Assignment -1

1) a) Master theorem

$$T(n) = aT(n/b) + f(n) \qquad a \geq 1 \; b \geq 1$$

Case ①: $(c < \log_b a)$

where $f(n) \in O(n^c)$, $c < \log_b a$

$$\Rightarrow T(n) = \Theta(n^{\log_b a})$$

② $c = \log_b(a)$

$$\rightarrow f(n) \in (n^c \log^k n) \quad \text{where} \quad k \geq 0, \; c = \log_b(a)$$

$$\rightarrow T(n) = \Theta(n^c \log^{k+1} n)$$

③ $c > \log_b(a)$

$$\rightarrow f(n) \in \Omega(n^q) \; ; \; c > \log_b(a)$$

$$f(n/b) \leq k f(n) \quad k < 1$$

$$\Rightarrow T(n) \in \Theta(f(n))$$

a) $T(n) = 2 \cdot T(n/2) + O(n \log n)$

$c=1 \quad a=2 \quad b=2 \quad f(n) = (n \log n)$

here $k=1$
according to case-2

$\Rightarrow$ Also must be $c = \log_2 2 \because (\log_b a) \Rightarrow c = 1$

This condition is also satisfied.

$$\therefore T(n) = \Theta(n^c \log^{k+1} n)$$

$$\Rightarrow \Theta(n^1 \log^{1+1} n)$$

$$\Rightarrow \Theta(n \log^2 n)$$

---

b) $T(n) = 2 \cdot T(n/2) + \Theta(n / \log n)$

$\Rightarrow$

$\Rightarrow 2 \cdot [2\, T(n/2^2) + O((n/2)/\log(n/2))] + \Theta(n/\log n)$

$\Rightarrow 2^2 \cdot T(n/2^2) + 2^*(n/2) / \lg(n/2) + n/\lg(n)$

$\Rightarrow 4^* T(n/4) + n/(\log n - 1) + n/\lg(n)$

$\Rightarrow 4[2\, T(n/2^3) + \frac{n/4}{\log(n/4)}] + n/\log n - 1 + n/\log n$

$$\Rightarrow 8\, T(n/8) + \frac{n}{\log n - 2} + \frac{n}{\log n - 1} + \frac{n}{\log n}$$

Generalizing it for k^{th} term

$$T(n) = a^k\, T(n/2^k) + \sum_{i=0}^{k-1} \left(\frac{n}{\log n - i} \right)$$

As n is a power of 2, and they are k terms in the series, $\therefore a^k = n$

$$T(n) = n \cdot T(1) + n \cdot \sum_{i=0}^{k-1} \frac{1}{\log(2^k) - i}$$

$$\Rightarrow n \cdot T(1) + n \sum_{i=0}^{k-1} \frac{1}{k - i}$$

It is a harmonic progression, where $\sum_{i=0}^{k-1} \frac{1}{k-i}$

from discussion,

$$H_k = 1/1 + 1/2 + \cdots - 1/k = \ln k + O(1)$$

$$\therefore T(n) = n \cdot (\ln k)$$

$$k = \log_2 n$$

$$\therefore T(n) = \Theta(n \cdot \ln(\log_2 n))$$

1 (c) $T(n) = \sqrt{n} * T\sqrt{n} + O(n)$

Sub $n = 2^k$

$$T(2^k) = 2^{(k/2)} \cdot T\left(2^{k/2}\right) + 2^k$$

Divide by 2^k to make constant 1

$$\frac{T(2^k)}{2^k} = \frac{2^{(k/2)} T(2^{k/2})}{2^k} + 1$$

$$\Rightarrow \frac{T(2^k)}{2^k} = \frac{T(2^{k/2})}{2^{k/2}} + 1$$

This fits as a Master theorem problem,

where $f(k) = T(2^k)/2^k$

$$\therefore f(k) = f(k/2) + 1$$

Here, $a = 1$, $b = 2$, $c = 0$.

