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.

Boston to Seattle

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.

> You're planning to drive from Boston to Seattle on I-90 this summer, and have a GPS programmed with the locations of gas stations along the way. Assuming that your car can go 400 miles on a tank of gas, determine where you should stop for gas in order to make as few stops as possible.
>
> Input: locations of n gas stations
>
> Output: which gas stations to stop at
>
> Legal solution: gas stations are not more than 400 miles apart
>
> Optimization goal: fewest number of stops

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?

> The brute force try-all-possible-subsets of gas stations would be exponential time.

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. Identify the particular greedy flavor, if applicable, as well as the applicable iterative approaches and what they look like for this problem.

> This is a "find a subset" problem — the gas stations we stop at are a subset of all of the gas stations along the route.
>
> Process input: for each gas station, decide whether or not to stop there.
>
> Produce output: repeatedly find the next gas station to stop at.

Greedy choice. The main steps involve repeatedly making a local decision. On what basis is that choice made?

Greedy strategies. Identify plausible greedy strategies for making each choice.

> For process input, the choice is whether or not to stop at a particular gas station. With the goal of the fewest number of stops, the only plausible choice would be to stop only if necessary i.e. if the next gas station is more than 400 miles from our last stop.
>
> For produce output, the choice is which gas station to stop at next. With the goal of the fewer number of stops, the only plausible choice would be to go as far as possible before stopping i.e. pick the farthest-away gas station that is no more than 400 miles from our last stop.

Counterexamples. Narrow down the candidates by looking for counterexamples to eliminate plausible-but-incorrect greedy strategies.

Note that the produce output approach reduces to the process input approach — we could find the farthest-away gas station within 400 miles by going through the gas stations and asking for each "is there another one within 400 miles of the last stop?" The answer will finally be no for the farthest-away gas station within 400 miles.

For a counterexample, we'll need a situation where stopping sooner than necessary gets up farther (and thus avoids a stop) in the future. But we are limited in how far we can go between stops — if we stop sooner at one point, then we are not going to get farther for some subsequent stop because we'd have to travel farther to get there than if we hadn't stopped early. So it seems unlikely we'll find a counterexample...

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

As noted already, the two approaches reduce to the same idea so it probably doesn't matter much which we pick. Let's try produce output.

```
repeatedly
  choose the farthest-away gas station that is no more than 400 miles from the
   last stop
```

Exit condition. Identify when the loop ends.

This is normally "we've produced all the output items" for a produce output pattern. That's still the case here, but we don't know the number of stops we are aiming for here — instead we know we are done when we've gotten close enough to Seattle.

We are done when the last stop identified is within 400 miles of Seattle, since no more stops are needed at that point.

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

The list of stops is the empty list and the last stop is Boston.

The last point — the car has a full tank of gas — isn't directly stated in the specifications, though it is reasonable to assume. (If the car doesn't start with a full tank, then how much gas it has initially should be part of the input.)

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

The last stop has been found, so the output (the list of stops) is complete. (In terms of the trip, we drive the rest of the way to Seattle.)

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.

What if there are several gas stations at the same exit? (they are the same distance from Boston) — if there are several farthest-along gas stations no more than 400 miles ahead, any of them can be picked as the next stop.

What if the car doesn't start with a full tank of gas? — the problem statement doesn't indicate the initial amount of gas, and it's not part of the problem input. So a reasonable assumption is that we start with a full tank of gas.

What if the next gas station is > 400 miles? — no solution is possible. Either add a precondition that there isn't a gap of more than 400 miles between gas stations, or add a case in the main steps to handle the "no such next station" case.

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

`findStops(stations)` —

Input: locations of n gas stations

Output: which gas stations to stop at

Legal solution: gas stations are not more than 400 miles apart

Optimization goal: fewest number of stops

```
the list of stops is the empty list
the last stop is Boston
repeatedly
  choose the farthest-away gas station that is no more than 400 miles from
   the last stop
  if there isn't one (or it is 0 miles from the last stop),
    break and report no solution
```

Termination. Show that the loop eventually terminates.

The measure of progress should be directly related to the exit condition.

Measure of progress. The measure of progress is the distance remaining to Seattle.

Making progress. Each iteration adds a new stop closer to Seattle than the last, even in the case of multiple gas stations in the same location — since we pick the farthest-away stop within 400 miles, a gas station at the same location will not be picked if there is any other. If there isn't, the special-case handling kicks in and ends the loop because no solution is possible.

Reaching the end. Since the distance to Seattle decreases with every iteration, it will eventually be less than 400 miles. Or there's no solution and the loop exits anyway.

