Question 1

Internalizing Min-cut Algorithm

1.

2.

Time Complexity :
Each contract step takes $O(n)$ and we are contracting $\frac{n}{2}$ times so this results in $O(n^2)$ work.
Thus if the time complexity of the algorithm is $T(n)$ then,

$$T(n) = O(n^2) + 2T(\frac{n}{2})$$

$$T(n) = O(n^2) + O(\frac{n^2}{2}) + 2T(\frac{n}{4})$$

$$T(n) = O(n^2) + O(\frac{n^2}{2}) + O(\frac{n^2}{8}) + \dots$$

$$T(n) = O(n^2)$$

Error Analysis :
Let the probability that algorithm preserves the min-cut be $q(n)$.
As shown in class, probability that min-cut is preserved after $\frac{n}{2}$ contractions is at least $\frac{1}{4}$. Thus, we can say that $q(n)$ is at least $\frac{1}{4}$ times the probability that at least one smaller recursive call of size $\frac{n}{2}$ preserves the min-cut. So,

$$q(n) \geq \frac{1}{4}\left(1 - \left(1 - q\left(\frac{n}{2}\right)\right)^2\right)$$

$$q(n) \geq \frac{1}{4}\left(2 - q\left(\frac{n}{2}\right)\right) q\left(\frac{n}{2}\right)$$

$$q(n) \geq \frac{1}{2}q\left(\frac{n}{2}\right) - \frac{1}{4}q\left(\frac{n}{2}\right)^2$$

Now let $p(k) = q(2^k)$. So we have,

$$p(k) \geq \frac{1}{2}p\left(k-1\right) - \frac{1}{4}p\left(k-1\right)^2$$

Call the above inequality to be I_1.
Since $p(k) \leq 1 \ \forall k$ we get

$$p(k) \geq \frac{1}{2}p\left(k-1\right) - \frac{1}{4}p\left(k-1\right)$$

or,

$$p(k) \geq \frac{1}{4}p\left(k-1\right)$$

By the above inequality and I_1 we have,

$$p(k) \geq \frac{1}{2}p\left(k-1\right) - p(k)p\left(k-1\right)$$

Thus,

$$\frac{1}{p(k)} \leq \frac{2}{p(k-1)} + 2$$

Base case - $p(1) = 1$

$$\frac{1}{p(k)} \leq 2\left(\frac{2}{p(k-2)} + 2\right) + 2$$

$$\frac{1}{p(k)} \leq \frac{2^2}{p(k-2)} + 2 + 2^2$$

In general,

$$\frac{1}{p(k)} \leq \frac{2^i}{p(k-i)} + 2 + 2^2 + \cdots + 2^i$$

So,

$$\frac{1}{p(k)} \leq \frac{2^{k-1}}{p(1)} + 2 + 2^2 + \cdots + 2^{k-1}$$

$$\frac{1}{p(k)} \leq 2^{k-1} + 2(2^{k-1} - 1)$$

$$\frac{1}{p(k)} \leq 2^k + 2^{k-1} - 2$$

Thus,

$$\frac{1}{q(n)} \leq n + \frac{n}{2} - 2$$

$$q(n) \geq \frac{2}{3n-4}$$

So the probability that the algorithm preserves min-cut is at least $\frac{2}{3n-4}$
To get error probability less than $\frac{1}{n^c}$ repeat our algorithm $cn\log(n)$ times.
Hence, the time complexity of the complete algorithm to get an inverse polynomial error bound is $O(n^3\log(n))$.

Question 2

A surprising problem from computational geometry

1.

We will construct the line segments in order they are to be selected in the permutation. Let's say the first line segment is a horizontal line segment of length 1 having its' left end at origin. Construct the second line segment such that it intersects the extended first line segment on the right side, and makes a small positive angle with the horizontal. Construct the third line segment such that it will intersect both the previous extended line segments, this line segment makes an angle with the horizontal which is just greater than the previous line segment. Now consider that we have constructed $i-1$ line segments in this manner, construct the i^{th} line segment intersecting all previous extended line segments and making an angle just greater than the $(i-1)^{th}$ line segment with the horizontal. We can always construct such a line segment by chosing a suitable length and suitable angle with the horizontal for the i^{th} line segment. Do this till we get n line segments.
Note - We are considering angle with the horizontal in the anticlockwise sense.
This procedure will result in $\binom{n}{2}$ points of intersection as every pair of line segments will have a distinct point of intersection if constructed this way.
Thus, there exist a set of n segments and a permutation for which the above algorithm will cause $\binom{n}{2}$ points of intersections.

2.

Take any line segment, call it L. We will look at all points of intersection which we get after extending this segment towards right, the other case is symmetrical.
Label the line segments which intersect the right extended part of L to be $1, 2, ..., i, ...$ in order from left to right.

Fact 1: If line segment L has a point of intersection with line labelled i then L should have been extended before all the line segments labelled 1 to $i-1$.

Using Fact 1 and the fact that initially line segments are permuted randomly uniformly(so the probability that any segment is picked first among a subset of size k is $\frac{1}{k}$), the probability that L has a point of intersection with line labelled i is at most $\frac{1}{i}$.
Consider random variable X_i taking value 1 if line segment i intersects with L otherwise 0.
So,

$$E[X_i] = P(X_i = 1) * 1 + P(X_i = 0) * 0 \implies E[X_i] = \frac{1}{i}$$

Let X be the number of points of intersection caused by line segment L when extended towards right.
By Linearity of Expectation,

$$E[X] \leq \sum_{i=1}^{n} E[X_i]$$

$$E[X] \leq \sum_{i=1}^{n} \frac{1}{i}$$

$$E[X] = O(\log n)$$

Similarly, we can get the same bound on the expected number of points of intersection caused by line segment L when extended towards left.
Thus expected number of points of intersection caused by the extension of line segment L is $O(\log n)$.
No two line segments intersected with each other initially, so all points of intersection will be due to extension of some line segment.