$$\log_b a = \log_2 1 = 0$$

which is equal to $c = \log_b a$

$\therefore$ from second case,

$$f(n) = O(n^0 \log(n)) = \log(n)$$

$$\Rightarrow \frac{T(2^m)}{2^m} = \log m$$

$$\Rightarrow T(2^m) = 2^m (\log m)$$

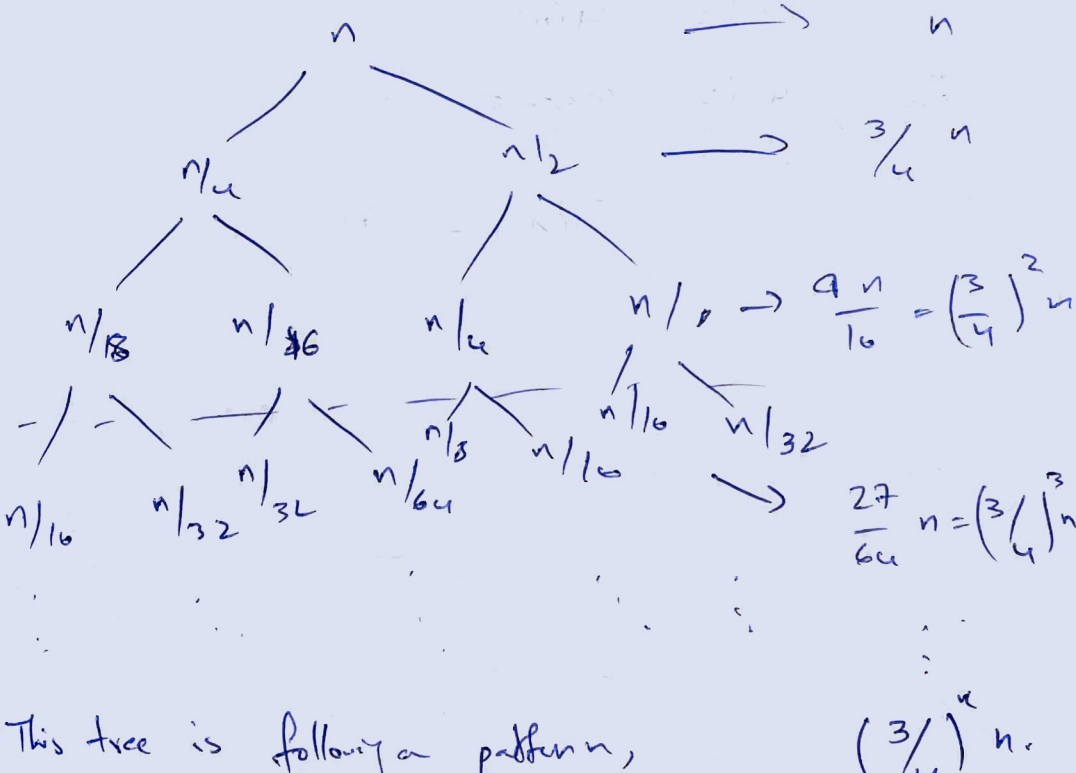
As 2^m is n

$\Rightarrow m = \log_2 n$

$$\Rightarrow T(n) = n \cdot \log(\log n)$$

1(d) $T(n) = T(n/4) + T(n/2) + O(n)$

We can solve this using recursion tree



⊙ This tree is following a pattern, and the least problem size is ①, so we can assume $n = 2^k$.

$\Rightarrow k = \log_4 n.$

$\therefore T(n) = n + (3/4)n + (3/4)^2 n \cdots (3/4)^{\log_4 n}\, n$

$\Rightarrow T(n) = n\left[1 + 3/4 + (3/4)^2 \cdots + (3/4)^{\log_4 n}\right]$

This is a geometric progression, where,

$a = 1, \quad r = 3/4$

$\therefore$ Sum of terms $= \dfrac{1}{1 - 3/4} = 4$

$\therefore T(n) = 4(n)$

Here, 4 is constant

$\therefore T(n) = n$

2. Suppose n is a positive integer and ω is an n-th root of unity, so ω_n=1. Define the n by n Fourier transform matrix $M_n(\omega)$ as in the book

A .

$M_n(\omega)$ is defined as below

$M_n(\omega) = \omega^{(j \cdot k)}$ for j ≥ 0, k<n

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ & & \vdots & \dots & \\ 1 & \omega^j & \omega^{2j} & \dots & \omega^{(n-1)j} \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)\,(n-2)} \end{bmatrix}$$

where (ω) is the nth root of unity and n is a power of 2.

