LEARN BASIC ALGORITHMIC THINKING BY BUILDING A NUMBER SORTER

Introduction:

In computer science, there are fundamental sorting algorithms that all developers should learn. In this number sorter project, you'll learn how to implement and visualize different sorting algorithms like bubble sort, selection sort, and insertion sort – all with JavaScript.

This project will help you understand the fundamental concepts behind these algorithms, and how you can apply them to sort numerical data in web applications.

Step 1:

In this project, you will be building a number sorter. The HTML and CSS have been provided for you. Feel free to explore them.

When you are ready, declare a `sortButton` variable and assign it the value of `.getElementById()` with the argument `"sort"`.

Step 2:

To prepare your project's logic, use `const` and arrow syntax to declare a `sortInputArray` function. It should take a single `event` parameter.

Step 3:

You will be using this as an event listener for the `sortButton`. Because buttons associated with a `form` element submit by default, you need to prevent that behavior. Call `event.preventDefault()` in your function to do this.

Step 4:

To test your code as you write it, mount an event listener to your `sortButton` element. It should listen for the `"click"` event, and take `sortInputArray` as the callback.

Step 5:

Back in your `sortInputArray` function, you need to get the values from your `select` elements. Since they all have the class `values-dropdown`, you can query them all at once.

Use `document.getElementsByClassName()` to get all the elements with this class by passing in the argument `"values-dropdown"`. Assign that to an `inputValues` variable with `const`.

Step 6:

Remember that `.getElementsByClassName()` method returns an HTMLCollection, which is an array-like object of all the elements that have a matching class name. You can use the spread operator to convert it into an array.

Convert the `document.getElementsByClassName()` call to an array with the spread operator and assign it to a variable called `inputValues`.

Step 7:

You need to get the values from your `select` elements. These values will currently be strings and you will convert them into numbers.

Use the `map` function to iterate over the array. Pass a callback function to `map` that takes a `dropdown` parameter and returns `dropdown.value`.

Step 8:

You should use `console.log()` to print out the result of `inputValues`. Write the code for this inside the `sortInputArray` function.

To see the logged `inputValues` array, click on the sort button and open up the console. You should see an array of strings like this:

Example Code:

```
[ "8", "2", "4", "1", "3" ]
```

Before going further, make sure you observe the data type of the printed result in the console.

In the next step, you will convert those strings into numbers.

Step 9:

Update your `.map()` callback to call the `Number()` function. Pass `dropdown.value` to that function call.

Open the Console tab to see that your `inputValues` is an array of numbers now.

Step 10:

Now that you have confirmed the data type of the `inputValues` elements, remove your `console.log()` call.

Step 11:

You need a function to update the display with the sorted numbers. Start by using arrow syntax to declare an `updateUI` function that takes a single `array` parameter.

Because you will be writing algorithms that won't immediately have a return value, set a fallback value for `array` to be an empty array. Here is an example of a function that has a fallback value:

Example Code:

```
const myFunction = (string = "") => {

}
```

Step 12:

To perform an action on each element in the array, use the method that is meant for iterating over arrays.

Use the `forEach()` method, and pass it an empty callback which takes `num` and `i` as the parameters.

Step 13:

Create a variable named `outputValueNode` and set its value to the result of calling the `document.getElementById()` method. Use template literal syntax to pass in the ``` `output-value-${i}` ``` string to `.getElementById()`.

Step 14:

Set the `innerText` property of `outputValueNode` to `num`.

Step 15:

In your `sortInputArray()` function, call your `updateUI()` function and pass `inputValues` as the argument.

You should now be able to click the `Sort` button and see the inputted array in the `Output` section.

Step 16:

Now you need to actually sort the array. The first sorting algorithm you will implement is the bubble sort, which starts at the beginning of the array and 'bubbles up' unsorted values towards the end, iterating through the array until it is completely sorted.

Begin by declaring a `bubbleSort` variable and assigning it an arrow function that takes an `array` parameter.

Step 17:

You'll need to iterate through the array. For simplicity, use a `for` loop to do so.

Step 18:

Because you need to compare elements, you'll need to use a nested `for` loop. This loop should iterate through every element in the array *except* the last one. Use `j` as your inner loop's iterator variable.

Step 19:

For debugging purposes, add a `console.log()` call in your inner loop. Pass it the arguments `array`, `array[j]`, and `array[j+1]`.

Step 20:

In your `sortInputArray()` function, declare a `sortedValues` variable. Assign it the value of calling `bubbleSort` with your `inputValues` array.

Then, update your `updateUI` call to pass `sortedValues` as the argument.

Step 21:

To achieve the "bubble up" result, you need to check if the current element is larger than the next element. You can do this by accessing the `array` at `j` and `j+1`.

Create an `if` condition that checks if the current element is larger than the next element.

Step 22:

When your `if` condition is true, you need to swap the two elements, "bubbling" the larger element up toward the end of the array.

To do this, declare a `temp` variable and assign it the value of `array[j]`. Then assign `array[j]` the value of `array[j + 1]`. Finally, assign `array[j + 1]` the value of `temp`.

Step 23:

Finally, after your outer loop has finished executing, return the sorted array.

