CS 3510: Design & Analysis of Algorithms 01/19/2023

Problem Set 2: Divide & Conquer, RSA and FFT

Abrahim Ladha Due: 01/26/2023 11:59pm

- **Please TYPE your solutions using Latex or any other software. Handwritten solutions *won't* be accepted.**
- **Your solutions must be in plain English and mathematical expressions. Show the running time using the Master Theorem (wherever applicable).**
- **Unless otherwise stated, use** log **to the base 2.**
- **Unless otherwise stated, the fastest (and correct) algorithms will receive more credit. If we ask for a specific running time, a correct solution achieving it will receive full credit even if a faster solution exists.**
- **Do not use pseudocode. Your answer will receive zero credit even if the pseudocode is correct.**

Name: Mohamed Ghanem

1.) Divide and Conquer (20 points)

An alien landed on our planet from a parallel universe and brought a new definition of string equivalency. They say that two equal-length strings p and q are equivalent in one of the following way:

(a.) Strings p and q are equal.

(b.) If string p is split into two parts p_1 and p_2, each of the same length, and string q into two parts q_1 and q_2, each of the same length, then one of the following is correct:

  (a) p_1 is equivalent to q_1, and p_2 is equivalent to q_2.

  (b) q_1 is equivalent to p_2, and p_1 is equivalent to q_2.

As a CS 3510 student, help the alien design a Divide & Conquer algorithm that runs specifically in $\mathcal{O}(N \log N)$ time complexity (where N represents the length of the two strings) to determine if two given strings are equivalent in his/her world. Make sure to provide proof of the correctness of your algorithm and the time complexity (using Master's Theorem).

Examples:

Input: $p = aabb$, $q = bbaa$
Output: True
Explanation: The second condition in b. holds true.

Input: $p = abbb$, $q = bbba$
Output: True
Explanation: The second half of p is equal to the first half of q. For the first half of p, $p_1 = "ab" =' a' +' b'$ and the second half of q, $q_2 = "ba" =' b' +' a' =' a' +' b'$. Hence, p_1 is equivalent to q_2.

Input: $p = aabb$, $q = baab$
Output: False
Explanation: None of the conditions hold.

Solution:

The algorithm consists of two functions, a sorting function that determines the minimum lexicographical ordering of a string recursively, and a equivalence function that calls the sorting function on the two input strings p and q then checks if their results are equal to determine if p and q are alien equivalent strings. The sort function uses divide and conquer strategy to find the minimum lexicographical ordering of the input string by dividing it into two parts and sorting them recursively and then comparing the two parts to determine the minimum final ordering.

Proposed sort function:

1. If the length of the input string s is equal to 1, return s, it is in minimum lexicographical ordering.
2. If the length of the input string s is greater than 1, split s into two equal length parts and recursively call the sort function on each to obtain s_1 and s_2.
3. Recombine s in two ways, s_1s_2 and s_2s_1, then iterate through them comparing the strings character by character to determine which order is the minimum lexicographical ordering for s.
4. Return the minimum lexicographical ordering to previous recursive call.

Master Theorem:

$$T(n) = aT([n/b]) + \mathcal{O}(n^d) \quad a > 0,\ b > 1,\ d \geq 0$$

$$f(n) = \begin{cases} \mathcal{O}(n^d), & \text{if } d > \log_b a \\ \mathcal{O}(n^d \log n), & \text{if } d = \log_b a \\ \mathcal{O}(n^{\log_b a}), & \text{if } d < \log_b a \end{cases}$$

In the proposed sorting function:

1. The branching factor (a) is 2 because each sort function call makes 2 more recursive calls.
2. The size of the sub-problem (b) is 2 because at each sort call the input string length n is halved, the recursive call is made on an input size $n/2$.
3. The cost of recombination (d) is 1 because at every level the two halves of the string are appended and the lexicographical minimum string is determined, both appending and finding the minimum are $\mathcal{O}(n)$ operations.

$$a = 2 \quad b = 2 \quad d = 1$$

The recurrence equation:

$$T(n) = 2T([n/2]) + \mathcal{O}(n)$$

Solving the recurrence equation using Master Theorem:

$$\log_b a = log_2 2 = 1$$

$$d = \log_b a \rightarrow 1 = 1$$

Runtime of sorting:

$$\mathcal{O}(n^d \log n) \rightarrow \mathcal{O}(n \log n)$$

The overall runtime of the algorithm consists of 2 sort calls each with $\mathcal{O}(n \log n)$ time and a single string comparison which is $\mathcal{O}(n)$ time, therefore the overall time complexity is:

$$2\mathcal{O}(n \log n) + \mathcal{O}(n) \rightarrow \mathcal{O}(n \log n)$$

Proof of Correctness:
The sorting algorithm continuously splits the input string s into two strings until the base case of single characters is reached. One level above the base case, the recursive call returns the minimum lexicographical sorting of two characters. Every level up from the base case the return string size increases by a power of two, preserving the lexicographical sorting of its two calls.

