Array.prototype.some()

Summary: in this tutorial, you will learn how to use the JavaScript Array `sort()` method to sort arrays of numbers, strings, and objects.

Introduction to JavaScript Array sort() method

The `sort()` method allows you to sort elements of an array in place. It changes the positions of the elements in the original array and returns the sorted array.

By default, the `sort()` method sorts the array elements in ascending order, placing the smallest value first and the largest value last.

When you sort an array of numbers, the `sort()` method converts these numbers to strings and compares the strings to determine the order. For example:

```
let numbers = [0, 2, 5, 3, 10];
numbers.sort();
console.log(numbers);
```

The output is:

```
[ 0, 10, 2, 3, 5 ]
```

In this example, the `sort()` method places `10` before `2` because the string `"10"` comes before the string `"2"` .

To change this behavior, you need to pass a *comparator function* to the `sort()` method. The `sort( )` method will use the comparator function to determine the order of elements.

The following illustrates the syntax of the `sort()` method with the comparator function:

```
array.sort(comparator)
```

In this syntax, the `comparator` function accepts two arguments and returns a value that determines the sort order.

The following illustrates the syntax of the comparator function:

```
function compare(a,b) {
  // ...
}
```

The `compare()` function accepts two arguments `a` and `b` .

The `sort()` method sorts elements based on the return value of the `compare()` function with the following rules:

- If the comparator function returns a negative number, the `sort()` method place `a` before `b` .
- If the comparator function returns a positive number, the `sort()` method place `b` before `a` .
- If the comparator function returns zero, the `sort()` method considers `a` and `b` are equal, and leaves their positions unchanged.

So to sort an array of numbers in ascending order, you can use the comparator function as follows:

```
const numbers = [0, 2, 5, 3, 10];
numbers.sort((a, b) => a - b);

console.log(numbers);
```

Output:

```
[ 0, 2, 3, 5, 10 ]
```

To sort the numbers array in descending order, you can change the return value of the comparator function as follows:

```
const numbers = [0, 2, 5, 3, 10];
numbers.sort((a, b) => b - a);

console.log(numbers);
```

Output:

```
[ 10, 5, 3, 2, 0 ]
```

Sorting an array of strings

Suppose you have an array of strings named `animals` as follows:

```
let animals = ['cat', 'dog', 'elephant', 'bee', 'ant'];
```

To sort the elements of the `animals` array in ascending order alphabetically, you use the `sort()` method without passing the compare function as shown in the following example:

```
let animals = ['cat', 'dog', 'elephant', 'bee', 'ant'];
animals.sort();

console.log(animals);
```

Output:

```
[ 'ant', 'bee', 'cat', 'dog', 'elephant' ]
```

To sort the `animals` array in descending order, you need to change the logic of the comparator function and pass it to the `sort()` method as the following example.

```
let animals = ['cat', 'dog', 'elephant', 'bee', 'ant'];

animals.sort((a, b) => {
  if (a > b) return -1;
  if (a < b) return 1;
  return 0;
});

console.log(animals);
```

Output:

```
[ 'elephant', 'dog', 'cat', 'bee', 'ant' ]
```

Suppose you have an array that contains elements in both uppercase and lowercase as follows:

```
let mixedCaseAnimals = ['Cat', 'dog', 'Elephant', 'bee', 'ant'];
```

To sort this array alphabetically, you need to use a custom comparator function to convert all elements to the same case e.g., uppercase for comparison, and pass that function to the `sort()` method.

```
let mixedCaseAnimals = ['Cat', 'dog', 'Elephant', 'bee', 'ant'];

mixedCaseAnimals.sort(function (a, b) {
  let x = a.toUpperCase(),
    y = b.toUpperCase();
  return x == y ? 0 : x > y ? 1 : -1;
});
```

Output:

```
[ 'ant', 'bee', 'Cat', 'dog', 'Elephant' ]
```

Sorting an array of strings with non-ASCII characters

The `sort()` method is working fine with the strings with ASCII characters. However, for the strings with non-ASCII characters e.g., é, è, etc., the `sort()` method will not work correctly. For example:

```
let animaux = ['zèbre', 'abeille', 'écureuil', 'chat'];
animaux.sort();

console.log(animaux);
```

Output:

```
[ 'abeille', 'chat', 'zèbre', 'écureuil' ]
```

The code returns an unexpected output because `écureuil` should come before `zèbre` .

To fix this, you use the `localeCompare()` method of the `String` object to compare strings in a specific locale, like this:

```
let animaux = ['zèbre', 'abeille', 'écureuil', 'chat'];
animaux.sort((a, b) => a.localeCompare(b));

console.log(animaux);
```

Output:

```
[ 'abeille', 'chat', 'écureuil', 'zèbre' ]
```

The elements of the `animaux` array now is in the correct order.

