Problem Set 4

Due dates: Electronic submission of the pdf file of this homework is due on **2/14/2025 before 11.59pm** on canvas.

Name: (put your name here)

Resources. (All people, books, articles, web pages, etc. that have been consulted when producing your answers to this homework)

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework. The solutions given in this homework are my own work.

Signature: _____

Make sure that you describe all solutions in your own words. Typeset your solutions in LaTeX. Read chapter 30 on "Polynomials and the FFT" and chapter 15 on "Greedy Algorithms" in our textbook.

Problem 1 (20 points)**.** The polynomial $A(x) = 1 + x + x^2$ can be represented by the vector $(1, 1, 1, 0)^t$. (a) Use a 4×4 DFT and transform this vector. (b) Explicitly describe the resulting vector in terms of a vector of the form $(A_(x_0), A(x_1), A(x_2), A(x_3))^t$ for some complex numbers x_0, x_1, x_2, and x_3. Make sure that you verify your result.

Solution. We will solve this problem step by step, performing the DFT, interpreting the result, and verifying our solution.

(a) Performing the 4×4 DFT:

The 4×4 DFT matrix W is defined as:

$$W = \begin{bmatrix} 1+0j & 1+0j & 1+0j & 1+0j \\ 1+0j & 0-1j & -1-0j & -0+1j \\ 1+0j & -1-0j & 1+0j & -1-0j \\ 1+0j & -0+1j & -1-0j & 0-1j \end{bmatrix}$$

Our input vector $a = (1, 1, 1, 0)^t$

Performing the DFT: $result = W \cdot a$

$$result = \begin{bmatrix} 3.0000000 + 0.0000000j \\ 6.1232340 \times 10^{-17} - 1.0000000j \\ 1.0000000 + 1.2246468 \times 10^{-16}j \\ -1.8369702 \times 10^{-16} + 1.0000000j \end{bmatrix}$$

(b) Interpreting the Result:

The resulting vector can be interpreted as $(A(x_0), A(x_1), A(x_2), A(x_3))^t$ where:

- $A(x_0) = 3 + 0j$
- $A(x_1) \approx 0 - 1j$ (small numerical error in the real part)
- $A(x_2) = 1 + 0j$
- $A(x_3) \approx 0 + 1j$ (small numerical error in the real part)

These values correspond to the evaluations of our polynomial $A(x) = 1 + x + x^2$ at the fourth roots of unity:

- $x_0 = 1$
- $x_1 = -i$
- $x_2 = -1$
- $x_3 = i$

Verification:

To verify our result, we evaluate the polynomial $A(x) = 1 + x + x^2$ at the fourth roots of unity:

- $A(1) = 1 + 1 + 1^2 = 3$
- $A(-i) = 1 + (-i) + (-i)^2 = 1 - i - 1 = -i$
- $A(-1) = 1 + (-1) + (-1)^2 = 1 - 1 + 1 = 1$
- $A(i) = 1 + i + i^2 = 1 + i - 1 = i$

Comparing these evaluations with our DFT result:

Point	DFT value	Polynomial value
x_0 (1)	$3.000 + 0.000j$	$3.000 + 0.000j$
x_1 (-i)	$0.000 - 1.000j$	$0.000 - 1.000j$
x_2 (-1)	$1.000 + 0.000j$	$1.000 + 0.000j$
x_3 (i)	$-0.000 + 1.000j$	$-0.000 + 1.000j$

The DFT values match exactly with the polynomial evaluations at the fourth roots of unity, confirming the correctness of our computation.

Conclusion:

We have successfully performed a 4×4 DFT on the vector representation of the polynomial $A(x) = 1 + x + x^2$. The resulting vector $(3, -i, 1, i)$ correctly represents the evaluations of $A(x)$ at the fourth roots of unity. This demonstrates the powerful connection between polynomial evaluation and the Discrete Fourier Transform, a fundamental concept in signal processing and computational mathematics.

