

# Functionality Investigation Sheet

<b>Feature:</b> Global search bar	<b>Feature #2</b>
<b>Problem:</b> To be able to filter recipes using the “global search bar” using	

<b>Option 1:</b> Linear search In this option, the linear search algorithm is used to search for the keyword in the array of keywords of each recipe. <b>Linear Search:</b> Search an array starting from the leftmost element and one by one comparing the searched value with each element of the array. If a match is found return a Boolean true, If no match is found with any element, return a Boolean false.	
<b>Benefits</b> <ul style="list-style-type: none"> <li>No specific array pre-treatment</li> <li>Relatively efficient on small arrays</li> </ul>	<b>Disadvantages</b> <ul style="list-style-type: none"> <li>The bigger the array, the slower the search</li> <li>Access data sequentially</li> </ul>
Equality comparison Time complexity: $-O(n)$	

<b>Option 2:</b> Binary search In this option, the binary search algorithm is used to search for the keyword in the sorted array of keywords of each recipe. <b>Binary Search:</b> Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.	
<b>Benefits</b> <ul style="list-style-type: none"> <li>Efficient on big array</li> </ul>	<b>Disadvantages</b> <ul style="list-style-type: none"> <li>Array needs to be sorted</li> <li>Access data randomly</li> </ul>
Ordering comparison Time complexity: $-O(\log n)$	

<b>Option 3:</b> Interpolation search In this option, the interpolation search algorithm is used to search for the keyword in the sorted array of keywords of each recipe. <b>Interpolation Search:</b> it is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. Binary Search always goes to the middle element to check. On the other hand, interpolation search may go to different locations according to the value of the key being searched. For example, if the value of the key is closer to the last element, interpolation search is likely to start search toward the end side.	
<b>Benefits</b> <ul style="list-style-type: none"> <li>Very efficient on uniform big array</li> </ul>	<b>Disadvantages</b> <ul style="list-style-type: none"> <li>Array needs to be sorted</li> <li>Better on uniformly distributed array</li> <li>Not made for string search</li> </ul>
Ordering comparison Time complexity: $-\log(\log n)$ up to $O(n)$ (depends on the uniformity of the distribution)	

**Solution retention:**

As said in the disadvantages of the 3<sup>rd</sup> option, the interpolation search is not made for a string search, to be able to still use it, it is necessary to add a function to convert the strings in numerical value (with many different possible implementation, more or less complex, and more or less working) therefor adding quite some processing time.

Regarding the comparison between option 1 & 2:

**Theory wise:**

On the case of the option 1, if the array is not sorted, then the processing time cannot be guessed, if it is sorted, then the 'lower' the searched value, the faster the search will be. In any case, at best, the searched word is found instantly, in worst case, the searched is found on the very last iteration.

On the case of the option 2: each iteration of the algorithm, divides by 2 the amount of word in the array. Therefore, at best, the searched word is found on the first iteration, and at worst on the  $\log_2(N)$ ,  $N = 50$  in our case, so on the ~6<sup>th</sup> iteration.

**In practice:**

After execution time testing using the Chrome Dev-tool, if we look at the table Figure1 (and Figure5 for details on the method). We can see that the binary search is almost 2 times faster than the linear one. Moreover, the binary search also offers an execution time 2 times more stable than the linear one.

Additionally, studies on both JSBench.me and JSBench.ch shows a similar result but more exact (more iterations of the code). See Figures 6-7-8-9. The binary search appears to be about 80% faster than the linear in case of a match, and about 55% faster in the case of a no match  $((70 + 40) / 2)$ .

It is thus the binary search that has been chosen for the search algorithm.

Keyword	Execution time (in ms)	
	Linear Search	Binary Search
app	0.316894531	0.248779297
appl	0.201171875	0.128173828
apple	0.178955078	0.126220703
egg	0.213134766	0.116943359
lem	0.227783203	0.124023438
lemo	0.206054688	0.115966797
lemon	0.187988281	0.123046875
mas	0.217041016	0.133056641
mash	0.197021484	0.108154297
mashe	0.180175781	0.105957031
mashed	0.159912109	0.126953125
tun	0.225097656	0.136230469
tuna	0.260253906	0.136230469
whi	0.212890625	0.127929688
whit	0.400878906	0.125
white	0.115722656	0.129882813
average	0.218811035	0.132034302
diff max-min	0.28515625	0.142822266

*Figure 1 : Comparative table of Linear / Binary search algorithm's execution time using console to get the values (see Figure 5)*

