Basic Algorithms (CS 301, Section 1), Professor Yap – Fall 2023

HOMEWORK SOLUTION

December 14, 2023

Homework 7 — Due: December 13, 2023

INSTRUCTIONS:

- Carefully read the standard Homework Instructions (Integrity Policy, justifying answers, etc) in hw1 or hw2.
- This is the last homework.
- ABSOLUTELY NO LATE SUBMISSIONS FOR THIS HOMEWORK. We will publish the solution immediately after due date.

QUESTIONS on Probability, Divide-and-Conquer, FFT, Hashing

> These topics are in **Lecture Slides 16, 17, 18, 19, 20, 21**.
> You may, if you like, look at corresponding chapters in my notes **Lecture VIII, II, XI** (approx)
> in ClassWiki→Schedule Page.

Q 1. (5+5+10+10 Points) Probability (Simulating Dice with Coins)

Professor X likes to play a dice game in his probability class, but has no dice. To simulate a dice roll, he asks three students to each toss a fair coin, yielding a binary number between 0 and 7. If 0 or 7 are tossed, the three coins are tossed again. The process is repeated until a number between 1 and 6 is tossed.

(a) The Craps Principle says: if A, B are disjoint events, then

$$\Pr(A|\ A \cup B) = \frac{\Pr(A)}{\Pr(A \cup B)}.$$

Note that we do not assume that $A \cup B$ are exhaustive, i.e., $\Pr(A \cup B) < 1$ is OK. This principle can be used to calculate the odds of a casino game called Craps.

SOLUTION:

$$\begin{aligned} \Pr(A|\ A \cup B) &= \Pr(A \cap (A \cup B))/\Pr(A \cup B) \\ &= \Pr(A)/(\Pr(A) + \Pr(B)) \qquad \text{(since } A, B \text{ are disjoint)} \end{aligned}$$

(b) Suppose you and I are playing a game in which there is one winner. Let A mean "I win", and B mean "you win". But we could also draw, or perhaps somebody else also playing the game could win. Suppose $\Pr(A) = 0.2$ and $\Pr(B) = 0.3$. Then if we are told that either you or I have won. What is the probability that I had won?

> **SOLUTION:** Applying the Craps Principle, I win with probability $\Pr(A|\, A \cup B) = \frac{0.2}{0.2+0.3} = 0.4$.

(c) Prove that Professor X's dice-from-coins simulation does produce a fair dice.

> **SOLUTION:** This is an application of the Craps principle: for $i = 1, \ldots, 6$, we have $\Pr\{i|1-6\} = \Pr\{i\} / \Pr\{1-6\}$. Since $\Pr\{i\} = 1/8$ and $\Pr\{1-6\} = 6/8$. Therefore, $\Pr\{i|1-6\} = 1/6$.

(d) What is the expected number of individual coin tosses needed to get a dice roll? Note that the number of coin tosses is always a multiple of 3.
HINT: Let C be the expected number of coin tosses. Write a recurrence for C (i.e., $C = \cdots C \cdots$) and solve.

> **SOLUTION:** If the expected number of coin tosses is C, then $C = 3 + \frac{1}{4}C$ since, after the first 3 coin tosses, there is $\frac{1}{4}$ chance that we have to repeat the process. Solving, $C = 4$.
> **Comments:** An alternative (more painful) solution is this: let N be the number of attempts until we get a number between 1 and 6. Since N is also a random variable, we can compute its expectation: $\mathtt{E}[N] = \sum_{i \geqslant 1} i \cdot \Pr(N = i) = \sum_{i \geqslant 1} i \cdot pq^{i-1}$ where $p = 3/4$ (probability of success) and $q = 1/4$ (probability of failure). It is well-known that $\sum_{i \geqslant 1} i \cdot pq^{i-1} = 1/p$. Thus $\mathtt{E}[N] = 1/p = 4/3$. Then $C = 3 \cdot \mathtt{E}[N] = 4$.

Q 2. (12+4+6+8 Points) Probability (Decision Process)

Consider the following "Decision Process" which consists of a sequence of steps. At each step, we roll a dice that has the usual possible outcomes: $1, 2, 3, 4, 5, 6$. In the i-th step, if the outcome is $\leqslant 2i$, we stop. Otherwise, we go to the next step.

For the first step, $i = 1$, we stop iff outcome is 1 or 2. Note that we will surely stop after the 3rd step. Let X be the random variable corresponding to the number of steps.