Problem 2. (20 points) Let ω be a primitive nth root of unity. The fast Fourier transform implements the multiplication with the matrix

$$F = (\omega^{ij})_{i,j \in [0..n-1]}.$$

Show that the inverse of the matrix F is given by

$$F^{-1} = \frac{1}{n}(\omega^{-jk})_{j,k \in [0..n-1]}$$

[Hint: $x^n - 1 = (x-1)(x^{n-1} + \cdots + x + 1)$, so any power $\omega^\ell \neq 1$ must be a root of $x^{n-1} + \cdots + x + 1$.] Thus, the inverse FFT, called IFFT, is nothing but the FFT using ω^{-1} instead of ω, and multiplying the result with $1/n$.

Solution. *Proof.* To prove that $F^{-1} = \frac{1}{n}(\omega^{-jk})_{j,k \in [0..n-1]}$ is the inverse of $F = (\omega^{ij})_{i,j \in [0..n-1]}$, we need to show that $F \cdot F^{-1} = I$, where I is the $n \times n$ identity matrix.

Let's consider the (i,k)-th element of the product $F \cdot F^{-1}$:

$$
\begin{aligned}
(F \cdot F^{-1})_{i,k} &= \sum_{j=0}^{n-1} F_{ij} \cdot (F^{-1})_{jk} \\
&= \sum_{j=0}^{n-1} \omega^{ij} \cdot \frac{1}{n} \omega^{-jk} \\
&= \frac{1}{n} \sum_{j=0}^{n-1} \omega^{ij-jk} \\
&= \frac{1}{n} \sum_{j=0}^{n-1} \omega^{j(i-k)}
\end{aligned}
$$

Let $m = i - k$. Then we have:

$$
(F \cdot F^{-1})_{i,k} = \frac{1}{n} \sum_{j=0}^{n-1} (\omega^m)^j
$$

This is a geometric series with n terms. We consider two cases:

1. If $i = k$ (i.e., $m = 0$): Each term in the sum is 1, so the sum is n. Therefore,

$$
(F \cdot F^{-1})_{i,k} = \frac{1}{n} \cdot n = 1
$$

2. If $i \neq k$ (i.e., $m \neq 0$): We know that $\omega^n = 1$ (since ω is an n-th root of unity), so $\omega^{mn} = 1$. The sum of the geometric series is:

$$
\sum_{j=0}^{n-1} (\omega^m)^j = \frac{1 - (\omega^m)^n}{1 - \omega^m} = \frac{1 - 1}{1 - \omega^m} = 0
$$

Therefore,

$$
(F \cdot F^{-1})_{i,k} = \frac{1}{n} \cdot 0 = 0
$$

Combining these results, we can conclude that:

$$
(F \cdot F^{-1})_{i,k} = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases}
$$

This is precisely the definition of the identity matrix I. Therefore, we have proven that $F \cdot F^{-1} = I$, which means that F^{-1} is indeed the inverse of F. $\square$

Problem 3. (20 points) Describe in your own words how to do a polynomial multiplication using the FFT and IFFT for polynomials $A(x)$ and $B(x)$ of degree $\leq n - 1$. Make sure that you describe the length of the FFT and IFFT needed for this task. Be concise and precise. Illustrate how to multiply the polynomials $A(x) = x^2 + 2x + 1$ and $B(x) = x^3 + 2x^2 + 1$ using this approach.

Solution. Polynomial multiplication is a fundamental operation in algebra and computer science. While traditional methods have a time complexity of $O(n^2)$ for polynomials of degree n, using the Fast Fourier Transform (FFT) and its inverse (IFFT) can reduce this to O(n log n).

1 Theory Context

Given two polynomials $A(x)$ and $B(x)$ of degrees m and k respectively, their product $C(x) = A(x) \cdot B(x)$ will have a degree of $m + k$. The key idea is to:

1. Convert the polynomials from coefficient representation to point-value representation using FFT.
2. Perform pointwise multiplication in the point-value representation.
3. Convert the result back to coefficient representation using IFFT.