## Annexes

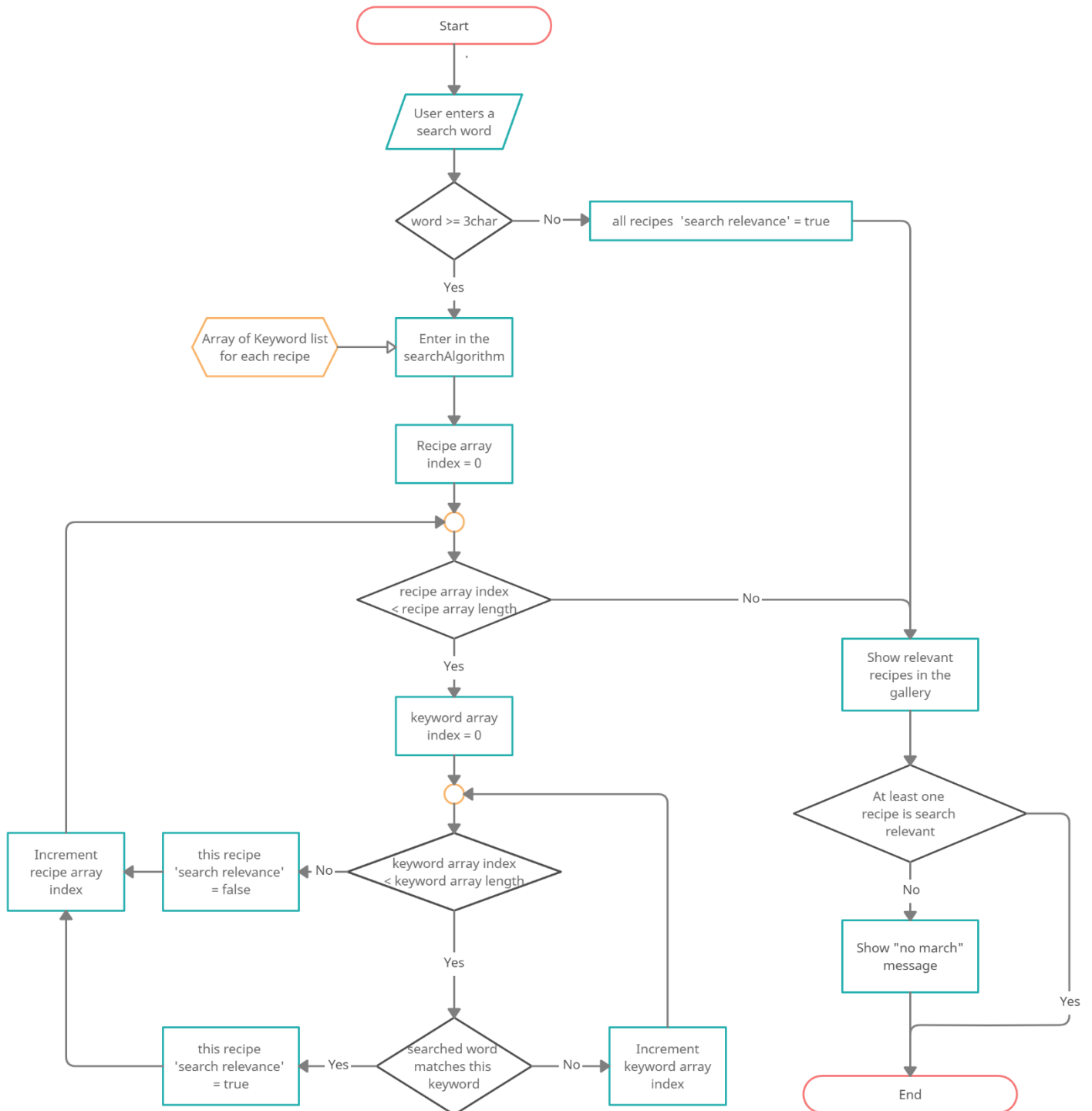


Figure 2: Workflow Diagram Linear Search Algorithm

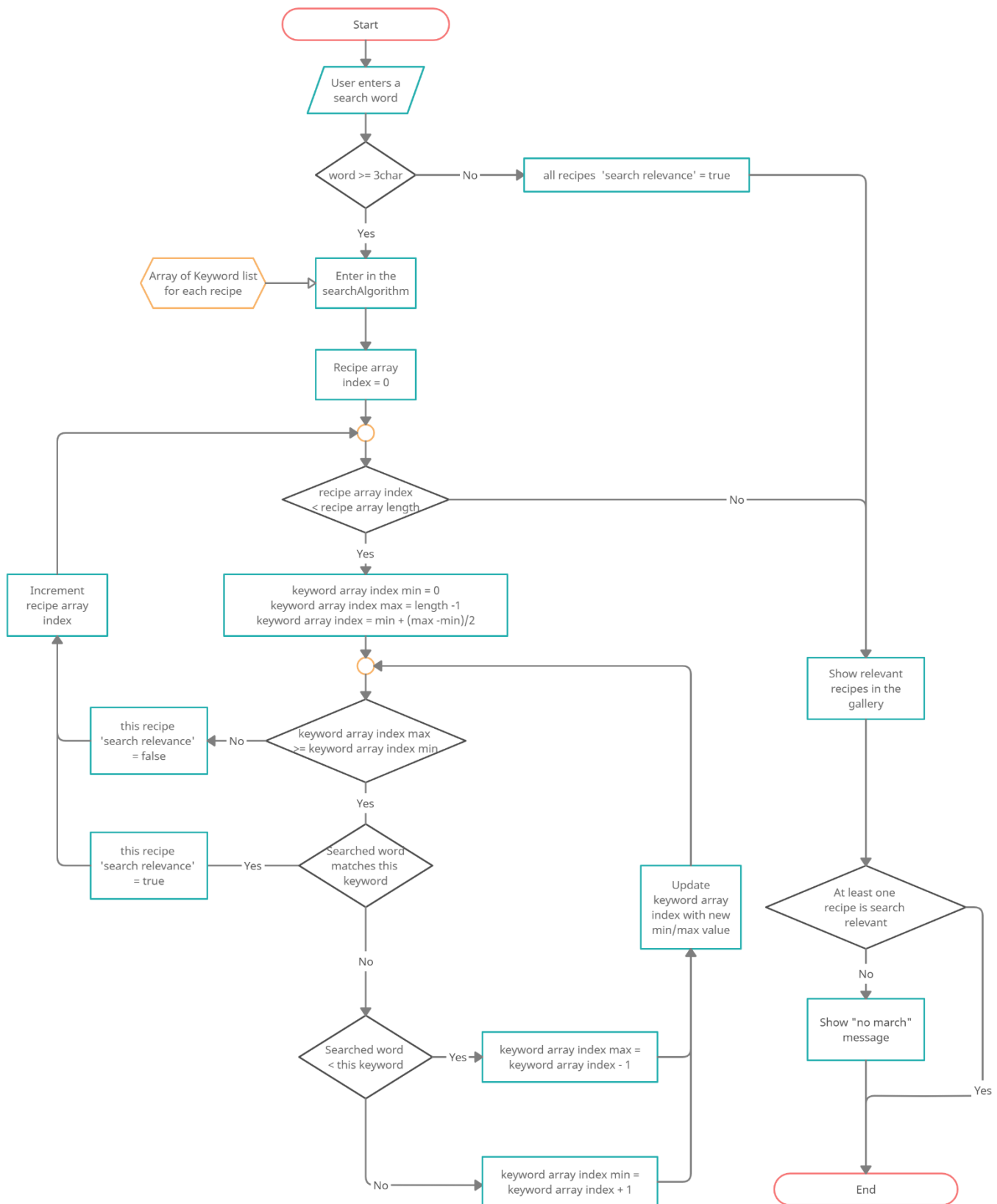


Figure 3: Workflow Diagram Binary Search Algorithm

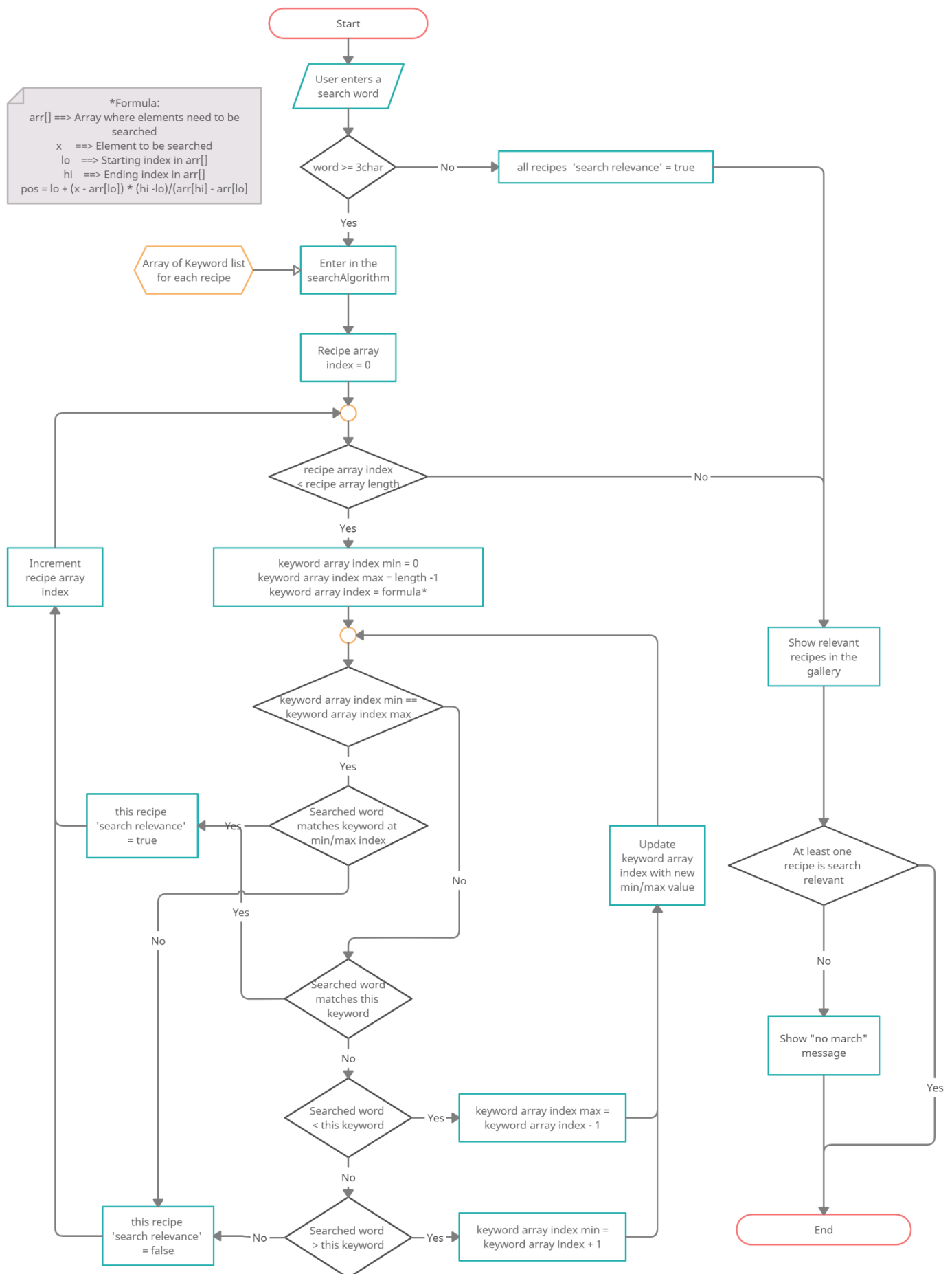


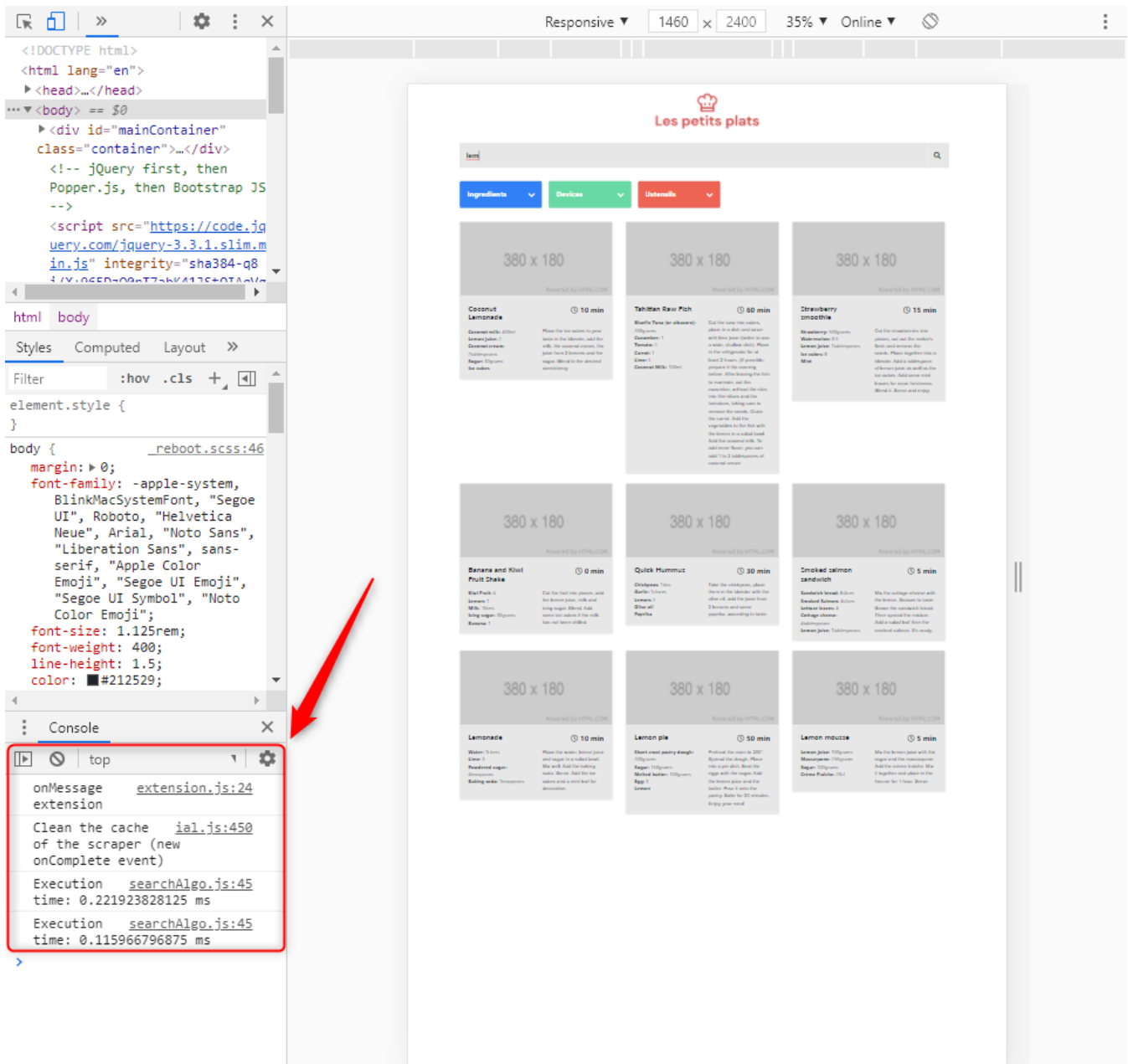
Figure 4: Workflow Diagram Interpolation Search Algorithm

```
function searchEngine(keywordStruct, searchWord){
  console.time('Execution time');

  var noMatch = true;
  for(let i=0; i<keywordStruct.length;i++){
    if(searchAlgorithm(keywordStruct[i].keywordList, searchWord)){
      keywordStruct[i].searchRelevant = true;
      noMatch = false;
    }
    else{
      keywordStruct[i].searchRelevant = false;
    }
  }

  if(noMatch){
    noMatchMessage.show();
  }

  console.timeEnd('Execution time');
}
```



