Instructions: For each problem description, show, in outline form, how the problem can be fully factored and give an algorithm to solve the problem based on your factoring. There may be multiple solutions, and you may not need all of the variables provided.

0. Example problem description for "Minimum and Maximum": `first_number` and `second_number` each hold an input; `minimum` should, after the algorithm runs, hold their minimum, whereas `maximum` should hold their maximum.

Example solution:

- Finding the minimum and maximum
  Splits into alternative subproblems:
  - Finding the minimum and maximum when `first_number` is less than `second_number`
    Splits into parallel subproblems:
    * Finding the minimum when `first_number` is less than `second_number`
    * Finding the maximum when `first_number` is less than `second_number`
  - Finding the minimum and maximum when `first_number` is greater than or equal to `second_number`
    Splits into parallel subproblems:
    * Finding the minimum when `first_number` is greater than or equal to than `second_number`
    * Finding the maximum when `first_number` is greater than or equal to than `second_number`

```
if first_number < second_number then
    let minimum ← first_number
    let maximum ← second_number
else
    let minimum ← second_number
    let maximum ← first_number
end
```

1. Problem description for "Clamping": `value`, `minimum`, and `maximum` each hold an input. If `value` is between `minimum` and `maximum`, inclusive, it should remain unchanged. Otherwise, it should be changed to the nearest value that is in that range. You may assume that `maximum` is at least `minimum`.

> *Clamping happens in many places, but especially in user interfaces. For instance, clamping mouse coordinates is what keeps you from dragging your cursor off the edge of the screen, clamping color values is what prevents visual effects from trying to display colors brighter than white or darker than black, etc.*

2. Problem description for "Median of Three": `first_number`, `second_number`, and `third_number` each hold an input, and `result` should, after the algorithm runs, hold their median.

> *Any time a computer sorts a list, there's a good chance it's using a technique called quicksort. Quicksort, like its name suggests, is usually fast, but, depending on the way it chooses elements to focus on, can sometimes end up making decisions that actually slow the sorting process down. A popular way to avoid most bad choices is to always pick the median of the first, middle, and last elements from the part of the list quicksort is working on.*

3. Problem description for "Reverse Five": `first_number` through `fifth_number` each hold an input, which the algorithm should put in reverse order. `copy` is available for temporarily holding copies of other values.

> *When a computer sends a packet over the internet, part of the information attached to the packet is a 5-tuple, five numbers, two of which describe the sender, one of which describes the protocol, and two more of which describe the recipient. Preparing a reply means reversing the sender and recipient roles, which is very similar to reversing a five-element list.*

4. Problem description for "Sort Two": `first_number` and `second_number` each hold an input. The algorithm should put them in ascending order. `copy` is available for temporarily holding copies of other values.

> *When you select text on a computer, you can either select from left to right or from right to left (assuming the text is written horizontally). But most of the operations on selections are written assuming that the first selection endpoint comes before the second. Internally, the computer must sort these two values to ensure that that really is the case.*

5. Problem description for "Move Maximum to Middle": `first_number`, `second_number`, and `third_number` each hold an input. The algorithm should rearrange them so that the maximum input is in `second_number`. `copy` and `maximum` are available for temporarily holding copies of other values.

> *Efficient route-finding, like that used to compute driving directions or in the navigation part of game AI, involves taking a set of candidate partial routes and repeatedly extending the most promising one until a path to the destination is found. Keeping track of which route is most promising is the job of a priority queue. The most popular kind of priority queue is called a heap, which works by solving simultaneous instances of this kind of problem.*

6. Problem description for "Sort Three": `first_number`, `second_number`, and `third_number` each hold an input. The algorithm should put them in ascending order. `copy` and `maximum` are available for temporarily holding copies of other values.

> *The generalization from sorting two elements to sorting three is the first step towards building general-purpose sorting algorithms like the aforementioned quicksort.*