CS345: Design and Analysis of Algorithms

Tushar Vatsal(14766), Ankur Kumar(14109)

Theoritical Assignment 1

1 Non-Dominated Points

1.1 Output sensitive Algorithm

Non_Dominated_Points(P) {
x ← Compute-x-median(P); ——————— O(n);
(P_L, P_R) ← Partition(P,x); ——————— O(n);
(x_R, y_R) ← find point with-maximum-y in P_R;——O(n);
Remove points in P_R with y≤ y_R and x ≤ x_R;——-O($\frac{n}{2}$);
Remove points in P_L with y < y_R;——————-O($\frac{n}{2}$);
A ← $A \cup \{(x_R, y_R)\}$;
$A_L \leftarrow Non\_Dominated\_Points(P_L)$;
$A_R \leftarrow Non\_Dominated\_Points(P_R)$;
$A \leftarrow A \cup A_L \cup A_R$;
return A;
}

1.1.1 Time Complexity Analysis

We can see that whenever we do any partition, we get one non-dominated point. Therefore, the algorithm will execute a maximum of 2*h partitions where h is the number of non-dominated points. Since the algorithm is recursive in nature, execution of algorithm can be illustrated by a binary tree. Each node in this binary tree represents execution of the algorithm on a set of n_k = O($\frac{n}{2^k}$) points where k is the depth of the node in the tree. If we focus on a node at level k in the tree, we will spend T_k = O(n_k) time in the execution of this set of points except the recursion calls as evident from the above pseudo code. Time taken in the recursion call is taken care by nodes deeper in the tree. Therefore, total time taken will be sum of all such T_k associated with each node in the tree.

$$T = \sum N_k T_k = \sum N_k O(n_k) = \sum N_k \frac{cn}{2^k} = cn \sum \frac{N_k}{2^k}$$

where N_k is the number of nodes at depth k.
To maximize T, there should be more number of nodes in a level and the time associated with each level is maximized(which can be done by keeping smaller number of levels in the tree) so that N_k is greater and 2^k becomes lesser. This is exactly the case of complete binary tree. In this case $0 \leq k \leq log(2h)$ and $N_k = 2^k$.

$$T = cn \sum \frac{N_k}{2^k} \leq cn \sum_{k=1}^{log(2h)} \frac{2^k}{2^k} = cnlog(2h) = O(nlog(h))$$

1.2 Extension to three dimensions

1.2.1 Part a

We can maintain a balanced BST for non-dominated points among 'i' points seen so far with key as x-co-ordinate and augmenting each node with y-co-ordinate. Now let's say P(x_p,y_p) is the next point.
Case 1: We can find the successor say Q, of x_p in the BST in O(logi) time and compare y_p with y_q. If $y_p < y_q \Rightarrow$ P is dominated by Q. Thus BST remains unchanged as P is not non-dominated point else P is non-dominated. Overall time O(logi).
Case 2: If P is non-dominated, we insert P in the BST in O(logi) time. Now we look for predecessor, say Q, of x_p in the resulting BST in O(logi) time. If $y_q > y_p$ then our insert operation is done in O(logi) time else if $y_q < y_p \Rightarrow$ Q is dominated by P. Hence, Q is not non-dominated point. We can delete Q in O(logi) time maintaining a balanced BST. We will again find out predecessor of P and carry out the whole process till no

predecessor of Q is dominated by P. We can delete a maximum of 'i' number of points till the insertion of P, each deletion in O(logi) time, therefore we spend only O(logi) time on an average for deletion and insertion of P. Therefore overall time complexity is O(ilogi) till insertion of i points.

