# Q. Faster Algorithm for Non dominated points in plane-

## O(n log(n)) approach

Lets divide the set into 2 halves according to their x coordinate. Let say the points in both halves separately in ascending order by their y coordinates. Now, look at the lowest y-valued point in both halves. If the lowest point on the left has a lower y value than the lowest point on the right, then that point is dominated by all points on the right. Otherwise, the bottom point on the right doesn't dominate anything on the left.

In both the cases, we are able to remove one point from one of the two halves and repeat the process with the remaining sorted lists. This takes O(1) time per point, so if there are n total points, this takes O(n) timr (after sorting) to count the number of dominating pairs across the two halves.

So the conquer step is taking O(nLogn) time because of sorting. And if we simply divide the array into 2 parts according to their x coordinate then the divide part is taking O(logn) time. So the total time taken is O(nLog(n)log(n)). However, we can speed this up if before the start of the divide and conquer step we store the points in increasing y coordinates order, doing the algorithm works in O(n log n).

The points given to us are already sorted with respect to their x coordinate...

## Approach - So in divide and conquer we can optimize either divide part or conquer part. So we can delete all the points that are dominated by the point who have maximum coordinate in right side of the division. And then we add that point to our set of Non-Dominated points.

## Psuedo Code –

```
1.  List          //it will store all dominated points
2.  Funct( p )   // p is set of points at any state
3.  {
4.      Point1=max_x_coordinate(p); Point2=max_y_coordinate(p)
5.      If Point1 == Point2 then
6.          List.add(Point1)     // the same point has max x and max y so its non domi…
7.      Else
8.          Right_list <- points having greater x coordinate then median
9.          Point3 = max_y_coordinate(Right_list)
10.         p=p-(points dominated by Point3)
11.         List.add(Point3)
12.         Right_list <-Points to right of Point3
13.         Left_list <- Points in the left division
14.         Funct(Right_list)
15.         Funct(Left_list)
16.}
```

## Correctness – So our algorithm first splits the set into 2 wrt to the median of x coordinates points. Now we pick the the point who have maximum coordinate in right side of the division.

Now right division lets say we split it again then the maximum y coordinate (YMax) point in right part of it is surely non-dominated as its x coordinate is greater then all left points and

in its own division it have maximum y coordinate. After this we delete all the points dominated by this point.
In the left division let say we split it again then the maximum y coordinate point in the right half of the left division is the non dominated in the sub problem only. We are not sure about right part of the initial split. But from above paragraph we have already removed all the points that were dominated by the our right side (YMax).
So in this way we are finding a non dominated point in every recursive call.

## Time Complexity -

At every level of the recursion, we take O(n) time in the conquer step. The depth of the recursion tree will be bounded by log h, since at level k there are $2^k$ subproblems, each finding one non dominated point before calling more child processes.

So finally we are taking O(n) time at every level, and total number of levels are logh, so the final complexity is O(nLog(h)).