2a) Suppose ω^k= 1 for some smaller integer k, 0<k<n. (This means ω is not a *primitive* n-th root of unity.) Argue that the matrix $M_n(\omega)$ is not invertible.

A.

Given $\omega^k = 1$ for 0 < k <n, that means ω is a root of unity of order k.

We can prove that matrix $M_n(\omega)$ is not invertible as follows:

Calculate the determinant of $M_n(\omega)$:

For the first row,

$\text{Det}(M_n(\omega))$ = $\omega^{0.k}$ * cofactor(0,0) + $\omega^{1.k}$ * cofactor(0,1).....+$\omega^{(n-1)k}$ * cofactor(0,n-1)

⇒ 1 * cofactor(0,0) + 1 * cofactor(0,1).....+1 * cofactor(0,n-1)

Since the determinant is sum of all factors in first row and all of them are 1's (as $\omega^k = 1$) , The determinant of $M_n(\omega)$ is the sum of equal terms which makes it zero.

⇒ $\text{Det}(M_n(\omega)) = 0$

When a determinant of a matrix becomes zero, that implies it is not invertible.

2b) Over the complex numbers, suppose n=4 and ω=i (the square root of -1). Write out $M_n(\omega)$ as the product of two "2-sparse" matrices. That is, matrices with at most two non-zero entries in each row and each column.

A. Given n=4, so

$$M_4(\omega) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}$$

Simplifying it, we get

$$M_4(\omega) =' \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

Given in question is that we have to find 2 sparse matrices at most two non-zero entries in each row and each column.

By analyzing the fast fourier circuit, the sparse matrices I got are :

$$A= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & i \\ 1 & 0 & -i & 0 \\ 0 & 1 & 0 & -i \end{bmatrix}$$

$$B= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

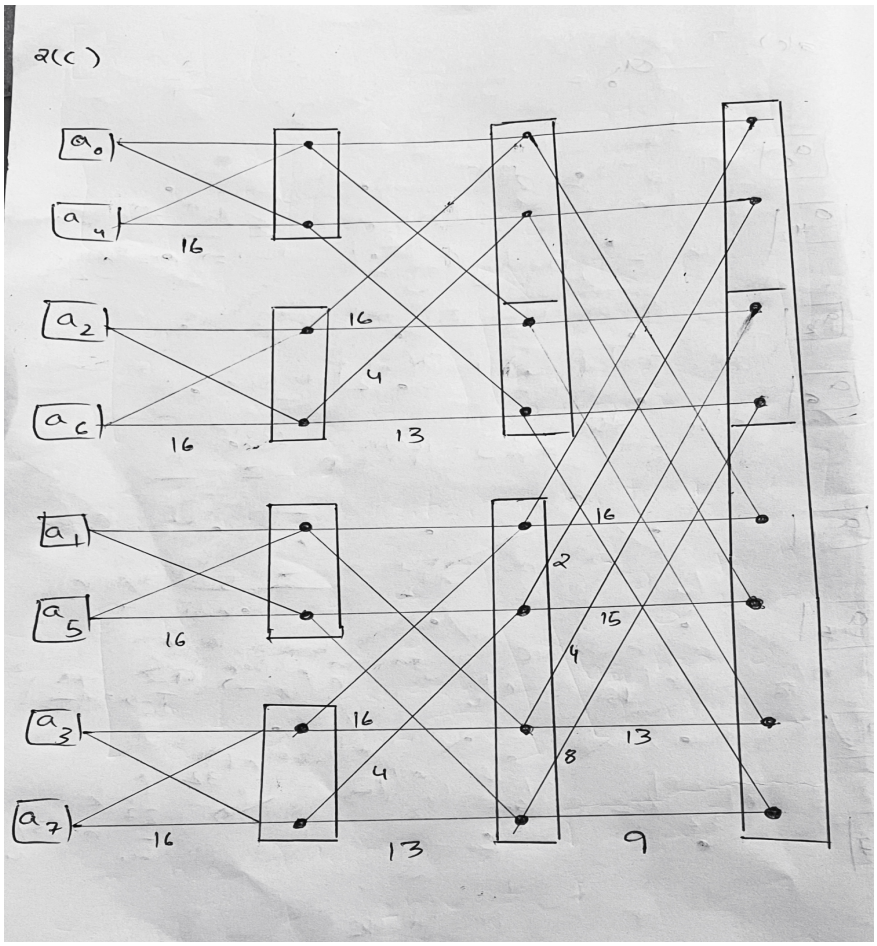
2c) Over the integers modulo 17, we choose n=8 and ω=2 (you may want to verify that 2 is a primitive 8-th root of unity, mod 17). We want to compute $M_n(\omega)$ using the Fast Fourier transform circuit (the butterfly network below, from DPV page 69/77). Draw a copy of the network, where you label each edge by its scalar multiplier (some remainder modulo 17). You may omit the label when it equals 1.

