CSCI 5454: Algorithms: Homework 3

Ashutosh Gandhi

February 18, 2023

Problem 1

1.1 Part A

$$
\begin{aligned}
p(x) &= \sum_{k=0}^{n-1} a_k x^k \\
&= a_0 + a_1 x^1 + a_2 x^2 + ... + a_{n/2-1} x^{n/2-1} + a_{n/2} x^{n/2} + a_{n/2+1} x^{n/2+1} + ... + a_{n-1} x^{n-1} \\
&= \sum_{k=0}^{n/2-1} a_k x^k + x^{n/2} [a_{n/2+0} x^0 + a_{n/2+1} x^1 + ... + a_{n/2+(n/2-1)} x^{n/2-1}] \\
&= \sum_{k=0}^{n/2-1} a_k x^k + x^{n/2} \sum_{k=0}^{n/2-1} a_{n/2+k} x^k \\
p(x) &= p_1(x) + x^{n/2} p_2(x)
\end{aligned}
$$

1.2 Part B

$$
\begin{aligned}
A_{2j} &= \sum_{k=0}^{n-1} a_k (\omega_n^{2j})^k \\
&= \sum_{k=0}^{n-1} a_k (\omega_{n/2}^{j})^k \\
&= a_0 + a_1 \omega_{n/2}^j + a_2 \omega_{n/2}^{2j} + ... + a_{n/2} (\omega_{n/2}^j)^{n/2} + a_{n/2+1} (\omega_{n/2}^j)^{n/2+1} + ... + a_{n-1} (\omega_{n/2}^j)^{n-1} \\
&= [a_0 + a_{n/2} (\omega_{n/2}^{n/2})^j] + [a_1 \omega_{n/2}^j + a_{n/2+1} (\omega_{n/2}^{n/2})^j (\omega_{n/2}^j)] + ... + [a_{n/2-1} (\omega_{n/2}^j)^{n/2-1} + a_{n-1} (\omega_{n/2}^j)^{n/2+n/2-1}] \\
&= [a_0 + a_{n/2}] + (\omega_{n/2}^j)[a_1 + a_{n/2+1}] + ... + (\omega_{n/2}^j)^{n/2-1} [a_{n/2-1} + a_{n-1} (\omega_{n/2}^{n/2})^j] \quad (\omega_{n/2}^{n/2} = 1) \\
&= [a_0 + a_{n/2}] + (\omega_{n/2}^j)[a_1 + a_{n/2+1}] + ... + (\omega_{n/2}^j)^{n/2-1} [a_{n/2-1} + a_{n-1}] \\
&= b_0 + b_1 \omega_{n/2}^j + ... + b_{n/2-1} (\omega_{n/2}^j)^{n/2-1} \quad (given \quad b_k = a_k + a_{n/2+k}) \\
&= \sum_{k=0}^{n/2-1} b_k (\omega_{n/2}^j)^k \\
A_{2j} &= B_j
\end{aligned}
$$

1.3 Part C

$$
\begin{aligned}
A_{2k+1} &= \sum_{j=0}^{n-1} a_k(\omega_n^{2k+1})^j \\
&= \sum_{j=0}^{n-1} a_k(\omega_n^{2k})^j\omega_n^j \\
&= \sum_{j=0}^{n-1} a_k(\omega_{n/2}^{k})^j\omega_n^j \\
&= a_0 + a_1(\omega_{n/2}^k)\omega_n^1 + ... + a_{n/2}(\omega_{n/2}^k)^{n/2}\omega_n^{n/2} + a_{n/2+1}(\omega_{n/2}^k)^{n/2+1}\omega_n^{n/2+1} + ... + a_{n-1}(\omega_{n/2}^k)^{n-1}\omega_n^{n-1} \\
&= [a_0 + a_{n/2}(\omega_{n/2}^{n/2})^k\omega_2] + [a_1(\omega_{n/2}^k)\omega_n^1 + a_{n/2+1}(\omega_{n/2}^{n/2})^k\omega_{n/2}^k\omega_n^{n/2}\omega_n^1] + ... \\
&\quad + [a_{n/2-1}(\omega_{n/2}^k)^{n/2-1}\omega_n^{n/2-1} + a_{n-1}(\omega_{n/2}^{n/2+n/2-1})^k\omega_n^{n/2+n/2-1}] \\
&= [a_0 - a_{n/2}] + \omega_{n/2}^k\omega_n^1[a_1 + a_{n/2+1}\omega_2] + ... \\
&\quad + [a_{n/2-1}(\omega_{n/2}^k)^{n/2-1}\omega_n^{n/2-1} + a_{n-1}(\omega_{n/2}^k)^{n/2-1}(\omega_{n/2}^k)^{n/2}\omega_n^{n/2-1}\omega_n^{n/2}] \\
&= [a_0 - a_{n/2}] + \omega_{n/2}^k\omega_n^1[a_1 - a_{n/2+1}] + ... + (\omega_{n/2}^k)^{n/2-1}\omega_n^{n/2-1}[a_{n/2-1} + a_{n-1}\omega_2] \\
&= \omega_{n/2}^0[a_0 - a_{n/2}] + \omega_{n/2}^k\omega_n^1[a_1 - a_{n/2+1}] + ... + (\omega_{n/2}^k)^{n/2-1}\omega_n^{n/2-1}[a_{n/2-1} - a_{n/2+(n/2-1)}] \\
&= c_0 + \omega_{n/2}^k c_1 + ... + (\omega_{n/2}^k)^{n/2-1}c_k \quad (given \quad c_j = (a_j - a_{n/2+k}) * \omega_n^j) \\
&= \sum_{j=0}^{n/2-1} c_k(\omega_{n/2}^k)^j \\
A_{2k+1} &= C_k
\end{aligned}
$$