The screenshot shows a web browser window displaying the 'Les petits plats' website. The website has a header with the logo and navigation tabs for 'Ingredients', 'Devices', and 'Listenable'. Below the header, there is a grid of recipe cards, each with a title, a duration, and a description. The browser's developer console is open, showing the execution time of the search algorithm. The console output includes:

```
onMessage extension.js:24
extension
Clean the cache ial.js:450
of the scraper (new
onComplete event)
Execution searchAlgo.js:45
time: 0.221923828125 ms
Execution searchAlgo.js:45
time: 0.115966796875 ms
```

Figure 5 : Execution time 'calculation' using the console feature

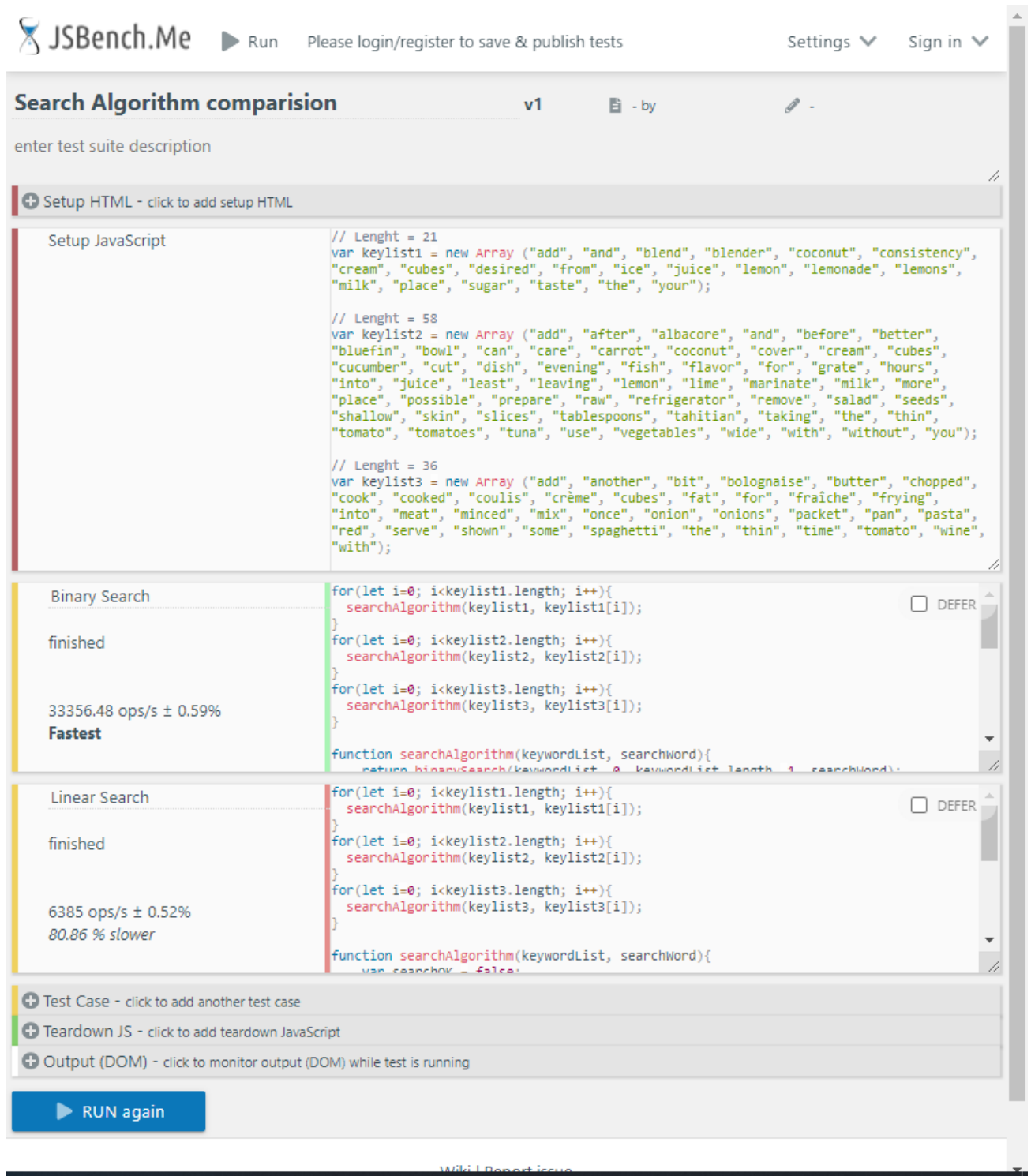


Figure 6 : Result JSBench.me – testing full march on a sample of 3 existing keylist