This method of lexicographical sorting **does not** sort in alphabetical order. It preserves the sorting of every half, and every half of every half, and so on until individual characters. Using this any two alien equivalent strings can be identified if they are first recursively lexicographical sort then checked directly for character by character equivalence. Therefore this algorithm must be correct as it is able to determine if any two strings are alien equivalent strings.

Code:

```
def lexicographicalSort(s):
    if (len(s) == 1):
        return s
    s1 = lexicographicalSort(s[0:int(len(s) / 2)])
    s2 = lexicographicalSort(s[int(len(s) / 2):len(s)])
    return min(s1 + s2, s2 + s1)

def alienString(p, q):
    return (lexicographicalSort(p) == lexicographicalSort(q))
```

Solution:

Possible Alternate Solution (Letter Frequency):

1. If the then length of string p is not equal to the length of string q return False, the two strings cannot be equal with different lengths.
2. If the length of string p is 1 (single character), check if the character p is equal to the character q and return the result of the comparison (true if they match, else false).
3. Split p into two parts p_1 and p_2, each of the same length, and split string q into two

parts q_1 and q_2, each of the same length.

4. If all the characters in string p_1 are present in string q_1, return the "and" of two recursive calls. The first with inputs p_1 and q_1 and the second with inputs p_2 and q_2.
5. Else if all the characters in string p_1 are present in string q_2, return the "and" of two recursive calls. The first with inputs p_1 and q_2 and the second with inputs p_2 and q_1
6. If neither condition (4) or (5) are satisfied return False.

Master Theorem:

$$T(n) = aT([n/b]) + \mathcal{O}(n^d) \quad a > 0,\ b > 1,\ d \geq 0$$

$$f(n) = \begin{cases} \mathcal{O}(n^d), & \text{if } d > \log_b a \\ \mathcal{O}(n^d \log n), & \text{if } d = \log_b a \\ \mathcal{O}(n^{\log_b a}), & \text{if } d < \log_b a \end{cases}$$

In the proposed algorithm:

1. The branching factor (a) is 2 because each function call makes 2 more recursive function calls.
2. The size of the sub-problem (b) is 2 because at each function call the input string length n is halved, the recursive call is made on an input size $n/2$.
3. The cost of additional computational checks (d) is 1 because at every level there is at most four checks to see if a two strings consist of the same characters which is each an $\mathcal{O}(2n)$ operation.

$$a = 2 \quad b = 2 \quad d = 1$$

The recurrence equation:

$$T(n) = 2T([n/2]) + \mathcal{O}(n)$$

Solving the recurrence equation using Master Theorem:

$$\log_b a = log_2 2 = 1$$

$$d = \log_b a \rightarrow 1 = 1$$

Runtime of algorithm:

$$\mathcal{O}(n^d \log n) \rightarrow \mathcal{O}(n \log n)$$

Proof of correctness:
At any point during the recursion, for the algorithm to return true either $p1$ must be alien equivalent to $q1$ or $p1$ must be alien equivalent to $q2$. Instead of checking all the possible results we generate frequency maps for $p1$, $q1$, and $q2$, where every character maps to its frequency. If the frequency maps of $p1$ and $q1$ are not equal then there is no point in checking equivalency between $q1$ and $p1$ using a recursive call because we know two strings cannot be alien equivalent if they do not consist of the exact same characters. This ensures that

we are not making extra unnecessary recursive calls and that we are always checking the relevant alien string equivalency case. The algorithm covers all cases therefore it msut be correct. Code:

```
def alienStringEquivalency(p, q):
    if len(p) != len(q):
        return False
    if len(p) == 1:
        return p == q
    p1 = p[:len(p)//2]
    p2 = p[len(p)//2:]
    q1 = q[:len(q)//2]
    q2 = q[len(q)//2:]
    if Counter(p1) == Counter(q1):
        return alienStringEquivalency(p2, q2)
        and alienStringEquivalency(p1, q1)
    elif Counter(p2) == Counter(q1):
        return alienStringEquivalency(p2, q1)
        and alienStringEquivalency(p1, q2)
    return False
```

2.) Divide and Conquer (20 points)

Our wonderful TA, Bharat, has C cats and H cat houses where $C < H$. These cat houses are located in a straight line, and the positions are given as an array A of 1D coordinates (the coordinates are guaranteed to be in strictly increasing order i.e. A is given in a sorted order). Bharat is concerned about cat fights in his house, if the cats are placed too closely. To prevent the cats from hurting each other, Bharat wants to assign cats to their houses, such that the minimum distance between any two of them is as large as possible.