(a) What is the sample space Ω and the probability function Pr for this problem?
SOLUTION FIGURE:

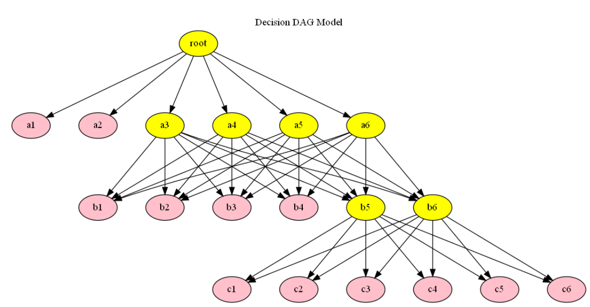


Figure 1: DAG Model for Sample Space

HINT: draw a DAG (directed acyclic graph) with **root node**, with 6 children $(a_1, a_2, \ldots, a_6)$ corresponding to the possibilities after the first step. The nodes $a_1, \ldots, a_6$ constitute **level 1**. In general, each level has 6 nodes, and edges goes from level i to level $i+1$. A node either has no children (terminal) or has 6 children of the next level. All the nodes at **level 3** are terminal. Each $\omega \in \Omega$ may be identified with the terminal nodes. We can assign a probability to each node in the DAG. This may be called a **decision DAG**.

SOLUTION: Consider the "decision DAG" as in Figure 1 where, starting from the root, we roll successive dice. We stop if the ith roll is $\leqslant 2i$. The terminal nodes in this DAG are shown in pink, and they constitute the sample space

$$\Omega = \{a_1, a_2, b_1, b_2, b_3, b_4, c_1, c_2, c_3, c_4, c_5, c_6\}$$

where $\Pr(a_i) = 1/6$, $\Pr(b_i) = (2/3)(1/6) = 1/9$ and $\Pr(c_i) = (2/3)(1/3)(1/6) = 1/27$.

Comments: The model in Figure 1 has $|\Omega| = 12$ sample points. You could also just give a model based on a tree (instead of a dag): begin with a root, it has 6 children corresponding to outcomes of $1, \ldots, 6$. But 1 and 2 are terminal, and for the non-terminal nodes, you again have 6 children. Thus there are $4 \times 6 = 24$ nodes at level 2. But $4 \times 4 = 16$ of these are terminal, and the final level will have $6 \times 2 \times 4 = 48$ nodes. Let Ω be the set of the terminal nodes. Then $|\Omega| = 66$.

(b) Describe the function $X : \Omega \to \mathbb{N}$.

SOLUTION: $X(a_i) = 1$, $X(b_i) = 2$ and $X(c_i) = 3$ for the various i's.

(c) Compute the expected value of X.

SOLUTION:

$$\begin{aligned}
\mathtt{E}[X] &= \Big(\Pr(a_1) + \Pr(a_2)\Big) + 2\Big(\sum_{i=1}^{4} \Pr(b_i)\Big) + 3\Big(\sum_{i=1}^{6} \Pr(c_i)\Big) \\
&= 1/3 + +2(4/9) + 3(6/27) = 17/9.
\end{aligned}$$

(d) Compute the variance of X.

SOLUTION:

$$\begin{aligned}
\mathtt{E}[X^2] &= 1(\Big(\Pr(a_1) + \Pr(a_2)\Big) + 4\Big(\sum_{i=1}^{4} \Pr(b_i)\Big) + 9\Big(\sum_{i=1}^{6} \Pr(c_i)\Big) \\
&= 1/3 + +4(4/9) + 9(6/27) = 37/9. \\
\mathtt{Var}(X) &= \mathtt{E}[X^2] - \mathtt{E}[X]^2 \\
&= 44/81.
\end{aligned}$$

Q 3. (8+12+10+15+15 Points) Divide-and-Conquer (Simulations)

Figure 2 illustrates how to simulate a divide-and-conquer algorithm like MergeSort: there is a tree representing the "divide" stages, and an inverted tree representing the "conquer" stages. These 2 trees are joined together at their leaves.

(a) (MergeSort) Please simulate MergeSort on the input

$$(5, 7, 3, 11; 35, 27, 46, 12; 1, 4, 8)$$

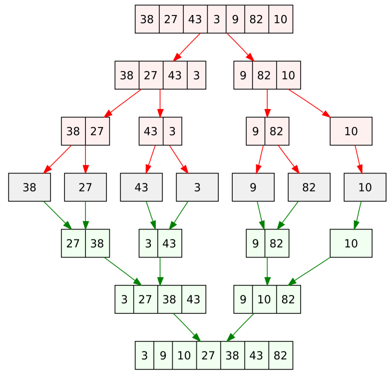


Figure 2: MergeSort simulation

You must use the following rule to "divide" a problem of size n: the first recursive call has size $\lceil n/2 \rceil$, the second has size $\lfloor n/2 \rfloor$.