1.4 Part D

procedure RECURSIVE-FFT(a)
    $n \leftarrow len(A)$ # n is power of 2
    if $n == 1$ **then** # base case goes here
    $\omega_n \leftarrow e^{2\pi i/n}$
    $\omega \leftarrow 1$
    $a^{[0]} \leftarrow (a_0, a_1, a_2, ..., a_{n/2-1})$
    $a^{[1]} \leftarrow (a_{n/2}, a_{n/2+1}, ..., a_{n-1})$
    $y^{[0]} \leftarrow Recursive - FFT(a^{[0]})$
    $y^{[1]} \leftarrow Recursive - FFT(a^{[1]})$
    for $k = 0 \quad to \quad n/2 - 1$ **do**
        $y_{2k} = y_k^{[0]} + y_k^{[1]}$ #proved in 1.b
        $y_{2k+1} = (y_k^{[0]} - y_k^{[1]}) * \omega$ #proved in 1.c
        $\omega \leftarrow \omega\omega_n$
    return y

The time complexity of the above algorithm is:

$$T(n) = 2T(n/2) + n/2$$
$$T(n) = O(nlog(n))$$

Problem 2

2.1 Part A

1. Recursively divide the given set of files into 2 halves each of similar length. (0 to n/2-1 and n/2 to n-1)
2. The base case is when there is only 1 file in that case return that file number
3. Compare the file from the left and right halves if they are the same then return the file number. The file comparison is done using rapid bitwise comparison.
4. Otherwise check if the number of occurrences of the left file in the left half and right file in the right half and return the left file if left count is more else return the right file. — $O(n)$
5. the file number returned by the above function is a possible candidate for a majority.
6. to confirm it, compare the file with each of the n files and check if the count is more than the majority of files.
7. print the file if the count is more than half the size of n else print none. — $O(n)$

The time complexity of the algorithm is:

$$T(n) = 2T(n/2) + O(2n) + O(n)$$
$$T(n) = 2T(n/2) + O(n) \quad \#from \quad master \quad theorem$$
$$T(n) = O(nlog(n))$$

2.2 Part B

1. create pairs (i, i+1) from the list of n files.
2. if both the files in the pair are the same then add one of them to a list of candidates
3. in case n is odd then discard one file and recurse over the remaining n-1 files.
4. Recursively computes the new set of candidates by forming pairs from the previous candidates.
5. the base case being when only one file is remaining in the set return it.

6. The file returned by the above function is a possible majority candidate, to confirm it counts the number of matches between the file returned and all the other files.

7. print the file if the count is more than half the size of n else print none.

The time complexity of the algorithm is:

$$T(n) = T(n/2) + O(n) + O(n) \quad \#solving \quad the \quad recurrence$$
$$T(n) = O(n)$$

Problem 3

(i) The time taken for each step is:

1. Sort the points using their x coordinates. — $O(nlog(n))$
2. Recursively carry out the following steps:
   (a) If the number of points is less than 10 just do pairwise comparisons. — C_0
   (b) Otherwise, split the list into two lists of size n/2 along the median x coordinate. — C_0
   (c) Recursively compute the closest pair for both halves. — $2T(n/2)$
   (d) Take the minimum of the two distances obtained from the two calls. — C_0
   (e) Sort the points in the "central strip" along the y coordinate. — $O(nlog(n))$
   (f) Iterate through the list and compare each point in the sorted list from the previous step to the eight previous points. — $O(n)$

The time complexity of the recursive step is:

$$T(n) = 2T(n/2) + O(nlog(n))$$
$$T(n) = O(n(log(n))^2)$$

The overall time complexity of the algorithm is:

$$T(n) = O(nlog(n)) + O(n(log(n))^2) \quad (1 \quad merge \quad and \quad recursive \quad part)$$
$$T(n) = O(n(log(n))^2)$$

(ii) an $O(nlog(n))$ algorithm

1. Sort the points using their x coordinates P_x. — $O(nlog(n))$
2. Sort the points using their y coordinates P_y. — $O(nlog(n))$
3. Recursively carry out the following steps:

(a) If the number of points is less than 10 just do pairwise comparisons. — C_0

(b) Otherwise, split the list into two lists of size n/2 along the median x coordinate $P_{xl}\&P_{xr}$. — C_0

(c) maintain the corresponding split in P_y. This can be done by comparing the x coordinates in P_y with the median x coordinate and calculating $P_{yl}\&P_{yr}$. — $O(n)$

(d) Recursively compute the closest pair for both halves. — $2T(n/2)$

(e) Take the minimum of the two distances obtained from the two calls. — C_0

(f) Iterate through the "central strip" list P_y and compare each point in the sorted list from the previous step to the eight previous points. — $O(n)$

The time complexity of the recursive step is:

$$T(n) = 2T(n/2) + O(n)$$
$$T(n) = O(nlog(n))$$

The overall time complexity of the algorithm is:

$$T(n) = O(nlog(n)) + O(nlog(n)) + O(nlog(n)) \quad (2 \quad merges \quad and \quad recursive \quad part)$$
$$T(n) = O(nlog(n))$$