Step 24:

Click your `Sort` button to see your bubble sort algorithm in action! If you open the console, you can watch the steps the algorithm takes.

Now that you have confirmed it works, remove your `console.log()` call.

Step 25:

Time to implement another sorting algorithm. This time, you'll be implementing a selection sort. Selection sort works by finding the smallest value in the array, then swapping it with the first value in the array. Then, it finds the next smallest value in the array, and swaps it with the second value in the array. It continues iterating through the array until it is completely sorted.

Start by declaring a `selectionSort` variable and assigning it an arrow function that takes an `array` parameter.

Step 26:

Update your `sortedValues` variable to be the result of calling `selectionSort` instead of `bubbleSort`.

Step 27:

Like a bubble sort, a selection sort needs to iterate through the array. Declare a `for` loop to do so.

Step 28:

A selection sort relies on tracking the index of the smallest value in the array. Declare a variable `minIndex` and set it to `i` - this ensures that if your current value is the smallest, it will be swapped with itself and not be moved. You will need to be able to reassign the value of `minIndex` as you iterate through the array.

Then, write another `for` loop, using `j` as the iterator. This loop needs to start at the index after `i` and iterate through the rest of the array.

Step 29:

Inside your nested `for` loop, add a `console.log()` call to check the values of `array`, `array[j]`, and `array[minIndex]` at each iteration. You can click the `Sort` button to see how your algorithm is traversing the array.

Then write an `if` statement that checks if the value at `array[j]` is smaller than the value at `array[minIndex]`. If it is, set `minIndex` to `j`.

Step 30:

After your nested `for` loop, you've found the smallest value. You need to swap it with your current value.

Like you did in your bubble sort, use a `temp` variable to extract the value at `array[i]`, then swap the values at `array[i]` and `array[minIndex]`.

Step 31:

Finally, after your outer loop has finished, you need to return the array. Once you've done so, you should be able to see the `Output` change when you click the `Sort` button again.

Step 32:

With your selection sort now functional, remove your `console.log()` statement.

Step 33:

The last sorting algorithm you will implement is the insertion sort. This algorithm works by building up a sorted array at the beginning of the list. It begins the sorted array with the first element. Then it inspects the next element and swaps it backward into the sorted array until it is in a sorted position, and so on.

Start by declaring an `insertionSort` variable and assigning it an arrow function which takes an `array` parameter.

Step 34:

As before, update your `sortedValues` variable to be the result of `insertionSort` instead of `selectionSort`.

Step 35:

An insertion sort algorithm starts the sort at the beginning of the list, meaning the first element is already sorted. With this in mind, create a `for` loop that starts at the second element in the array - it should still iterate through the rest of the array.

Step 36:

Declare a `currValue` variable and assign it the value at `array[i]`. Then, declare a `j` variable and assign it `i - 1`. Your `j` variable should be re-assignable.

Step 37:

For this algorithm, you'll want to use a `while` loop. This loop needs two conditions:

- First, it should not run beyond the beginning of the array (accessed with `j`).
- Second, the loop should not run after it finds a value smaller than the current value.

To prevent an infinite loop, decrement `j` inside your loop.

Step 38:

On each iteration of your `while` loop, it is finding an element that is larger than your current value. You need to move that element to the right to make room for your current value.

Do so by assigning the value at the `j` index to the next index.

Step 39:

After your while loop, you need to insert your current value. Remember that your loop ends when `j` is either out of the array bounds, or when the value at `j` is less than your current value.

Use the assignment operator to insert your current value into the correct index

Step 40:

After your `for` loop has finished, you need to return the array. You should then be able to see the `Output` change when you click the `Sort` button again.

Step 41:

To sort the elements of an array, you can use the built-in method called `.sort()`. Therefore, you can update the `sortedValues` variable by assigning it the result of calling `.sort()` on the `inputValues` array.

Step 42:

The `Sort` button may appear to work correctly when clicked, but this is only because all the values in the array are single digits, and the sorting may not work as expected with more complex values.

Change the `value` and text of the `option` element that is `selected` from `1` to `10`, and click the `Sort` button again.

Step 43:

Notice how the number `10` is placed at the beginning of the array. This is because the default behavior of `.sort()` is to convert the numbers values to strings, and sort them alphabetically. And `"10"` comes before `"2"` alphabetically.

To fix this, you can pass a callback function to the `.sort()` method. The callback function has two parameters - for yours, use `a` and `b`. The parameters of `a` and `b` represent the number values in the array that will be sorted.

Leave the function empty for now.

Step 44:

The callback to `.sort()` should return a number. That number determines how to sort the elements `a` and `b`:

- If the number is negative, sort `a` before `b`.
- If the number is positive, sort `b` before `a`.
- If the number is zero, do not change the order of `a` and `b`.

Keeping in mind that you want the numbers to be sorted in ascending order (smallest to largest), return a single subtraction calculation using `a` and `b` that will correctly sort the numbers with the above logic.

Step 45:

If you press the `Sort` button again, you should see that `10` is now in the correct position of the `Output`.

To finish this project, change your `option` back to a `value` and text of `1`.