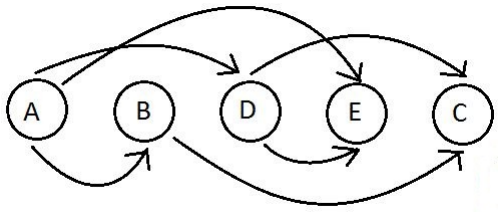
CS 610: Assignment 2

Name: Chanda Grover, 17310004

November 5, 2017

Ans 1: Since it is a directed acyclic graph, We can do a topological sort on the graph using 's' as the source in $O(m+n)$.

For example consider an example: Let 'A' be the source and 'C' be the destination for the topologically sorted graph below. [2]



Let us beginning from the leftmost vertex and increment the counts of all the vertices that have an edge to them from the vertex.

Initialize an array(counts) of 5 elements to 0.
0th iteration 0 0 0 0 0
First iteration 0 1 1 1 0
Second iteration 0 1 1 1 1
Third iteration 0 1 1 2 2
Fourth iteration 0 1 1 2 2
Fifth iteration 0 1 1 2 2
So we have two paths from A to C.

Algorithm:Count-DAG-PathsG=(V,E), s,t

1. Run the topological sort on all vertices V of G.
2. for i=1 to n
3. $count[v_i] = 0$ //initialize the count for all vertices
4. $count[s] = 1$
5. for i=1 to n

6. for each $u \in adj[v_i]$

7. $count[u] = count[u] + count[v_i]$

8. return count[t]

Time Complexity: $O(n^2)$. As it can be seen from the algo , the loop runs at max n^2 times.

Ans 2:

$$\mu_\infty = \max_i |x_i - \mu|$$

(a) Example of $(x_1, x_2, ...x_n)$ where μ_∞ is different from median and mean of the numbers. Let n=3 to generate such example.
Consider $X_1 = \{10, 10, 10\}$. Here $\mu_1 = 10$, $M_1 = 10$
$X_2 = \{2, 6, 22\}$. Here $\mu_2 = 10$, $M_2 = 6$
$X_3 = \{7, 11, 12\}$. Here $\mu_3 = 10$, $M_3 = 11$
We can see mean is same in all three examples but median is different in all. In X_1, mean=median. In X_2, median is less than mean, and in X_3 median is greater than mean. We have just tried to generate the required example with different types of distributions.[5]

Let's see if we can get required μ_∞ in all three.
For X_1, $\mu_\infty = max\{0, 0, 0\} = 0$
For X_2, $\mu_\infty = max\{8, 4, 12\} = 12$
For X_3, $\mu_\infty = max\{3, 1, 2\} = 3$
Therefore all three examples are the ones which are required in question.

(b) Algorithm to compute μ_∞ in $O(n)$

1) Compute mean, μ of the the distribution in $O(n)$ time.
2) max=0
3) for i=1 to n
4) {
5) $temp = |x_i - \mu|$
6) if(max<temp)
7) max=temp
8) }
9) return max

Time Complexity: Since only one for loop is there and it is executing n number of times, therefore it will take $O(n)$ time.

Ans: 3 *Input:* Two separate databases, each containing n elements, so there are $2n$ elements. All elements are distinct.

Output: We have to find median value of the two databases in $O(\log n)$ time so that minimum number of queries hit the database.

We will try to solve this problem with the Divide and Conquer technique, Since in the question it is demanding $O(\log n)$ time, Therefore it is giving intuition that we have to apply divide and conquer algo i.e. divide by two. Let us assume that values in both the databases are already sorted.[3]
Let us start solving problem with smaller number of elements, say $n = 3$ in both lists. For Example, $d_1 = \{1, 3, 6\}$ $d_2 = \{2, 3, 9\}$,
Sorted list of both databases will be: {1,2,3,3,6,9} and hence $median = (3 + 3)/2 = 3$. This giving intuition that if median of both the lists are same then return that, it will be the median of both databases.