A. As the question said, we want to verify that 2 is a primitive 8th root of unity. To do that,

We check that 2^8 is the first power of ω=2 that equals 1, mod 17

i	1	2	3	4	5	6	7	8
2^i	2	4	8	16	15	13	9	1

The circuit is given below :



3. Given complex numbers r1, r2, ..., rn, we want to compute the polynomial A(x) = (x-r1)(x-r2)...(x-rn). That is, we want to compute the coefficients a0, a1, ..., an-1 such that A(x) = a0 + a1 x + a2 x2 + ... + xn. Find an efficient divide-and-conquer algorithm, solving this in near-linear time. What is your running time?

A.

Given n complex numbers r1,r2,...,rn and A(x)= (x-r1)(x-r2)....(x-rn). For computing the terms a0, a1, … a(n-1) such that A(x) = a0 + a1 x + ... + x^n. We can calculate the coefficients of A(x) using Fast Fourier Theorem (FFT), which is a divide and conquer algorithm, in linear time.

In FFT, we divide the problem into smaller subproblems, solve them using recursion and merge the results to get the final result. I am writing the steps below:

- Select a value of N : Our first step is finding out value of N, which is a power of 2 where $N \geq 2n$. To reach this step we can pad the input complex numbers to 0.
- Divide into subproblems : In the second step we create two arrays namely 'B(x)' and 'C(x)', both having a size of N. These arrays will become representations for the polynomials associated with r1, r2, ..., rn and 1, respectively. As discussed in the earlier step, initialize C(x) with zeros and B(x) such that B(xi) = 1- ri, where ‘ri’ is each complex number in the set. Now perform FFT on both the arrays. Keep the resultant values as F(B) and F(C). This step is also known as Evaluation.
- Multiplication : In the third step, we conduct a multiplication between F(B) and F(C). Store the resultant as F(R).
- Merge : In this step, we use interpolation on F(R) to deduct the coefficients of A(x).

Running time :

The Evaluation and Interpolation can be done with remarkable efficiency in a near linear time O(N log N). Therefore, the total run time is O(n log n).

4.Search for information on the polynomial multipoint evaluation problem, similar to problem `E' of our first lecture:

We are given a polynomial A(x) = a0 + a1 x + ... + an-1 xn-1, and n numbers w0, w1, ..., wn-1. (Suppose all the a's and w's are complex numbers.) We want to output the n numbers A(w0), A(w1), ..., A(wn-1).

This time the wi's are part of the input (not fixed). You may use the internet, journals, books, etc, just not people. Search for and state some result, showing that this problem can be solved in time O(n log^C n), for some constant c. Give a full citation for your source(s).

A.

Given question is regarding the polynomial multipoint evaluation problem, where A(x) = a0 + a1 x + ... + an-1 x^(n-1) and n complex numbers w0,w1,...w(n-1). We should calculate A(x) for each w0,w1… and output those n numbers. From browsing, I found out an efficient algorithm which uses recursion that is used to divide the problem into subproblems, finally leading to a time complexity to O(M(n) log n), where M(n) represents the cost of multiplying 2 elements in the n complex numbers. So we can consider it as n in the final case.