2 Algorithm

Algorithm 1 Polynomial Multiplication using FFT and IFFT

procedure PolyMultiply(A, B)
  $m \leftarrow \text{degree}(A)$
  $k \leftarrow \text{degree}(B)$
  $n \leftarrow 2^{\lceil \log_2(m+k+1) \rceil}$
  $A' \leftarrow \text{PadZeros}(A, n)$
  $B' \leftarrow \text{PadZeros}(B, n)$
  $\hat{A} \leftarrow \text{FFT}(A')$
  $\hat{B} \leftarrow \text{FFT}(B')$
  $\hat{C} \leftarrow \hat{A} \cdot \hat{B}$ ▷ Pointwise multiplication
  $C \leftarrow \text{IFFT}(\hat{C})$
  return C
end procedure

3 Step-by-Step Process

3.1 Determine FFT Length

The length of the FFT and IFFT is determined by:

$$n = 2^{\lceil \log_2(m+k+1) \rceil}$$

where m and k are the degrees of $A(x)$ and $B(x)$ respectively.

3.2 Evaluate Polynomials using FFT

Apply FFT to both $A(x)$ and $B(x)$, evaluating them at n equally spaced points on the unit circle in the complex plane:

$$\hat{A} = \text{FFT}(A), \quad \hat{B} = \text{FFT}(B)$$

3.3 Pointwise Multiplication

Multiply the evaluated points pointwise:

$$\hat{C}[i] = \hat{A}[i] \cdot \hat{B}[i], \quad \text{for } i = 0, 1, \ldots, n-1$$

3.4 Inverse FFT

Apply IFFT to convert the pointwise product back to coefficient form:

$$C = \text{IFFT}(\hat{C})$$

4 Example

Consider multiplying $A(x) = x^2 + 2x + 1$ and $B(x) = x^3 + 2x^2 + 1$:

1. Degree calculation:
   - Degree of $A(x)$ is 2
   - Degree of $B(x)$ is 3
   - Resulting polynomial $C(x)$ will have degree $2 + 3 = 5$
2. FFT length: $n = 8$ (next power of 2 greater than 5)
3. Evaluate $A(x)$ and $B(x)$ at 8 points using FFT
4. Multiply the results pointwise
5. Apply IFFT to obtain the coefficients of $C(x)$

5 Conclusion

Polynomial multiplication using FFT and IFFT provides a significant speedup over traditional methods, especially for large polynomials. This approach leverages the efficiency of the FFT algorithm to reduce the time complexity from $O(n^2)$ to $O(n \log n)$, making it invaluable for applications involving large-scale polynomial operations.

Problem 4. (20 points) How can you modify the polynomial multiplication algorithm based on FFT and IFFT to do multiplication of long integers in base 10? Make sure that you take care of carries in a proper way. Write your algorithm in pseudocode and give a brief explanation.

Solution. To modify the polynomial multiplication algorithm using FFT and IFFT for long integer multiplication in base 10, we need to: 1) Convert the integers to polynomials 2) Perform FFT-based multiplication 3) Handle the carries properly

Algorithm in Pseudocode:

```
Algorithm LongIntegerMultiply(a, b):
Input: Two long integers a and b in base 10
Output: Product c = a × b

1. // Convert integers to polynomial coefficients
   A[] ← DigitsToCoefficients(a)  // least significant digit first
   B[] ← DigitsToCoefficients(b)
   n ← NextPowerOf2(2 * max(len(A), len(B)))

2. // Pad arrays with zeros to length n
   PadWithZeros(A, n)
   PadWithZeros(B, n)

3. // Perform FFT-based multiplication
   FA[] ← FFT(A, n)
   FB[] ← FFT(B, n)
   FC[] ← PointwiseMultiply(FA, FB)
   C[] ← IFFT(FC, n)

4. // Handle carries
   result ← 0
   carry ← 0
   for i from 0 to length(C)-1:
       digit ← round(real(C[i])) + carry
       carry ← digit ÷ 10
       result ← result + (digit mod 10) × 10^i

5. return result
```

Explanation:

- The algorithm first converts each input integer into a polynomial representation where each digit becomes a coefficient.
- We use FFT of size n, where n is the next power of 2 greater than twice the maximum length of input numbers. This ensures no wrap-around in multiplication.