MergeSort Solution in Figure 3

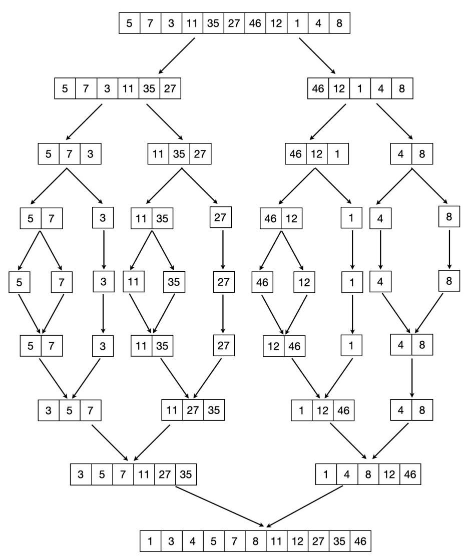


Figure 3: MergeSort of $(5, 7, 3, 11; 35, 27, 46, 12; 1, 4, 8)$

(b) (QuickSort) The QuickSort algorithm works by first calling a subroutine called **partition**. It first picks an arbitrary element p from the input; call this the **pivot element**. Then it splits the input into two lists, L and R where L contains those elements smaller than p, and R contains those larger than p. For simplicity, assume the elements are distinct. It recursively sorts L and R. See Figure 4 for simulation of QuickSort in which we always pick the last element of the array as the pivot.

Notice that we do not have a "conquer" stage in Figure 4. That is because we assume that the input is an array $A[0..n) = A[0..n-1]$, and the **partition** subroutine re-shuffles array A so that L is in $A[0..i)$ and R is in $A[i+1..n)$ and $A[i] = p$. After the recursive sorting of $A[0..i)$ and $A[i+1..n)$, the entire array is already

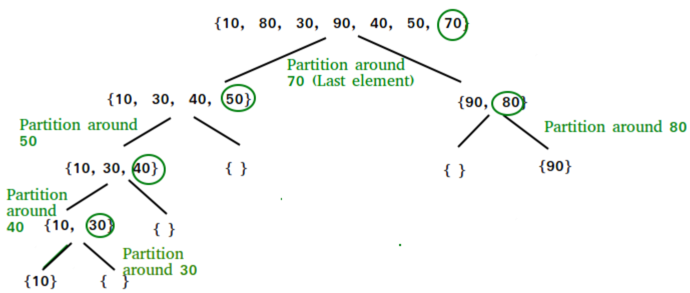


Figure 4: QuickSort simulation

sorted! We call this an **in place** algorithm because we do not need extra array, as the elements always remain in the original. This partition subroutine is said to be **stable** if the relative ordering of elements in $A[0..i)$ is the same as their original ordering, and similarly for $A[i+1..n)$. Note that Figure 4 above is not stable (but it does not affect the correctness of the algorithm). In any case, all recursive calls of QuickSort are on subarrays of the form $A[i..j)$ where $0 \leqslant i \leqslant j \leqslant n$. FINALLY, we can read off the sorting order of the elements using a **in-order traversal** of the nodes of the subdivision tree.

Please simulate QuickSort on the input

$$(5, 7, 3, 11; 35, 27, 46, 12; 1, 4, 8) \tag{1}$$

QuickSort Solution in Figure 5

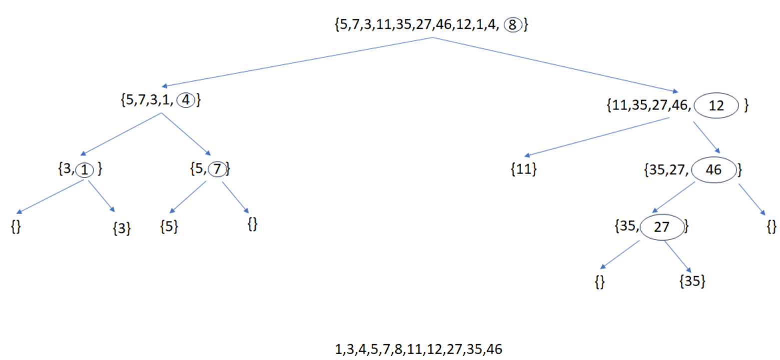


Figure 5: QuickSort of $(5, 7, 3, 11; 35, 27, 46, 12; 1, 4, 8)$

(c) The real Quicksort algorithm picks the pivot p randomly. Then we can prove that its expected running time is $O(n \log n)$. *But for our simulation purposes*, we always *pick the last element of* $A[i..j)$ *as pivot*. Please re-order the elements of (1) so that our Quicksort simulation has the worst possible performance. Assume that partition on $A[i..j)$ takes time $O(j - i + 1)$.

SOLUTION: Bingwei, please provide solution: I am not sure of this solution:

The sorted sequence is $(1, 3, 4, 5; 7, 8, 11, 12; 27, 35, 48)$. The worst case is

$$(1, 7, 5, 3; 4, 11, 48, 12; 35, 27, 8).$$

(d) Same as previous question, but re-order the elements so that the QuickSort performance is the best possible. Please explain how you created this answer.

SOLUTION: Bingwei, please provide solution.

(e) We had explained how to simulate Karatsuba's multiplication of Integers in class. This is illustrated in Figure 6.