You are given a black-box algorithm called $placeCats(A, d, C)$. It takes in three parameters: the position array A, the distance d that you would like to test, and the number of cats C. In $O(H)$ time, it outputs a boolean indicating whether there's a placement of cats such that the minimum distance between any two cats from C cats is d.

Example:

Input: $H = 5$, $C = 3$, $A = [1, 2, 4, 8, 9]$
Output: 3

Explanation:
We want to choose 3 of the houses (for $C = 3$ cats) out of $[1, 2, 4, 8, 9]$ so that the minimum distance between any two of them is maximal. For this case, we can place the cats at position 1, 4 and 9, and the minimum distance between any of the two cats will be 3 as $min(4 - 1, 9 - 4) = 3$.
Why not 4? If we place the first cat at position 1, the second cat at position 8 and the third cat at position 9, we will get $min(8 - 1, 9 - 8) = 1$. Hence, there's no way we can get a minimum distance of 4.
Why not 2? If we place the first cat at position 2, the second cat at position 4 and the third cat at position 9, we will get $min(4 - 2, 9 - 4) = 2$. But, this is not the largest of all the minimum distances. Hence, 3 is the largest minimum distance that the cats can be placed.

As a CS 3510 student, please help Bharat design an algorithm that outputs the largest minimum distance that the cats can be assigned a house on the given location array A. The time complexity of your solution should be $O(H \log M)$ where H is the number of cat houses and M is the max difference of array A i.e. $max(A) - min(A)$. Make sure to provide the proof of correctness of your algorithm. Also, please give a brief intuitive justification for the runtime (proof using Master's Theorem is not required).

Hint: Compute the maximum separation from array A, and then try to optimize your search for distances to achieve the given time complexity.

Solution:

The algorithm is a binary search algorithm that finds the largest minimum distance that the cats can be assigned a house on the given location array A by searching across the solution space of possible distances.

Proposed Algorithm:

1. Initialize the result variable $result$ to -1, and two variables $left$ and $right$ to the 0 and the maximum separation between the elements in array A respectively.
2. A loop is used to iterate while $left$ is less than $right$.
3. In each iteration of the while loop, a variable mid is calculated as the floor of the sum of $left$ and $right$ by 2.
4. The function $placeCats$ is called with the parameters A, mid, and C. If the function call $placeCats(A, mid, C)$ returns True, the result variable $result$ is updated to the maximum of $result$ and mid and $left$ is incremented by 1.
5. If the function $placeCats(A, mid, C)$ returns False, $right$ is set to mid.
6. After the loop has completed, the final value of the $result$ variable is returned.

Runtime of algorithm:

$$\mathcal{O}(H \log(M))$$

The range from 0 to maximum separation is bounded by $M = max(A) - min(A)$ because M is the maximum difference between any two elements in the array A. The algorithm performs a binary search across a range $\leq M$, in each iteration, the $placeCats$ function is called which takes $\mathcal{O}(H)$ time. This means the overall runtime of the algorithm is, $\mathcal{O}(H \log(M)) = \mathcal{O}(H \log(M))$.

Proof of Correctness:

The array A is sorted, this means the largest possible solution is the separation between the last element in the array and the first element in the array, this is equal to M our upper bound in the solution space. In the worst case the smallest possible separation between any two indices in the array A (assuming distinct integers), that is a valid solution is 1, this is equal to the lower bound of our solution space. The The array A may not contain a solution for distance d, that is why the result is initialized to -1 which if returned indicates there is no valid solution. Let S be the solution space array we are performing binary search on, and i be the current solution we are checking. If S[i] is a solution there is no need to check $S[0], S[1]...S[i-1]$ because all those solutions will be smaller than our current solution, and we are trying to find the **largest** minimum distance. If S[i] is not a valid solution there is no need to check $S[i], S[i+1]...S[M]$ because all those are also not valid solutions. This designed algorithm must be correct because it uses binary search to explore all possible solutions for the largest minimum distance, therefore if there is a valid solution this algorithm will find it.

