Functionality Investigation Shee7

Feature: Global search bar	**Feature #2**
Problem: To be able to filter recipes using the “global search bar” using	

Option 1: Linear search In this option, the linear search algorithm is used to search for the keyword in the array of keywords of each recipe. **Linear Search:** 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.	
Benefits • No specific array pre-treatment • Relatively efficient on small arrays	**Disadvantages** • The bigger the array, the slower the search • Access data sequentially
Equality comparison Time complexity: - O(n)	

Option 2: 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. **Binary Search:** 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.	
Benefits • Efficient on big array	**Disadvantages** • Array needs to be sorted • Access data randomly
Ordering comparison Time complexity: -O(log n)	

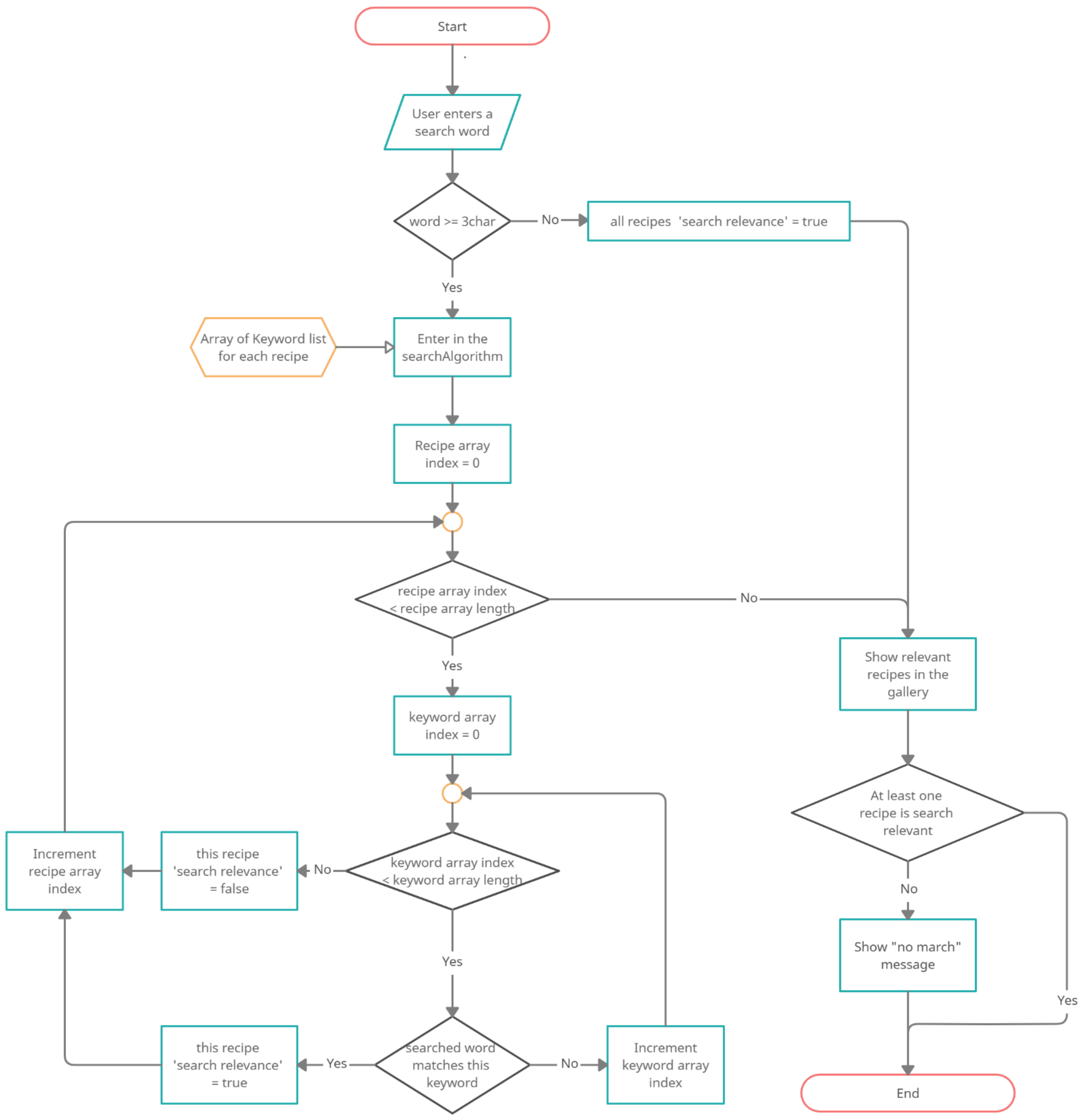
Option 3: Interpolation search In this option, the interpolation search algorithm is used to search for the keyword in the sorted array of keywords of each recipe. **Interpolation Search:** it is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. 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.	
Benefits • Very efficient on uniform big array	**Disadvantages** • Array needs to be sorted • Better on uniformly distributed array • Not made for string search
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 3rd 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 log2(N), N = 50 in our case, so on the ~6th iteration.