The Algorithm :

Algorithm 1 Multipoint evaluation

(Pre)computation step: Compute $P_{i,j}$ for all $0 \leq i \leq k$ and $0 \leq j < 2^{k-i}$.
1. If $n = 1$ return f.
2. Let $r_0 = f \mod P_{k-1,0}$ and $r_1 = f \mod P_{k-1,1}$.
3. Recursively, evaluate r_0 at $u_0, \ldots, u_{n/2-1}$.
4. Recursively, evaluate r_1 at $u_{n/2}, \ldots, u_{n-1}$.
5. Output $r_0(u_0), \ldots, r_0(u_{n/2-1}), r_1(u_{n/2}), \ldots, r_1(u_{n-1})$.

Analysis :

- We start initially with a polynomial f(x) of degree less than n = 2k. Define two polynomials P0 and P1 such that :

P0 = ∏(x - ui) for i in the range [0, n/2 - 1].

P1 = ∏(x - ui) for i in the range [n/2, n - 1].

- Calculate and store the remainders of f divided by P0 and P1. Store the results as r0, r1. We can deduce that $r_0(u_i)$ = f(u_i) for all 0≤ $i \leq$ n/2 -1 and same goes for $r_1(u_i)$ = f(u_i) for all n/2 -1 ≤ $i \leq$ n. So we can apply the multipoint evaluation algorithm recursively on r0 and r1 where we have the reduced degree n/2=2k-1.
- In the third step we precompute the polynomial $P_{i,j}$ for optimizing the recursive step where $P_{i,j} = \prod_{l=0}^{2^i-1} (x = u_{j \cdot 2^i + l})$, 0 ≤ i ≤ k and 0 ≤ j < 2^(k-i). The precomputation is done using:

$$P_{0,j} = x - u_j \text{ for i=0}$$

$$P_{i+1,j} = P_{i,2j} + P_{i,2j+1} \text{ when i>1.}$$

- The above step computes $P_{i,j}$ in O($M(2^i)$) operations. This step has a time complexity of O(M(n)). Adding overall i from 0 to k-1, The total complexity of this step is O((n) log n).

Work Cited :“Lecture 6 1 Multipoint evaluation of a polynomial.” *CSA – IISc Bangalore*, 7 May 2012, https://www.csa.iisc.ac.in/~chandan/courses/CNT/notes/lec6.pdf. Accessed 7 September 2023.

5. Suppose we have an input graph G with vertices numbered 1 through n, such that |i-j|<10 whenever (i,j) is an edge. We want to find an MIS (maximum independent subset) in such a graph. Propose an algorithm solving this problem in polynomial time. Hint: a subset of *nine* consecutive vertices forms a "separator", so you could try divide-and-conquer.

A.

This algorithm splits the graphs into smaller subgraphs using separators and uses the graph features to find MIS of those.

Divide and Conquer algorithm:

Initially, divide the graph into blocks of 9 consecutive vertices. This means we will have n/9 subgraphs.

Separator set :

It is a set of vertices whose removal will divide the graph into subgraphs.

Now find a separator set S of vertices in the graph . In this case, a subset of 9 consecutive vertices form a separator.

Finding a separator:

In this case, select the middlemost nine vertices. It divides the graph into vertices to the left and right. That acts as a separator.

Using this we recursively find Maximum independent subsets of smaller subgraphs as per Planar separator theorem

Merge:

In this step we combine all the MIS of subgraphs with the separator set to obtain the final Maximum independent subset for the graph.

Time Complexity:

The overall time complexity is determined by the divide and conquer step using separators

$$T(n) = O(n) + 2^{|s|} [2 . T(n/2)]$$

Here s=9, substitute and we get

$$\Rightarrow T(n) = O(n) + 2^{9} [2 . T(n/2)]$$

$$\Rightarrow T(n) = O(n) + 2^{10} . T(n/2)$$

We can solve this using masters theorem, where

a= 2^{10} , b=2, d=1 (As per DPV pg : 49 or 60)

$$\Rightarrow T(n) = O(n^{log_b a})$$

$$\Rightarrow T(n) = O(n^{log_2 2^{10}})$$

$$\Rightarrow T(n) = O(n^{10})$$

The final time complexity in this case is O(n^{10})