Code:

```
    def largestMinDist(A, H, C):
        res = -1
        left = 1
        right = A[H - 1] - A[0]
        while (left < right):
            mid = (left + right) // 2
            if (placeCats(A, mid, C)):
                left = mid + 1
                res = max(res, mid)
            else:
                right = mid
        return res
```

3.) Modular Arithmetic (10 points)

Use Fermat's Little Theorem to evaluate the following (all moduli are prime):

(a.) $100^{199} \pmod{199}$

(b.) $4^{63} \pmod{61}$

(c.) $9^{82} \pmod{163}$

(d.) $3^{176} \pmod{179}$

Solution:

Fermat's Little Theorem:
if p is prime, then for every $1 \leq a < p$,

$$a^{p-1} \equiv 1(\mod p)$$

(a.) $100^{199} \pmod{199}$

$$100^{198} \equiv 1(\mod 199)$$
$$100(100^{198}) \equiv 100(\mod 199)$$
$$100^{199} \equiv 100(\mod 199)$$
$$100(\mod 199) = 100$$

(b.) $4^{63} \pmod{61}$

$$4^{60} \equiv 1(\mod 61)$$
$$4^3(4^{60}) \equiv 4^3(\mod 61)$$
$$4^{63} \equiv 4^3(\mod 61)$$
$$4^3(\mod 61) = 64(\mod 61) = 3$$

(c.) $9^{82} \pmod{163}$

$$9^{82}(\mod 163) = 3^{2^{82}}(\mod 163) = 3^{164}(\mod 163)$$

$$3^{162} \equiv 1(\mod 163)$$
$$3^2(3^{162}) \equiv 3^2(\mod 163)$$
$$3^{164} \equiv 3^2(\mod 163)$$
$$3^2(\mod 163) = 9(\mod 163) = 9$$

(d.) $3^{176} \pmod{179}$

$$3^{178} \equiv 1(\mod 179)$$

$$3^{-2}(3^{178}) \equiv 3^{-2}(\mod 179)$$

$$3^{-2}(\mod 179) = 9^{-1}(\mod 179)$$

$$9^{-1}(\mod 179) = 20$$

This is solve by finding v, the multiplicative inverse of $9(\mod 179)$.

$$9v \equiv 1(\mod 179)$$

$$v = 20$$

$$9^{-1}(\mod 179) = 20$$

4.) RSA Cryptosystem (20 points)

Part A

Suppose Malav wants to send a secured email to Stephen using the RSA scheme. Help your TAs answer these questions.

(a.) Who should generate the RSA public and private keys? What are the quantities that the other party will need in order to generate an encrypted message?

(b.) Suppose the person generating the key chooses the primes $p = 29$ and $q = 7$ and computes $e = 9$. Suppose the message that Malav sent is $m = 9$, what's the encrypted message received by Stephen? You may use calculator for this part, but show your work for full credit.

(c.) Now suppose, Malav wants to send a secured email to both Stephen and Rohit. Normally, Malav would generate a unique public, private key pair for each, but he has bad randomness. Instead of generating four unique primes, two for each, he only generates three. He constructs the moduli for the two public keys as $N_1 = pq$ and $N_2 = qr$. The public exponents are chosen normally. Show this is insecure. Explain your response.

Solution:

(a.) Stephen should generate the public and private keys. Malav will need Stephen's public key to encrypt the email before sending it to Stephan. Stephen will then use his private key to decrypt the email from Malav.

(b.) Stephen generates public key (N, e), where $N = pq$.

$$\text{Public Key: } (203, 29)$$

Malav uses Stephen's public key to encrypt his email:

$$y = (m^e \mod N) = (9^9 \mod 203) = 64$$

The encrypted message y is sent to Stephen.

(c.) The prime q can be computed by taking the $gcd(N_1, N_2)$, which is a polynomial time algorithm. The primes p and r can then be determined respectively by $p = N_1/q$ and $r = N_2/q$. The decryption keys can then be computed: $d_1 = e_1^{-1} \mod ((p-1)(q-1))$ and $d_2 = e_2^{-1} \mod ((q-1)(r-1))$. This is insecure because with the decryption keys the emails being sent to Stephen and Rohit can be accessed.

Part B

Suppose there is a ballot initiave, a simple yes or no vote in your hometown. An RSA public, private key pair is generated of the form $(pk, sk) = ((N, e), d)$, and every vote is encrypted with the same key pair. These encrypted votes are of the form $[(\texttt{"yes"})^e \pmod N, \texttt{NAME}]$ or $[(\texttt{"no"})^e$

(mod N), `NAME`]. The vote itself is encrypted, but the name is not. They are sent over a public network to the vote counting server. Suppose you have access to every encrypted vote, including your own. Show this is insecure, and that you can tell exactly how everyone voted. Give a suggestion on how to fix it.