Annexes

Figure 2: Workflow Diagram Linear Search Algorithm


Start
User enters a search word
word >= 3char
No
all recipes 'search relevance' = true
Yes
Array of Keyword list for each recipe
Enter in the searchAlgorithm
Recipe array index = 0
recipe array index < recipe array length
No
Yes
Show relevant recipes in the gallery
keyword array index = 0
At least one recipe is search relevant
Increment recipe array index
this recipe 'search relevance' = false
No
keyword array index < keyword array length
No
Yes
Show "no march" message
Yes
this recipe 'search relevance' = true
Yes
searched word matches this keyword
No
Increment keyword array index
End

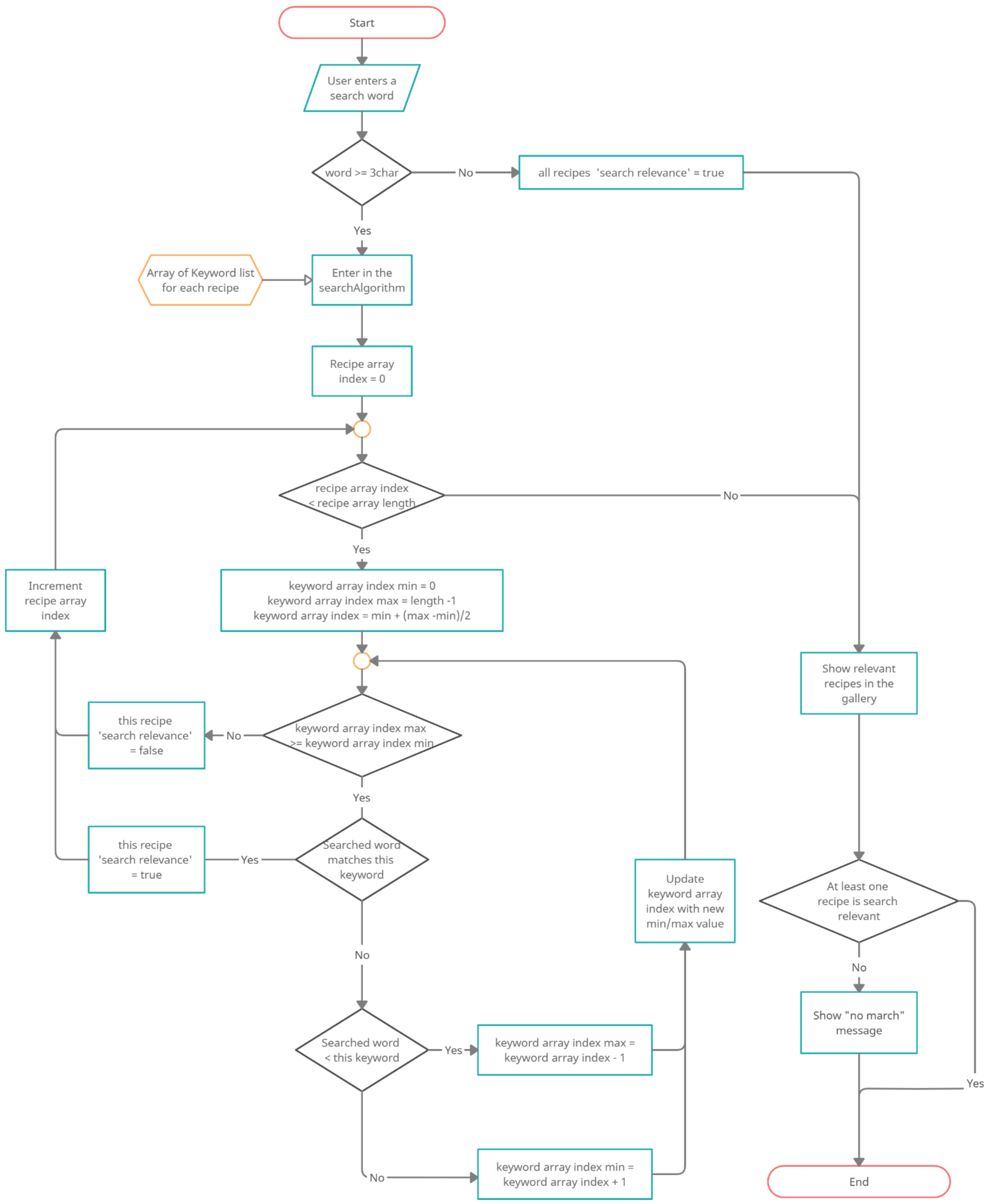

Start
User enters a search word
word >= 3char
No
all recipes 'search relevance' = true
Yes
Array of Keyword list for each recipe
Enter in the searchAlgorithm
Recipe array index = 0
recipe array index < recipe array length
No
Yes
Increment recipe array index
keyword array index min = 0
keyword array index max = length -1
keyword array index = min + (max -min)/2
Show relevant recipes in the gallery
this recipe 'search relevance' = false
No
keyword array index max >= keyword array index min
Yes
this recipe 'search relevance' = true
Yes
Searched word matches this keyword
Update keyword array index with new min/max value
At least one recipe is search relevant
No
No
Show "no march" message
Searched word < this keyword
Yes
keyword array index max = keyword array index - 1
Yes
No
keyword array index min = keyword array index + 1
End

