e-PG PATHSHALA- Computer Science
Design and Analysis of Algorithms
Module 13
Component-I (A) - Personal Details

Role	Name	Designation
Principal Investigator	Dr.T.V.Geetha	Senior Professor, Department of Computer Science &Engineering, Anna University, Chennai
Content Writer (CW)	Dr.S.Sridhar	Associate Professor Department of Information Science and Technology, Anna University, Chennai
Content Reviewer (CR)	Dr.K.S.Easwarakumar	Professor Department of Computer Science &Engineering, Anna University, Chennai
Language Editor (LE)		

e-PG PATHSHALA- Computer Science
Design and Analysis of Algorithms
Module 13

Component-I (B) Description of Module

Items	Description of Module
Subject Name	Computer Science
Paper Name	Design and Analysis of Algorithms
Module Name/Title	**More Applications of Divide and Conquer**
Module Id	CS/DAA/13
Pre-requisites	None
Objectives	To understand algorithms to find maximum/minimum, Tiling problem, Fourier Transform and its usage for

	polynomial multiplication.
Keywords	**Divide and conquer, Max and Min, Tiling Problem, FFT, Polynomial Multiplication**

Module 13: More Applications of Divide and Conquer

This module 13 focuses on three important algorithms of divide and conquer. One is to find maximum and minimum in an array of numbers, Algorithm to solve Tiling problem of a defective chessboardand Fast Fourier Transform algorithm and its usage for finding polynomial multiplication. The Learning objectives of this module are as follows:

- To **find minimum and maximum in an array using divide and conquer**
- **To understand the use of divide and conquer for Tiling problem**
- **To understand implement Fast Fourier Transform**
- **To implement polynomial multiplication problem**

What is divide and Conquer design paradigm?

Divide and conquer is a design paradigm. It involves the following three components:

Step 1: (Divide) The problem is divided into subproblems. It must be noted that the subproblems are similar to the original problem but smaller in size.

Step 2: (Conquer) after division of the original problem into subproblems, the subproblems are solved iteratively or recursively. If the problems are small enough, then they are solved in a straightforward manner.

Step 3: (Combine) Then, the solutions of the subproblems are combined to create a solution to the original problem

Finding Maximum and Minimum Elements

Finding maximum and minimum of an array is one of the most commonly used routine in many applications. Maximum and minimum are called order statistics.

The conventional algorithm for finding the maximum and minimum elements in a given array is given as follows:

1. Set largest = A[1]
2. Set index = 2 and N = length(A)
3. While (index <= N) do
   if A[index] > largest then
   largest = A[index]
4. Print the largest
5. End.

Complexity Analysis:

It can be observed that the conventional algorithm requires 2n-2 comparisons for finding maximum and minimum in an array. Therefore, the complexity of the algorithm is O(n).

Idea of Divide and Conquer Approach

One can use the divide-and-conquer strategy to improve the performance of the algorithm. The idea is to divide the array into subarrays and to find recursively the maximum and minimum elements of the subarrays. Then, the results can be combined by comparing the maximum and minimum of the subarray to find the global maximum and minimum of an array.

To illustrate this concept, let us assume that the given problem is to find the maximum and minimum of an array that has 100 elements. The idea of divide and conquer is to divide the array into two subarrays

of fifty elements each. Then the maximum element in each group is obtained recursively or iteratively. Then, the maximum of each group can be computed to determine the overall maximum.

This logic can be repeated for find minimum also.

Informal Algorithm

This idea can be generalized to an informal algorithm as follows:

1. Divide the n elements into 2 groups A and B with floor(n/2) and ceil(n/2) elements, respectively.

2. Find the min and max of each group recursively.

3. Overall min is min{min(A), min(B)}.

4. Overall max is max{max(A), max(B)}.

This idea is illustrated in the following Example 1 and Example 2.

Example 1: Find the maximum of an array {2,5,8,1,3,10,6,7} using the idea of divide and conquer.

