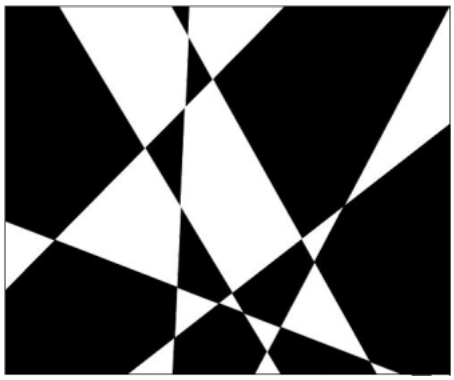
A sidebar indicates the "answer" for each step — it is content that is part of following the algorithm development process, separate from commentary about the process or the solution itself.

Coloring Regions

Specifications. State complete specifications for the problem. What is the problem? What do you start with (input) and what is the end result (output)? What are the legal input instances and the required output for each? For optimization problems, distinguish between legal solutions and optimal ones.

> Given a subdivision of a plane defined by n lines, color each region either black or white so that any two regions sharing a boundary have different colors.
>
> Input: n lines
>
> Output: a coloring of the regions (assignment of color to region)
>
> Legal solution: every region has a color, and any two regions sharing a boundary have different colors

Examples. If needed, give examples (specific inputs and the corresponding outputs) of typical and special cases to clarify the specifications.



Targets. What are the time and space requirements for your solution?

> No time or space constraints are given as part of the problem. There are n lines so it seems difficult to do better than $\Theta(n)$.

Tactics. The time and space constraints may narrow down the algorithmic options and/or may guide you in particular directions. Consider both things you can and can't do.

Approaches. Consider specific iterative patterns — what would they look like if applied to this problem?

The input involves lines and the output is a coloring of regions.

> Process input: for each line, incorporate it into the coloring.
>
> Produce output: for each region, assign it a color.

Narrowing the search space isn't applicable, as this isn't a search problem.

Main steps. This is the core of the algorithm — the main loop, focusing on the loop body. What's being repeated?

It's not clear how to iterate through all of the regions in any consistent way, plus the regions are going to have to be computed from the lines themselves. So, go with the process input approach, which means incorporating each line into the coloring. How? This is a case where working on an example can be useful. Start with a coloring with no lines, then add one, adjust the coloring, add another, ...

```
for each line
  add it to the subdivision, computing new regions
  flip the colors (black to white, white to black) of all of the regions on
   one side of the line
```

Exit condition. Identify when the loop ends.

We are done when all of the lines have been added to the subdivision.

Setup. Whatever must happen before the loop starts (initialization, etc).

With no lines added yet, there is one region. It needs a color — color it black (or white).

Wrapup. Whatever must happen after the loop ends to produce the final solution.

Nothing additional is needed; the coloring is the goal.

Special cases. Make sure the algorithm works for all legal inputs — revise the previous steps to add handling for special cases as needed.

Consider things that might cause problems for the algorithm outlined so far.

A line that just clips the corner and doesn't intersect any other lines — impossible because the lines and the plane are infinite.

Two parallel lines — no problem, algorithm works as-is.

Two identical lines problem! — Either update the preconditions to require distinct lines, or adjust the algorithm to detect and handle them. There's no reason to allow them, and detecting duplicate lines may be expensive, so updating the preconditions seems like a better solution.

A line that passes exactly through an intersection of two other lines — no problem, it still just splits the regions it passes through, and it doesn't matter how many lines go through a single intersection.

Algorithm. Assemble the algorithm from the previous steps and state it.

`color(lines)` —

Input: n distinct lines

Output: a coloring of the regions (assignment of color to region)

Legal solution: every region has a color, and any two regions sharing a boundary have different colors

```
there is a single region; color it black
for each line
  add it to the subdivision, computing new regions
  flip the colors (black to white, white to black) of all of the regions on
   one side of the line
```

Termination. Show that the loop eventually terminates.

These elements come from the process input pattern.

Measure of progress. The measure of progress is the number of lines that have been added to the subdivision.

Making progress. Each iteration adds a new line, increasing the number added by one.