Now, let $d_1 = \{1, 3, 6\}$ $d_2 = \{2, 4, 8, 9\}$
Sorted list of both databases will be: {1,2,3,4,6,8,9} and hence $median = 4$.
Here $M1 = 3$ and $M2 = 6$, i.e. M1 < M2, it means the median of both the databases could be possibly present in either the second half of d_1 or in the first half of d_2, since first half of d_1 are obviously smaller than M1 and for the same reason second half of d_2 are larger than d_2, therefore we can discard them. New median of both the databases could be present between and M2. Here is the formal explanation.

Algo: Median(d_1,d_2) //Let d_1 and d_2 are the two databases

1) Determine the medians of each databases, say $M1$ and $M2$.
2) If $M1$ and $M2$ are equal, then return $M1$ or $M2$.
3) else
4) If $M1 > M2$, then median could be present in possibly two halves:
   a) First half of d_1
   b) Second half of d_2
5) If M1 < M2, then median could be present in possibly two halves:
   a) Second half of d_1
   b) First half of d_2
6) Repeat the above process until the size becomes 2
7) If the size of two databases is 2, then $median = (max(d_1[0], d_2[0]))+(min(d_1[1], d_2[0]))/2$

Time Complexity: $O(\log n)$, Since we are dividing the list in halves and recurrence relation is given by :

$$T(n) = T(\frac{n}{2}) + \emptyset(1)$$

$$T(n) = O(\log n)$$

Ans: 4 *Input:* A sequence of n numbers $a_1, a_2, ...a_n$,all are distinct.

Output: Determine Significant inversion such that for every i<j , we have $a_i > 2a_j$ in $O(n \log n)$ time.[6]
Since output is demanding in $O(n \log n)$ time, it gives us intuition that we have to apply divide and conquer technique. If we have two elements, simply check if i < j and $a_i > 2a_j$, then return 1 else return 0. This will take $O(1)$ time.
Otherwise if we more than two elements, then we will divide the list into two halves. We will keep the count of inversions in each half and in the merged array again we will count the number of inversions in $O(n)$ time. Below is the algorithm:

Algo: Count Significant inversion(A, temp,left, right)

1. invct=0
2. if(right>left)
3. {
4. mid=(left+right)/2
5. invct=Count Significant inversion(A,temp, 0,mid)
6. invct+=Count Significant inversion(A,temp,mid+1, right)
7. invct+=MergeCount(A,temp, left,mid, right)
8. }
9. return invct

MergeCount(A, temp, left, mid, right)

1. i=left
2. j=mid
3. k=right
4. while(i < mid-1) and (j < right)
5. if (arr[i] *leq* arr[j])

6. temp[k++]=arr[i]
7. else
8. temp[k++]=arr[j]
9. invct=invct+(mid-i)
10. copy the remaining elements of left subarray to temp.
11. copy the remaining elements of right subarray to temp.
12. return invct

Time Complexity: $T(n) = 2T(\frac{n}{2}) + O(n)$
$T(n) = \emptyset(nlogn)$

Ans: 5

$$F_j = \Sigma_{i<j} \frac{Cq_iq_j}{(j-i)^2} - \Sigma_{i>j} \frac{Cq_iq_j}{(j-i)^2}$$

Given: An algorithm for computing the force between charged particles is in $O(n^2)$.
Output: Optimize the algorithm and have to compute the force between each particle in $O(n \log n)$

Claim: The given problem can be reduced to **Convolution**.
We can keep one charged particle in one vector and distance between each of these particle with other fixed charged particle say q_j in other vector. Finally multiplying both of these using convolution, we can find the force in $O(n \log n)$
Let $q_i = \{q_1, q_2, q_3, ...q_n\}$
Let $D_i = \{\frac{1}{n^2}, \frac{1}{(n-1)^2}, \frac{1}{(n-2)^2}, ...\frac{1}{2^2}, \frac{1}{1^2}, 0, 1, ...\frac{1}{(n-1)^2}, \frac{1}{n^2}\}$
Applying convolution over these two vectors, we get $(q_i * D_i)$=

$$\Sigma_{i<j} \frac{Cq_i}{(j-i)^2} - \Sigma_{i>j} \frac{Cq_i}{(j-i)^2}$$

Now, multiplying this result with q_j, we get Force acting on jth particle, as it was asked in question.i.e

$$F_j = \Sigma_{i<j} \frac{Cq_iq_j}{(j-i)^2} - \Sigma_{i>j} \frac{Cq_iq_j}{(j-i)^2}$$

