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 Pl and right half as Pr. Then, we find the point ymax with maximum y coordinate in Pr. ymax has to be a non-dominated point in Pr as discussed in method 1 in lectures, and because every non-dominated point in Pr has to be a non-dominated point for Pl, ymax 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 ymax as they cannot be non-dominated points. Then recursion step is followed to repeat the same algorithm on Pl and Pr to find non-dominated points in both sets. The final set of non-dominated points has to be ymax, non-dominated points in Pl and non-dominated points in Pr.

**Pseudo Code:**

NonDomPoints(P) {

if P = φ : return φ

if |P| = 1 : return P

set ans = φ

Let xmed be the median of x-coordinate of the points in P ……………………….O(n)

Partition P into two sets Pl & Pr about median xmed  ……………………….O(n)

Let Pl be all the points with x-coordinate ≤ x.

Let Pr be all the points with x-coordinate > x.

Let ymax be the point in Pr with the maximum y-coordinate ……………………….O(n)

ans = ans ∪ ymax

Delete every point dominated by ymax in Pl ……………………….O(n)

Delete every point dominated by ymax in Pr ……………………….O(n)

L = NonDomPoints(Pl) ……………………….….T(n/2,h1)

R = NonDomPoints(Pr) ……………………….….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 ymax with max y coordinate in Pr. And after removing all the points dominated by ymax from Pl & Pr, we are running this algorithm separately for Pl & Pr. 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 ymax beforehand, we ensure that the recursion step for Pl doesn’t return any point which will be dominated by any point in Pr. After recursion for Pr & Pl we’ll have all non dominated points form Pl & Pr.  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 Pl does not have any points remaining. This is because there were no non-dominated points in Pl 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 ymax 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

`

Induction:

)

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