Design and Analysis of Algorithms
Indian Institute of Technology Kanpur

Worked out Assignment on *Divide and Conquer* paradigm

An Important guideline

- It is only through the assignments that one learns the most about the algorithms and data structures. For the current assignment, a sequence of hints are given. Try to look at a hint only after trying on your own for sufficiently long period of time.
- Ideally, one should look at one hint per day and then spend around 20-25 minutes to materialize the hint. The best way - go for a stroll after dinner pondering over the hint(s).
- You should judge your performance by the number of hints you indeed use – more hints you use, less is your performance.
- The onus of learning from a course lies first on you. So act wisely while working on this assignment.

An $O(n \log h)$ time algorithm for non dominated points.

Let P be a set of n points in plane. In class we discussed two algorithms to compute all non-dominated points of P. The first algorithm was a simple algorithm which was output sensitive - its time complexity is $\mathcal{O}(nh)$, where h is the number of non-dominated points in P. The second algorithm was based on the divide and conquer paradigm, and its time complexity is $\mathcal{O}(n \log n)$. Observe that none of the algorithm is better than the other - for $h << \log n$, the first algorithm is better, whereas for $h >> \log n$, the second algorithm is better.

Your task is to design an $\mathcal{O}(n \log h)$ time algorithm for the problem.

Hint: You just need to internalize the 2 algorithms to design the desired algorithm. Make sincere attempts before proceeding further.

Few Observations to get you started :

1. Looking at just the time complexity $\mathcal{O}(n \log h)$, it is easy to observe that we can not do even simple preprocessing like sorting to achieve the goal.
2. The first algorithm is already so basic that there does not seem to be any way to alter it.
3. Focus on the second algorithm (based on divide and conquer). In particular, let us look at the conquer step carefully ...

Most Powerful Hint: The $O(n \log h)$ time algorithm is designed by performing very minor changes to the $O(n \log n)$ time divide and conquer algorithm. Ponder over this hint ...

Observe that each non-dominated point of the right set is also a non-dominated point of the original set. However, a non-dominated point of the left half set is not necessarily a non-dominated point of the original set.

As an example, imagine that there are *many* non-dominated points from the left set, but majority of them (in fact, possibly all of them) turn out to be discarded during the conquer step. In a way, it was quite wasteful to compute all such non-dominated points of the left set. Isn't it ? Can we avoid this wastage ? Notice that we are anyway spending $\mathcal{O}(n)$ time in the divide step. So it will be quite good if we can avoid this wastage by spending *extra* $\mathcal{O}(n)$ time. Think hard to exploit this hint.

Let $y_{\max}$ be the y-coordinate of the highest point from the right set. Before invoking the recursive call for the left half set, we eliminate all points of the left half set whose y-coordinate is smaller than $y_{\max}$. Can you see the algorithm now ? Try to make useful observations about this algorithm to see why its time complexity is $\mathcal{O}(n \log h)$. What can you say about the recursion tree of the new algorithm ?

Note that the new algorithm avoids the wastage that was there in the second algorithm. A recursive call is invoked by the new algorithm only if the corresponding point set has at least one non-dominated point. Hence, it can be seen that the number of leaves of the recursion tree is h. This is significant since for the divide and conquer based algorithm, there are n leaves in the recursion tree. Can you analyse the algorithm and establish that its time complexity is $\mathcal{O}(n \log h)$?

You might feel tempted to quickly claim the following:

Since the number of leaves in the recursion tree of the new algorithm is h, so the height of the recursion tree is $\log h$*; and hence the time complexity is* $\mathcal{O}(n \log h)$.

But, this claim is wrong. Can you see this ? Think of an alternate way to analyse the time complexity. Try induction. But what will you induct on ?

For the time complexity of the new algorithm, one might write the following recurrence.

$$T(n) = T(\frac{n}{2} - x) + T(\frac{n}{2}) + cn$$

where x is the number of points removed from first half. This might not work. After all, h has to somehow appear in the recurrence. Isn't it ? How should you define the recursive term then ?

Let $T(n, h)$ be the time complexity of the new algorithm. What is the recurrence for $T(n, h)$? Try sincerely to come up with the recurrence and prove it using induction.

The recurrence relation is the following.

$$T(n,h) = T(\frac{n}{2}, h_r) + T(n', h - h_r) + cn$$

where h_r is the number of non-dominated points from the right set and $n' \leq n/2$ is the number of points of the left recursive call (after pruning the points dominated by the highest point from the right set). Give suitable arguments to establish the validity of the above recurrence and then prove using induction that $T(n,h) = \mathcal{O}(n \log h)$.

Interestingly, there is alternate proof to establish $\mathcal{O}(n \log h)$ time complexity of the new algorithm. It is based on an elegant analysis of the recursion tree alone. Ponder over it...