Solution: The idea is to split the above array into subarrays A and B such that

A = {2,5,8,1} and

B = {3,10,6,7}

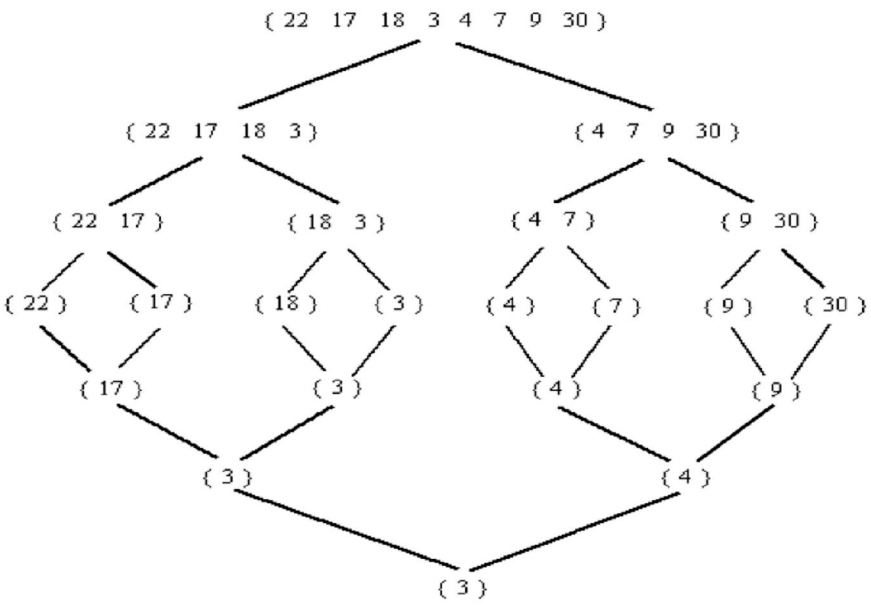
The idea can be repeated to split subarrays A and B further. Then, it can be found that

max(A) = 8, max(B) = 10.

Therefore, the maximum of the array is - max{max(A), max(B)} = 10.

Example 2: Find the minimum of the array {22,17,18,3,4,7,9,30} using divide and conquer idea?

Solution: The idea of the previous problem can be repeated. This results in the following Fig. 1.



∴ The minimum element is 3.

Fig 1: Finding Minimum in an array

Formal Algorithm

The formal algorithm based on [1] is given as follows:

```
Algorithm minimummaximum A(i,j)

Begin

  mid = floor of (i + j) / 2

  [max, min] = minimummaximum(A[i,mid])

  [max1,min1] = minumummaximum(A[mid+1,j])

  globalmax = max(max,max1)

  globalmin = min(min,min1)

End
```


It can be observed that, this algorithm formally divides the given array into two subarrays. The subarrays are subdivided further if necessary. It can be observed that only the maximum and minimum elements of the subarrays are compared to get the maximum/minimum element of the parent list.

Complexity Analysis of Finding Maximum/Minimum

The recurrence equation for the max/min algorithm [1,2] can be given as follows:

$$T(n) = \begin{cases} 2T\left(n/2\right) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

Assume that $n = 2^k$. By repeated substitution, one can obtain that the following relations:

$$T(n) = 2T\left(\frac{n}{2}\right) + 2$$

$$= 2\left[\left\{2T\left(n/4\right)\right\} + 2\right] + 2$$

$$= 4T\left(n/4\right) + 4 + 2$$

$$\vdots$$

$$= 2^{k-1} \cdot T(2) + \sum_{i=1}^{k-1} 2^i$$

$$= 2^{k-1} + 2^k - 2$$

$$= \frac{2^k}{2} + 2^k - 2$$

$$= \frac{n}{2} + n - 2 \quad , \text{since } n = 2^k.$$

$$= \frac{3n}{2} - 2$$

The solution of the recurrence equations gives (3*n*/2) − 2 comparisons.

Tiling Problem

Another important problem is Tiling problem [3]. The problem can be stated as follows: Given a Region and a Tile T, Is it possible to tile R with T? A defective chessboard is a chessboard that has one unavailable (defective) position. A tromino is an L shaped object as shown in Fig. 2.