We can consider q_i and D_i as two polynomials and their multiplication as convolution.[6][7]

Generally, a polynomial $A(x) = a_0 + a_1x + ... + a_dx_d$ can be specified by either one of the following:

1. Its coefficients $a_0, a_1, ..., a_d$

2. The values $A(x0), A(x1), ..., A(xd)$

Of these two representations, the second is the more useful for polynomial multiplication. Value of the product C(x) at any given point z can be easily determined, just A(z) times B(z). Thus polynomial multiplication takes linear time in the value representation.i.e. $O(n)$ But we expect input polynomials to be specified by coefficients. So we need to first translate from coefficients to values which means evaluating the polynomial at the chosen points and then multiply in the value representation, and Finally translate back to coefficients, a process called interpolation

Polynomial multiplication(A,B) //Let degree of polynomial=d

1. **Selection**: Pick some points $x_0, x_1, ...x_n - 1$
2. **Evaluation**: Compute $A(x_0), A(x_1), ..., A(x_n-1)$ and $B(x_0), B(x_1), ..., B(x_n-1)$
3. **Multiplication**: Compute $C(x_k)$: $A(x_k)B(x_k) \forall k = 0, 1, ...n - 1$.
4. **Interpolation**: Recover $C(x) : c_0 + c_1x + c_2x^2.....c_2dx^2d$

Time-Complexity: $O(n \log n)$

Now translation from coefficients of polynomials to value ans vice versa problem is called FFT, Which can be solved in $O(n \log n)$ time. Therefore Overall solving this problem takes $O(n \log n)$ time, Although multiplication of polynomials simply takes $O(n)$ time.

Ans : 6 Given: n non vertical lines in the plane labelled $L_1, L_2, ...L_n$with the ith line specified by $y = a_ix + b_i$

Assumption: No three lines meet at a single point

Output: To show set of visible line in $O(n \log n)$ time.Visible lines are defined as if there is some x-coordinate at which it is uppermost.

Solution: Since we have to output the result in$O(n \log n)$ time, it gives us intuition that we can apply Divide and Conquer technique.
Let us sort the given set of lines in increasing order of their slopes.

Claim: First and last lines in this sorted will always be visible. or we can say that the line with the minimum and maximum slopes in a set will always be visible.

Proof of Claim: L_i is visible if there is some x-coordinate at which it is uppermost and line L_i is uppermost at a given co-ordinate x0 if its y coordinate at x0 > the y-coordinates of all other lines.

(6) Condition for min slope (visible line)

$m_j^* x^* + c_j^* > m_j^* x + c_j$

$x^* (m_j^* - m_j) > c_j - c_j^*$

$x^* (m_j - m_j^*) > c_j^* - c_j$

$x^* < \frac{c_j^* - c_j}{m_j - m_j^*}$

Where m_j^* is the min. slope of all lines.

Condition for maximum slope (visible Line)