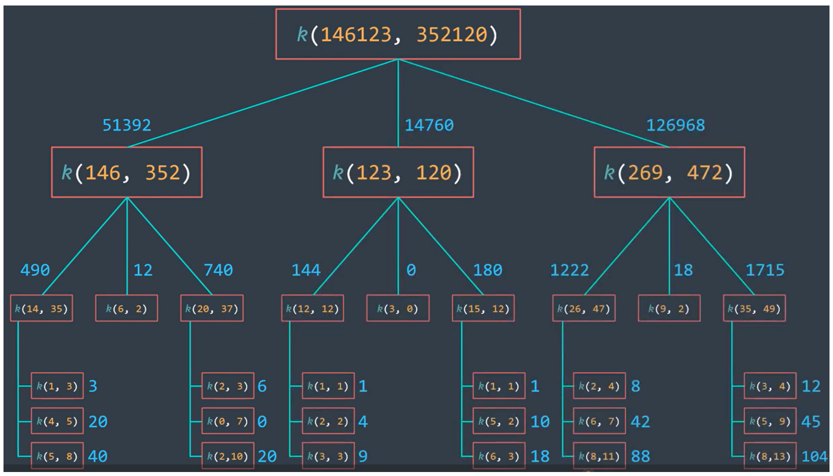


Figure 6: Karatsuba simulation

Please simulate Karatsuba's algorithm for multiplying $X = 123456$ and $Y = 78965$.

Karatsuba Solution in Figure 7

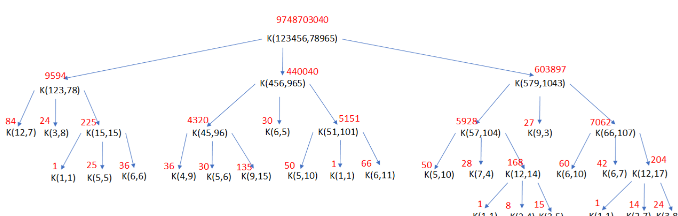


Figure 7: Karatsuba simulation for 123456×78965

Q 4. (4x5+10 Points) Master Theorem and Solving Recurrences

Please solve the following recurrences. Use the Master Theorem whenever possible.

(a) $T(n) = 2T(\frac{n}{4}) + 1$.

(b) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$.

(c) $T(n) = 2T(\frac{n}{4}) + n$.

(d) $T(n) = 2T(\frac{n}{4}) + n^2$.

(e) $T(n) = 2T(n-4) + n$.

PLEASE NOTE: To get full credit, when you use the Master Theorem, you MUST explicitly state the values of the constants a, b, e, f of the Master Theorem, and also which of the 3 CASES is used: $e = f$ or $e < f$ or $e > f$.

SOLUTION:

(a) $(a, b, e) = (2, 4, 0)$ so $f = \log_4 2 = \frac{1}{2}$. We have CASE $e < f$, so

$$T(n) = O(n^f) = O(n^{\frac{1}{2}}) = O(\sqrt{n}).$$

(b) $(a, b, e) = (2, 4, \frac{1}{2})$ so $f = \log_4 2 = \frac{1}{2}$. We have CASE $e = f$, so

$$T(n) = O(n^e \log n) = O(\sqrt{n} \cdot \log n).$$

(c) $(a, b, e) = (2, 4, 1)$ so $f = \log_4 2 = \frac{1}{2}$. We have CASE $e > f$, so

$$T(n) = O(n^e) = O(n).$$

(d) $(a, b, e) = (2, 4, 2)$ so $f = \log_4 2 = \frac{1}{2}$. We have CASE $e > f$, so

$$T(n) = O(n^e) = O(n^2).$$

SOLUTION: Part(e) This does not fit the Master Theorem template. As noted in our lecture, we can use the "rote" method, or the **EGVS method** to prove this result: $T(n) = \Theta(2^{n/4})$.

$$\begin{array}{rcll}
T(n) & = & 2\boxed{T(n-4)} + n & \text{(1st expansion)} \\
& = & 2\boxed{2T(n-8) + (n-4)} + n & \text{(2nd expansion)} \\
& = & 4\boxed{T(n-8)} + (3n-8) & \text{(simplification)} \\
& = & 4\boxed{2T(n-12) + (n-8)} + (3n-8) & \text{(3rd expansion)} \\
& = & 8\boxed{T(n-12)} + 7n - 40 & \text{(simplification)}
\end{array}$$

At this point you may guess that the recursive term is $2^i T(n-4i)$ after the ith expansion. But what to make of the term "$7n-40$"? THIS LOOKS DIFFICULT until you re-write it as a sum of the form

$$S_i := \sum_{j=0}^{i-1} 2^j (n - 4j).$$

This sum is $S_1 = n$, $S_2 = n+2(n-4) = 3n-8$, and $S_3 = n+2(n-4)+4(n-4\cdot 2) = 7n-40$. Thus, our formula agrees with the data when $i = 1, 2, 3$. We are ready to do the "Guess Part" of EGVS:

$$\begin{array}{rcll}
T(n) & = & 2^i \boxed{T(n-4i)} + \sum_{j=0}^{i-1} 2^j (n-4j) & \text{(ith expansion, GUESS!)} \\
& = & 2^i \boxed{2T(n-4(i+1)) + (n-4i)} + \sum_{j=0}^{i-1} 2^j (n-4j) & \text{($i+1$st expansion)} \\
& = & 2^{i+1} \boxed{T(n-4(i+1))} + \sum_{j=0}^{i} 2^j (n-4j) & \text{(simplification, VERIFIED!!)}
\end{array}$$