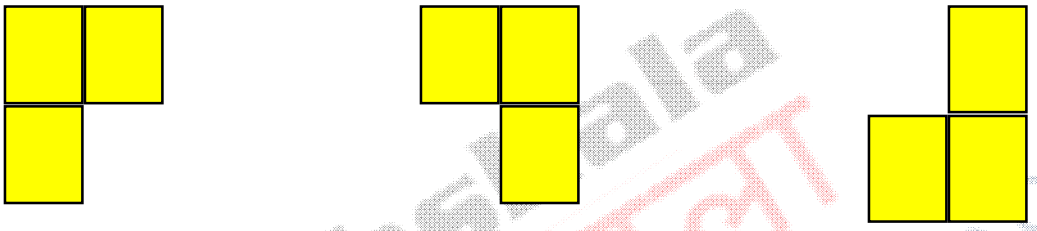


Fig. 2: Examples of Tromino

The idea is to tile the defective chess board with a tromino. The divide and conquer paradigm can be applied to this problem. The procedure for applying divide and conquer paradigm is given below:

If board is small, Directly tile,

else

- Divide the board into four smaller boards
- Conquer using the tiling algorithm recursively
- Combine it

It can be understood as follows. If the board is small, then the tromino can be applied manually and checked. Else, divide and conquer paradigm can be applied. The board configurations can be

divided further, then the subboards can be tiled, finally the results can be combined to find solution of the given larger board.

The formal Algorithm based on [1] for Tiling problem is given as follows:

INPUT: *n* ï the board size (2^nx2^n board), *L* – location of the defective hole.

OUTPUT: tiling of the board

Algorithm Tile(n, L)

Input : n ï order of the board,

L ï Tromino

Begin

if n = 1 **then**

Tile with one tromino directly

return

Else

Divide the board into four equal-sized boards

Place one tromino at the centre to cover the defective hole by assuming the extra 3 additional holes, L1, L2, L3, L4 denote the positions of the 4 holes

Tile(n-1, L1)

Tile(n-1, L2)

Tile(n-1, L3)

Tile(n-1, L4)

End

The complexity analysis of this is given based on [1] below.

Complexity Analysis

The recurrence equation of the defective chess board problem is given as follows:

$$T(n) = \begin{cases} c & \text{for } n = 0 \\ 4T\left(\frac{n}{2}\right) + c & \text{for } n > 0 \end{cases}$$

Therefore, the complexity analysis is O(n^2).

Fourier Transform

Fourier transform [4] is used for polynomial multiplication because it helps to convert one representation of a polynomial (coefficient representation) to another representation (value representation). Thus, computation using Fourier transform can be carried out in the following two ways:

Evaluate Fourier transforms help in converting a coefficient representation to a value representation. This is given as follows:

<values> = Fourier transform((coefficients), ω).

Interpolation After computation, conversion of a value representation to a coefficient form can be performed using inverse Fourier transform. This is given as follows:

<coefficients> = Fourier transform((values), ω^{-1})

One may view Fourier transform as a method of changing the representation or coding of a polynomial. Fourier transform has many applications. One of its important applications is polynomial multiplication. A polynomial is used to represent a function in terms of variables. A polynomial can be represented as follows:

$$A = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

Here n is referred to as the degree bound of the polynomial and $a_0, a_1, \ldots, a_{n-1}$ are called its coefficients. The polynomial is said to be of degree k, if the highest coefficient of the polynomial is a_k.

Fourier transform is then given as follows:

$$A_i = \sum_{k=0}^{n-1} a_k e^{-j\frac{2\pi ik}{n}}, \quad 0 \le i \le n-1,$$

a_k, where k ranges from 0 to n ī 1, represents the set of coefficients of a polynomial $\{a_0, a_1, \cdots a_{n-1}\}$; n is the length of the coefficient vector that represents the degree of the given polynomial. In other words, Fourier transform also represents a polynomial as the nth root of unity [3]. The nth root is the solution of the polynomial $x^n - 1 = 0$, which is given as $\omega_n = e^{-j2\pi}$. The nth root at all points of input x is given as $e^{2\pi jx/n}$. Using Eulerôs formula, the evaluation of $e^{2\pi jx/n}$ yields $\cos(\frac{2\pi}{n}x) + j\sin(\frac{2\pi}{n}x)$, where $j = \sqrt{-1}$. The output of a Fourier transform thus can also be a complex number.

