Problem 1. (10 points)

Give a divide-and-conquer algorithm for multiplying two polynomials of degree \( n \) in time \( O(n \log_2^3) \).

**Answer:** Suppose the two polynomials we want to multiply are \( A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1} \) and \( B(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{n-1}x^{n-1} \). We assume that \( n \) is a power of two (otherwise, we can always pad the coefficients with zeros to reach the "next" power of two). Let us break \( A(x) \) and \( B(x) \) into two polynomials as follows.

\[
A(x) = A_0(x) + x^{n/2}A_1(x)
\]
\[
B(x) = B_0(x) + x^{n/2}B_1(x)
\]

where

\[
A_0(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n/2-1}x^{n/2-1}
\]
\[
A_1(x) = a_{n/2} + a_{n/2+1}x + a_{n/2+2}x^2 + \ldots + a_{n-1}x^{n/2-1}
\]
\[
B_0(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{n/2-1}x^{n/2-1}
\]
\[
B_1(x) = b_{n/2} + b_{n/2+1}x + b_{n/2+2}x^2 + \ldots + b_{n-1}x^{n/2-1}
\]

Then the problem of multiplying \( A(x)B(x) \) can be decomposed in the problem of multiplying \( A_0(x), A_1(x), B_1(x), B_1(x) \) as follows. We omit "\((x)\)" to reduce the clutter.

\[
AB = (A_0 + x^{n/2}A_1)(B_0 + x^{n/2}B_1)
\]
\[
= A_0B_0 + x^{n/2}(A_0B_1 + A_1B_0) + x^nA_1B_1
\]
\[
= A_0B_0 + x^{n/2}((A_0 - A_1)(B_1 - B_0) + A_0B_0 + A_1B_1) + x^nA_1B_1
\]

Therefore, we need 3 multiplications of two polynomials of degree \( n/2 \) (namely, \( A_0B_0, A_1B_1 \) and \( (A_0 - A_1)(B_1 - B_0) \)) and \( O(n) \) additional work for the sum and the differences.

The recurrence relations is

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \\
3T(n/2) + O(n) & \text{if } n > 1
\end{cases}
\]

which can be solved using the Master Theorem, concluding that \( T(n) \in O(n \log_2^3) \).

Problem 2. (10 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

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

where \( W_{n/2} \) is the \( n/2 \times n/2 \) Weirdo matrix.
where $I_k$ denotes the $k \times k$ identity matrix. For example,

\[
W_2 = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad W_4 = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & 1 & -1 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \quad W_8 = \begin{bmatrix} 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 0 & 1 & -1 & 0 & -1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & -1 & -1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.
\]

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** ($n$: integer, $x[1...n]$: vector)

1. if $n = 1$ then return $x$
2. else
3. $x_L \leftarrow x[1...n/2]$
4. $x_R \leftarrow x[n/2 + 1...n]$
5. $U \leftarrow \text{WPRODUCT}(n/2, x_L)$
6. $V \leftarrow \text{WPRODUCT}(n/2, x_R)$
7. $y_L \leftarrow U - V$
8. $y_R \leftarrow x_L + V$
9. 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.** (10 points)

Given an array $A$ of $n$ (possibly negative) integers, find two indices $1 \leq i \leq n$ and $1 \leq j \leq n$ such that the value of $\sum_{k=i}^j a_k$ is maximized.

Write an $O(n \log n)$-time divide and conquer\footnote{A $O(n)$ dynamic programming algorithm for this problem exists, but here you are supposed to give the slower divide and conquer algorithm.} algorithm for the problem described above. The algorithm should return $i$ and $j$. If all elements of the array are negative, the algorithm should return $i = j = 0$.

**Answer:** The general strategy is based on divide and conquer.

Divide the array in two halves. Either the maximum is on the left half, or the maximum is on the right half, or the maximum is the sum of a region on the left half all the way to the center and a region on right half starting from the center.
Algorithm MAXSUB$(p, q)$
if $p + 1 = q$ then \{ if $A[p] < 0$ then return 0 else return $A[p]$ \} 
else \{ /* $p + 2 \leq q$ */ 
k $\left\lfloor \frac{(p + q)}{2} \right\rfloor$
lmax $\leftarrow$ MAXSUB$(p, k)$
rmx $\leftarrow$ MAXSUB$(k + 1, q)$
lbdmax $\leftarrow$ 0
lbd $\leftarrow$ 0
for $i$ $\leftarrow$ $k$ to $p$
  lbd $\leftarrow$ lbd $+$ $a[i]$
  if lbdmax $<$ lbd then lbdmax $\leftarrow$ lbd
rbdmax $\leftarrow$ 0
rbd $\leftarrow$ 0
for $i$ $\leftarrow$ $k$ $+$ $1$ to $q$
  rbd $\leftarrow$ rbd $+$ $a[i]$
  if rbdmax $<$ rbd then rbdmax $\leftarrow$ rbd
return max(lmax, rmax, lbdmax $+$ rbdmax)
\}

We solve the problem by calling the function MAXSUB$(1, n)$.

Time complexity: This algorithm has the same recurrence relation of Mergesort, which has solution $O(n \log n)$. 