Answer 1:

The problem is similar to the convex hull problem. Hence, we will try to solve it via similar approach: 1. Sorting the points (in this problem lines) via abscissa (in this problem slope) and 2. adding the points (or lines) one by one and checking condition for a viable convex hull (or in this problem a viable intersection of half planes). We know that the procedure described above for the convex hull problem takes $O(n \log n)$.

In the given problem the half planes are sorted by their slopes and we need to find the intersection of the given half planes in $O(n)$ time complexity.

We can solve the problem in the following manner. First, divide the set of half planes into upper half planes (U) and lower half planes (L) (What to do when we have vertical half planes?). In this way we have two sets of half planes. In the second step, we will find the intersection of all the half planes present in just the set U and similarly find the intersection of all the half planes in just the set L. This step will provide us two sets of line segment chains. In the last step, we just need to find intersection between these two sets of line segment chains to obtain the common intersection of all the half planes given.

First step: Can easily be done in $O(n)$. Just scan each half plane and put them into either the set U or the set L. For set U, if there are multiple half planes with same slope then only keep the half plane with highest Y intercept. For set L, if there are multiple half planes with same slope then keep the half plane with lowest Y intercept.

Second step: For this step, we scan the half planes that are sorted by their slopes and construct viable intersection iteratively. We will look into the pseudocode for finding intersection for the set U. For the set L, we can use the similar approach.

*Please report if you find any error in the solutions*
Pseudocode:
Let $U$ be the set of $n$ upper half planes, sorted by their slopes.
Let $Q$ be a priority queue storing valid half planes.
Let $P$ be a priority queue storing intersection points

\begin{verbatim}
findCommonIntersect(U):
    $h_1 =$ first half plane in $U$;
    Store $h_1$ in Q;
    $S = U - h_1$;
    \textbf{for} half plane $k$ in set $S$ \textbf{do}
        \textbf{for} $i$ from 1 to length(Q) \textbf{do}
            top = Top element from Q;
            top' = Top element from P;
            \textbf{if} top == NULL \textbf{then}
                Store k in Q;
                \textbf{break};
            \textbf{else}
                temp = Compute intersection of half planes top and k;
                \textbf{if} temp is valid \textbf{then}
                    Store temp in P;
                    Store k in Q;
                    \textbf{break};
                \textbf{end}
            \textbf{else}
                Remove top from Q;
                Remove top' from P;
            \textbf{end}
        \textbf{end}
    \textbf{end}
\end{verbatim}

Similarly for set L, we can obtain sets $P'$ and $Q'$ corresponding to $P$ and $Q$ of $U$ respectively.
Third step: Compute intersection of two regions obtained from half planes in $U$ and $L$. We can compute the intersection of upper and lower sets in $O(n)$. 
merge(X,Y):
    Let A and B be two intersection points between X and Y;
    Scan X and Y from left to right using plane sweeping.
    
    if $A \neq B$ then
    | Return concatenation of chains X and Y;
    end
    
    else
    | if $A=B$ is the only intersection point then
    | | Return A;
    | end
    | else
    | | The region is unbounded;
    | | Return chains of X and Y to the left or right of A;
    | end
    end

halfPlanesIntersect(X,Y):
    X = findCommonIntersect(U);
    Y = findCommonIntersect(L);
    merge(X,Y)

Complexity analysis:
findCommonIntersect() takes $O(n)$.
merge() takes $O(n)$.
Hence, total time complexity is $O(n)$.

Correctness: We define two procedures to solve the problem. The correctness of the procedure merge() can be trivially verified. We argue that findCommonIntersect() correctly finds the common intersection. Similar case of convex hull construction, in each step, a line is included in the intersection of half planes so far. Hence, by induction we can prove that this procedure will find the correct intersection of half planes.
**Answer 2:**

**Approach:** We can solve the given problem using the concept of plane sweeping. Here, instead of line segment intersection we are looking into intersection of disks. Any given two disks intersect iff the distance between their center positions is less than or equal to the sum of their radii.

**Pseudocode:**

Let $D$ be the set of $n$ disks. Each disk $d_i$ has its center position $c_i = x_i, y_i$ and radius $r_i$. First we will transform each disk into their corresponding line segments (by defining the left and the right points of each disk as $(x_i - r_i, y_i)$ and $(x_i + r_i, y_i)$)

\[
diskIntersect(d_1, d_2): \\
\quad \text{return } |c_2 - c_1| \leq r_1 + r_2
\]

\[
anyDiskIntersect(D): \\
\quad T = \text{an empty balanced binary tree.}; \\
\quad \text{Sort the line segments corresponding to the disks.}; \\
\quad \text{for each point } p \text{ in the sorted list of end points do} \\
\quad \quad \quad \text{if } p \text{ is the left end of disk } d \text{ then} \\
\quad \quad \quad \quad \quad \text{INSERT}(T,d); \\
\quad \quad \quad \text{end} \\
\quad \quad \quad \text{if } (\text{ABOVE}(T,d) \text{ exists and intersects } diskIntersect(\text{Above}(T,d),d) \text{ OR} \\
\quad \quad \quad \quad \quad \text{BELOW}(T,d) \text{ exists and intersects } diskIntersect(\text{BELOW}(T,d),d) \text{ then} \\
\quad \quad \quad \quad \quad \text{return TRUE;} \\
\quad \quad \quad \text{end} \\
\quad \quad \quad \text{else if } p \text{ is the right end of a disk } d \text{ then} \\
\quad \quad \quad \quad \quad \text{if both } \text{Above}(T,d) \text{ and } \text{BELOW}(T,d) \text{ exist and} \\
\quad \quad \quad \quad \quad \quad \text{diskIntersect(\text{BELOW}(T,d),\text{Above}(T,d)) \text{ then} \\
\quad \quad \quad \quad \quad \quad \quad \quad \text{return TRUE;} \\
\quad \quad \quad \quad \quad \text{end} \\
\quad \quad \quad \text{DELETE}(T,d); \\
\quad \text{end} \\
\quad \text{return FALSE;}
\]
Complexity analysis: The problem of disk intersection was reduced to the problem of finding the line segment intersection. Hence, the time complexity would be same as for the line segment intersection problem, i.e., $O(n \log n)$

Correctness: The problem is solved using plane sweeping. The procedure checks for an intersection every time a new pair of adjacent disks is added in the binary tree (T). Because, every pair of adjacent disks are checked for the intersection, the algorithms returns TRUE if it finds an intersection and FALSE if not.

