CS345
Assignment 1
Hard

Siddharth Agrawal(150716)
AviRaj(150168)

August 14, 2017

Question 1

Part 1

Algorithm

Algorithm 1 Algorithm to find the non dominated points in a graph

1: **function** $NDP(P)$
2: **if** $|P| = 1$ **then**
3: **return** P
4: **end if**
5: $p_{med} \leftarrow$ Compute x-median of P
6: $(P_L, P_R) \leftarrow$ Split-by-x-median(P)
7: $H_R \leftarrow NDP(P_R)$
8: $y_{max} \leftarrow$ Point with maximum y-coordinate in H_R
9: $P_L \leftarrow P_L \backslash$(Points in P_L which are dominated by y_{max})
10: $H_L \leftarrow NDP(P_L)$
11: **return** $H_L \cup H_R$
12: **end function**

Proof of Correctness

Notice that the above algorithm is almost similar to the divide and conquer algorithm discussed in class. The only special operation done (as shown in step 8 & 9) is to delete points in the left half(P_L) which are dominated by the point in H_R with maximum y-coordinate.
So, we have to prove that H_L and H_R are mutually exclusive (for step 11)

Lemma: *H_L and H_R are mutually exclusive*
Proof:
Note that in step 9, we are removing all points which are dominated by y_{max}. This ensures that all remaining points in P_L are NOT dominated by any non-dominated point in P_R. Thus, H_L returns only those non-dominated points which are not dominated by any point in P_R. Hence, H_L and H_R are mutually exclusive.

Thus, combining the above lemma with the proof of the normal divide and conquer method proves our algorithm.

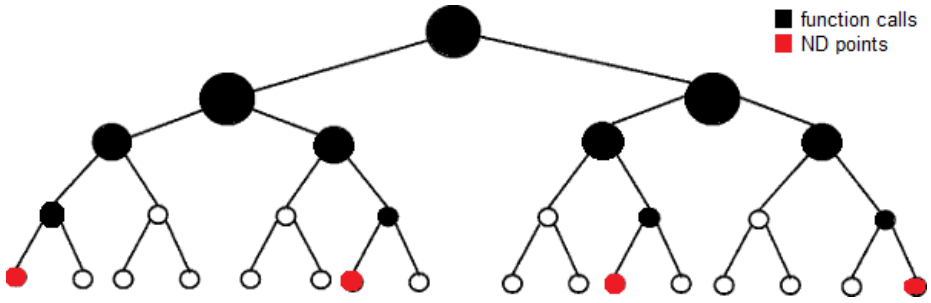
Time Complexity Analysis

Let $T(n, h)$ be the time taken to calculate all the h non-dominated points in a set of size n.
Then,

$$T(n,h) = T(\frac{n}{2}, k) + T(\frac{n}{2}, h-k) + O(n) \qquad 0 <= k <= n \tag{1}$$

(Since deletion (step 9), finding maximum (step 8) and finding median (step 5), all take O(n) time)

Worst Case Time Analysis



- This tree represents each call of divide and conquer method.
- Each leaf denotes a point among n points in the plane.
- Our h non-dominated points are among any h of these n leaves.

Lemma 1: *Time taken to reach from root to leaf is O(n)*
Proof:
Let $T(n)$ be the time taken to reach from root to leaf.
Then,

$$\begin{aligned} T(n) &= O(n) + T(n/2) \\ \therefore T(n) &= 2cn \\ &= O(n) \end{aligned}$$

Lemma 2: *Sum of Time taken to reach any 2 leaf nodes is maximized when their Least Common Ancestor (LCA) is as near to the root as possible*
Proof:
On the contrary, letus assume that there is a worse case b where L.C.A. is at a higher level 'B' (lesser height) than L.C.A. in case a (level 'A').
i.e,

$$\begin{aligned} A < B \text{ and } T(\text{case b}) &> T(\text{case a}) \\ T(\text{case a}) &= 2cn + 2cn * (2^{-A}) \\ \text{Similarly, } T(\text{case b}) &= 2cn + 2cn * (2^{-B}) \\ T(\text{case b}) - T(\text{case a}) &= 2cn * (2^{-B} - 2^{-A}) < 0 \end{aligned}$$

CONTRADICTION! Hence our assumption that we got a worse case b was incorrect.

Using lemma 2, we can maximize the time to reach 'h' leaf nodes. Therefore in the first 'log h' levels, all nodes will be traversed.

