Problem 1: Suppose you are given a set P of n points in the plane. A point p ∈ P is maximal in P if no other point in P is both above and to the right of p. Intuitively, the maximal points define a "staircase" with all the other points of P below it.

Algorithm:
First we will define the coordinates into x and y and make them into an array so that we can go through all the data points. Let's say an array of P[$(X_i, Y_i),(X_{i+1}, Y_{i+1})\ldots\ldots(X_n, Y_n)$]

Step 1] Since there are many coordinated we can do a merge sort with respect to the X-coordinates and their related Y-coordinates so that we can keep a track of which X is related to which Y

Step 2] Noting the largest X- coordinates we will keep their Y-coordinates saved in a temporary variable.

Step 3] If we have more than one same X-coordinates, we will compare their Y-coordinates and take the large one. If we have the same Y –Coordinates we will compare their X-coordinates and select the large one.

Analysis:

Since we sorting the coordinates using the merge sort the time complexity would be O(n log n). Since I am going through the entire array it would have a time complexity of O(n).

T(n)=O(n.log n) + O(n) , removing the constants I will get O(n log n), hence solved.

Problem 2: Call a sequence X[1 · · n] of numbers bitonic if there is an index i with 1 < i < n, such that the prefix X[1·· i] is increasing and the suffix X[i·· n] is decreasing. Describe an O(logn) time algorithm to search a bitonic sequence of length n for a number k.

Like we learnt in the class about longest increasing subsequence. Start with diving the arrays into two parts as shown in the question and then find the length of the largest increasing sequence and then the length of the largest decreasing sequence. The result will be sum of these two created arrays minus one.

Algorithm:

```
A[1..n]                 // will be the array
LIS(A[1..n])            //return length of the largest increasing sub sequence

for i=1 to n
```

for j=0 to I

if((A[i]>A[j])&&(LIS[i]<LIS[j]))

LIS[i]=LIS[j+1]

End if
End for
End for
for i=n-2 to 0

for j=n-i to i-1

if ((A[i] > A[j]) and (LDS [i] < LDS [j])

LDS[i] = LDS[j] +1

End if
End for
End for

Return Max(LIS[i]+LDS[i]-1)

Analysis:
This program would not return a time complexity of O(log n) but it would return time complexity as $O(n^2)$.

Reference: http://www.geeksforgeeks.org/dynamic-programming-set-15-longest-bitonic-subsequence/

Problem 3: Call a sequence X[1 · · n] of numbers oscillating if X[i] < X[i + 1] for all even i, and X[i] > X[i + 1] for all odd i. Describe an efficient algorithm to compute the length of the longest oscillating subsequence of an arbitrary array A of integers.

Algorithm:

I may not be entirely correct but as much I have understood the algorithm starts from running through the array lets say an A[1..n] then the current number let say A[i] is less then A[i+1] for all even i in the array.
And the same will be for odd numbers but there A[i] would be greater than the next odd value A[i+1] .
Then we have to find the maximum length of the longest oscillating sequence in the array which can be either odd or even.