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.

The produce output loop invariant pattern says to try something of the form "After the first k output elements have been generated..." This is an optimization problem, so we need to cover both legality and optimality of the solution in order to establish correctness. A direct statement ("we have an optimal solution so far") is often not successful, so we'll try the staying ahead pattern. In this, we compare the algorithm's progress at a certain point to an optimal solution at the same point. Since the goal is the fewest stops —

After the first k stops, no stops are more than 400 miles apart and the algorithm's solution has no more stops than the optimal solution does.

But the optimal part here is trivial — we can't use the same thing to both define the comparison point between the algorithm's solution and an optimal solution and say how the two are doing relative to each other. Instead, let's use the measure of progress —

> After the first k stops, no stops are more than 400 miles apart and the algorithm's solution has gotten at least as close to Seattle as the optimal solution has.

Remember that the loop invariant is about showing correctness, but it should seem plausible that something about how far the algorithm's solution has gotten towards Seattle is related to having fewer stops — the closer to Seattle, the fewer additional stops are needed.

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): with 0 stops identified, no stops are more than 400 miles apart and both the algorithm and an optimal solution are in Boston — the same distance from Seattle. The algorithm is at least as close to Seattle as an optimal solution at this stage.
>
> For $k = 1$ (after the first iteration / before the second iteration): the algorithm picks the farthest stop from Boston within 400 miles (or concludes no solution). Thus the algorithm hasn't picked any stops more than 400 miles apart. We don't know which stop is first in an optimal solution, but since the algorithm has picked the farthest stop from Boston (and thus the closest reachable stop to Seattle) and both algorithm and optimal are starting from Boston, the optimal couldn't have picked a stop closer to Seattle.

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: After the first k stops, no stops are more than 400 miles apart and the algorithm's solution has gotten at least as close to Seattle as the optimal solution has.
>
> We need to show that the invariant is maintained with the addition of stop $k+1$. What happens when stop $k+1$ is picked?
>
> Since the algorithm always picks a legal stop (within 400 miles of the last) if it can, no stops are more than 400 miles apart. ("No solution" is declared if it is not possible to pick a legal stop.)

Proof by contradiction is a common tactic for the optimality part.

> Assume that stop $k+1$ is where the algorithm falls behind — the optimal's stop $k+1$ is closer to Seattle than the algorithm's.
>
> Observe that "closer to Seattle" is the same thing as "farther from Boston". Let A_i and O_i be the distance the algorithm and the optimal, respectively, have covered from Boston after i stops.
>
> The loop invariant means that $A_k \geq O_k$. The assumption that the algorithm has now fallen behind means that $A_{k+1} < O_{k+1}$. Because the optimal solution is a legal solution, $O_{k+1} - O_k \leq 400$. But it must also be the case that $O_{k+1} - A_k \leq 400$ because $A_k \geq O_k$. However, that means that A_{k+1} wasn't the farthest station within 400 miles - O_{k+1} is reachable and the algorithm would have chosen that instead. Thus it can't be the case that the algorithm fell behind at step $k+1$.

Having shown both parts of the invariant, we can conclude that the whole invariant holds.

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 loop invariant gives us that after k stops, the algorithm's solution is legal (no stops more than 400 miles apart) and that the algorithm has gotten at least as close to Seattle as an optimal solution.

The exit condition gives us that algorithm's kth stop is at most 400 miles from Seattle, so the algorithm has a complete solution with k stops.

Since the optimal solution hasn't gotten any farther than the algorithm has in k stops, it is at best the same distance from Seattle as the algorithm and at worst farther from Seattle. This means that at best the optimal solution is also complete with k stops and at worst the optimal needs additional stops to get the rest of the way to Seattle.

Since the optimal is optimal, it's not possible that the algorithm has a legal solution with fewer stops than the optimal, so it must be the case that the algorithm's solution has the same number of stops as the optimal and is thus optimal.

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 main task is to repeatedly find the farthest gas station within 400 miles of certain point. Note that we are only looking ahead — once a gas station has been passed, we never consider going back to it. Observe, then, that if the gas stations are sorted by distance, we only ever need to scan forward from the last stop to find the next one. This makes for a total of $O(n)$ work to carry out all of the find-next-stop tasks for the loop.

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

Sorting the list of gas stations is $O(n \log n)$. Then the main loop is $O(n)$ as we only have to scan forward to find the next stop, and adding to the end of a list is O(1). Thus the running time of the algorithm is $O(n \log n)$ due to having to sort the gas stations, and $O(n)$ if the list is provided in sorted order.