Note that this verification requires some actual computation! We have proved the guessed formula **by induction**. Finally, we must STOP the recursion by choosing i: if we choose $i = \lfloor n/4 \rfloor$ (note we need to take floor since i must be an integer), then $n - 4i < 4$. Then

$$T(n) = 2^{\lfloor n/4 \rfloor} T(n - 4i) + \sum_{j=0}^{\lfloor n/4 \rfloor - 1} 2^j (n - 4j)$$

Let the "**default initial condition**" be $T(n) = 0$ for $n < 4$. Then

$$T(n) = \sum_{j=0}^{\lfloor n/4 \rfloor - 1} 2^j (n - 4j) \tag{2}$$

But how to do this summation? For simplicity, assume n is a multiple of 4 so that $\lfloor n/4 \rfloor = n/4$. Then (2) becomes

$$T(n) = \sum_{j=0}^{n/4} 2^j (n - 4j) \tag{3}$$

We introduce a new variable k that is related to j via the identity

$$k \equiv (n/4) - j. \tag{4}$$

This means $j = 0 \leftrightarrow k = n/4$ and $j = (n/4) - 1 \leftrightarrow k = 1$. Moreover, $n - 4j = 4k$. Then (3) becomes

$$\begin{array}{rcll} T(n) & = & \sum_{k=1}^{n/4} 2^{(n/4)-k} 4k & \text{(from (3) using variable } k\text{)} \\ & = & 4 \cdot 2^{(n/4)} \sum_{k=1}^{(n/4)} k 2^{-k} & \\ & \leqslant & 4 \cdot 2^{(n/4)} \sum_{k=1}^{\infty} k 2^{-k} & \text{(summing to } k = \infty\text{)} \\ & = & 4 \cdot 2^{(n/4)} \cdot 2 & \text{(the build-heap identity)} \end{array}$$

The "build-heap identity" $\sum_{k=1}^{\infty} k 2^{-k} = 2$ was proved in p.7 of `4-pqueues.pdf` (in the analysis of build-heap algorithm). It is not hard to see that we have actually proved an asymptotically tight bound, namely, $T(n) = \Theta(2^{n/4})$ for all n.

Comments: Here is a simple proof of the build-heap identity: start from the another famous identity on variable x:

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k$$

(the "mother=of=series"). Differentiating by x,

$$\frac{1}{(1-x)^2} = \sum_{k=1}^{\infty} k x^{k-1}$$

Thus

$$\frac{x}{(1-x)^2} = \sum_{k=1}^{\infty} k x^k$$

The series is valid when $|x| < 1$. So we substitute $x = 1/2$ to get our build-heap identity

$$\frac{1/2}{(1-(1/2))^2} = \sum_{k=1}^{\infty} k 2^{-k}$$

Q 5. (8+6+4+10+12+15 Points) FFT over Finite Fields

We want to multiply two polynomials $f(x), g(x) \in F[x]$ where F is a field. Assume

$f(x), g(n)$ have degrees $< n$ where n is a power of 2. To use the FFT algorithm, we need an $\omega \in F$ that is a primitive nth root of unity. In this question, $F = \mathbb{Z}_{17}$ and we let $\omega = 3$ be a primitive 16th root of unity (see page 9, `18-polynom-fft.pdf`).

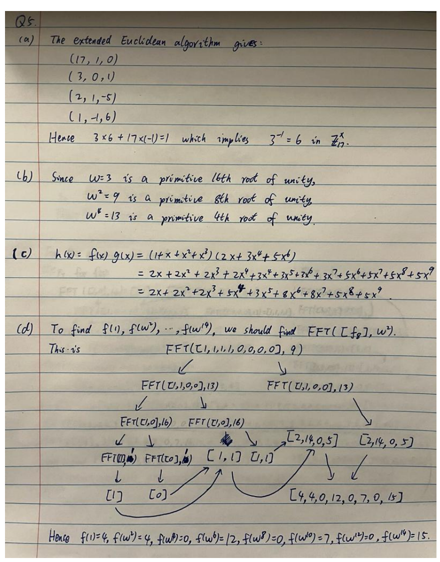
(a) To do the inverse FFT, we need the inverse ω^{-1}. Determine ω^{-1} from ω using the extended Euclidean algorithm. Show your steps.

> **SOLUTION:** Answer is $\omega^{-1} = 6$.
> COMPUTATION: The triples of the extended Euclidean algorithm are
>
> $$(17, 1, 0) \to (3, 0, 1) \to (2, 1, -5) \to (1, -1, 6)$$
>
> Hence $(-1) \cdot 17 + (6) \cdot 3 = 1$ or $(6) \cdot 3 \equiv 1(\mod 17)$. Thus $6 = 3^{-1}$.

(b) Please determine a primitive 8th root of unity. Determine a primitive 4th root of unity in $\mathbb{Z}_p$.

