Problem 1. [**A variant of Partition**.]
Design a variant of the *Partition*(A, p, r) procedure that runs in time $O(r - p + 1)$ and divides the input array $A[p \ldots r]$ into three subarrays $A[p, \ldots, q-1]$, $A[q, \ldots, s-1]$ and $A[s, \ldots, r]$ such that each element of $A[p, \ldots, q-1]$ is strictly less than $A[q]$, the elements $A[q, \ldots, s-1]$ are identical in value to each other, and each element of $A[s, \ldots, r]$ is strictly greater than $A[q]$. The subarrays $A[p, \ldots, q-1]$ and $A[s, \ldots, r]$ may be empty, depending on the input values. Call this procedure *New_Partition*. This procedure returns the pair (q, s), where, $p \le q < s \le r+1$.

Explain the loop invariant satisfied by your algorithm, prove that it holds initially and is maintained after each iteration and write pseudo-code. The time complexity should be $\Theta(r - p + 1)$ and you should make a single pass over the array $A[p, \ldots, r]$. (i.e., do not run the textbook *Partition* and then make a second pass over the array).

Problem 2. Maximum contiguous subarray sum problem You are given an array $A[1, \ldots, n]$ consisting of positive and/or negative numbers. Given indices $1 \le i \le j \le n$, the contiguous subarray sum $A[i, \ldots, j]$ is $\sum_{k=i}^{j} A[k]$. Over the possible $n(n+1)/2$ possible values of i, j, find the pair (i, j) which has the largest contiguous sub-array sum. Return i, j and the subarray sum.

(*Note 1:* You can use divide and conquer. Divide a given array $A[l, \ldots, h]$ into two halves $A[l \ldots, mid]$ and $A[\ mid{+}1, \ldots, h]$. Recursively, find the maximum contiguous subarray sum in the left half and in the right half. Now, find the maximum contiguous subarray sum ending in $A[mid]$ and the maximum contiguous subarray starting at $mid + 1$ and return their sum. This should give an $O(n \log n)$ time algorithm.)

(*Note 2:* A better (non-recursive) algorithm can be designed as follows. For $1 \le j \le n$, let $C[j]$ denote the sum of the maximum contiguous sub-array starting at some point at or prior to j till j. Then, show that

$$C[j+1] = \max(C[j] + A[j+1], C[j+1])$$

that is, either we add the current element $A[j+1]$ to the maximum subarray ending at j, or we start a new subarray at $j+1$, whichever has larger sum. This is the dynamic programming method, and gives an $O(n)$ time solution to the problem.)

Problem 3. Longest increasing contiguous subarray. Given an array $A[1, \ldots, n]$, a subarray $A[p \ldots q]$ is said to be an increasing contiguous subarray if $A[p] < A[p+1] < A[p+2] < \ldots < A[q]$ and is of length $q - p + 1$. The problem is to find the length of the *longest* increasing contiguous subarray. (*Note*: A divide and conquer approach similar to the maximum contiguous subarray sum problem can be designed to work in $O(n \log n)$ time.)

(*Note 2*: For $1 \le i \le n$, let $L[i]$ denote the length of the longest increasing contiguous subarray ending at i. Then, the following recurrence equation holds $L[i+1] = L[i] + 1$ if $L[i+1] > L[i]$; otherwise, $L[i+1] = 1$ (corresponding to the singleton subarray $[i+1, ..., i+1]$. This dynamic programming algorithm takes time $\Theta(n)$.

Problem 4. Divide and Conquer: Monge Arrays [Problem 4-6 from CLRS.] An $m \times n$ array A of real numbers is a *Monge array* if for all i, j, k and l such that $1 \le i < k \le m$ and $1 \le j < l \le n$, we have,

$$A[i, j] + A[k, l] \le A[i, l] + A[k, j] \ .$$

In other words, whenever we pick two rows and two columns of a Monge array and consider the four elements at the intersections of the rows and columns, the sum of the upper-left and lower-right elements is less than or equal to the sum of the lower-left and upper-right elements.

a. Prove that an array is Monge if and only if for all $i = 1, 2, \ldots, m-1$ and $j = 1, 2, \ldots, n-1$, we have,

$$A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j]$$

(*Hint*: For the "if" part, use induction separately on rows and columns.)

b. Let $f(i)$ be the index of the column containing the leftmost minimum element of row i. Prove that for any $m \times n$ Monge array, $f(1) \leq f(2) \leq \cdots \leq f(m)$.

c. The following describes a divide-and-conquer algorithm that computes the leftmost miminum element in each row of an $m \times n$ Monge array A:

> Construct a submatrix A' of A consisting of the even-numbered rows of A. Recursively determine the leftmost minimum for each row of A'. Then compute the leftmost minimum in the odd numbered rows of A.

Explain how to compute the leftmost minimum in the odd-numbered rows of A (given that the leftmost minimum of the even-umbered rows is known) in $O(m + n)$ time.

d. Write the recurrence describing the running time of the algorithm described in part (d). Show that its solution is $O(m + n \log m)$.