Test 1

Given a set of data with the following structure:

```
{
"metadata": {
        "patientA": "cancer",
        "patientB": "cancer",
        "patientC": "control",
        "patientD": "cancer",
        "patientE": "control"
},
"mutations": {
        "patientA": {
               // key is the location, value is the amount of mutations there
               "KRAS/10394954": 5,
               "KRAS/3958838": 20,
               "TP53/94959003": 10,
               "TP53/12931920": 2,
               "NRAS/399322": 75,
        },
        "patientB": {
               "KRAS/10394954": 31,
               "KRAS/3958838": 122,
               "TP53/94959003": 45,
               "TP53/12931920": 11,
               "NRAS/399322": 52,
        },
        "patientC": {
               "KRAS/10394954": 10,
               "KRAS/3958838": 74,
               "TP53/94959003": 4,
               "TP53/12931920": 0,
               "NRAS/399322": 39,
                "NRAS/19929330": 20,
        },
        "patientD": {
                  "KRAS/10394954": 7,
                  "KRAS/3958838": 3,
                  "TP53/94959003": 11,
                  "TP53/12931920": 92,
                  "NRAS/399322": 0,
                  "NRAS/19929330": 3,
          },
          "patientE": {
                  "KRAS/10394954": 7,
                  "KRAS/3958838": 3,
                  "TP53/94959003": 11,
                  "TP53/12931920": 92,
                  "NRAS/399322": 0,
                  "NRAS/19929330": 3,
        }
}
```

Based on the above data, we can calculate the specificity and sensitivity with the following approach:

- Sensitivity: after sorting the patients by sum of mutations left to right, count n patients from the left, whereas n equals the number of cancers in the set. Then count how many cancer patients are in this sorted subset. Divide this by n.
- Specificity : same thing, but 1-n with controls and counting from the right

Write a function to sort the data by the total mutation count of each patient and calculate the specificity and sensitivity for the result.

Test 2

Given the following scenario: we have a table with multiple thousand rows and some thousands of columns. Since rendering in that size into the browser's DOM will cause the UI to be too slow to be useful, we're only rendering a chunk - based on what can actually be seen on the page at the moment.

The data is coming in full from a worker thread. A small fragment of that data is then actually rendered. Every interaction with the page sends the entire data set back to the worker thread in order to receive an updated version of the data from the worker thread.

Despite the efforts to render only what can be seen, the performance is still lacking. A further performance analysis revealed that the lack of performance is caused by marshaling the huge amount of data to transfer it between the worker thread and the main thread.

What is your approach to refactoring the code to allow for 60fps independent of the amount of data?