Figure 3: Workflow Diagram Binary Search Algorithm

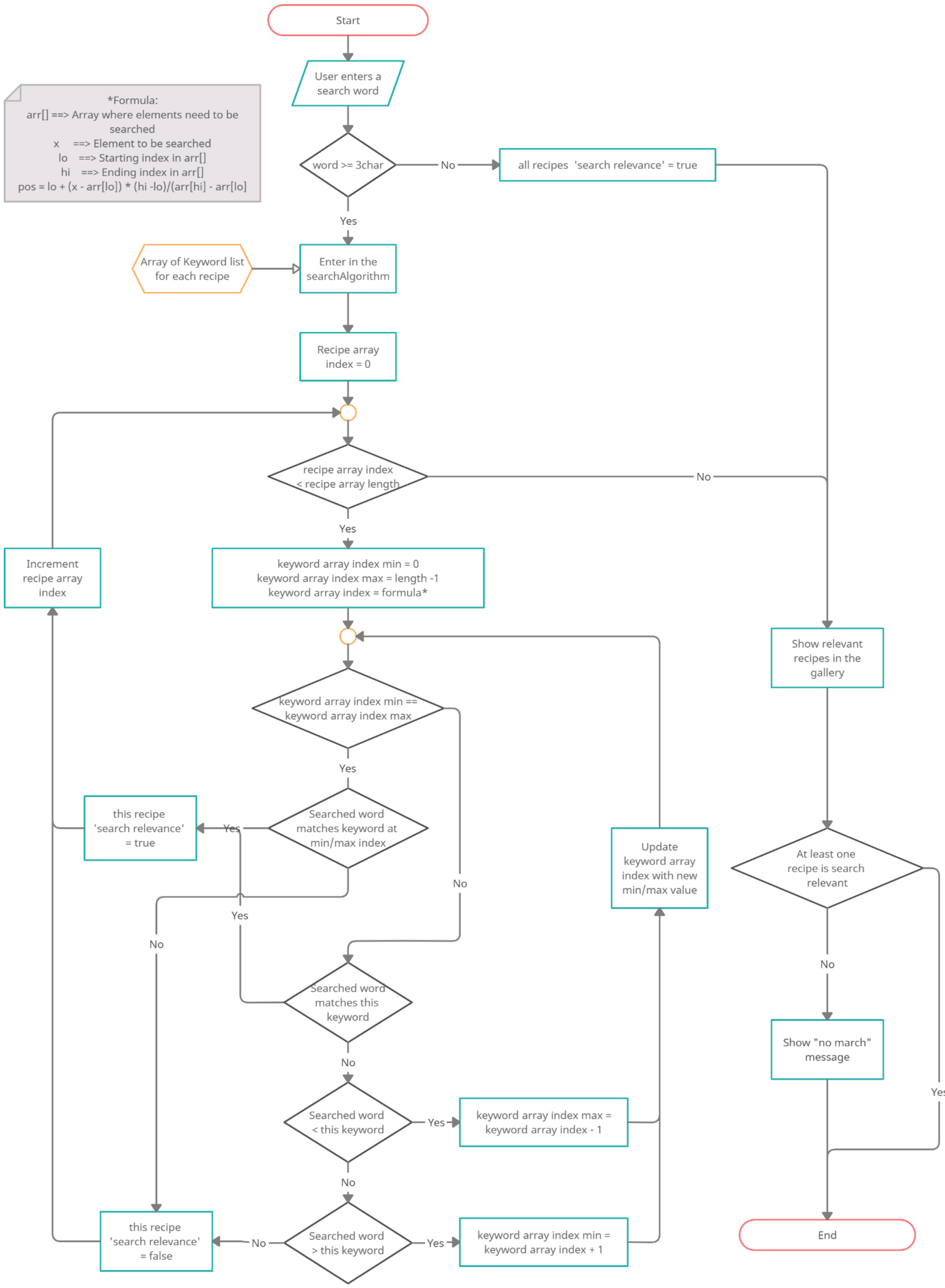
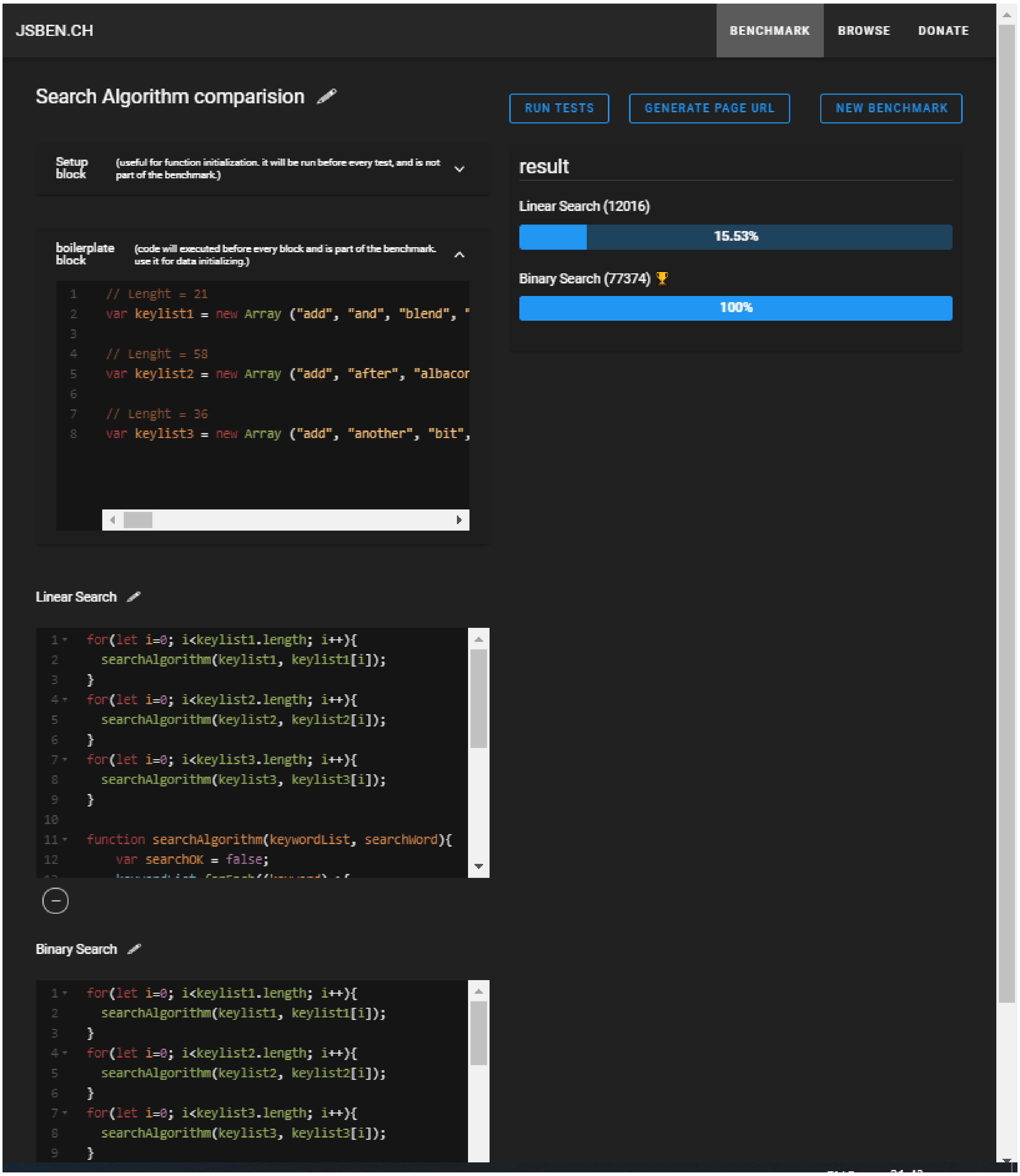