- After FFT multiplication, we use IFFT to get back the coefficient representation.
- The crucial modification is in the carry handling step:
  - We process the coefficients from least to most significant
  - For each position, we round to nearest integer (to handle FFT numerical errors)
  - We compute the carry by integer division by 10
  - The digit at current position is the remainder after division by 10
- The final result is constructed by combining the processed digits with appropriate powers of 10

This algorithm maintains the O(n log n) complexity of FFT-based multiplication while properly handling base-10 arithmetic operations.

Problem 5 (20 points)**.** Describe an efficient algorithm that, given a set

$$\{x_1, x_2, \ldots, x_n\}$$

of n points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

Why do we care? This is nice opportunity to design a simple greedy algorithm. Arguing the correctness of a greedy algorithm is essential, and this is not too difficult to do in this case.

Solution. We will develop a greedy algorithm to cover a set of points on the real line with the smallest number of unit-length closed intervals. Then, we will prove its correctness.

Algorithm: Greedy Get It

```
1: procedure CoverPoints(P)
2:     Sort points P in ascending order
3:     intervals ← ∅
4:     current_right ← −∞
5:     for each point p in P do
6:         if p > current_right then
7:             intervals ← intervals ∪ [p, p + 1]
8:             current_right ← p + 1
9:         end if
10:    end for
11:    return intervals
12: end procedure
```

Explanation

The algorithm works as follows:

1. Sort the input points in ascending order.
2. Initialize an empty set of intervals and set the rightmost covered point to negative infinity.
3. Iterate through the sorted points:
    - If the current point is not covered by the last interval (i.e., it's to the right of *current_right*), create a new interval starting at this point and extending one unit to the right.
    - Update *current_right* to be the right endpoint of the new interval.
4. Return the set of intervals.

Proof of Correctness

To prove the correctness of this algorithm, we need to show that it produces a valid covering and that this covering is optimal (uses the minimum number of intervals).

Lemma 1: The algorithm produces a valid covering.
Proof:

- Each point p in the input set P is covered by an interval.
- This is guaranteed because we only create a new interval when we encounter a point that is not covered by the previous interval.
- Each interval created always covers the point that triggered its creation.

Lemma 2: The covering produced by the algorithm is optimal.
Proof by contradiction:

- Assume there exists a covering with fewer intervals than our algorithm produces.
- Let the first difference occur at the k-th interval in our solution.
- This means the optimal solution must cover the point p_k that our algorithm uses to start the k-th interval with an earlier interval.
- However, our algorithm only starts a new interval when the current point is not covered by the previous interval.
- Therefore, no interval starting before p_k can cover p_k in a unit-length interval.
- This contradicts our assumption that the optimal solution covers p_k with an earlier interval.

Therefore, our algorithm produces an optimal covering.

Time Complexity

- Sorting the points takes $O(n \log n)$ time, where n is the number of points.
- The main loop iterates through each point once, taking $O(n)$ time.
- Thus, the overall time complexity is $O(n \log n)$, dominated by the sorting step.

Space Complexity

- The space complexity is $O(n)$ to store the sorted points and the output intervals.
- In the worst case, we might need an interval for each point.

Conclusion

The proposed greedy algorithm efficiently solves the problem of covering points with the minimum number of unit-length intervals. It has a time complexity of $O(n \log n)$ and a space complexity of $O(n)$. The algorithm's correctness is proven through its construction and the optimality of its solution.

Checklist:
☐ Did you add your name?
☐ Did you disclose all resources that you have used?
(This includes all people, books, websites, etc. that you have consulted)
☐ Did you sign that you followed the Aggie honor code?
☐ Did you solve all problems?
☐ Did you submit the pdf file of your homework?