$m_i^* x^* + c_i^* > m_i x^* + c_i$

$x^* (m_i^* - m_i) > c_i - c_i^*$

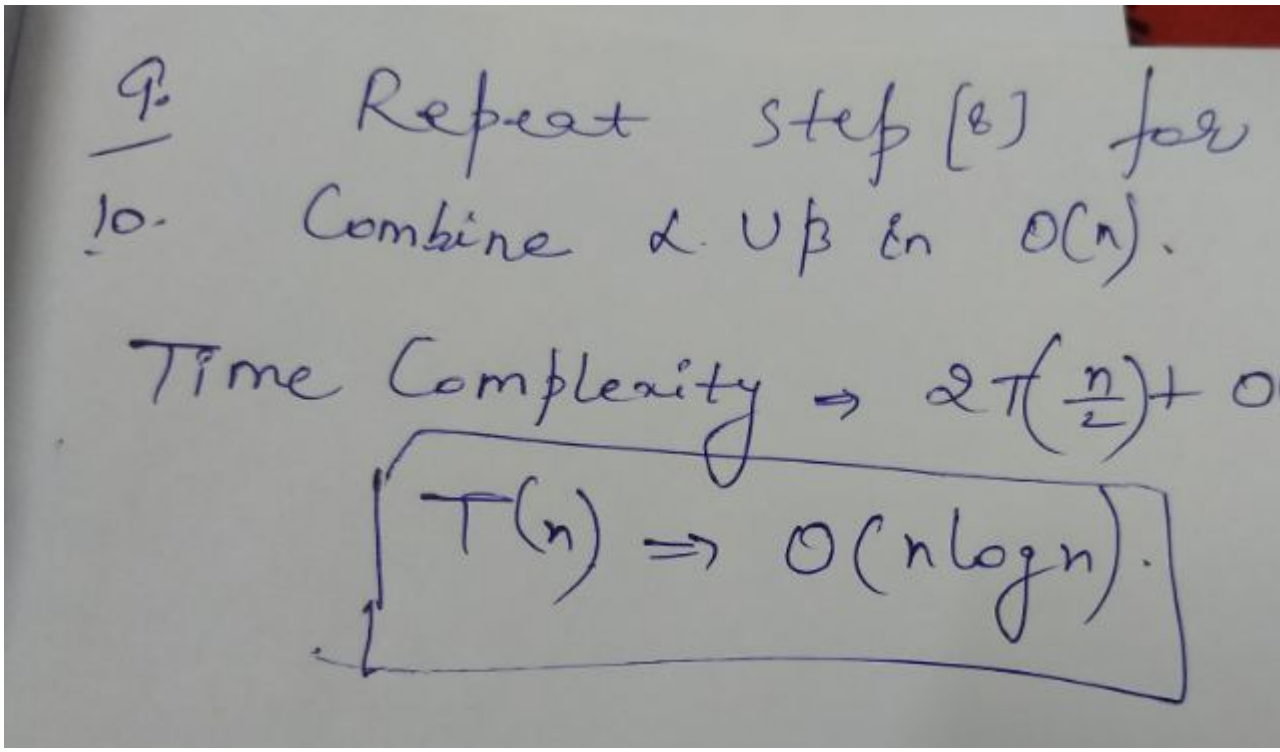
$x^* > \frac{c_i - c_i^*}{m_i^* - m_i}$

Where m_i^* is max. slope of all lines.

∴ We can claim that the line with max. & min. slope will always be visible.

Algo:- HSR(Li).

1. Sort all the lines in increasing order of slopes. ⇒ O(nlogn)
2. If ($n \leq 3$) // base case
3. I st & third line will always be visible. 2nd line will be visible iff it meets the first line to the left of where the third line meets the first line. ⇒ O(1).
4. else
5. Divide the set of lines into 2
6. $\alpha = L_1, \ldots L_m \Rightarrow T(n/2)$
7. $\beta = L_{m+1} \ldots L_n \Rightarrow T(n/2)$.
8. Compute $a_1, a_2, \ldots a_p$ of the point of intersection of line L_k with $L_{i<k}$. $a_1, a_2 \ldots a_p$ are in order for α.



Ans: 7 part a) Given: Minkowski sum $X + Y$ as the following set of integers : $\{x + y | x \in X, y \in Y\}$. Naive Algorithm, Time Complexity = $O(n^2 \log n)$ time.

Claim: This problem is reduced to $X + Y$ Sorting problem.
[9][8][10][1]

