Closest Prime

In this coding entry, we will be creating a method that will identify the closest prime number relative to the desired number.

To do this we need to create:

An array of Prime Number Generators using a simple (a * b) loop.

A Variable Representing our chosen number.

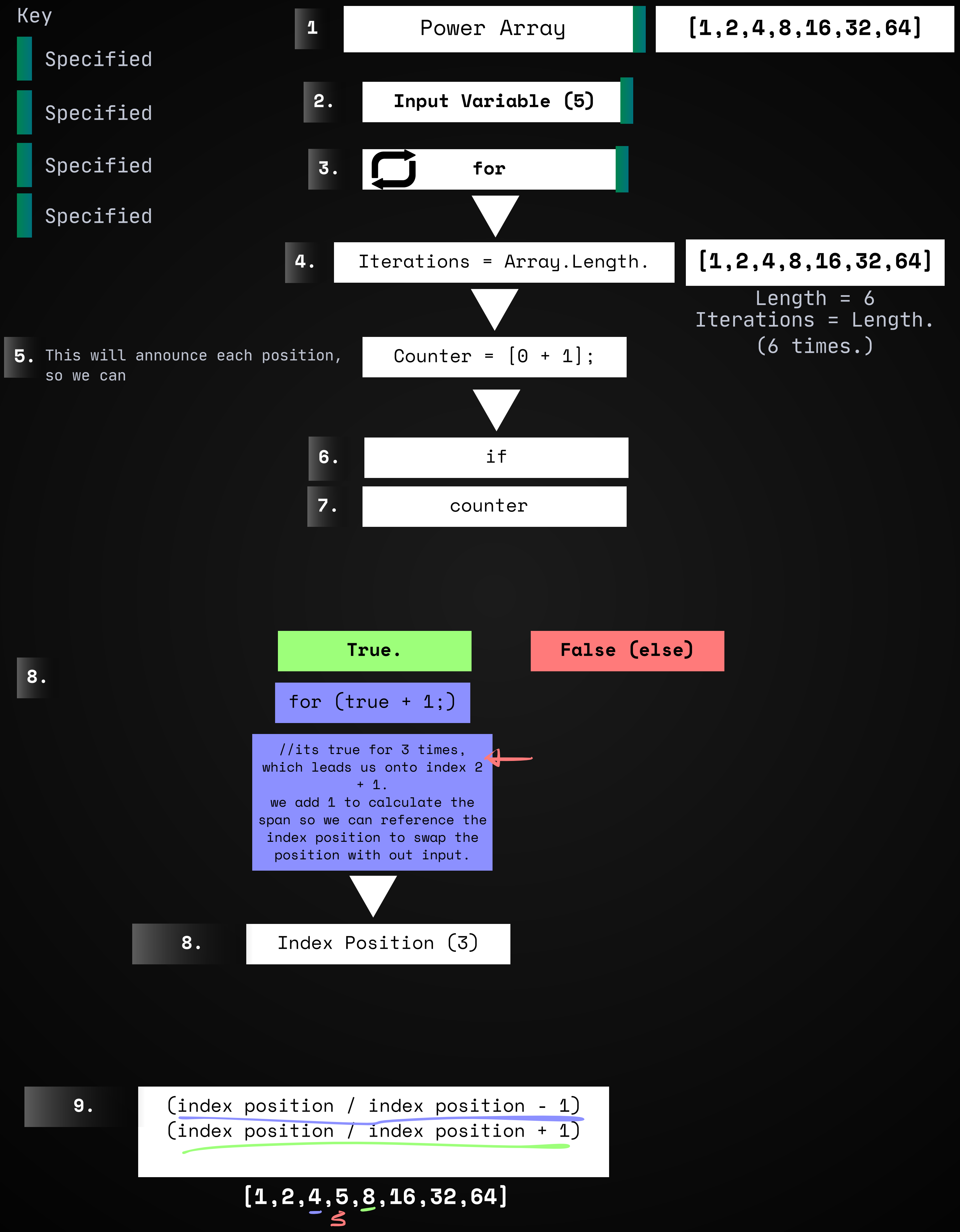
An insert and sorting algorithm that will sort the array from biggest to smallest with the input variable.

And finally, a comparison operator will compare the left and right of our input variable.

Outline

JL

Flow Chart



Key

🟩 Specified

🟩 Specified

🟩 Specified

🟩 Specified

1 Power Array [1,2,4,8,16,32,64]

2. Input Variable (5)

3. for

4. Iterations = Array.Length. [1,2,4,8,16,32,64]

Length = 6
Iterations = Length.
(6 times.)

5. This will announce each position, so we can

Counter = [0 + 1];

6. if

7. counter

8.