Start
User enters a search word
*Formula:
arr[] ==> Array where elements need to be searched
x ==> Element to be searched
lo ==> Starting index in arr[]
hi ==> Ending index in arr[]
pos = lo + (x - arr[lo]) * (hi -lo)/(arr[hi] - arr[lo]
word >= 3char
No
all recipes 'search relevance' = true
Yes
Array of Keyword list for each recipe
Enter in the searchAlgorithm
Recipe array index = 0
recipe array index < recipe array length
No
Yes
Increment recipe array index
keyword array index min = 0
keyword array index max = length -1
keyword array index = formula*
Show relevant recipes in the gallery
keyword array index min == keyword array index max
Yes
this recipe 'search relevance' = true
Yes
Searched word matches keyword at min/max index
No
Update keyword array index with new min/max value
At least one recipe is search relevant
Yes
No
No
Searched word matches this keyword
No
Show "no march" message
Yes
Searched word < this keyword
Yes
keyword array index max = keyword array index - 1
No
this recipe 'search relevance' = false
No
Searched word > this keyword
Yes
keyword array index min = keyword array index + 1
End

Figure 4: Workflow Diagram Interpolation Search Algorithm

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



Search Algorithm comparision

Setup block (useful for function initialization. it will be run before every test, and is not part of the benchmark.)

boilerplate block (code will executed before every block and is part of the benchmark. use it for data initializing.)

```
// Lenght = 21
var keylist1 = new Array ("add", "and", "blend", "
                                                   
// Lenght = 58
var keylist2 = new Array ("add", "after", "albacor

// Lenght = 36
var keylist3 = new Array ("add", "another", "bit",
```

Linear Search

```
for(let i=0; i<keylist1.length; i++){
  searchAlgorithm(keylist1, keylist1[i]);
}
for(let i=0; i<keylist2.length; i++){
  searchAlgorithm(keylist2, keylist2[i]);
}
for(let i=0; i<keylist3.length; i++){
  searchAlgorithm(keylist3, keylist3[i]);
}

function searchAlgorithm(keywordList, searchWord){
    var searchOK = false;
```