Answer 3:

(A): The problem can be decomposed into two subproblems of finding upper and lower convex hulls. In this way we have two sets: one consisting of upper halves of polygon $P_1$ and $P_2$, and other sets consisting of lower halves of polygon $P_1$ and $P_2$. Apply plane sweeping on both the sets to find upper and lower convex hulls. Finally, merge the upper and lower convex hulls to find the final convex hull.

Pseudocode:

Let $P_1$ and $P_2$ be two convex polygons with $n_1$ and $n_2$ vertices, respectively. The upper convex hull of a convex polygon can be found by scanning the vertices from left until first decrease in the x-value of a point. The lower points can be extracted when we know all the upper points.

mergeConvexPoly($P_1, P_2$):

- $U_1 = \text{the upper convex hull of } P_1$;
- $U_2 = \text{the lower convex hull of } P_2$;
- $L_1 = \text{the upper convex hull of } P_1$;
- $L_2 = \text{the lower convex hull of } P_2$;
- $U = \text{upperConvex}(P_1, P_2)$;
- $L = \text{lowerConvex}(P_1, P_2)$;

The procedures upperConvex() and lowerConvex() are nothing but the procedures to find convex hulls of upper and lower points respectively. This can be done by simple plane sweeping (For algorithm on how to find convex hull please see CLRS page 1029).

Complexity analysis: Since, the vertices are ordered, the plane sweeping will have $O(n_1 + n_2)$ time complexity.

(B): The problem can be solved by using divide and conquer strategy to recursively find convex hulls. First, divide the points in arbitrary manner into two equal sets. Then, recursively find the convex hull of the two sets. Finally, compute the convex hull of the two sets.

Pseudocode:
Let $P$ be points drawn from the sparse-hulled distribution.

```plaintext
if $|P| \leq 3$ then
    return $P$;
else
    Split the points into two sets $P_1$ and $P_2$, arbitrarily;
    temp = mergeConvexHulls($P_1, P_2$);
    return temp;
end
```

The mergeConvexHulls() is nothing but the solution for the first part of this problem. As we have seen, this can be done in linear time of the size of the convex hulls.

**Complexity analysis:** The expected size of the convex hull be given as $O(n^{1/3})$. The recursive relation for the procedure is given by

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^{1/3})$$

Using master theorem we can prove that $T(n) = \theta(n)$

**Correctness:**

The base case is correct since convex hull of three points contains all the three points. As we have proved the correctness of merging two convex hulls in the previous part of this question. By induction, the proposed procedure is correct.

**Answer 4:** We can solve this problem by using plane sweep technique. But, instead of traditional plane sweep we will make some modifications. From any given endpoint, we will do angular sweep of all other endpoints. First, sort all other $2n-1$ end points by their angle from the origin (current endpoint). Then, perform angular sweep, maintaining sorted order of line segments by their y coordinates (from the origin or current endpoint). In general, this procedure is like performing plane sweeping from $2n$ endpoints. Hence, we expect the time complexity to be $O(n^2\log n)$

**Time complexity:**

At each $2n$ endpoints $O(n)$ we need to sort ($O(n\log n)$) other $2n - 1$ endpoints. Hence, the algorithm takes $O(n^2\log n)$.

**Correctness:**

In order to calculate all the edges of the visibility graph, the algorithm visits every endpoint from the current origin and connects if there is no obstacle between them. In a sense, the procedure does not miss to check any other endpoint for its involvement in the visibility graph.

**Pseudocode:**
Compilability Graph (S)

- V = all vertices in S
- E = empty
- G = (V, E)
- sorted V = sorted the remaining points in V (other than V_i) in a clockwise order around V_i
- obstacles = bi-edges are broken by giving preference to points close to V_i
- V_i = (V_i) + \infty, V_{i-1}/ horizontal line from V_i to the right
- obstacles = balanced search tree initially filled with segments intersected by horizontal sweep line
- the segments are added in the order they intersect the sweep line
- for y = 0 \Rightarrow j = sorted V_i, count
- obstacles if (isVisible (V_i, sorted V)) suspended until intersection
- obstacles = add (V_i, sorted V_i), distance (V_i, sorted V_i)
- obstacles = add (sorted V_i, V_j)
- obstacles = delete (segment V_j)

return n

isVisible (origin, end, keep, prevEnd)

1. if (loop == 0) or (prevEnd is not on segment origin \rightarrow end)
2. if (isLeft (V_i, sorted V)) || add that origin intersects (until now...)
3. if (obstacle_one == null) and (origin \rightarrow end intersects obstacle) and (origin and end
4. not part of obstacle segment)
5. return false
6. else return true
7. else if (prevEnd is not visible) return false.
8. else
9. obstacle = edge in obstacles that intersects prevEnd \rightarrow end
10. if (obstacle_one == null) return false
11. else return true