NOTE: *We are considering time only for the nodes which are being traversed. Those which are not traversed can be neglected because we have deleted those points at each level in our algorithm and ensure that each call (denoted by a node) returns a non-dominated point.*

Time for Worst Case

$$\begin{aligned}
T(n) &= n + 2\frac{n}{2} + 2^2\frac{n}{2^2} + ... + 2^{logh}\frac{n}{2^{logh}} + h(logn - logh) \\
&= (n + n + n + ...logh \text{ times}) + hlogn - hlogh \\
&\leq nlogh + hlogn \\
&\leq 2nlogh && (h < n \implies n^h < h^n \implies hlogn < nlogh, \text{ for large } n, h) \\
&= O(nlogh)
\end{aligned}$$

$\therefore$ the time complexity of this algorithm is $O(nlogh)$

Part 2a

Algorithm

We maintain a Red-Black Binary Tree, whose each node represents a non-dominated point.
Let the RB Tree be filled till $i-1$ points. Now we insert the i^{th} point in the tree using the subsequent steps:

1. insert the point in the tree using $O(logi)$ time.
2. check if the newly inserted point is dominated by it's successor.
   - `if yes:` delete the newly inserted point. exit the loop.
   - `if no:` go to step 3.
3. check if the newly inserted point dominates it's predecessor.
   - `if yes:` delete the predecessor point. go back to step 3.
   - `if no:` exit the loop.

Proof of Correctness

This is an inductive algorithm. Thus, it is sufficient to prove only the inductive step.
Let the RB Tree consist of all the non-dominated points in the $i-1$ checked points.
Upon insertion of the i^{th} point, if this point is NOT a non- dominated point, step of our algorithm rejects it, else all those points in the tree which are dominated by this i^{th} point will get deleted in step 3, and the RB Tree will update the set of non-dominant points for ll the i points read till now.

Time Complexity Analysis

Time taken to insert the i^{th} point in the tree = $O(logi)$.
Time taken to find predecessor/sucessor in the tree = $O(logi)$.
Suppose that the number of deletions (while updating the tree on each insertion) is as follows -

- i_1 deletions for i^{th} insertion.
- i_2 deletions for $(i-1)^{th}$ insertion.
- ... and so on.

Note: $i_1 + i_2 + \ldots\ldots\ldots \leq i$ $\because$ number of deletions cannot exceed total number of insertions (or the points traversed).
Thus,

$$\begin{aligned} T(i) &= i_1 log(i) + i_2 log(i-1) + i_3 log(i-2) + \ldots\ldots\ldots && \text{(T(i) is the total time for } i \text{ insertions)} \\ &\leq i_1 log(i) + i_2 log(i) + i_3 log(i) + \ldots\ldots\ldots \\ &= log(i) * (i_1 + i_2 + i_3 + \ldots\ldots\ldots) \\ &\leq log(i) * i && \text{(using the result of the note specified above)} \\ &= O(ilogi) \end{aligned}$$

Thus, the total time for i insertions has a bound of $O(ilogi)$

Part 2b

Algorithm

Algorithm 2 Algorithm to find the non dominated points in a 3D graph

S denotes the set of n points in 3d plane
Sort S in decreasing order of $z-$coordinate
$D \leftarrow \phi$ ▷ D denotes the set of non-dominated points
$P \leftarrow \phi$ ▷ P denotes the RB Tree as defined in part a
for each point p in S in decreasing z-coordinate **do**
    Insert p in P according to the algorithm in part a ▷ (Addition in P is only dependent on x, y-coordinates of the point p)
    if p gets inserted successfully in P **then**
        $D \leftarrow D \cup \{p\}$
    else
        ignore p
    end if
end for

Proof of Correctness

Since, the above algorithm is inductive, it is sufficient to prove that the i^{th} insertion successfully updates our set of non-dominated points (viz. D)

Proof:

- The i^{th} point being inserted CANNOT be dominated by the subsequent $(i+1)^{th}, (i+2)^{th}, (i+3)^{th}, \ldots\ldots\ldots$ inserted points because $z_i > z_{i+1} > z_{i+2} > z_{i+3} > \ldots\ldots\ldots$ (by the algorithm).
- Thus we only need to check the previously inserted points, i.e, $1^{st}, 2^{nd}, 3^{rd}, \ldots\ldots\ldots, (i-1)^{th}$ points.
- According to the algorithm, $z_1 > z_2 > z_3 > \ldots\ldots\ldots > z_{i-1} > z_i$.