> **SOLUTION:** Answer: $\omega^2 = 3^2 = 9$ is a primitive 8th root of unity. $\omega^4 = 9^2 = 13$ is a primitive 4th root of unity. HAND COMPUTATION: $9^2 = (-8)^2 = 64 = 64 - 34 = 30 = 30 - 17 = 13$.
>
> Why is $order(\omega^2) = 8$? Let $x = \omega^2$. Then $x^2 = \omega^4$, $x^3 = \omega^6, \ldots, x^7 = \omega^{14}$ are all different from 1. But $x^8 = \omega^{16} = 1$. Hence $order(x) = order(\omega^2) = 8$. A smilar argument shows $order(\omega^4) = 4$.

(c) Let $f(x) = 1 + x + x^2 + x^3$ and $g(x) = 2x + 3x^4 + 5x^6$. Please comupte $h(x) = f(x)g(x)$ (using High School Algorithm).
Solution to parts(c-d) in Figure 8



Q5.

(a) The extended Euclidean algorithm gives:

(17, 1, 0)
(3, 0, 1)
(2, 1, -5)
(1, -1, 6)

Hence $3 \times 6 + 17 \times (-1) = 1$ which implies $3^{-1} = 6$ in $\mathbb{Z}_{17}^{\times}$.

(b) Since $w = 3$ is a primitive 16th root of unity,
$w^2 = 9$ is a primitive 8th root of unity,
$w^4 = 13$ is a primitive 4th root of unity.

(c) $h(x) = f(x)\, g(x) = (1 + x + x^2 + x^3)(2x + 3x^4 + 5x^6)$
$= 2x + 2x^2 + 2x^3 + 2x^4 + 3x^4 + 3x^5 + 3x^6 + 3x^7 + 5x^6 + 5x^7 + 5x^8 + 5x^9$
$= 2x + 2x^2 + 2x^3 + 5x^4 + 3x^5 + 8x^6 + 8x^7 + 5x^8 + 5x^9$.

(d) To find $f(1), f(w^2), \ldots, f(w^{14})$, we should find $FFT([f_8], w^2)$.
This is $FFT([1,1,1,1,0,0,0,0], 9)$

$FFT([1,1,0,0],13)$ $FFT([1,1,0,0],13)$

$FFT([1,0],16)$ $FFT([1,0],16)$ $[2,14,0,5]$ $[2,14,0,5]$

$FFT([1], 1)$ $FFT([0], 1)$ $[1,1]$ $[1,1]$

$[1]$ $[0]$ $[4,4,0,12,0,7,0,15]$

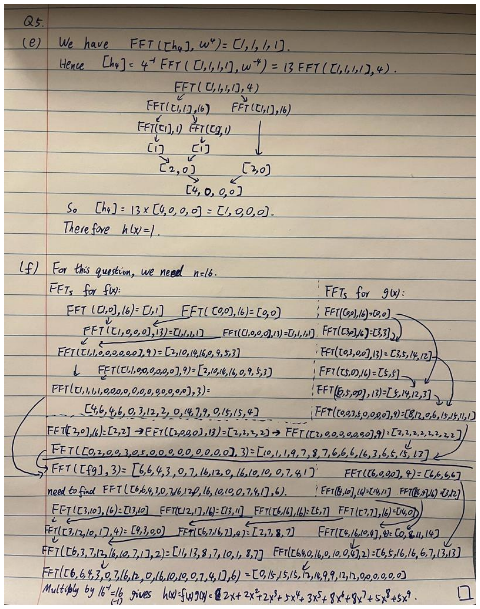
Hence $f(1)=4$, $f(w^2)=4$, $f(w^4)=0$, $f(w^6)=12$, $f(w^8)=0$, $f(w^{10})=7$, $f(w^{12})=0$, $f(w^{14})=15$.

Figure 8: (c) $f(x) \cdot g(x)$, (d) $FFT(f(x), \omega^2)$

(d) Please evaluate $f(x)$ at the points $\omega^0, \omega^2, \omega^4, \omega^6, \ldots, \omega^{14}$ using the FFT algorithm. Please show your simulation.

(e) Suppose $h(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$ is unknown. But you are told that the evaluation of $h(x)$ at each of the points $(\omega^0, \omega^4, \omega^8, \omega^{12})$ is $(1, 1, 1, 1)$. What is $h(x)$? NOTE: we are asking to solve the polynomial interpolation problem. You need the inverse FFT.

Solution to parts(e-f) in Figure 9


Q5.
(e) We have FFT([h_4], w^4) = [1,1,1,1].
Hence [h_4] = 4^{-1} FFT([1,1,1,1], w^{-4}) = 13 FFT([1,1,1,1], 4).
FFT([1,1,1,1], 4)
FFT([1,1],16) FFT([1,1],16)
FFT([1],1) FFT([1],1)
[1] [1]
[2,0] [2,0]
[4,0,0,0]
So [h_4] = 13 × [4,0,0,0] = [1,0,0,0].
Therefore h(x) = 1.