Reaching the end. The number lines added to the subdivision keeps increasing, so eventually all have been added.

Correctness. Show that the algorithm is correct.

Loop invariant. A loop invariant is a boolean statement about the state at the start of the loop body. It purpose is to ensure that the algorithm remains on track towards the desired solution.

This isn't an optimization problem, so try a direct statement. For process input, this is to state that the solution so far is correct.

After k lines have been added to the subdivision, every region is colored with one of two colors (black or white) and no two regions sharing a boundary have the same color.

Establish the loop invariant. Explain why the invariant is true after the setup and before the first iteration of the loop. Assuming the invariant is true before the first iteration, explain why it is still true after the first iteration and before the second.

For $k = 0$ (before the first iteration): the setup initializes a single region colored black. Every region has a color, and there's only one region so there's no boundary-sharing.

For $k = 1$ (after the first iteration / before the second iteration): the loop body adds a line, splitting one region into two. These regions now share a boundary (along the line that was added), but the color is flipped for the region on one side. Every region has a color, and the two regions sharing a boundary are different colors.

Maintain loop invariant. Assume that the loop invariant is true at the start of an iteration, and explain why the invariant is still true at the end of that iteration.

Assume the invariant holds after k iterations: the subdivision contains $k \geq 1$ lines and that every region is colored with one of two colors (black or white) with no two regions sharing a boundary having the same color.

We need to show that the invariant is maintained with the addition of line $k+1$. What happens when line $k+1$ is added?

Observe that the opposite of a valid coloring (that is, a coloring where black and white are swapped) is itself a valid coloring — only two colors are used, and no two regions sharing a boundary have the same color.

Adding a line splits the regions that the line passes through so that now all of the regions adjacent to the new line violate the "no two regions sharing a boundary have the same color" rule. Flipping the colors of every region on one side of the line maintains a valid coloring amongst those regions (they now have the opposite of the valid coloring they had before), and means that the regions split by the new line have different colors on edge side of the new line.

Thus, after $k+1$ lines, every region is colored with one of two colors (black or white) with no two regions sharing a boundary having the same color.

Final answer. Explain why, with the loop invariant being true and the exit condition being false when the loop completes, the wrapup steps lead to a correct result.

The invariant states that after k lines have been added to the subdivision, every region is colored with one of two colors (black or white) and no two regions sharing a boundary have the same color. Since this is true after every iteration, it is also true after the main loop ends — which, from the exit condition, is when $k = n$. We thus have that after n lines have been added to the subdivision, every region is colored with one of two colors (black or white) and no two regions sharing a boundary have the same color — exactly what is required.

Implementation. Identify data structures and, as necessary, specific implementations of those data structures to efficiently support the algorithm. Also fill in any algorithmic steps that haven't been specified.

The subdivision needs to be updated with each line, which would seem to require intersecting line k with each of the $k-1$ other lines already added in order to determine the regions to split ($O(k)$), then some number of regions need their colors flipped. Intersecting two lines and flipping a color are both $O(1)$.

Time and space. Assess the running time and space requirements of the algorithm given the implementation identified.

The main loop repeats n times. Intersecting line k with each of the other $k-1$ lines is $O(k)$. How many regions might need to be split, and how many might need colors flipped? The number split won't exceed the number of colors flipped, so we can focus on the latter. The best case would be if all of the lines are parallel — each additional line results in one new region. An upper bound would be if each line splits all of the existing regions — k lines would result in 2^k regions and thus up to 2^{k-1} regions to flip. This yields a worst-case running time of

$$\sum_{k=1}^{n} \left((k-1) + 2^{k-1} \right) = \Theta(2^n)$$

Exponential time is rarely considered good, so can we do better? One observation is that since n lines can lead to worst-case 2^{n-1} regions, simply going through the regions and assigning a color to each is $\Theta(2^n)$ — so if the output is to be a listing of regions and colors, we can't do better. If, however, the goal is some sort of structure that could be efficiently queried to find out the color for each region... Then we'll need a way to be able to flip the colors of a whole collection of regions without having to individually change each one, as well as to establish that it's the assignment of colors that is the bottleneck and not the computation of the regions.