- Hence, the i^{th} point is non-dominated ONLY IF its projection is non-dominated by the projections of the previous $i-1$ points. (Otherwise not only it's z-coordinate but x, y-coordinates also get dominated, which implies that it is NOT a non-dominated point).
- The above point is successfully checked by the algorithm specified in part a.
- Hence, each i^{th} insertion successfully updates our set D of non-dominated points.

Time Complexity Analysis

Our algorithm takes-

- $O(nlogn)$ for sorting the n points according to z-coordinate
- $O(nlogn)$ for inserting the n points in the RB Tree P (as proved in part a)

$$\begin{aligned} \therefore T(n) &= O(nlogn) + O(nlogn) \\ &= O(nlogn) \end{aligned}$$

Therefore, it takes $O(nlogn)$ time for our algorithm to find the non-dominated points among n 3d points.

Question 2

Algorithm

We use the multiplication of polynomials algorithm (discussed in class) to compute this problem. For this, we are required to construct two polynomials.
Also, we can use the following law -

$$F_j = q_j * E_j \tag{2}$$

where E_j denotes the electric field at point j

Hence our original equation reduces to -

$$E_j = \sum_{i<j} \frac{Cq_iq_j}{(j-i)^2} - \sum_{i>j} \frac{Cq_iq_j}{(j-i)^2} \tag{3}$$

Now, consider the following two polynomials (and their product $C(x)$)-

$$A(x) = \frac{1}{(n-1)^2} + \frac{1}{(n-2)^2}x + \text{.....} + \frac{1}{1^2}x^{n-2} + 0.x^{n-1} + \frac{-1}{1^2}x^n + \text{.....} + \frac{-1}{(n-1)^2}x^{2n-2} \tag{4}$$

$$B(x) = 0 + 0.x + \text{.....} + 0.x^{n-2} + q_nx^{n-1} + q_{n-1}x^n + \text{.....} + q_1x^{2n-2} \tag{5}$$

$$C(x) = A(x) * B(x) \tag{6}$$

Using the relation in equation (5), we can safely establish the following relation -

$$E_j = C * \text{Coefficient of } x^{3n-2-j} \text{ in } C(x) \tag{7}$$

Hence, since we can find the polynomial $C(x)(= A(x) * B(x))$ from the multiplication of polynomials algorithm (discussed in class), we also know all the coefficients of $C(x)$. Using relation (6), we can know the electric fields at all the n points. Then we use the relation (1) to find the force at all n points.

Proof of Correctness

For the multiplication of polynomials algorithm, its proof was already discussed in class. Also, the two polynomials being assumed as such is just intelligent manipulation and doesn't require any proof. The relation (1) follows from definition of electric field.
Hence, it is sufficient to prove the correctness of the relation (6).

Proof for Relation (6)

$$\begin{aligned}
\text{Coefficient of } x^{3n-2-j} \text{ in } C(x) &= \sum_{0<=i<=(3n-2-j)} a_i b_{(3n-2-j)-i} \\
&= \sum_{i=0}^{3n-2-j} b_i a_{(3n-2-j)-i} \\
&= \sum_{i=0}^{n-2} 0.a_{(3n-2-j)-i} + \sum_{i=n-1}^{2n-2} b_i a_{(3n-2-j)-i} && (\because b_i = 0 \ \forall \ 0 <= i <= n-2) \\
&= \sum_{i=2n-2}^{n-1} b_i a_{(3n-2-j)-i} \\
&= q_1 * \frac{1}{(j-1)^2} + q_2 * \frac{1}{(j-2)^2} + ... + 0 + ... + q_n * \frac{1}{(n-j)^2} \\
&= \sum_{i<j} \frac{q_i q_j}{(j-i)^2} - \sum_{i>j} \frac{q_i q_j}{(j-i)^2} \\
&= \frac{E_j}{C} && \text{(Using relation (2))} \\
\therefore E_j &= C * \text{Coefficient of } x^{3n-2-j} \text{ in } C(x)
\end{aligned}$$

Time Complexity Analysis

The multiplication of two polynomials takes $O(nlogn)$ time (discussed in class).
After that, finding all E_j's takes $O(n)$ time ($\because$ we already know all the coefficients beforehand, so $O(1)$ time for each E_j).
After that, finding all F_j's again takes $O(n)$ time ($\because$ we already know all E_j's, so $O(1)$ tome for each F_j).

$$\begin{aligned}
\therefore T(n) &= O(nlogn) + O(n) + O(n) \\
&= O(nlogn)
\end{aligned}$$

Hence our algorithm takes $O(nlogn)$ time to compute the force on each particle.