Overview: The goal of this algorithm is to construct a polynomial with specific roots, where the roots are represented by a set of distinct x-values. The polynomial we aim to find will have these x-values as its roots, and all y-values (coefficients) will be non-zero.

1. Input:
    - `x_values`: An array containing distinct x-values x0, x1, ..., xn-1, which represent the roots of the polynomial.
2. Initialization:
    - Initialize `result` as a polynomial with a constant term of 1. This represents the initial state of our polynomial.
3. Polynomial Construction:
    - For each root `x_values[i]` in the input array:
        - Multiply the current polynomial `result` by the factor `(x - x_values[i])`. This operation effectively adds the root `x_values[i]` to the polynomial.
        - The multiplication is performed using a helper function `multiply_polynomials`, which computes the product of two polynomials.
4. Helper Function `multiply_polynomials`:
    - Input: Two polynomials `poly1` and `poly2`.
    - Initialization:
        - Determine the degrees `m` and `n` of `poly1` and `poly2`, respectively.
        - Initialize `result` as a polynomial with the degree `(m + n - 1)` to accommodate the product.
    - Product Calculation:
        - For each term in `poly1`, and for each term in `poly2`, calculate the product of their coefficients and add it to the corresponding position in the `result` polynomial.
        - This multiplication process effectively computes the product of the two input polynomials.
5. Output:
    - The `find_polynomial` function returns the final polynomial `result`, which has the specified x-values as its roots and non-zero coefficients for all terms.

Overall, this algorithm constructs a polynomial with the given x-values as its roots by iteratively multiplying the current polynomial by factors corresponding to each root. The result is a polynomial that satisfies the conditions specified in the problem statement, with the provided x-values as its roots and non-zero coefficients for all terms.

Pseudocode: These methods will both go in the main class. The input for find_polynomial is an array of numbers that represent the zeroes of the polynomial while the input for multiply_polynomials is 2 arrays of numbers that represent the coefficients of the polynomials being multiplied.

- function find_polynomial(x_values):

```
n = length(x_values)
result = [1]  # Start with a constant term of 1
for i from 0 to n-1:
    result = multiply_polynomials(result, [-x_values[i], 1])
return result
```

- function multiply_polynomials(poly1, poly2):

```
m = length(poly1)
n = length(poly2)
result = [0] * (m + n - 1)
for i from 0 to m-1:
    for j from 0 to n-1:
        result[i + j] += poly1[i] * poly2[j]
return result
```

Justification: The algorithm for constructing a polynomial with given roots is based on fundamental principles of polynomial interpolation and polynomial multiplication. To justify its correctness, we can break down the algorithm into its key components and explain why each step is valid:

1. **Initialization with a Constant Term of 1:**
    - The algorithm begins by initializing the `result` polynomial with a constant term of 1. This step is correct because it ensures that the initial state of the polynomial is not zero, and it won't affect the subsequent root addition steps. Since 1 is a non-zero constant, it ensures that the polynomial will have non-zero coefficients for all terms.
2. **Polynomial Construction (Root Addition):**
    - For each root `x_values[i]` in the input array:
        - The algorithm multiplies the current polynomial `result` by the factor `(x - x_values[i])`. This step is based on the fundamental property of polynomials, where if `r` is a root of a polynomial `p(x)`, then `(x - r)` is a factor of `p(x)`. Therefore, adding a root to the polynomial by multiplying it by `(x - x_values[i])` is a valid operation.

        - The algorithm correctly accumulates the roots into the polynomial `result` in each iteration, ensuring that the polynomial has the specified x-values as its roots.
3. **Helper Function `multiply_polynomials`:**
    - This function is responsible for computing the product of two polynomials. It operates by multiplying the coefficients of corresponding terms from `poly1` and `poly2` and correctly placing them in the result polynomial.
    - The correctness of this function relies on the distributive property of polynomial multiplication, which ensures that the coefficients of corresponding terms in the product are computed correctly.
    - The function's logic ensures that the product of the input polynomials is correctly calculated, and this correctness is propagated back to the main algorithm.
4. **Overall Correctness:**
    - The main algorithm uses a combination of polynomial addition and multiplication operations to construct a polynomial that has the specified x-values as its roots and non-zero coefficients for all terms.
    - At each step of the algorithm, the operations are consistent with the properties of polynomials, ensuring that the resulting polynomial has the desired roots and non-zero coefficients.
    - The algorithm's correctness is further guaranteed by the correctness of the helper function `multiply_polynomials`, which is designed to calculate polynomial products correctly.

