CS345 Assignment-1

Ayush Shakya
180178

Soham Ghosal
180771

August 20, 2021

1 Faster algorithm for Non-Dominated Points in plane

In the lecture, two approaches were discussed to evaluate non-dominated points in a plane. The first approach was **output sensitive**and worked in a time-complexity of $O(N * h)$ where h is the number of dominated points in the **plane consisting of N points**. This algorithm was $O(N)$ in best case but $O(N^2)$ in the worst case.

We then improved this approach to $O(NlogN)$. This new algorithm was based on **divide-and-conquer paradigm**, but there were scenarios where the first algorithm could outperform the second algorithm. In this assignment, we devise an algorithm with a **time complexity of** $O(Nlogh)$ which is faster than both these algorithms, specifically in cases where the number of non-dominated points are very few. As mentioned in the problem statement, it can be shown that if n points are selected randomly uniformly from a unit square, then the expected(average) number of non-dominated points is just $O(logn)$.

2 Sketch of Algorithm

2.1 Assumptions

- We assume that no two points in the plane have **same x-coordinate** or **the same y-coordinate**.
- There will be only one point on the vertical line passing through the median of x- coordinate of the given points.
- We define the x-median to be the median of x coordinates of all given points-in case there are even number of points, we take the floor of the actual median.
- The set of points S received as an argument to the function is has a non-zero size.

2.2 Intuition

We begin by computing the x-median of the input points, and splitting the given sample space into two halves, via a vertical line passing through the x-median **[Divide Step]**.The median can be computed using an $O(N)$ algorithm, as discussed in class. Since we are splitting on the basis of x-median, the sizes of the two halves **would differ by at most one**. Let's call these partitions **left set(LS)** and **right set(RS)**.

We invoke a recursive call to the divide function for the right set. Among all those points, we choose the point with maximum value of y-coordinate. Let the y-coordinate of this point be y_{max}.

We collect all the points in the left set whose y-coordinate is greater than y_{max}. Unless this set is empty, we invoke another recursive call for this left set.

3 Pseudocode

Algorithm 1: Faster Algorithm for non-dominated points

Function `fasterNDP(`S`)`
  $x_m \leftarrow \text{getmax}_x(S)$, $y_m \leftarrow \text{getmax}_y(S)$ `/* Can be done in O(N) time */`
  $p_{max} \leftarrow (\mathrm{x}_m, y_m)$
  $finalNDP \leftarrow \emptyset$ `/* List to store all non-dominated points */`
  `/* Base Case:If` x_m `and` y_m `correspond to a single point in the set, then there are no more non-dominated points, since this point encompasses all others. */`
  if $p_{max} \in S$ **then**
    finalNDP.push(p_{max})
    return finalNDP
  $(LS, RS) \leftarrow SplitByMedian(S)$ `/* Can be done in O(N) time */`
  $rightNDP \leftarrow fasterNDP(RS)$ `/* Recursive call on the right set */`
  $y_{max} \leftarrow getmax_y(rightNDP)$ `/* Can be done in O(N) time */`
  $reduced_{LS} \leftarrow \emptyset$ `/* List to store the candidate points from the left set */`
  `/* We iterate on the left set and add only those points whose y-coordinate is greater than` y_{max} `*/`
  for *point* $p \in LS$ **do**
    if $p.y > y_{max}$ **then**
      $reduced_{LS}$.push(p)
  finalNDP.push(rightNDP)
  if $reduced_{LS}.size > 0$ **then**
    $leftNDP \leftarrow fasterNDP(reduced_{LS})$ `/* Recursive call on the left set */`
    finalNDP.push(leftNDP)
  return finalNDP

4 Proof of Correctness

First we consider the base case. Since we assumed that x-coordinate and y-coordinate of all points are unique, if there exists a point whose both x-coordinate and y-coordinate are maximum among all points in that current set, it dominates all other points in that set(since, it encompasses all other points). Hence it is the only non-dominated point in the given set and we return it as an answer and exit from the recursive call.

Next, we need to prove that set of points $rightNDP$ obtained in recursive call on right set(RS) are non-dominated points. Since their x-coordinates are greater than all the points in the left set(LS), they are non-dominated with respect to the points in the left set, and we already know, they are non-dominated among themselves. Thus we add $rightNDP$ into the final list of non-dominated points.