Binary Search

```
for(let i=0; i<keylist1.length; i++){
  searchAlgorithm(keylist1, keylist1[i]);
}
for(let i=0; i<keylist2.length; i++){
  searchAlgorithm(keylist2, keylist2[i]);
}
for(let i=0; i<keylist3.length; i++){
  searchAlgorithm(keylist3, keylist3[i]);
}
```

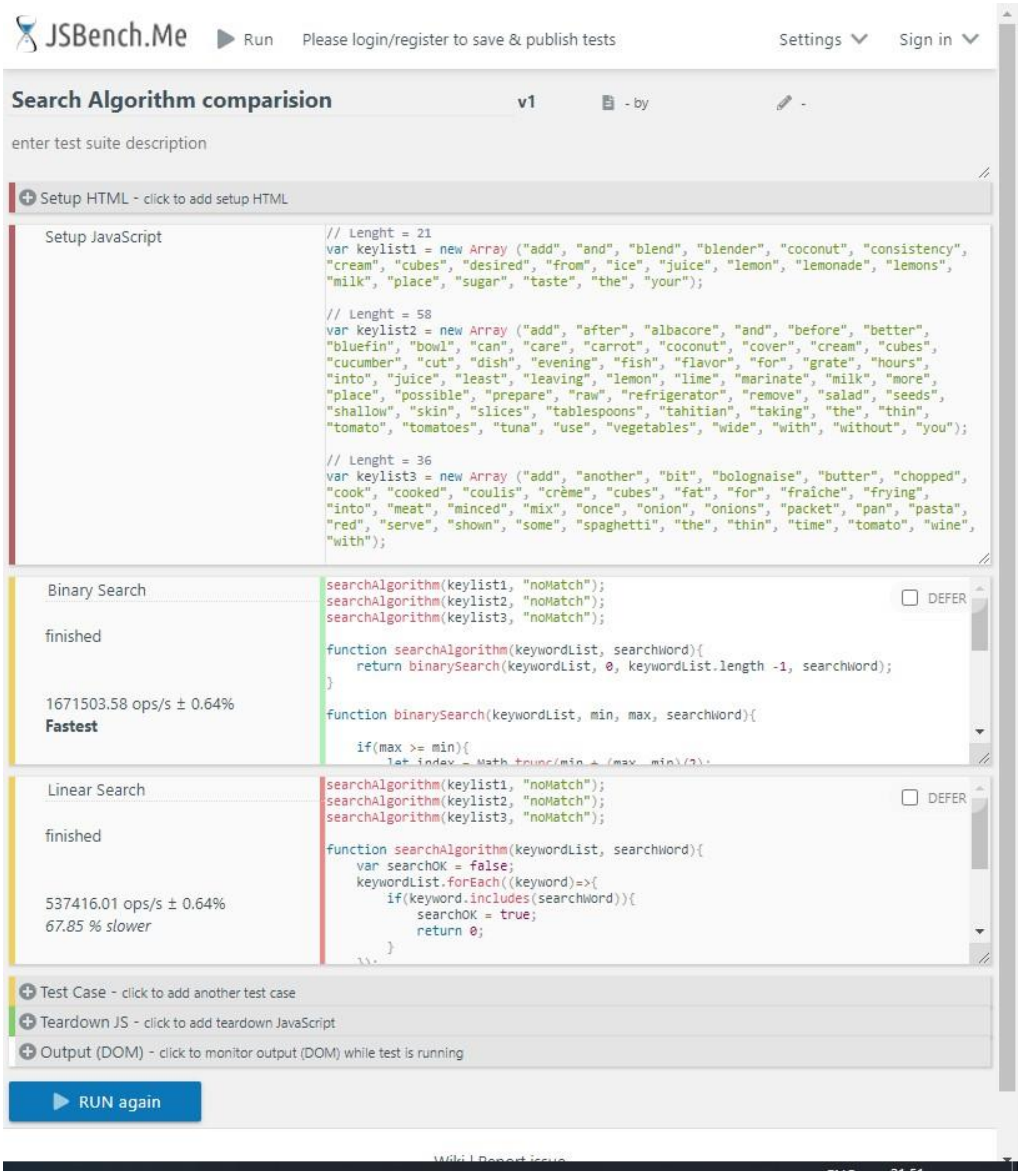
result

Linear Search (12016)