One can also design inverse Fourier transform to convert a value form to a coefficient form. Inverse Fourier transform can be given as follows:

$$a_i = \frac{1}{n}\sum_{k=0}^{n-1} A_k e^{j\frac{2\pi ik}{n}}, \quad 0 \le i \le n-1$$

To multiply faster and effectively, it is better to use matrix representation for implementing Fourier transform. The matrix representation is given as follows:

$$A = Va$$

where V is an $n \times n$ matrix, called the Vandermonde matrix, and a is the vector of coefficients given as $\{a_0, a_1, \cdots, a_{n-1}\}$. Here n represents the number of coefficients of the given polynomial. The resultant vector A is a set of values given as $\{A_0, A_1, \cdots, A_{n-1}\}$, which represent the transformed coefficients of Fourier transform. The matrix V can be given as follows:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)^2} \end{pmatrix}$$

Thus, the resultant matrix A of Fourier transform can be given as follows:

$$A=\begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{1} & \omega^{2} & \omega^{3} & \cdots & \omega^{n-1} \\ 1 & \omega^{2} & \omega^{4} & \omega^{6} & \cdots & \omega^{2(n-1)} \\ 1 & \omega^{3} & \omega^{6} & \omega^{9} & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)^2} \end{pmatrix} \times a$$

Inverse Fourier transform can be obtained as follows:

As $A = Va$, the coefficients a can be retrieved as follows:

$$a = V^{-1} \times A$$

Here, the matrix V^{-1} can be obtained by taking the complex conjugate of the matrix V by replacing ω by ϖ, as $\varpi = \frac{1}{\omega}$ or ω^{-1}. Complex conjugate means the sign of the imaginary component of a complex number is changed. Therefore, substituting this in the matrix, one gets the inverse matrix V ($V^{\bar{1}}$), which is as follows:

$$V^{\bar{1}} = 1/n \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \varpi^{1} & \varpi^{2} & \varpi^{3} & \cdots & \varpi^{n-1} \\ 1 & \varpi^{2} & \varpi^{4} & \varpi^{6} & \cdots & \varpi^{2(n-1)} \\ 1 & \varpi^{3} & \varpi^{6} & \varpi^{9} & \cdots & \varpi^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \varpi^{(n-1)} & \varpi^{2(n-1)} & \varpi^{3(n-1)} & \cdots & \varpi^{(n-1)^2} \end{pmatrix}$$

Thus, the resultant matrix a of inverse Fourier transform can be given as follows:

$$a = 1/n \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)^2} \end{pmatrix} \times A$$

Let us try to design the matrix V and V^{-1} for four sample points. Therefore, $n = 4$ and let $a = \{a_0, a_1, a_2, a_3\}$. Then one can find ω by substituting $e^{2\pi jx/n}$ as follows:

$$\omega = e^{-j\frac{2\pi}{n}} = e^{-j\frac{2\pi}{4}} = e^{-j\frac{\pi}{2}} = \cos\frac{\pi}{2} - j\sin\frac{\pi}{2} = -j \quad (\text{as } n = 4)$$

On substituting this value of ω in the matrix V of order 4×4, one gets the following matrix:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & (-j)^2 & (-j)^3 \\ 1 & (-j)^2 & (-j)^4 & (-j)^6 \\ 1 & (-j)^3 & (-j)^6 & (-j)^9 \end{pmatrix}$$

Here, j is a complex number and is equal to $\sqrt{-1}$. Therefore, one can observe that the resultant matrix V that involves complex numbers is as follows:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix}$$

Thus, the resultant matrix A can be given as follows:

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The coefficients can be retrieved using inverse Fourier transform. For this case where $n = 4$ and $A_k = \{A_0, A_1, A_2, A_3\}$, the matrix V^{-1} of order 4 × 4 can be obtained by substituting $\varpi = \frac{1}{\omega} = e^{j\frac{2\pi}{n}} = e^{j\frac{\pi}{2}} = -j$ in the general matrix. This is the complex conjugate of the matrix *V*. For finding the complex conjugate, one has to change the sign of the imaginary component of the complex number. For *n* = 4, the inverse matrix *V* (V^{-1}) is given as follows:

$$V^{-1} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & +j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{pmatrix}$$

Thus, the resultant matrix for finding coefficients from values is given as follows:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \frac{1}{n} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \times \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix}$$

One can verify that the product of V and V^{-1} is a unit matrix as they are complex conjugates of each other. In addition, one can check that the original coefficients are obtained using inverse Fourier transform and there is no information loss. This is demonstrated in the following numerical example 1.

