1. This innocent-looking piece of dynamic programming code does a surprising task:

   Input: Array $A$ of size $n$
   Set $x = 0$ and set $y = 0$
   For $i = 1$ to $n$ do
     $x = \max(x + A[i], 0)$
     $y = \max(x, y)$
     Print $A[i]$, $x$, and $y$
   Return $y$

   (2 pts) (a) Trace the above algorithm on the input $A[1], \ldots, A[6] = (-2, 2, -1, 3, -6, 1)$, printing out the values of $A[i]$, $x$ and $y$ whenever the line “Print $A[i]$, $x$, and $y$” occurs. Do this in a table with columns labeled by values of $i$, and rows labeled by $A[i]$, $x$, and $y$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$A[i]$</th>
<th>$x$</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

   (2 pts) (b) Briefly describe what this algorithm does on a general array of integers. (hint: test on other arrays if necessary.)

   This algorithm finds the largest sum of any subsequence of $A[1], A[2], \ldots, A[n]$.

   (2 pts) (c) Interpret the value printed for $y$ when $i = 2$.

2. The algorithm Bubble Sort sorts a list of integers by successively comparing the \( i \)th number to the \((i + 1)\)st number in the list and interchanging them if the \( i \)th number is larger. Here is the full algorithm:

**Bubble Sort**

- **Output**: A listing of \( \{A[1], A[2], \ldots, A[n]\} \) in increasing order.

Step 1. Set \( m = n - 1 \).
Step 2. For \( i = 1, 2, \ldots, m \), if \( A[i] > A[i + 1] \), interchange \( A[i] \) and \( A[i + 1] \).
   Print \( A[1], A[2], \ldots, A[n] \).
Step 3. Decrease \( m \) by 1. If \( m \neq 0 \), return to Step 2. If \( m = 0 \) stop and return \( A[1], \ldots, A[n] \).


\[
\begin{array}{c|c|c|c|c|c}
  \hline
  4 & 7 & 8 & 4 & 1 & 10 \\
  3 & 7 & 4 & 1 & 8 & 10 \\
  2 & 4 & 1 & 7 & 8 & 10 \\
  1 & 1 & 4 & 7 & 8 & 10 \\
\end{array}
\]

(3 pts) (b) Determine the worst-case time complexity of Bubble Sort. Be sure to fully justify your answer.

The algorithm requires \( n - 1 \) comparisons the first time Step 2 is executed, \( n - 2 \) comparisons the second time, etc. So the total number of comparisons is \( (n - 1) + (n - 2) + \cdots + 1 \). It is well known that this sum is equal to \( \frac{n(n-1)}{2} \). So Bubble Sort is \( O(n^2) \).
3. Suppose that you are given a sorted sequence of distinct integers $A = (a_1, a_2, \ldots, a_n)$. Give a divide-and-conquer $O(\log n)$ algorithm in pseudocode to determine if there exists an index $i$ such that $a_i = i$. For example, in $(-10, -3, 3, 5, 7)$, $a_3 = 3$. In $(2, 3, 4, 5, 6, 7)$ there is no such $i$.

(hint: test the middle element, and break into cases.)

Note that since the $a_i$'s are all distinct, if $a_i > i$ for some $i$ then $a_j > j$ for all $j > i$. Similarly if $a_i < i$ for some $i$ then $a_j < j$ for all $j < i$. So use binary search as follows: first check if the middle element satisfies $a_i = i$. If so, return $a_i = i$. If not and $a_i > i$, repeat the search on the left half of the list and if $a_i < i$, repeat the search on the right half of the list. The algorithm looks like:

\[
\begin{align*}
    l &= 1 \\
    r &= n \\
    \text{Find}(A, l, r) \\
    m &= \left\lfloor \frac{l + r}{2} \right\rfloor \\
    \text{if } a_m = m \text{ then return}(a_m = m) \\
    \text{if } a_m < m \text{ then } l = m + 1 \\
    \text{if } a_m > m \text{ then } r = m - 1 \\
    \text{if } r < l \text{ return}(\text{"no index } i \text{ such that } a_i = i\) \text{ else } \text{Find}(A, l, r)
\end{align*}
\]