15.53%

Binary Search (77374) 🏆

100%



Search Algorithm comparision v1 - by -

enter test suite description

Setup HTML - click to add setup HTML

Setup JavaScript

```
// Lenght = 21
var keylist1 = new Array ("add", "and", "blend", "blender", "coconut", "consistency",
"cream", "cubes", "desired", "from", "ice", "juice", "lemon", "lemonade", "lemons",
"milk", "place", "sugar", "taste", "the", "your");

// Lenght = 58
var keylist2 = new Array ("add", "after", "albacore", "and", "before", "better",
"bluefin", "bowl", "can", "care", "carrot", "coconut", "cover", "cream", "cubes",
"cucumber", "cut", "dish", "evening", "fish", "flavor", "for", "grate", "hours",
"into", "juice", "least", "leaving", "lemon", "lime", "marinate", "milk", "more",
"place", "possible", "prepare", "raw", "refrigerator", "remove", "salad", "seeds",
"shallow", "skin", "slices", "tablespoons", "tahitian", "taking", "the", "thin",
"tomato", "tomatoes", "tuna", "use", "vegetables", "wide", "with", "without", "you");

// Lenght = 36
var keylist3 = new Array ("add", "another", "bit", "bolognaise", "butter", "chopped",
"cook", "cooked", "coulis", "crème", "cubes", "fat", "for", "fraîche", "frying",
"into", "meat", "minced", "mix", "once", "onion", "onions", "packet", "pan", "pasta",
"red", "serve", "shown", "some", "spaghetti", "the", "thin", "time", "tomato", "wine",
"with");
```

Binary Search

finished

1671503.58 ops/s ± 0.64%
Fastest

```
searchAlgorithm(keylist1, "noMatch");
searchAlgorithm(keylist2, "noMatch");
searchAlgorithm(keylist3, "noMatch");

function searchAlgorithm(keywordList, searchWord){
    return binarySearch(keywordList, 0, keywordList.length -1, searchWord);
}

function binarySearch(keywordList, min, max, searchWord){

    if(max >= min){
        let index = Math.trunc(min + (max - min)/2);
```

DEFER

Linear Search

finished

537416.01 ops/s ± 0.64%
67.85 % slower

```
searchAlgorithm(keylist1, "noMatch");
searchAlgorithm(keylist2, "noMatch");
searchAlgorithm(keylist3, "noMatch");

function searchAlgorithm(keywordList, searchWord){
    var searchOK = false;
    keywordList.forEach((keyword)=>{
        if(keyword.includes(searchWord)){
            searchOK = true;
            return 0;
        }
    });
```