In conclusion, the algorithm correctly constructs a polynomial with the specified roots by performing valid polynomial operations and adhering to the fundamental principles of polynomial interpolation and multiplication. The correctness of each step is based on well-established mathematical properties of polynomials.

Runtime Analysis: The runtime analysis for the algorithm to construct a polynomial with given roots involves evaluating the time complexity of the main steps in the algorithm: polynomial initialization, polynomial construction, and the helper function `multiply_polynomials`.

Let's denote the number of roots as `n`.

1. **Initialization (Constant Time):**
    - Initializing `result` as a polynomial with a constant term of 1 takes constant time since it does not depend on the number of roots.
2. **Polynomial Construction (Linear Time):**
    - For each of the `n` roots in `x_values`, we perform the following steps:

        - Multiplying the current polynomial `result` by a factor `(x - x_values[i])`.
        - This involves updating the coefficients of `result`.
    - Since we perform this operation for each of the `n` roots, the polynomial construction step runs in O(n) time.
3. **Helper Function `multiply_polynomials` (Quadratic Time):**
    - The `multiply_polynomials` function computes the product of two polynomials. In the worst case, if `poly1` has `m` terms and `poly2` has `n` terms, the product will have `(m + n - 1)` terms.
    - To compute each term of the product, we iterate over all pairs of terms from `poly1` and `poly2`. This involves nested loops, resulting in a quadratic time complexity in terms of the number of terms in the input polynomials.
    - In the context of our algorithm, `multiply_polynomials` is called once for each root addition. Therefore, the total time spent in this function over all iterations of the main loop is O(n^2).

Overall, the runtime analysis of the algorithm can be summarized as follows:

- The initialization step takes constant time, O(1).
- The polynomial construction step takes linear time, O(n), as it iterates over each root.
- The `multiply_polynomials` function, which is called once for each root addition, takes quadratic time, O(n^2).

Hence, the overall time complexity of the algorithm is dominated by the `multiply_polynomials` function, resulting in a worst-case time complexity of O(n^2). This means that the algorithm's runtime grows quadratically with the number of roots.

Examples:

Ex. 1: Simple Case Suppose we have one root, `x_values = [2]`. We want to construct a polynomial with this root.

1. Initialization:
    - Initialize `result` with a constant term of 1: `result = [1]`.
2. Polynomial Construction (Root Addition):
    - For `x_values[0] = 2`, multiply `result` by `(x - 2)`:
        - `result = [1] * (x - 2) = [1, -2]`.

The resulting polynomial is `p(x) = 1x - 2`, which has `x = 2` as its root.

Example 2: Multiple Roots Suppose we have multiple roots, `x_values = [1, 2, 3]`. We want to construct a polynomial with these roots.

1. Initialization:
    - Initialize `result` with a constant term of 1: `result = [1]`.
2. Polynomial Construction (Root Addition):
    - For `x_values[0] = 1`, multiply `result` by `(x - 1)`:
        - `result = [1] * (x - 1) = [1, -1]`.
    - For `x_values[1] = 2`, multiply `result` by `(x - 2)`:
        - `result = [1, -1] * (x - 2) = [1, -3, 2]`.
    - For `x_values[2] = 3`, multiply `result` by `(x - 3)`:
        - `result = [1, -3, 2] * (x - 3) = [1, -6, 11, -6]`.

The resulting polynomial is `p(x) = 1x^3 - 6x^2 + 11x - 6`, which has roots at `x = 1`, `x = 2`, and `x = 3`.