(f) For this question, we need n=16.
FFTs for f(x):
FFT([1,0],16) = [1,1] FFT([0,0],16) = [0,0]
FFT([1,0,0,0],13) = [1,1,1,1] FFT([1,0,0,0],13) = [1,1,1,1]
FFT([1,1,0,0,0,0,0,0],9) = [2,10,14,16,0,9,5,3]
FFT([1,1,0,0,0,0,0,0],9) = [2,10,14,16,0,9,5,3]
FFT([1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0],3) =
[4,6,4,6,0,3,12,2,0,14,7,9,0,15,15,4]
FFTs for g(x):
FFT([0,0],16) = [0,0]
FFT([3,0],16) = [3,3]
FFT([0,3,0,0],13) = [3,5,14,12]
FFT([5,0],16) = [5,5]
FFT([0,5,0,0],13) = [5,14,12,3]
FFT([0,0,3,5,0,0,0,0],9) = [8,12,0,6,15,15,11,1]
FFT([2,0],16) = [2,2] → FFT([2,0,0,0],13) = [2,2,2,2] → FFT([2,0,0,0,0,0,0,0],9) = [2,2,2,2,2,2,2,2]
FFT([0,2,0,0,3,0,5,0,0,0,0,0,0,0,0,0],3) = [10,1,1,9,7,8,7,6,6,6,16,3,6,5,15,13]
FFT([fg],3) = [6,6,4,3,0,7,16,12,0,16,10,10,0,7,4,1]
need to find FFT([6,6,4,3,0,7,16,12,0,16,10,10,0,7,4,1],6).
FFT([6,0,0,0],4) = [6,6,6,6]
FFT([4,10],16) = [14,11] FFT([16,9],16) = [3,12]
FFT([3,10],16) = [13,10] FFT([12,1],16) = [13,11] FFT([6,16],16) = [5,7] FFT([7,7],16) = [14,0]
FFT([3,12,10,1],4) = [9,3,0,0] FFT([6,7,16,7],4) = [2,7,8,7] FFT([4,16,10,4],4) = [0,8,11,14]
FFT([6,3,7,12,16,10,7,1],2) = [11,13,8,7,10,1,8,7] FFT([6,4,0,16,0,10,0,4],2) = [6,5,16,16,6,7,13,13]
FFT([6,6,4,3,0,7,16,12,0,16,10,10,0,7,4,1],6) = [0,15,15,15,12,14,9,9,12,12,0,0,0,0,0,0]
Multiply by 16^{-1} = 16 gives h(x) = f(x)g(x) = 2x + 2x^2 + 2x^3 + 5x^4 + 3x^5 + 8x^6 + 8x^7 + 5x^8 + 5x^9.

Figure 9: (e) Inverse FFT, (f) Simulation FFT polynomial multiplication.

(f) Simulate the FFT multiplication the polynomials $f(x), g(x)$ from part(c). Note that this should give the same answer as part(c).

> **SOLUTION:** See Figure 9.

Q 6. (20+25+5 Points) Multiplying Polynomials (FFT implementation)

Please study the Java program `PolyMul.java` that we provide for multiplying two polynomials using the FFT algorithm. This is a fully working program, except that we have body of the `FFT(...)` method for you to fill in.

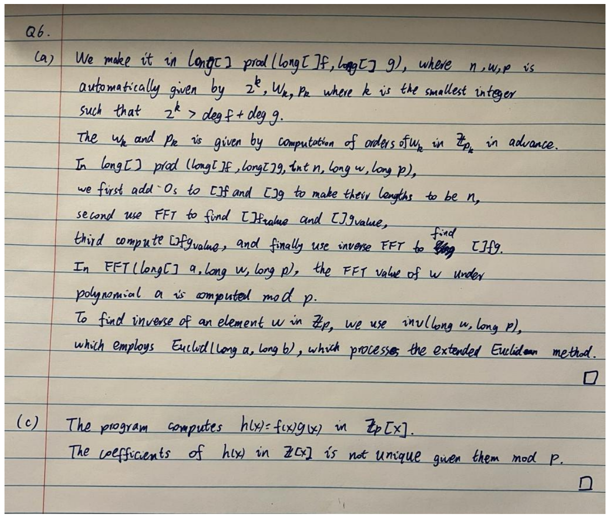
(a) `PolyMul.java` is able to multiply two integer polynomials of very high degrees (but there is a limit). Please explain how we do it.

Solution to part(a) in Figure 9

(b) Please do the programming part of this assignment: basically, you must fill in the body of the method called `FFT(int[] a, int w, int p)`. This method evaluates the input polynomial (represented by array `a`) at the powers of the primitive root `w` (mod `p`). If you do it correctly, then running the program in modes `mm=1,2,3` will reproduce an output that looks like the provided files `TESTOUTPUT-mm`.

> **SOLUTION:** See the `PolyMul.java` file in the solution zip folder.