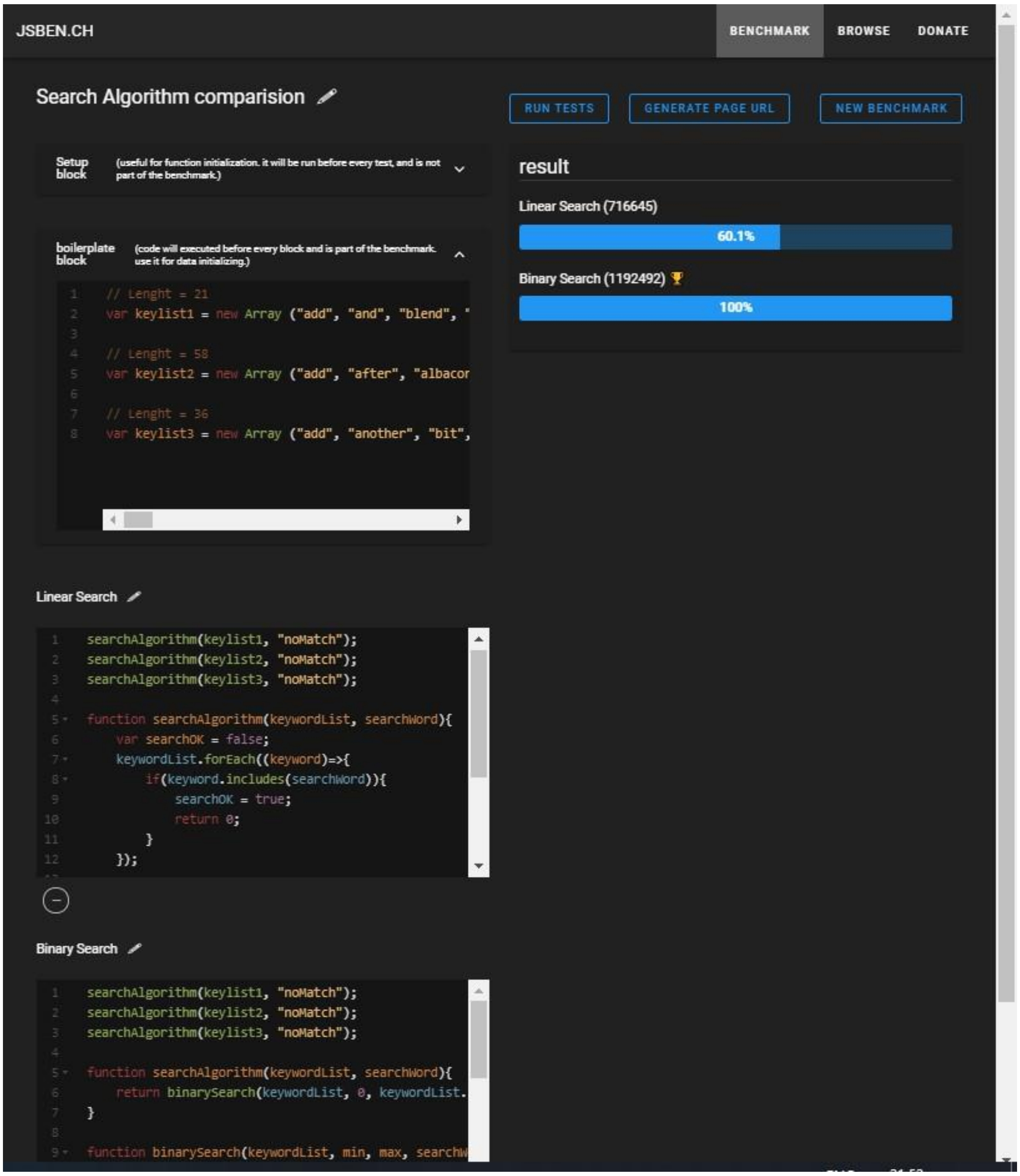
DEFER

Test Case - click to add another test case

Teardown JS - click to add teardown JavaScript

Output (DOM) - click to monitor output (DOM) while test is running

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



Search Algorithm comparision

Setup block (useful for function initialization. it will be run before every test, and is not part of the benchmark.)

boilerplate block (code will executed before every block and is part of the benchmark. use it for data initializing.)

```
// Lenght = 21
var keylist1 = new Array ("add", "and", "blend", "

// Lenght = 58
var keylist2 = new Array ("add", "after", "albacor

// Lenght = 36
var keylist3 = new Array ("add", "another", "bit",
```

result

Linear Search (716645)

60.1%

Binary Search (1192492)

100%

Linear Search

```
searchAlgorithm(keylist1, "noMatch");
searchAlgorithm(keylist2, "noMatch");
searchAlgorithm(keylist3, "noMatch");

function searchAlgorithm(keywordList, searchWord){
    var searchOK = false;
    keywordList.forEach((keyword)=>{
        if(keyword.includes(searchWord)){
            searchOK = true;
            return 0;
        }
    });
```

Binary Search

```
searchAlgorithm(keylist1, "noMatch");
searchAlgorithm(keylist2, "noMatch");
searchAlgorithm(keylist3, "noMatch");

function searchAlgorithm(keywordList, searchWord){
    return binarySearch(keywordList, 0, keywordList.
}

function binarySearch(keywordList, min, max, searchW
```