Problem 1. (25 points [writing/solving recurrence relations])

Solve one (but NOT both) of the following problems. You are NOT required to verify the correctness of your solution.

1. Solve exactly (that is, without using any asymptotic notation) the following recurrence relation by iterative substitutions

\[ T(n) = \begin{cases} 
 1 & n = 1 \\
 3T\left(\frac{n}{2}\right) + 2 & n > 1 
\end{cases} \]

**Answer:** We have

\[
T(n) = 3T\left(\frac{n}{2}\right) + 2 = 3 \left(3T\left(\frac{n}{4}\right) + 2\right) + 2 = 3^2T\left(\frac{n}{4}\right) + 3 \cdot 2 + 2 = 3^3T\left(\frac{n}{8}\right) + 3^2 \cdot 2 + 3 \cdot 2 + 2 \\
\vdots
\]

\[
= 3^iT\left(\frac{n}{2^i}\right) + \left(3^{i-1} + 3^{i-2} + \ldots + 1\right) \cdot 2
\]

\[
= 3^iT\left(\frac{n}{2^i}\right) + 3^i - 1
\]

now we set \( n/2^i = 1 \) which is \( i = \log_2 n \) and we get

\[
T(n) = 3^{\log_2 n}T(1) + 3^{\log_2 n} - 1 = 2n^{\log_2 3} - 1
\]

2. Using the Master method, give an asymptotic tight bound for \( T(n) \) in the following recurrence relation

\[ T(n) = \begin{cases} 
 1 & n = 1 \\
 T\left(\frac{n}{2}\right) + n \log_2 n & n > 1 
\end{cases} \]

**Answer:** Case 3 of Master theorem applies. First note that \( n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \). The first condition for case 3 is \( n \log_2 n \in \Omega(n^\epsilon) \) which is satisfied for \( \epsilon = 1 \). The second condition is \( af(n/b) \leq \delta f(n) \), which translates to

\[
(n/2) \log_2 (n/2) = (n/2) \log_2 n - (n/2) \log_2 2
\]

\[
= (n/2) \log_2 n - (n/2)
\]

\[
\leq \delta n \log_2 n
\]

The last inequality is satisfied by \( \delta = 1/2 < 1 \).

The conclusion is \( T(n) = \Theta(n \log n) \).
Problem 2. (25 points [writing/solving recurrence relations])
Consider the following variant for the closest-pair algorithm we discussed in class.

Algorithm Closest-Pair (A : array of 2D points)
if |A| ≤ 2 then return the pair in O(1) time
sort set A by x coordinate
dL ← Closest-Pair(A[1...n/2])
dR ← Closest-Pair(A[n/2+1...n])
d ← min(dL,dR)
S ← {p ∈ A : |p.x − A[n/2].x| ≤ d} be the points in the vertical strip around the median
dS ← the closest-pair distance of points in S, computed in O(n log n) time
return min(d,dS)

Assume each basic operation on points (computing distance, comparing coordinates, . . . ) takes O(1) time. Define T(n) to be the worst-case running time of Closest-Pair(A) on any array A of n points. Write a recurrence relation for T(n) and give a tight bound on T(n).

Answer: The recurrence relation for T(n) is

\[ T(n) = 2T(n/2) + n \log n \] for n ≥ 3

and

\[ T(n) = 1 \] for n ≤ 2

By Master theorem Case 2 (k = 1) we have T(n) ∈ Θ(n log^2 n).

Problem 3. (25 points [divide & conquer])
Another version of Strassen’s algorithm uses the following identities. To compute

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix} = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \begin{bmatrix} I & J \\ K & L \end{bmatrix}
\]

first compute the following values:

\[
s_1 = G + H \\
s_2 = s_1 - E \\
s_3 = E - G \\
s_4 = F - s_2 \\
s_5 = J - I \\
s_6 = L - s_5 \\
s_7 = L - J \\
s_8 = s_6 - K
\]

\[
m_1 = s_2s_6 \\
m_2 = EI \\
m_3 = FK \\
m_4 = s_3s_7 \\
m_5 = s_1s_5 \\
m_6 = s_4L \\
m_7 = Hs_8 \\
m_8 = s_6 - K
\]

then

\[
A = m_2 + m_3 \\
B = m_1 + m_2 + m_5 + m_6 \\
C = m_1 + m_2 + m_4 - m_7 \\
D = m_1 + m_2 + m_4 + m_5
\]
1. Analyze the time complexity $T(n)$ of this new algorithm assuming that the input are two $n \times n$ matrices

2. Compare $T(n)$ against that of the original Strassen. Is it faster, slower, or the same?

Answer:

1. Since there are 7 multiplications and 18 additions, the recurrence relation is

$$T(n) = \begin{cases} 
  c & n = 2 \\
  7T \left( \frac{n}{2} \right) + 18n^2 & n > 2 
\end{cases}$$

2. The solution of the recurrence relation is $T(n) \in O(n^{\log_2 7})$, which is the same time complexity as Strassen’s.

Problem 4. (25 points [divide & conquer])

You are given an unsorted array $A[1 \ldots n]$ of $n$ distinct integers. We say that $A[i]$ is a local maximum if $A[i]$ is bigger than its neighbors, that is, $A[i] > A[i - 1]$ (if $i \neq 1$) and $A[i] > A[i + 1]$ (if $i \neq n$). For instance in $A = \{3, 4, 1, 2, 5, 6, 0, 8, 9\}$, 4 is a local maximum, 6 is a local maximum, and 9 is a local maximum. Give an $O(\log n)$-time algorithm to find any of the local maximum in $A$. If there is more than one local maximum in $A$, we are OK with any one of them. Explain briefly how the algorithm works, and why it runs on $O(\log n)$ time. You can assume that $n$ is a power of two. Hint: If $A[i]$ is not a local maximum because $A[i] < A[i + 1]$, then must there be some local maximum $A[j]$ with $j > i$?

Answer: The answer to the hint is “yes”, because if $A[i] < A[i + 1]$, then the first $j > i$ such that $A[j] > A[j + 1]$ or $j = n$ will be a local maximum. Likewise, if $A[i] < A[i - 1]$, then there must be a local maximum to the left of $i$. Thus, we can do binary search over $\{1, 2, \ldots, n\}$ to find a local maximum. Here is pseudo-code:

Algorithm findPeak($A$)
\begin{align*}
  &\text{if } n = 1 \text{ then return } A[1] \\
  &\quad \quad i \leftarrow \lfloor n/2 \rfloor \\
  &\text{if } A[i] < A[i + 1] \text{ then return findPeak($A[i + 1 \ldots n]$)} \\
  &\text{if } A[i] < A[i - 1] \text{ then return findPeak($A[1 \ldots i - 1]$)} \\
  &\text{return } i
\end{align*}

Each recursive call takes constant time and cuts the problem size in half, so the total time is $O(\log n)$. 