Sorting an array of objects by a property

The following is an array of `employee` objects, where each object contains three properties: `name` , `salary` and `hireDate` .

```
let employees = [
    {name: 'John', salary: 90000, hireDate: "July 1, 2010"},
```

```
    {name: 'David', salary: 75000, hireDate: "August 15, 2009"},
    {name: 'Ana', salary: 80000, hireDate: "December 12, 2011"}
];
```

Sorting objects by a numerical property

The following example shows how to sort the employees by `salary` in ascending order.

```
let employees = [
  { name: 'John', salary: 90000, hireDate: 'July 1, 2010' },
  { name: 'David', salary: 75000, hireDate: 'August 15, 2009' },
  { name: 'Ana', salary: 80000, hireDate: 'December 12, 2011' },
];

// sort by salary
employees.sort((x, y) => x.salary - y.salary);

console.table(employees);
```

Output:

(index)	name	salary	hireDate
0	'David'	75000	'August 15, 2009'
1	'Ana'	80000	'December 12, 2011'
2	'John'	90000	'July 1, 2010'

This example works the same as sorting an array of numbers in ascending order. The difference is that it compares the `salary` property of two objects.

Sorting objects by a string property

To sort the `employees` array by `name` property case-insensitively, you pass the compare function that compares two strings case-insensitively as follows:

```
let employees = [
  { name: 'John', salary: 90000, hireDate: 'July 1, 2010' },
  { name: 'David', salary: 75000, hireDate: 'August 15, 2009' },
  { name: 'Ana', salary: 80000, hireDate: 'December 12, 2011' },
```

```
];

employees.sort((x, y) => {
  let a = x.name.toUpperCase(),
    b = y.name.toUpperCase();
  return a == b ? 0 : a > b ? 1 : -1;
});

console.table(employees);
```

(index)	name	salary	hireDate
0	'Ana'	80000	'December 12, 2011'
1	'David'	75000	'August 15, 2009'
2	'John'	90000	'July 1, 2010'

Sorting objects by the date property

Suppose, you wish to sort employees based on each employee's hire date.

The hire date data is stored in the `hireDate` property of the employee object. However, it is just a string that represents a valid date, not the `Date` object.

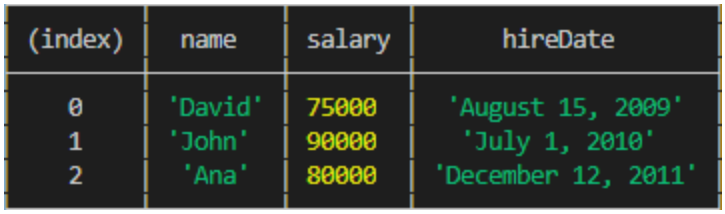
Therefore, to sort employees by hire date, you first have to create a valid `Date` object from the date string, and then compare two dates, which is the same as comparing two numbers.

Here is the solution:

```
let employees = [
  { name: 'John', salary: 90000, hireDate: 'July 1, 2010' },
  { name: 'David', salary: 75000, hireDate: 'August 15, 2009' },
  { name: 'Ana', salary: 80000, hireDate: 'December 12, 2011' },
];

employees.sort((x, y) => {
  let a = new Date(x.hireDate),
    b = new Date(y.hireDate);
  return a - b;
});
```

```
console.table(employees);
```



(index)	name	salary	hireDate
0	'David'	75000	'August 15, 2009'
1	'John'	90000	'July 1, 2010'
2	'Ana'	80000	'December 12, 2011'

Optimizing JavaScript Array sort() method

The `sort()` method calls the comparator function multiple times for each element in the array.

```
let rivers = ['Nile', 'Amazon', 'Congo', 'Mississippi', 'Rio-Grande'];

rivers.sort((a, b) => {
  console.log(a, b);
  return a.length - b.length;
});
```

Output:

```
Amazon Nile
Congo Amazon
Congo Amazon
Congo Nile
Mississippi Congo
Mississippi Amazon
Rio-Grande Amazon
Rio-Grande Mississippi
```

How it works:

1. First, declare an array `rivers` that consists of the famous river names.
2. Second, sort the `rivers` array by the length of its element using the `sort()` method. We output the elements of the `rivers` array to the web console whenever the `sort()` method invokes the comparator function.

The output shows that the `sort()` method evaluates each element multiple times e.g., Amazon 4 times, Congo 2 times.

If the number of array elements is increasing, it will impact the performance.

We cannot reduce the number of times that the comparator function is executed. However, we can reduce the work that the comparison has to do. This technique is called the Schwartzian Transform.

To implement this, you follow these steps:

1. First, extract the actual values into a temporary array using the map() method.
2. Second, sort the temporary array with the elements already evaluated (or transformed).
3. Third, walk the temporary array to get the sorted array.

Here is the solution:

```
let rivers = ['Nile', 'Amazon', 'Congo', 'Mississippi', 'Rio-Grande'];

// temporary array holds objects with position and length of element
var lengths = rivers.map(function (e, i) {
  return { index: i, value: e.length };
});

// sorting the lengths array containing the lengths of river names
lengths.sort(function (a, b) {
  return +(a.value > b.value) || +(a.value === b.value) - 1;
});

// copy element back to the array
var sortedRivers = lengths.map(function (e) {
  return rivers[e.index];
});

console.log(sortedRivers);
```

Output:

```
[ 'Nile', 'Congo', 'Amazon', 'Rio-Grande', 'Mississippi' ]
```

Summary

- The `sort()` method sorts array elements in place and returns the sorted array.
- The `sort()` method converts numbers into strings before sorting.