For the left part, we construct a set of points $reduced_{LS}$ whose y-coordinate is greater than y_{max}. This means these points are not dominated by any point in right set(RS). After a recursive call on this set, we get a set of non-dominated points $leftNDP$ among the $reduced_{LS}$. We already proved that these points are not dominated by right set(RS) due to the condition that their y-coordinate is greater than y_{max}, and after the recursive call, we know that they are non-dominated among themselves. Hence, this set of points also holds the non-dominated conditions for the current set S. Thus we add $leftNDP$ into the final list of non-dominated points and return this as an answer.

5 Time Complexity Analysis

We try to analyse the time complexity using induction. Let T(h) denote the time-complexity of the proposed function when there are exactly "h" non-dominated points among the current set of points S, of size N. The functions $getmax_x$, $getmax_y$ and p_{max} checking condition, x-median calculation and splitting in LS and RS, filtering points on basis of y_{max} and add to final list are linear time functions in N. So, for cases where h=1, the time complexity is O(N) (Since log(1)=0).
Note that, after division of the sample space into two halves, there might be an arbitrary number of non-dominated points in the left half(let's call this h_1), and as a consequence, the number of non-dominated points in the right half would be $h-h_1$. Keeping this in mind, we would have to recur on the left half(containing n/2 points) and the right half(containing n/2 points).Thus the time complexity when there are h non-dominated points would be bounded by the maximum of all such divisions, hence the recurrence relation (equation 2).

The inductive hypothesis is assumed to be as follows :

$$\textbf{Induction Hypothesis}: \boxed{\mathrm{T(h)} \leq c*(N+N*log_2(h))} \textbf{ for N points}. \tag{1}$$

When left set(LS) is completely deleted by y_{max}, the size of h is 0. For such a case, T(h) for any size of N is 0. For the case of h=1, the algorithm is simply linear time due to the p_{max} if condition.

The recurrence relation for this pseudocode can be written as :

$$\boxed{\mathrm{T(h)} \leq max(T(h_1)+T(h-h_1)+c_1*N) \;\; \forall \;\; 0 \leq h_1 \leq h} \tag{2}$$

Please note that T(h) is defined for all N points, but T(h_1), T($h-h_1$) are defined for N/2 points. Value of h_1 can vary from 0 to N, thus, we need to take the max of all the right recurrences and compare it with the T(h) for N points.

$$\boxed{\mathrm{T(h_1)} \leq c*(N/2+N/2*log_2(h_1)) \;\; \text{and} \;\; T(h-h_1) \leq c*(N/2+N/2*log_2(h_1))} \tag{3}$$

Let us substitute these induction hypothesis for N/2 points into equation 2.

$$\mathrm{T(h)} \leq \max\ (\ \mathrm{T(h_1)}+T(h-h_1)+c_1*N) \;\; \forall \;\; 0 \leq h_1 \leq h$$

$$=>\mathrm{T(h)} \leq \max\ (\ \mathrm{c*(N/2 + N/2*log_2}(h_1)+c*(N/2+N/2*log_2(h_1))+c_1*N) \;\; \forall \;\; 0 \leq h_1 \leq h$$

$$=>\mathrm{T(h)} \leq \max\ (\ (\mathrm{c+c_1})*N+c*N/2*(log_2(h_1*(h-h_1)))) \;\; \forall \;\; 0 \leq h_1 \leq h$$

The $(c+c_1)$ component is independent of h_1. It can be easily shown through differentiation that the max value of h_1*h-h_1 is achieved at $h_1=h/2$, which gives the value of $h_1*h-h_1=h^2/4$. If h is an odd number, then too the value of $h_1*h-h_1<h^2/4$. Thus, we can say that, $h_1*h-h_1 \leq h^2/4$. $log_2(x)$ is a monotonically increasing function in x (input). So we can say:

$$\mathrm{T(h)} \leq \max\ (\ (\mathrm{c+c_1})*N+c*N/2*(log_2(h^2/4))$$

$$=>\mathrm{T(h)} \leq \max\ (\ (\mathrm{c+c_1})*N+c*N*(log_2(h/2))$$

$$=>\mathrm{T(h)} \leq \max\ (\ (\mathrm{c+c_1})*N+c*N*(log_2(h)-1))$$

$$=>\mathrm{T(h)} \leq \max\ (\ \mathrm{c_1}*N+c*N*(log_2(h)))$$

We can always choose a **c which is greater than** c_1. On doing so, we get the desired bound:

$$\boxed{\mathrm{T(h)} \leq c*N(1+log_2(h)) \;\; \text{where } \mathrm{c \geq c_1}}$$

Thus, our initial hypothesis is true; the time complexity of the proposed algorithm is **O(Nlog h)**.