1.2.2 Part b

Let P be my set of points in 3-d.
P' ← Sort P according to z-co-ordinate.
A ← NULL;
(A is a BST storing all the non-dominated points found till now).
We shall sort P according to z-co-ordinate. Now suppose there are k different values of z ⇒ z_1 to z_k in increasing order. We shall process points in decreasing order of z ⇒ all the points having $z = z_k$ will be processed first, then all the points with $z = z_{k-1}$ and so on. All the non-dominated points with $z = z_i$ can't be dominated by any point with $z < z_i$. We will take points with $z = z_k$ one by one. Since z is same for all such points, it essentially reduces to 2-dimensional case. Using part(a), we can find non-dominated points in O($n_k log n_k$) time by maintaining a BST A with key as x-co-ordinate and augmenting the nodes with y-co-ordinate. Now we take points with $z = z_{k-1}$ one by one say p. We will search in A w.r.t x and y only in O(logn_k) time. If p is dominated by any point in A, say q, then p is certainly not a non-dominated point since $z_p < z_q$ and q already dominated p in terms of x and y. ———— O(logn_k) time.
If p is not dominated by any point in A then we will maintain a BST for non-dominated points with $z = z_{k-1}$ and proceed as in part(a). suppose there are n_{k-1} points with $z = z_{k-1}$. Maintaining a separate BST, say B for all the non-dominated points with $z = z_{k-1}$ will take O($n_{k-1} log n_{k-1}$) time from part(a).
Note: This set of non-dominated points are not going to be non-dominated in the entire set of n points in 3-dimension. But if there are any non-dominated points with $z = z_{k-1}$ in the entire 3-d set, then they will also lie in B. Now we are left with the task of separating those points in B which are non-dominated points for the entire set of n points.For each point in B, we will do binary search in A to find if that point is non-dominated till now or not and for each point this can be acheived in O(logn_k) time as explained earlier in this section. If we find a point in B which is dominated by any point in A then we will delete that point from B in O(logn_{k-1}). So to make decision for any point in B and updating the BST B we take O($log(n_k + logn_{k-1})$) time. Then after completely updating B we will modify A as A ← merge(A,B). This will take O($log(n_k + logn_{k-1})$) time. So total time taken will be O($n_{k-1} log(n_k + n_{k-1})$) time and then we will continue further updating our A.
T_i is the total time to process all the points with $z = z_i$.
⇒ $T_k = O(n_k logn_k)$;
$T_{k-1} = O(n_{k-1} log(n_k + n_{k-1}))$;
$T_{k-2} = O(n_{k-2} log(n_k + n_{k-1} + n_{k-2}))$;
.
.
.
⇒ $T = T_1 + T_2 + T_3 + ... + T_k$
$= O(n_1 log(n_k + ... + n_1))$
$+O(n_2 log(n_k + ... + n_2))$
.
.
.
$+O(n_k logn_k)$
$\leq O(n_1 logn) + O(n_2 logn) + ... + O(n_k logn)$
$= O((n_1 + n_2 + ... + n_k) logn)$
=O(nlogn)

2 A computational problem of an experimental physicist

Consider

$$F_j = \sum_{i<j} C \frac{q_i q_j}{(j-i)^2} - \sum_{i>j} C \frac{q_i q_j}{(j-i)^2}$$

where F_j is as defined in the problem
⇒ $F_j = Cq_i F_{j1} - Cq_i F_{j2} = Cq_i(F_{j1} - F_{j2})$
where $F_{j1} = \sum_{i<j} \frac{q_i}{(j-i)^2} = \sum_{i=1}^{j-1} \frac{q_i}{(j-i)^2}$ and $F_{j2} = \sum_{i>j} \frac{q_i}{(j-i)^2}$
If we can compute F_{j1} and F_{j2} in optimal time, then we can compute F_j by just multiplying Cq_i
For computing F_{j1}
$F_{11} = 0$

$F_{21} = \sum_{i=1}^{1} \frac{q_i}{(j-i)^2} = \frac{q_1}{1^2}$
$F_{31} = \sum_{i=1}^{2} \frac{q_i}{(j-i)^2} = \frac{q_1}{2^2} + \frac{q_2}{2^2}$
$F_{41} = \sum_{i=1}^{3} \frac{q_i}{(j-i)^2} = \frac{q_1}{3^2} + \frac{q_2}{2^2} + \frac{q_3}{1^2}$
So consider two polynomials :-
A(x) = $\frac{x}{1^2} + \frac{x^2}{2^2} + \frac{x^3}{3^2} + \frac{x^4}{4^2} + ... + \frac{x^{n-1}}{(n-1)^2}$
$B(x) = q_1x + q_2x^2 + q_3x^3 + ... + q_{n-1}x^{n-1}$
Now let C(x) = $C_0 + C_1x + C_2x^2 + C_3x^3 + ... + C_{2n-2}x^{2n-2}$
where $C_k = \frac{q_1}{(k-1)^2} + \frac{q_2}{(k-2)^2} + ... + \frac{q_{k-1}}{(k-(k-1))^2}$
which is exactly F_{k1}
For computing F_{j2} :-
$F_{12} = \frac{q_2}{1^2} + \frac{q_3}{2^2} + \frac{q_4}{3^2} + ... + \frac{q_n}{(n-1)^2}$
$F_{22} = \frac{q_3}{1^2} + \frac{q_4}{2^2} + ... + \frac{q_n}{(n-2)^2}$
Again consider two polynomilas, A(x) and P(x)
P(x) = $q_nx + q_{n-1}x^2 + q_{n-2}x^3 + ... + q_2x^{n-1}$
Let Q(x) = P(x) × A(x)
Q(x) = $\lambda_0 + \lambda_1x + \lambda_2x^2 + \lambda_3x^3 + ... + \lambda_{2n-2}x^{2n-2}$
where $\lambda_k = \frac{q_n}{(k-1)^2} + \frac{q_{n-1}}{(k-2)^2} + ... + \frac{q_{n-(k-2)}}{(k-(k-1))^2}$
$\Rightarrow \lambda_{n-k+1} = \frac{q_{k+1}}{1^2} + \frac{q_{k+2}}{2^2} + ... + \frac{q_n}{(n-k)^2}$
which is F_{k2}
Both the multiplications can be carried out in O(nlogn) time as all the polynomials have degree less than n. Therefore, all F_{j1} and F_{j2} can be calculated in O(nlogn) time. Hence, all F_j can be calculated in O(nlogn) time.