

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. Let the length of text  $t$  be denoted by  $|t|$  and the length of pattern  $p$  be denoted by  $|p|$ . Let  $t = t_1, t_2 \dots t_{|t|}$ ,  $p = p_1, p_2 \dots p_{|p|}$ . Let  $t(i, |p| + i - 1) = t_i, t_{i+1} \dots t_{|p|+i-2}, t_{|p|+i-1}$  denote a substring of  $t$  of length  $|p|$  starting at the  $i^{th}$  position. It is possible to check in  $O(1)$  time using  $|p|$  processors on CRCW PRAM whether pattern  $p$  matches  $t(i, |p| + i - 1)$  for any fixed  $i$ , as follows. Every processor  $j$ ,  $1 \leq j \leq |p|$  reads  $p_j$  and  $t_{i+j-1}$ , compares them and if they are not equal writes '1' to memory location  $b[i]$  (initially set to zero). Strings match iff  $b[i] = 0$  after this step is done.

Since there are  $|t| - |p|$  possible positions  $i$ , computing all  $b[i]$  (in  $O(1)$  time) as described above requires  $|p|(|t| - |p|)$  processors. Once all the  $b[i]$  are computed, they determine all occurrences of  $p$  in  $t$  as follows: if  $b[i] = 0$  then  $p$  occurs in  $t$  starting at  $i^{th}$  position.

If  $|t| \leq n$  and  $|p| \leq n$  then this algorithm takes  $O(1)$  time on an  $n^2$ -processor CRCW PRAM.

4. Let  $k_1, k_2, \dots, k_n$  be the elements. Divide the elements into groups of size  $\log n$ . Assign the first  $\log n$  elements to the first processor and the second  $\log n$  elements to the second processor and so on.

Create a temporary array,  $A$ , of size  $n$ .

Step 1: At each processor do: compare all the elements assigned to that processor with  $x$  sequentially. If an element  $k_i$  is less than or equal to  $x$ , place 1 in the array  $A$  at index  $i$  else place a 0 at  $A_i$ . Time =  $O(\log n)$ .

Step 2: Compute the prefix sums of the elements of the array  $A$  using all the  $\frac{n}{\log n}$  processors. Complexity =  $O(\log n)$ .

Let us assume that the rearranged elements will be placed in an array  $B$ .

Step 3: Move the elements that are less than or equal to  $x$  into  $B$  first.

For each element  $k_i$  with a 1 in  $A_i$ , the value of the corresponding prefix sum gives the index in  $B$  where that element can be placed.

At each processor do: let  $p$  be the prefix sum of an element  $k_i$  with a 1 in  $A_i$ . Move  $k_i$  into  $B[p]$ . Time =  $O(\log n)$  (there are only  $\log n$  elements at each processor).

Step 3: Find the maximum prefix ( $maxPrefix$ ) of any element with a 1 in  $A_i$  in step 2.

Step 4: Now, the elements that are greater than  $x$  can be copied into  $B$  starting from  $B[maxPrefix + 1]$ .

Flip all the elements of  $A$  and repeat the procedure in steps 2 and 3 with the following difference: if  $p$  is the prefix sum of an element  $k_i$ , move  $k_i$  into  $B[p + maxPrefix]$ . Complexity =  $O(\log n)$ .

Total complexity =  $O(\log n)$ .

5. We know that  $\pi_1$  polynomially reduces to  $\pi_2$ . Let  $x$  be an instance of  $\pi_1$  with  $|x| = n$ . We can convert this into an instance  $x'$  of  $\pi_2$  in  $O(n^c)$  time (for some constant  $c$ ). Note that  $c$  could be any constant (10, for instance) and we can only say that  $|x'| = O(n^c)$  and in fact  $|x'|$  could be  $\Omega(n^c)$ . If  $|x'|$  is  $\Omega(n^c)$ , the run time needed for solving  $x'$  will be  $O(2^{\sqrt{\Omega(n^c)}})$  which can be asymptotically greater than  $2^{\sqrt{n}}$ . Thus the given statement is not correct.
6. Use the following algorithm, Size(Graph  $G$ ) -

```
for  $i := |V|$  to 0 do
  if CLQ( $i$ ) = yes then
    output  $i$ 
    quit
end
```

Note that we increase the runtime of the CLQ algorithm, by a factor of  $|V|$ , yet maintaining it polynomial.