Assignment 1: Difficult: Non Dominated Points

Solution:

Algorithm:
Let us consider all the points in set P. First, we divide P into two parts by the median of x-coordinate of all the points. Each part will have n/2 points each. Let us call left half as P_l and right half as P_r. Then, we find the point y_{max} with maximum y coordinate in P_r. y_{max} has to be a non-dominated point in P_r as discussed in method 1 in lectures, and because every non-dominated point in P_r has to be a non-dominated point for P_l, y_{max} turns out to be a non-dominated point for the whole set of points P. We then proceed to remove all the points from set P which are dominated by the point y_{max} as they cannot be non-dominated points. Then recursion step is followed to repeat the same algorithm on P_l and P_r to find non-dominated points in both sets. The final set of non-dominated points has to be y_{max}, non-dominated points in P_l and non-dominated points in P_r.

Pseudo Code:

```
NonDomPoints(P) {
    if P = φ : return φ
    if |P| = 1 : return P
    set ans = φ
    Let x_med be the median of x-coordinate of the points in P ……………………….O(n)
    Partition P into two sets P_l & P_r about median x_med    ……………………….O(n)
        Let P_l be all the points with x-coordinate ≤ x.
        Let P_r be all the points with x-coordinate > x.
    Let y_max be the point in P_r with the maximum y-coordinate ……………………….O(n)
    ans = ans ∪ y_max
    Delete every point dominated by y_max in P_l          ……………………….O(n)
    Delete every point dominated by y_max in Pr           ……………………….O(n)
    L = NonDomPoints(P_l)         ……………………….…T(n/2,h1)
    R = NonDomPoints(P_r)         ……………………….…T(n/2, h-1-h1)
    ans = ans ∪ L ∪ R
    return ans
}
```

Correctness :
We need to show that this algorithm correctly identifies all the non dominated points. This is a divide & conquer based algorithm. Assuming that the non dominated points in left & right subpart are correctly identified. At each step we are first removing the point y_{max} with max y coordinate in P_r. And after removing all the points dominated by y_{max} from P_l & P_r, we are running this algorithm separately for P_l & P_r. This ensures that at each step of recursion we add one point in our answer set unless the set is empty. Since we have removed all points dominated by y_{max} beforehand, we ensure that the recursion step for P_l doesn't return any point which will be dominated by any point in P_r. After recursion for P_r & P_l we'll have all non dominated points form P_l & P_r. Thus, at each step we get an exhaustive set of all non dominated points in a given set of points.

Time Complexity:

To get an intuition that time complexity has to be O(nlogh), consider the recurrence tree used to calculate time complexity in case of O(nlogn) solution. At each step we used to divide n into two parts of n/2 and we keep doing this till we have one or no points at the end. The time complexity comes out to be O(nlogn) because at each level of tree, O(n) time was spent and the height of tree was logn. In the case of this algorithm, at each recursion, we ensure that we have one non-dominated point discovered, as well as the points dominated by it. So, each recursion gives one dominated point and takes O(n) time for n points in the set. And if there is no dominated point in the set, we figure it out in O(1) time. So, the height of the tree can be considered to have decreased to logh in this case. Even when there is a case when there are no non-dominated points present, we invest only constant time O(1) on it, not adding to the total time complexity. Thus, our total time complexity turns out to be O(nlogh).

In simple terms, whenever we have non-dominated points in our set, we figure one of them for sure and also eliminate the points dominated under it, which can then result in such a case that P_l does not have any points remaining. This is because there were no non-dominated points in P_l and hence no extra time was spent on figuring this out. Hence time complexity is O(nlogh).

The formal proof can be done as:

To prove: Time Complexity of the given algorithm is O(nlogh).
Given that there are total h non-dominated points, if L has h1 elements R will have (h-h1) elements. More precisely, Since we are removing y_{max} beforehand R will have (h-h1-1) elements. Thus,
T(n, h) = O(n) + T(n/2, h1) + T(n/2, h-h1-1)   if h>=2
= O(1)   if h<=1

Claim: T(n, h) = O(nlogh)
For h = 1, this doesn't hold. Time complexity is O(1).

Base case: h = 2, there are 2 possibilities.
1.) Both non-dominated points are in R
2.) 1 is in R and 1 is in L.
 # Impossible case: Both non-dominated points can't be in the L set. As the point with maximum coordinate is always a non-dominated point.

In first case: h1 = 0, h-h1-1 = 1

$$
\begin{aligned}
T(n,\ h) &= \ cn + T(\tfrac{n}{2}, 1) \\
&= \ cn + b\frac{n}{2} \\
&<= anlog2
\end{aligned}
$$

`

Induction:

$$
\begin{aligned}
T(n,\ h) &= \ cn + T(\tfrac{n}{2}, h1) + T(\tfrac{n}{2}, h - h1 - 1) \\
&<= cn + b(\tfrac{n}{2} logh1 + \tfrac{n}{2}\log{(h - 1 - h1)}) \\
&= cn + b\frac{n}{2}(\log{(h1 * (h - 1 - h1)}) \\
&<= cn + b\frac{n}{2}(\log{(\tfrac{h}{2} * \tfrac{h}{2})}) \\
&<= cn + b\frac{n}{2} * 2 * \log{(\mathrm{h})} \\
&= O(nlogh)
\end{aligned}
$$

Thus, the time complexity of this algorithm is O(nlogh).