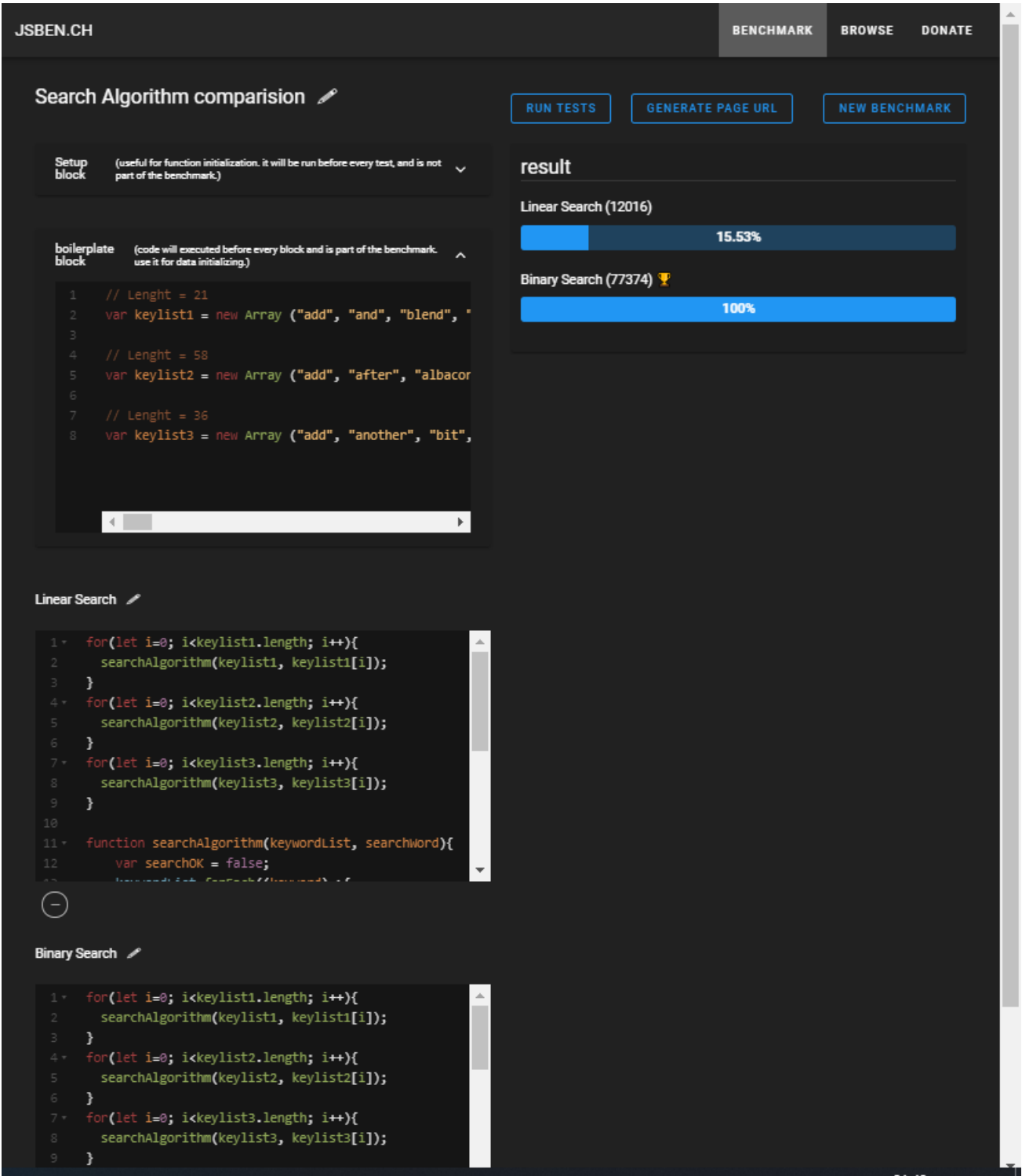


Figure 7 : Result JSBench.ch – testing full march on a sample of 3 existing keylist