Example 1

Find Fourier transforms of the following four coefficients and also verify that inverse Fourier transform gives the original coefficients without any loss.

$$x = \{1, 3,5,7\}$$

Solution

As there are four samples, $n = 4$. Fourier transform can be given as follows:

$$A = V \times a$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 5 \\ 7 \end{pmatrix}$$

$$= \begin{pmatrix} 16 \\ -4-4j \\ -4 \\ -4-4j \end{pmatrix}$$

One can verify that the inverse of this gives back the original coefficients. Therefore, take the inverse kernel and multiply the Fourier coefficients:

$$a = \frac{1}{n} \times (V^{-1} \times A)$$

$$a = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & +j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{pmatrix} \begin{pmatrix} 16 \\ -4-4j \\ -4 \\ -4-4j \end{pmatrix}$$

$$= a = \frac{1}{4}\begin{pmatrix} 4 \\ 12 \\ 20 \\ 28 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 5 \\ 7 \end{pmatrix}$$

It can be observed that one is able to get back the original coefficients.

Idea of FFT

One can implement a faster Fourier transform using an algorithm called an FFT algorithm. FFT is implemented using the divide-and-conquer strategy. The input array of points is divided into odd and even arrays of points. Individually, FFT is applied to the subarrays. Finally, the subarrays are merged. Informally, an FFT algorithm can be stated as follows:

Step 1: If $n = 1$, then solve it directly as a_0.

Step 2: Divide the input array into two arrays B and C such that B has all odd samples and C has all even samples. Continue division if the subproblems are large.

Step 3: Apply FFT recursively to arrays B and C to get subarrays B' and C'.

Step 4: Combine the results of subarrays B' and C' and return the final list.

Complexity Analysis of FFT algorithms

The recurrence equation of FFT is given as follows:

$$T(n) = 2T\left(\frac{n}{2}\right) + \mathrm{O}(n)$$

One can use the master theorem and solve this recurrence equation. One can observe that the complexity analysis of this algorithm turns out to be $\mathrm{O}(n \log n)$.

Polynomial Multiplication

Many polynomial operations are required in scientific applications. One of the most important applications is multiplying two polynomials. Let $a(x)$ and $b(x)$ be two polynomials; their product $C(x) = a(x) \times b(x)$ can be expressed [3] as follows:

Compute C(x) = A(x)B(x), where degree(A(x)) = m, and degree(B(x)) = n. Degree(C(x)) = m+n, and C(x) is uniquely determined by its value at m+n+1 distinct points.

The informal algorithm is given as follows:

Step 1: Let the polynomials A and B be of degree n. Then find $m = 2 \times n$ ī 1.

Step 2: Pick m points ranging from 0 to m ī 1.

Step 3: Evaluate polynomials A and B at m points.

Step 4: Compute $C(x) = A(x) \times B(x)$ using ordinary multiplication.

Step 5: Interpolate $C(x)$ to get the coefficients of the polynomial $C(x)$.

It can be observed that point-wise multiplication is enough to multiply polynomials. One can combine the idea of divide and conquer with this concept. The idea of division is that any function at sample points x can be divided into function samples at odd points and those at even points. Thus, a polynomial $A(x)$ can also be represented as $A_{odd}(x^2) + xA_{even}(x^2)$, where A_{odd} represents a set of odd sample points and A_{even} a set of even sample points of the given polynomials. Therefore, the advantage of using a divide-and-conquer algorithm is that only one-half of the resultant polynomial is calculated and the other half is a negative of the first half (i.e., $A_{odd}(x^2) - xA_{even}(x^2)$).

Summary

In short, one can conclude as part of this module 13 that

- Divide and conquer is effective in implementing Fourier Transform.

- DFT is important problem and can be solved by divide and conquer strategy.
- Polynomial multiplication and convolution are some of the examples of applications of FFT.

References:

1. *S.Sridhar , Design and Analysis of Algorithms , Oxford University Press, 2014.*
2. *A.Levitin, Introduction to the Design and Analysis of Algorithms, Pearson Education, New Delhi, 2012.*
3. T.H.Cormen, C.E. Leiserson, and R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA 1992.
4. URL: https://en.wikipedia.org/wiki/Joseph_Fourier