(c) Given $f, g \in \mathbb{Z}[x]$, let $h = f \cdot g \in \mathbb{Z}[x]$ denote their product as integer polynomials. Does our program really compute h? Explain.



Q6.

(a) We make it in long[] prod(long[]f, long[] g), where n, w, p is automatically given by $2^k, w_k, p_k$ where k is the smallest integer such that $2^k > \deg f + \deg g$.
The w_k and p_k is given by computation of orders of w_k in $\mathbb{Z}_{p_k}$ in advance.
In long[] prod(long[]f, long[]g, int n, long w, long p),
we first add 0s to []f and []g to make their lengths to be n,
second use FFT to find []fvalue and []gvalue,
third compute []fgvalue, and finally use inverse FFT to find []fg.
In FFT(long[] a, long w, long p), the FFT value of w under polynomial a is computed mod p.
To find inverse of an element w in $\mathbb{Z}_p$, we use inv(long w, long p), which employs Euclid(long a, long b), which processes the extended Euclidean method. □

(c) The program computes $h(x)=f(x)g(x)$ in $\mathbb{Z}_p[x]$.
The coefficients of $h(x)$ in $\mathbb{Z}[x]$ is not unique given them mod p. □

Figure 10: Design of the `PolyMul.java`: idea is to precompute a list of (p_k, ω_k) $(k = 1, 2, \ldots)$ where p_k is a prime and ω_k has order 2^k. The rest is standard FFT multiplication.

SOLUTION: We need to choose a prime p_k from our list such that $\omega_k \in \mathbb{Z}_{p_k}$ has order 2^k with $2^k > \deg(fg)$. Thus we treat f, g as elements of $\mathbb{Z}_p[x]$, not $\mathbb{Z}[x]$. Then our output is some polynomial in $\widetilde{h} \in \mathbb{Z}_p[x]$. How can we ensure that $\widetilde{h}(x)$ is really the same as $h(x) = f(x)g(x)$? In general, they are different. SUPPOSE the following condition holds:

(C) The coefficients of $f(x), g(x), h(x)$ are all less than p_k.

Under (C), we can conclude that $\widetilde{h}(x) = h(x)$.

We can easily give a **sufficient condition** to ensure (C): First compute M, the maximum of the absolute values of the coefficients of $f(x)$ and $g(x)$. Clearly, we must ensure that $M < p_k$. If $D = \deg(f(x)g(x)$, then the coefficients of $h(x)$ is at most $(D+1)M^2$. To see this, let $f = \sum_{i \geqslant 0} f_i x^i$, $g = \sum_{i \geqslant 0} g_i x^i$ and $h = \sum_{i \geqslant 0} h_i x^i$. Then $h_i = \sum_{j=0}^{i} f_j g_{i-j}$. Hence $|h_i| \leqslant (i+1)M^2 \leqslant (D+1)M^2$. To summarize, if $(D+1)M^2 < p_k$ then condition (C) holds, and therefore our program would output $h = fg$.

Q 7. (20+10 Points) Perfect Hashing

Assume a hash family $\mathcal{H} = \{h_\lambda : \lambda \in \Lambda\}$ that is **universal**, and $h_\lambda : \mathcal{U} \to \mathbb{Z}_m$ where $m \geqslant n(n-1)$. Given a set $K \subseteq \mathcal{U}$, we want to find a **perfect hash function** $h \in \mathcal{H}$ for K. "Perfect" means h has no collisions on K.

Please refer to slides `21-hash3.pdf`. On page 2, we stated a result called the **Union Bound**:

$$\Pr[collision] \leqslant \frac{n(n-1)}{2m}.$$

The slides suggest this "Strategy"

> Repeat
>   Choose a random hash function h
>   If h is perfect for K, return h

to find a perfect hash function for

$$K = \{a_1, \ldots, a_n\}.$$

(a) Use pseudo-code to flesh out the above "Strategy" a little more, writing a method called

$$\texttt{perfect}(K, H, \Lambda)$$

where $|K| = n$, H is a 2-argument function $H(\lambda, a)$ such that for all $\lambda \in \Lambda$ and $a \in \mathcal{U}$, we have $H(\lambda, a) := h_\lambda(a)$. The method $\texttt{perfect}(K, H, \Lambda)$ return a $\lambda \in \Lambda$ such that h_λ is perfect for K.

SOLUTION:

```
perfect(K, H, Λ)
    Let T[1..m] be a boolean array.
    Repeat
        Initialize T[i] = false for all i = 1..m.
        Choose a random λ ∈ Λ
        For (i = 1..m) h is perfect for K, return h.
            For each a ∈ K
                if T[H(λ, a) = true,
                    break out of both for loops
                Return λ.
```

(b) Let T be the number of iterations inside the Repeat-loop. Note that T is a random variable. Give an upper bound on the expected value of T. **HINT**: set up a recurrence for T.

SOLUTION: By the Union Bound, $\Pr[\textit{collision}]$ is $< \frac{1}{2}$. Hence, $T \leqslant 1 + \frac{1}{2}T$. THe solution is $T \leqslant 2$.