True.

for (true + 1;)

//its true for 3 times, which leads us onto index 2 + 1.
we add 1 to calculate the span so we can reference the index position to swap the position with out input.

False (else)

8. Index Position (3)

9. (index position / index position - 1)
(index position / index position + 1)

[1,2,4,5,8,16,32,64]

JL Flow Chart Correspondance.

```
    const { Console } = require("console");

    // Power Number Array
1   let Powers = [1, 2, 4, 8, 16, 32, 64];

    let inputNumber = 9; // assuming inputNumber is a single number

2   // Concatenate the input number to the array
    let InsertArray = Powers.concat(inputNumber);

    // Find the correct position to insert the input number
    let insertedIndex = -1;
    for (let i = 0; i < InsertArray.length - 1; i++) {
        if (InsertArray[i] > inputNumber) {
            // Insert input number at the correct position
            InsertArray.splice(i, 0, inputNumber);
            // Remove the duplicate input number at the end of the array
            InsertArray.pop();
            insertedIndex = i;
            break;
        }
    }

    // If inputNumber is larger than all elements in the array, no insertion was done in the loop
3   // In that case, the input number is already correctly placed at the end after concatenation
    if (insertedIndex === -1) {
        insertedIndex = InsertArray.length - 1;
4   }

5   let leftIndex = insertedIndex - 1;
    let rightIndex = insertedIndex + 1;
6
    console.log('Sorted array:', InsertArray);
7   console.log('Input number:', inputNumber);
    if (leftIndex >= 0) {
        console.log('Element to the left:', InsertArray[leftIndex], 'at index', leftIndex);
8   } else {
        console.log('No element to the left of the input number.');
    }
9   if (rightIndex < InsertArray.length) {
        console.log('Element to the right:', InsertArray[rightIndex], 'at index', rightIndex);
    } else {
        console.log('No element to the right of the input number.');
    }

    // Compare the differences to find out which element is closer to the input number
    if (leftIndex >= 0 && rightIndex < InsertArray.length) {
        let differenceToLeft = inputNumber - InsertArray[leftIndex];
        let differenceToRight = InsertArray[rightIndex] - inputNumber;

        if (differenceToLeft < differenceToRight) {
            console.log(`${InsertArray[leftIndex]} at index ${leftIndex} is closer to ${inputNumber}`);
        } else if (differenceToRight < differenceToLeft) {
            console.log(`${InsertArray[rightIndex]} at index ${rightIndex} is closer to ${inputNumber}`);
        } else {
            console.log(`${InsertArray[leftIndex]} at index ${leftIndex} and ${InsertArray[rightIndex]} at index
    ${rightIndex} are equally close to ${inputNumber}`);
        }
    } else if (leftIndex >= 0) {
        console.log(`${InsertArray[leftIndex]} at index ${leftIndex} is the only element close to ${inputNumber}`);
    } else if (rightIndex < InsertArray.length) {
        console.log(`${InsertArray[rightIndex]} at index ${rightIndex} is the only element close to ${inputNumber}`);
    }
```

Explanation.

splice method is a method that takes 3 parameters that serve its own function as follows: **(start, delete/count, items)**, in this instance, it first takes "i" which refers to the first instance of InsertArray > inputNumber and returns it as an index position, it then ends with 0, which means the entire array, and finally, it replaces it with the input number at the position of 3 without deleting its correspondent "8".

in short, it selects InsertArray

Applies splice Method.

Starts at i which is equal to 3, because of the for loop conditions.

ends with 0, meaning the entire array.

and replaces the start with Input Number (5)

```
InsertArray.splice(i, 0, inputNumber);
```

=

[1, 2, 4, 8, 16, 32, 64, 5]

=

5

3 of index

[1, 2, 4, 8, 16, 32, 64, 5]

0 1 2 3

output

[1, 2, 4, 5, 8, 16, 32, 64]

pre pop()[1, 2, 4, 8, 16, 32, 64, 5]

post pop() [1, 2, 4, 8, 16, 32, 64]

Explanation.

The pop() Method is u

```
InsertArray.pop();
```

=

[1, 2, 4, 8, 16, 32, 64, 5]

Explanation.

splice method is a method that takes 3 parameters that serve its own function as follows: **(start, delete/count, items)**, in this instance, it first takes "i" which refers to the first instance of InsertArray > inputNumber and returns it as an index position, it then ends with 0, which means the entire array, and finally, it replaces it with the input number at the position of 3 without deleting its correspondent "8".

in short, it selects InsertArray

Applies splice Method.

Starts at i which is equal to 3, because of the for loop conditions.

ends with 0, meaning the entire array.

and replaces the start with Input Number (5)

```
insertedIndex = i;
```

This Article is rated

Excellent

based on 20 reviews

★★★★★

Easy to follow

Straight Forward and Concise