profit is in the interval  $[0, n2^q]$ . Use binary search to identify this maximum profit. In particular check if the answer to  $\text{Knap}(n2^q/2)$  is *yes* or *no*. If the answer is *yes*, restrict the future search to  $[n2^q/2 + 1, n2^q]$ . If not, search in the interval  $[0, n2^q/2]$ . The number of calls made to  $\text{KnapD}(P)$  is no more than  $\log n + q = O(q)$ .
5. Let the transformation from  $\pi_1$  to  $\pi_2$  take time  $p(n)$ , for some polynomial  $p()$ . If  $n$  is the input size to  $\pi_1$ , then the input size to  $\pi_2$  can be as large as  $p(n)$ . Thus the solution of  $\pi_2$  on the transformed input can take time  $\Omega(2\sqrt{p(n)})$ . This time can be asymptotically larger than  $2\sqrt{n}$ . For example, if  $p(n) = n^4$ , the run time is  $2n^2$ .
6. Let  $G(V, E), k$  be any instance of the NCDP. Let  $|V| = n$ . Create the following instance of the set cover decision problem:  $F = \{S_1, S_2, \dots, S_n\}, k$ , where  $S_i$  is the set of edges incident on node  $i$ , for  $1 \leq i \leq n$ . Clearly, the union of all the sets in  $F$  is equal to  $E$ . If there are  $k$  sets in  $F$  whose union equals  $E$ , then the corresponding nodes will form a cover for  $G$ . Also, if  $G$  has a node cover of size  $k$ , then the union of the corresponding  $S_i$ 's will equal  $E$ .
7. It was shown in class that the maximum of  $n$  elements can be found in  $O(1)$  time using  $n^2$  common CRCW PRAM processors.

Consider the case when  $\epsilon = \frac{1}{2}$ . Divide the elements into groups of size  $\sqrt{n}$ . Assign the first  $\sqrt{n}$  elements to the first  $n$  processors and the second  $\sqrt{n}$  elements to the next  $n$  processors and so on. The maximum element in each group can be found in  $O(1)$  time. At this stage, we have  $\sqrt{n}$  elements and  $n\sqrt{n}$  processors. Hence, the maximum of these elements can be found in  $O(1)$  time. Total time =  $O(1)$ .

Next, consider the case when  $\epsilon = \frac{1}{3}$ . Here, divide the elements into groups of size  $n^{1/3}$ . Assign the first  $n^{1/3}$  elements to the first  $n^{2/3}$  processors and the second  $n^{1/3}$  elements to the next  $n^{2/3}$  processors and so on. The maximum element of each group can be found in  $O(1)$  time and using  $n^{4/3}$  processors the maximum of these maximum elements can be found in  $O(1)$  time.

For the general case, partition the input into groups with  $n^\epsilon$  elements in each group. Find the maximum of each group assigning  $n^{2\epsilon}$  processors to each group. This takes  $O(1)$  time.

Now the problem reduces to finding the maximum of  $n^{1-\epsilon}$  elements. Again, partition the elements with  $n^\epsilon$  elements in each group and find the maximum of each group. There will be only  $n^{1-2\epsilon}$  elements left. Proceed in a similar fashion until the number of remaining elements is  $\leq \sqrt{n}$ . The maximum of these can be found in  $O(1)$  time. Clearly, the run time of this algorithm is  $O(1/\epsilon)$ . This will be a constant if  $\epsilon$  is a constant.

8. Let  $A$  and  $B$  be the matrices to be multiplied and let  $C = A \times B$ .  $C[i][j] = \sum_{k=1}^n A[i][k]B[k][j]$ . Assign  $\frac{n}{\log n}$  processors to compute each of the  $C[i][j]$ 's. First multiply  $A[i][k]$  with  $B[k][j]$  for  $k = 1, 2, \dots, n$ . This can be done in one unit of time using  $n$  processors, or equivalently, in  $\log n$  time using  $\frac{n}{\log n}$  processors (applying the slow-down lemma). Now add these  $n$  products using the prefix computation algorithm. This takes  $O(\log n)$  time using  $\frac{n}{\log n}$  processors. Thus each of the  $C[i][j]$ 's can be computed in  $O(\log n)$  time using  $\frac{n}{\log n}$  processors.
9. Let  $t = t_1 t_2 \cdots t_n$  and  $p = p_1 p_2 \cdots p_m$ . Assign  $m$  processors to each symbol in  $t$ . Processors associated with the symbol  $t_i$  compare  $t_i t_{i+1} \cdots t_{i+m-1}$  with  $p$  to check if there is a match starting from  $t_i$ . Clearly this can be done in  $O(1)$  time using the common CRCW PRAM model. Thus the whole algorithm runs in  $O(1)$  time using  $mn$  processors.