Faster algorithm for Non-dominated points in a plane

1 Problem 1

1.1 Brief Description

We make a very small tweak with respect to the $O(nlogn)$ algorithm discussed in the class. We do the following at **each** recursion call -
After we compute the Non Dominated Points in the right sub-space, we 'prune' the points in the left sub-space such that it contains only those points which are non dominated by any other point in the right sub-space. We compute the non dominated points in the left sub-space **if and only if** we have a non-empty 'pruned' set of points belonging to the left sub-space.
If the set S is non empty, the right subspace is guaranteed to have a non dominated point.
Notations - We will represent Sets containing points with capital letters and the points themselves with small letters. $p.x$ denotes the x-coordinate of the point p and similarly $p.y$ denotes the y-coordinate of the point p.

1.2 PseudoCode

Algorithm 1 Algorithm to compute all Non Dominated Points in a given set

1: **procedure** NON_DOMINATED_POINTS(S) ▷ S is the given set of points
2: **if** $|S| = 1$ **then**
3: **return** S
4: $m_{med} \leftarrow Split\_By\_x\_Median(S)$ ▷ Computes x-median of the set S in O(n)
5: Split the points of S into S_l and S_r such that $S_l \cup S_r = S$ and all points to the left of m_{med} belong to S_l and the rest to S_r. ▷ Difference between the sizes of S_l and S_r is atmost 1 and $|S_r| = \lceil \frac{n}{2} \rceil$
6: $H_2 \leftarrow$ Non_Dominated_Points(S_r)
7: $p \leftarrow$ Point with the maximum y coordinate from H_2 ▷ Linear time computation
8: Remove all points c from S_l such that $c.y < p.y$, $c \in S_l$ ▷ Linear time operation
9: $H = H_2$ ▷ H is the final set of non dominated Points
10: **if** $S_l \neq \phi$ **then**
11: $H_1 \leftarrow$ Non_Dominated_Points(S_l)
12: $H = H_1 \cup H_2$
Return H

1.3 Proof of correctness

Note that we did not invoke any special 'Merge' procedure at all. We prove that we do not even need it using the below assertion.

Assertion - H_1 doesn't contain any points dominated by any other point in H_2. More formally, we claim that there exists no point $p_1 \in H_1$ which is dominated by any point $p_2 \in H_2$.

Proof - After execution of line 8, any point $p_1 \in H_1$ has a y coordinate greater than or equal to that of any other point p_2 in H_2. This is because

$$p_1.y \geq p.y$$

(see definition of p from the pseudocode)

$$\implies p_1.y \geq p_2.y \qquad \forall p_2 \in H_2 \quad \text{since} \quad p_2.y \leq p.y$$

For any point c to be dominated, there must exist a point q such that

$$q.x > c.x \quad and \quad q.y > c.y$$

Hence p_1 cannot be dominated by any point p_2 in H_2. Since p_1 was arbitrary, it holds true for all points in H_1

Assertion - H contains all the non dominated points of the set S

Proof - We assume that H_1 and H_2 are computed correctly and using them, try to prove that H is indeed computed correctly.
Since c is a non dominated point from the set S_l and it is not dominated by any other point q from S_r, c is a non dominated point from the set S. Hence c must be contained in the set H where H represents the final set of non dominated points.
Since c was arbitrary in H_1, all points in H_1 are contained inside the set H.
Since all points q in H_2 have the x-coordinate greater than or equal to all the points c in H_1, there exists no point c in H_1 which dominates the point q. Hence q is a non dominated point from the set S. Therefore q must be contained in the set H. Since q was arbitrary, all points H_2 must be contained in H.
Hence, $H = H_1 \cup H_2$. Therefore, H is computed correctly.

1.4 Time Complexity

If the set S is non empty, it is fairly easy to see that the rightmost point will surely be a non dominated point. Hence if the set S is non empty, H_2 is surely non empty. Therefore, the set S must contain at least one non dominated point.
To prove the time complexity, we will look at the recursion tree formed by calling the function Non_Dominated_Points on the whole input set S. Let the size of the input set S be n. Also, let the total number of non dominated points be h. Therefore, there will be atmost $\lceil log(n) \rceil$ levels of recursion where we assume that the first call i.e. on the set S is at a level 0. At each recursion level, there can be atmost 2^i nodes. By the execution of the code, it is fairly evident that before a call to a node has been made, it has been processed i.e. all the points in it which are dominated by the nodes to the right have been removed. Hence it will be called only when it is non empty i.e. it has at least one non dominated point. Therefore, at each recursion level i, we only call atmost h nodes since the number of non dominated points is bounded by h. Therefore, at each recursion level i, we only call $min(2^i, h)$ nodes. Note that each recursion level might not have 2^i nodes in case of a non balanced recursion tree hence the bound above is a fairly generous one. It is achieved only when our recursion tree is a perfectly balanced binary tree. Moreover, at each level i, the input size to the node is atmost $\lceil \frac{n}{2^i} \rceil$. Therefore at each level i, the number of operations our algorithm performs is

$$O(\frac{n}{2^i})min(2^i, h)$$

Now we simply need to sum this up from 0 to $\lceil log(n) \rceil$ to compute the final time complexity. Therefore,

$$T(n) \leq \sum_{i=0}^{\lceil log(n) \rceil} O(\frac{n}{2^i})min(2^i, h) \leq O\left(\sum_{i=0}^{\lceil log(n) \rceil} \frac{n}{2^i} min(2^i, h) \right)$$

Let

$$X = \left(\sum_{i=0}^{\lceil log(n) \rceil} \frac{n}{2^i} min(2^i, h) \right)$$

There must exist some k such that $2^{k-1} < h$ but $2^k \geq h$ since $0 \leq h \leq n$. Therefore X can be written as

-

$$X = \left(\sum_{i=0}^{k-1} \frac{n}{2^i} 2^i + \sum_{i=k}^{\lceil log(n) \rceil} \frac{n}{2^i} h \right)$$

$$\implies X = \left(\sum_{i=0}^{k-1} n + nh \sum_{i=k}^{\lceil log(n) \rceil} \frac{1}{2^i} \right) = \left(nk + nh(\frac{1}{2^{k-1}} - \frac{1}{2^{\lceil log(n) \rceil}}) \right)$$

Since $\frac{-1}{2^{\lceil log(n) \rceil}} \leq \frac{-1}{2^{log(n)+1}} = \frac{-1}{2n}$,

$$\implies X \leq \left(nk + nh(\frac{1}{2^{k-1}} - \frac{1}{2n}) \right)$$

$$\implies X \leq n \left(k + \frac{h}{2^{k-1}} \right) - \frac{h}{2}$$

$$\implies X \leq n \left(k + \frac{h}{2^{k-1}} \right)$$

Since k is bounded such that $2^{k-1} < h$ and $2^k \geq h$, this means that $log(h) \leq k \leq log(h) + 1$. We generously replace k by $log(h) + 1$ in the first term of X and by $log(h)$ in the second term. Therefore,

$$X \leq n \left(log(h) + 1 + \frac{h}{2^{log(h)-1}} \right) \implies X \leq n(log(h) + 3)$$

Therefore,

$$T(n) \leq O(X) \leq O(nlog(h) + 3n) = O(nlog(h)) + O(n) = O(nlog(h))$$

Now, since $T(n) \leq O(nlog(h)) \implies T(n) = O(nlog(h))$
Hence Proved