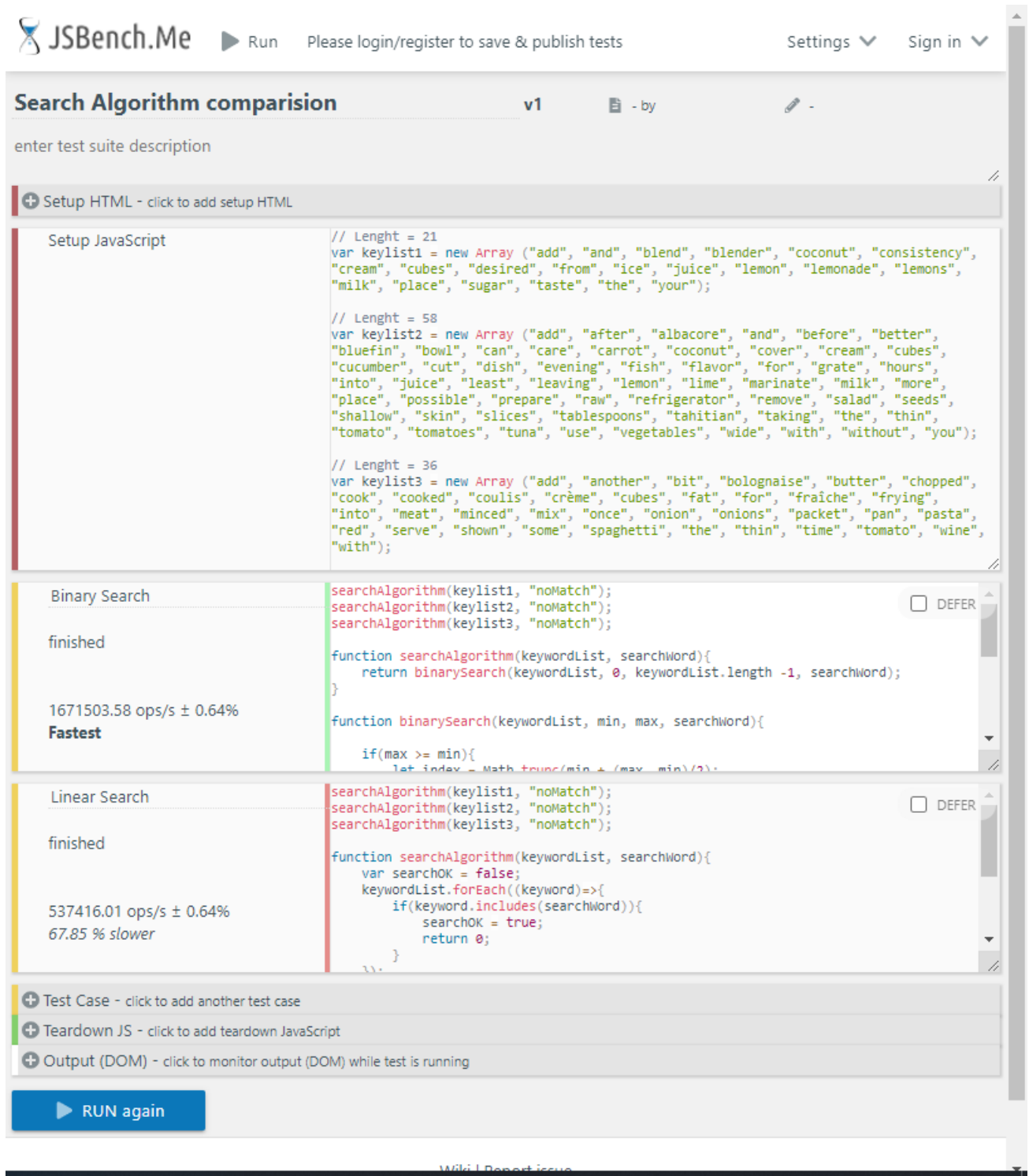


Figure 8 : Result JSBench.me – testing no march on a sample of 3 existing keylist

