Problem 1. (40 points)

Answer the following two questions about Karatsuba’s divide-and-conquer integer multiplication algorithm:

1. First, generalize Karatsuba’s algorithm to multiply an \( n \)-bit integer by an \( m \)-bit integer, where \( n \geq m \), in \( O(nm^{0.5849...}) \) time.

2. Second, consider this scenario. A student has been trying to speed-up Karatsuba’s divide-and-conquer integer multiplication algorithm. Given two numbers \( x, y \) with \( n \) bits each, her/his algorithm (1) first divides both \( x \) and \( y \) into four equal-length pieces, then (2) expresses the product \( x \cdot y \) using 4 multiplications of these \( n/4 \)-bit pieces, followed by a merging step that takes \( \Theta(n^p) \) where \( p > 1 \). What condition on \( p \) would give her/him a faster algorithm than the Karatsuba’s algorithm covered in class?

Answer:

1. First consider the case that \( n \) is divisible by \( m \). Break \( n \) into pieces long \( m \) bits. Each pairs of \( m \) bits costs \( O(m^{\log_2 3}) \), and there are \( n/m \) pieces. If \( n \) is not a multiple of \( m \), there will be a leftover of size \( O(m) \) that can be handled by an addition cost of \( O(m^{\log_2 3}) \).

2. The new divide and conquer algorithm has the following recurrence relation:

\[
T(n) = 4T(n/4) + n^p
\]

which is Master Theorem case III, \( n^p \in \Omega(n^{\log_4 4\epsilon}) \), given that \( p > 1 \) then \( \epsilon = p - \log_4 4 > 0 \). The second condition requires that \( af(n/b) \leq \delta f(n) \), that is \( 4(n/4)^p \leq \delta n^p \) that is \( \delta \geq 4^{1-p} \) if we pick \( \delta = 4^{1-p} < 1 \) as needed. The conclusion is that \( T(n) \in O(n^p) \). We need to find a condition on \( p \) such that this algorithm is asymptotically better than \( O(n^{\log_2 3}) \), i.e., we need \( p < \log_2 3 \approx 1.5849... \).

Problem 2. (30 points)

For an \( n \) that is a power of 2, the \( n \times n \) Weirdo matrix \( W_n \) is defined as follows. For \( n = 1 \), \( W_1 = [1] \). For \( n > 1 \), \( W_n \) is defined inductively by

\[
W_n = \begin{bmatrix}
W_{n/2} & -W_{n/2} \\
I_{n/2} & W_{n/2}
\end{bmatrix},
\]

where \( I_k \) denotes the \( k \times k \) identity matrix. For example,
\[
W_2 = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \quad W_4 = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & 1 & -1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad W_8 = \begin{bmatrix} 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 0 & 1 & -1 & 0 & -1 & 1 \\ 0 & 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & -1 & -1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}
\]

Give \( O(n \log n) \)-time algorithm that computes the product \( W_n \cdot \bar{x} \), where \( \bar{x} \) is a vector of length \( n \) and \( n \) is a power of 2.

**Answer:** Here is the pseudocode for this special matrix-vector product

**Algorithm** WPRODUCT \((n: \text{integer}, x[1...n]: \text{vector})\)

if \( n = 1 \) then return \( x \)
else
  \( x_L \leftarrow x[1...n/2] \)
  \( x_R \leftarrow x[n/2 + 1...n] \)
  \( U \leftarrow \text{WPRODUCT}(n/2, x_L) \)
  \( V \leftarrow \text{WPRODUCT}(n/2, x_R) \)
  \( y_L \leftarrow U - V \)
  \( y_R \leftarrow x_L + V \)
return \([y_L, y_R]\)

The recurrence relation for the time complexity is \( T(n) = 2T(n/2) + O(n) \) which has solution \( O(n \log n) \).

**Problem 3.** (30 points)

Given an array of numbers \( X = \{x_1, x_2, \ldots, x_n\} \), an exchanged pair in \( X \) is a pair \((x_i, x_j)\) such that \( i < j \) and \( x_i > x_j \). Note that an element \( x_i \) can be part of up to \( n - 1 \) exchanged pairs, and that the maximal possible number of exchanged pairs in \( X \) is \( n(n - 1)/2 \), which is achieved if the array is sorted in descending order. Give a divide-and-conquer algorithm that counts the number of exchanged pairs in \( X \) in \( O(n \log n) \) time.

**Answer:** As usual, we assume \( n \) to be a power of two, but this condition may be relaxed by adding special numbers to the array to get to the “next” power of two.

The main idea is that if we divide the array \( X \) into two halves, say \( L \) and \( R \), the total number of exchanged pairs in \( X \) is the sum of the number of exchanged pairs completely contained in \( L \) plus the number of exchanged pairs completely contained in \( R \) plus the exchanges \((x_i, x_j), i < j, x_i > x_j\) where \( x_i \in L \) and \( x_j \in R \) (let’s call these LR-exchanges). Clearly we can get the number of exchanges completely contained in \( L \) and \( R \) by calling
recursively our algorithm, but in order to meet the $O(n \log n)$ time bound we need to find a way to compute the number of LR-exchanges in linear time. The key observation is that if $L$ and $R$ are sorted, then a linear scan is enough to count them.

So our algorithm is a variant of MergeSort, where before we perform the actual merging we count in linear time the number of LR-pairs. The complexity is $O(n \log n)$ because the recurrence relation associated with our algorithm has the form $T(n) = 2T(n/2) + O(n)$. 