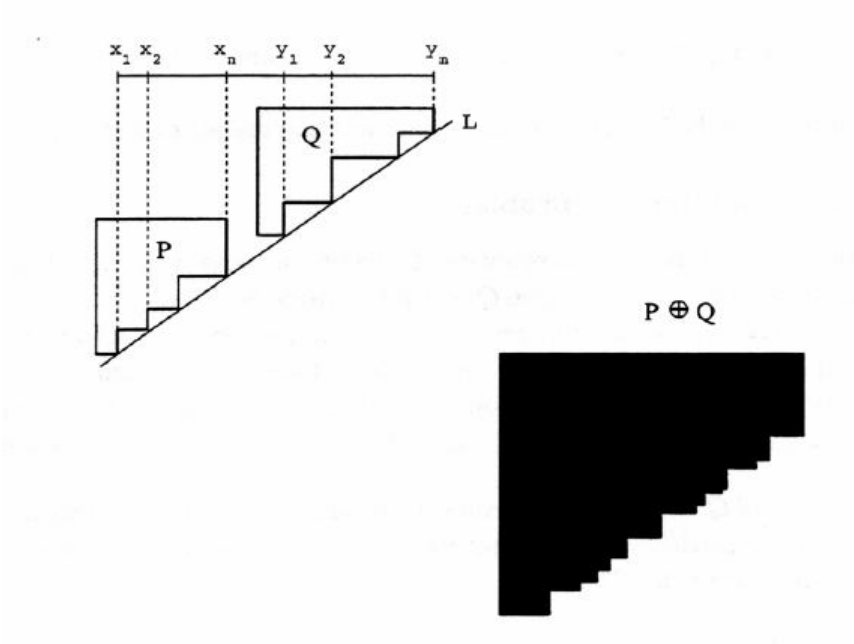
$X+Y$ sorting is the problem of sorting pairs of numbers by their sum. Given two finite sets X and Y, the problem is to order all pairs (x, y) in the Cartesian product XY by the key $x + y$.

Since size of both the sets is given to be n, Therefore time complexity for sorting n^2 elements will take $O(n^2 \log n^2)$ time.

$$O(2n^2 \log n) = O(n^2 \log n)$$

Proof of Claim:
Let P and Q be the two polynomials such that intersection of every line is orthogonal to L in a connected interval. We generate the vertices of stair case part in part P and Q by taking them along same line. The intersection of a vertical line at $x_i(y_j)$ with line L will be a vertex of P(Q) on the stair case. By

traversing the vertices on lower stair case $P + Q$, we will get sorted values of $X + Y$. Given figure has been taken from reference [9]



part b) Given: M is the largest absolute value of $X \cup Y$.
Output: Give $O(M \log M)$ algorithm for Minkowski Sum.

Claim: Minkowski Sum problem can be solved to FFT problem.

Proof of Claim: We can consider the given two sets as two polynomials and then adding each pair of elements . e.g.

$$A(x) = \{a_0 + a_1 x + a_2 x^2 + ...a_p x^p\}$$
$$B(x) = \{b_0 + b_1 x + b_2 x^2 + ...b_r x^r\}$$

1. Generate the vectors $A[i] = 1$, if $i \in A, 0$ otherwise and similarly for B, i.e. $B[j] = 1$, if $j \in B, 0$ otherwise. Each such vector is of size M.
2. Compute the convolution of A and B using FFT (time: $O(MlogM)$). Output size is $O(M)$.
3. Scan output O - at every index, O[i] is nonzero, iff i is an element of the Minkowski sum.

$O[i]! = 0$ iff $\exists$ k such that $A[k]! = 0$ and $B[i-k]! = 0$, iff $k \in A$ and $i - k \in B$, iff $k + i - k$, that is i, is in the Minkowski sum.[4]
Time Compexity: $O(MlogM)$

Collaborators: Ishita Doshi, Priyanka, Anirban Sir

References:

1. https://en.wikipedia.org/wiki/
2. https:// www.quora.com
3. https://www.geeksforgeeks.com
4. https://stackoverflow.com
5. https://sol.du.ac.in
6. Chapter 5: "Algorithms Design" by Kleinberg and Tardos
7. Chapter 2 "Algorithms" by S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani
8. Fast Implementations of Morphological Operations using Fast Fourier Transform(FFT) by Olga Kosheleva, Sergio D.Cabrera, Glenn A. Gibson and Misha Koshelev
9. Finding an $(O(n^2 logn))$algorithm is sometimes hard by Antonio Hernandez Barrera
10. A. Hernandez Barrera. Computing the Minkowski sum of monotone polygons. In abstracts of the 12th europeon workshop on computational geometry. pp 113-116, 1996.