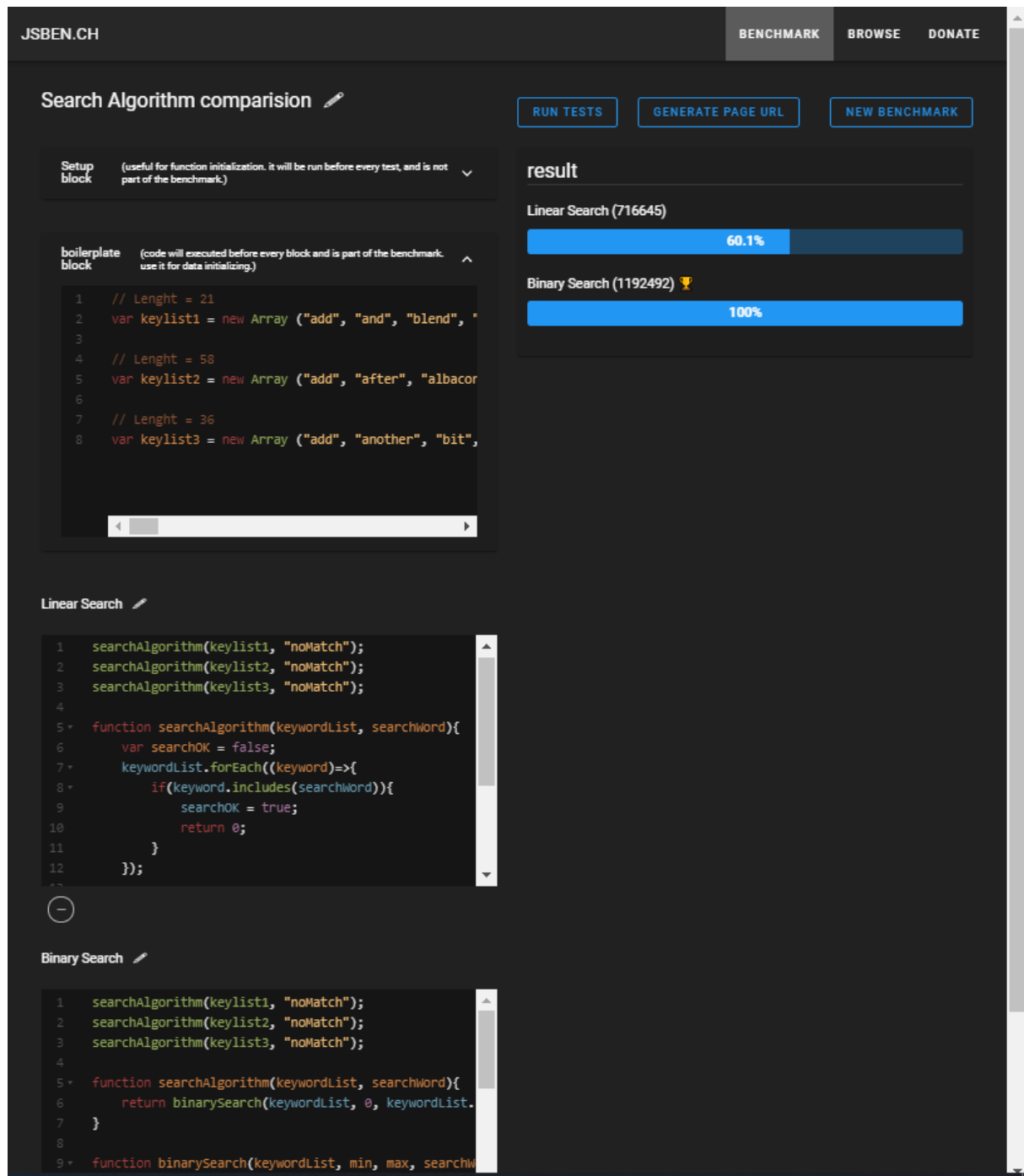


Figure 9 : Result JSBench.ch – testing no march on a sample of 3 existing keylist