Let the total number of points of intersection is N.
By linearity of expectation,

$$E[N] = n * O(\log n)$$

$$E[N] = O(n \log n)$$

Thus, for any arbitrary set of n line segments, if we permute them randomly uniformly and then execute the above algorithm, the expected number of points of intersection will be $O(n \log n)$.

Question 3

Internalizing Backward Analysis and Randomized Incremental Construction (RIC)

A point x_i, y_i is said to dominate another point x_j, y_j if $x_i > x_j$ and $y_i > y_j$.
Non dominant point: A point is said to be non dominant if there is no point in P which dominates it.
So we need to find an expected $O(n \log n)$ algorithm to compute all non-dominanted point in P. We'll solve this using *Randomized Incremental Construction*, using conflict graphs.

Some notations used in analysis are G_i : denotes the conflict graph after i^{th} step, p_i : i^{th} point in our incremental construction, I_i : i^{th} interval on X axis.

Conflict Graph Details

Our conflict graph to solve this problem is based on interval division. Formally at i^{th} stage, conflict graph C_i will store 2-way mapping between set of points and set of intervals (division of X axis).

If $I_1 = 0 \to x_1, I_2 = x_1 \to x_2, I_k \ldots x_{k-1} \to x_k$ denote the intervals at i^{th} stage (x_k being the point in P having largest x coordinate), then our conflict graph will store which interval, point p belongs to $\forall p \in P$. Also we store a list of all the points belonging to a particular interval. Lastly we'll also store the points which define this interval, ie $x_{i-1}, x_i \in P$ for $I_i, \forall i \in [1, k]$, and the interval height (defined by H_i for I_i interval, $H_i = \max y_j$ for $j \in [1, i]$). Note that these interval heights may change when we proceed incrementally.

Some operations and their complexity analysis are:

- **Locating interval of** $p_i \in P$: Well store explicit mapping of interval for each point, so this opearation will take $O(1)$ time.
- **Determining whether** G_{i+1} **will be different from** G_i : Note that G_{i+1} will change only when we change the intervals in our graph. Interval will change if y-coordinate of p_{i+1} is greater than H_t, where H_t is the height of interval at i^{th} stage to which p_{i+1} belong. We can get the interval to which p_{i+1} belongs in $O(1)$, also its height in $O(1)$, we determining whether G_{i+1} will be different will take $O(1)$ time.
- **Updating** G_{i+1}**:** We will update G_{i+1} only if p_{i+1} has height greater than interval height to which p_{i+1} belong to (let this interval be I_t). In that well remove $I_t = (x_{t-1} \to x_t)$ and add 2 new intervals: I_t^1, I_t^2, such that $I_t^1 = (x_{t-1} \to x_{p_i}), I_t^2 = (x_{p_i} \to x_t)$ and height of I_t^1 is equal to y-coordinate of p_{i+1} and height of I_t^2 is equal to height of I_t. Note that we have to change the mapping for all the points which belonged to I_k, by convention $p_{i+1} \in I_t^1$. Also we need to update neighbors of I_t^1, I_t^2 using neighbor data of I_t. So in worst case there can be n points whose mapping needs to be changed, ie worst case time = $O(n)$.

One last thing before we proceed further is that we'll also store the point p which is on the top right corner of a interval. We have seen above that this point p caused creation of this interval.

So using these details of implementation as background our algorithm to find non dominant points is:

Algorithm 1: RIC based algorithm for finding Non dominant points.

Input: $P = p_1, p_2, \ldots p_n$: set of n points in $2 - D$ plane
Output: Set of non dominant points in P
$G_1 = \{I = \{I_1\}, M\}$ (initial conflict graph, here $I_1 = (0 \rightarrow x_{mx})$, where x_{mx} is maximum X coordinate of all points in P)
/*
In subsequent step M will denote the map from set of points to set of intervals
I will denote the set of intervals
I_k will denote k^{th} interval, this will store data for height and interval limit(on X axis).
$H[I_k]$ will denote height of k^{th} interval.
I_k is defined by $x_{k-1} \rightarrow x_k$
$D[I_k]$ will denote the generating point for interval I_k
*/

```
for i ← 1 to n do
    M[p_i] = I_1
end
I_1 = (0 → x_mx) H[I_1] = 0
D[I_k] = p_{x_mx} // defined by point having maximum value of X coordinate. for i ← 1 to n do
    I_k = M[p_i]
    if y_{p_i} > H[I_k] then
        // have to update G
        I_k^1 = x_{k-1} → x_{p_i}
        I_k^2 = x_{p_i} → x_k (since I_k = x_{k-1} → x_k)
        for j ← 1 to n do
            if M[p_j] = I_k then
                if x_{p_j} ≥ x_{k-1} & x_{p_j} ≤ x_{p_i} then
                    M[p_j] = I_k^1
                else
                    M[p_j] = I_k^2
        end
        H[I_k^1] = y_{p_i}
        H[I_k^2] = H[I_k]
end
S = φ
for Q ∈ I do
    S = S ∪ {D[Q]} // include defining point of this interval
end
return S
```