Solution:

The same $(pk, sk) = ((N, e), d)$ is used to encrypt the votes "yes" and "no" for all votes. This means the encryption $(\texttt{"yes"})^e \pmod N$ and $(\texttt{"no"})^e \pmod N$ will be the same for every vote. Assuming I vote "yes", I can use my vote to figure out the encryption of $(\texttt{"yes"})^e \pmod N$. I can then filter through all the votes extracting all the "yes" votes based on the encryption $(\texttt{"yes"})^e$, the remaining votes would be all the "no" votes. I can now find out the result of the ballot by counting the votes, I can also find out if someone specific voted "yes" or "no" because the encrypted votes contain the voter's name in appended in plain text. This is obviously insecure. It can be fixed by appending a large random number to the vote before encrypting it with known length. For example, $(\texttt{"yes" + 128-bit random number})^e \pmod N$, decrypt like you usually would then shift by 128-bits to get ride of the appended random number and to obtain the result of the vote.

5.) Fast Fourier Transform (FFT) (10 points)

In this question, we'll see a basic example of using FFT to multiply two polynomials. Our polynomials here will be $f(x) = 10 + 3x + x^2$ and $g(x) = 1 - 3x$. Recall that $i = \sqrt{-1}$, which means that the 4^{th} roots of unity are 1, i, -1, and $-i$ (each of these complex numbers raised to the 4^{th} power is 1). **Show your work for all parts.**

(a.) Compute the FFT (i.e. the "value representation") of $f(x)$ and $g(x)$ by evaluating them at the 4^{th} roots of unity. You can just leave your answer as a tuple of complex numbers, for example, the FFT of $2 + x$ would be $(3,\ 2 + i,\ 1,\ 2 - i)$. (No need to do the actual recursive method, simply evaluating by hand is fine).

(b.) Now, take the element-wise product of the two FFTs you computed above (that is, multiply element 1 of FFT($f(x)$) with element 1 of FFT($g(x)$), element 2 with element 2, etc.). Then, separately, compute the product of the two polynomials, $h(x) = f(x)g(x)$ (just with normal polynomial multiplication). Compute the FFT of $h(x)$. Does it match the element-wise product from earlier?

Solution:

(a.) For $f(x) = 10 + 3x + x^2$:

$$f(1) = 10 + 3 + 1 = 14$$
$$f(i) = 10 + 3i + (-1) = 9 + 3i$$
$$f(-1) = 10 - 3 + 1 = 8$$
$$f(-i) = 10 - 3i + (-1) = 9 - 3i$$
$$\text{FFT}(f(x)) = (14, 9 + 3i, 8, 9 - 3i)$$

For $g(x) = 1 - 3x$:

$$g(1) = 1 - 3 = -2$$
$$g(i) = 1 - 3i$$
$$g(-1) = 1 + 3 = 4$$
$$g(-i) = 1 + 3i$$
$$\text{FFT}(g(x)) = (-2, 1 - 3i, 4, 1 + 3i)$$

(b.) Element-wise product:

$$f(1)g(1) = 14(-2) = -28$$
$$f(i)g(i) = (9 + 3i)(1 - 3i) = 18 - 24i$$
$$f(-1)g(-1) = 8(4) = 32$$
$$f(-i)g(-i) = (9 - 3i)(1 + 3i) = 18 + 24i$$
$$\text{FFT}(f(x))\text{FFT}(g(x)) = (-28, 18 - 24i, 32, 18 + 24i)$$

Product of polynomials $h(x)$:

$$h(x) = f(x)g(x) = (10 + 3x + x^2)(1 - 3x)$$

$$h(x) = 10 - 30x + 3x - 9x^2 + x^2 - 3x^3$$

$$h(x) = -3x^3 - 8x^2 - 27x + 10$$

$$h(1) = -3 + -8 - 27 + 10 = -28$$

$$h(i) = 3i + 8 - 27i + 10 = 18 - 24i$$

$$h(-1) = 3 - 8 + 27 + 10 = 32$$

$$h(-i) = -3i + 8 + 27i + 10 = 18 + 24i$$

$$\text{FFT}(h(x)) = (-28, 18 - 24i, 32, 18 + 24i)$$

The FFT($h(x)$), the product of the polynomials matches FFT($f(x)$)FFT($g(x)$), the element wise product of FFT($f(x)$) and FFT($g(x)$).

$$\text{FFT}(f(x)g(x)) = \text{FFT}(f(x))\text{FFT}(g(x)) = (-28, 18 - 24i, 32, 18 + 24i)$$