Binary Search in JavaScript

Complexity of Binary Search

A dataset of length n can be divided *log n* times until everything is completely divided. Therefore, the search complexity of binary search is O(log n).

Iterative Binary Search

A binary search can be performed in an iterative approach. Unlike calling a function within the function in a recursion, this approach uses a loop.

```
function binSearchIterative(target, array, left, right) {
  while(left < right) {
    let mid = (right + left) / 2;
    if (target < array[mid]) {
      right = mid;
    } else if (target > array[mid]) {
      left = mid;
    } else {
      return mid;
    }
  }
  return -1;
}
```

Base case in a binary search using recursion

In a recursive binary search, there are two cases for which that is no longer recursive. One case is when the middle is equal to the target. The other case is when the search value is absent in the list.

```
binary_search(sorted_list, left_pointer, right_pointer, ta
  if (left_pointer >= right_pointer)
    base case 1
  mid_val and mid_idx defined here
  if (mid_val == target)
    base case 2
  if (mid_val > target)
    recursive call with left pointer
  if (mid_val < target)
    recursive call with right pointer
```

Updating pointers in a recursive binary search

In a recursive binary search, if the value has not been found then the recursion must continue on the list by updating the left and right pointers after comparing the target value to the middle value.
If the target is less than the middle value, you know the target has to be somewhere on the left, so, the right pointer must be updated with the middle index. The left pointer will remain the same. Otherwise, the left pointer must be updated with the middle index while the right pointer remains the same. The given code block is a part of a function `binarySearchRecursive()` .

```
function binarySearchRecursive(array, first, last, target) {
  let middle = (first + last) / 2;
  // Base case implementation will be in here.

  if (target < array[middle]) {
    return binarySearchRecursive(array, first, middle,
target);
  } else {
    return binarySearchRecursive(array, middle, last,
target);
  }
}
```

Binary Search Sorted Dataset

Binary search performs the search for the target within a sorted array. Hence, to run a binary search on a dataset, it must be sorted prior to performing it.

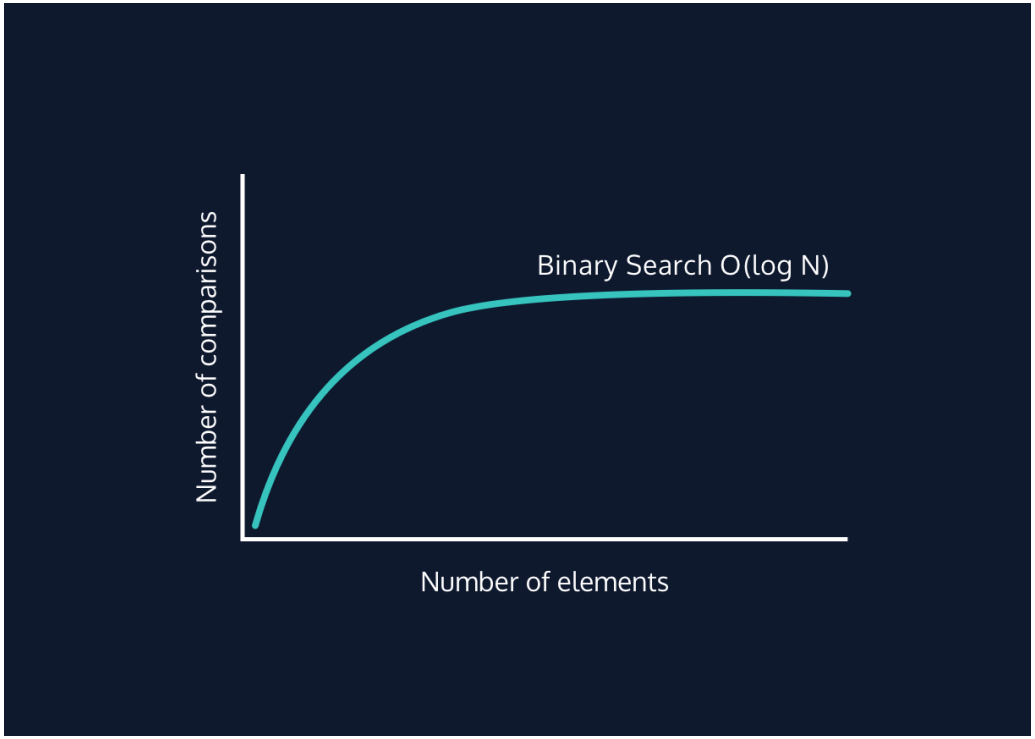
Operation of a Binary Search

The binary search starts the process by comparing the middle element of a sorted dataset with the target value for a match. If the middle element is equal to the target value, then the algorithm is complete. Otherwise, the half in which the target cannot logically exist is eliminated and the search continues on the remaining half in the same manner.
The decision of discarding one half is achievable since the dataset is sorted.

Binary Search Performance

The binary search algorithm takes time to complete, indicated by its time complexity. The worst-case time complexity is $O(\log N)$. This means that as the number of values in a dataset increases, the performance time of the algorithm (the number of comparisons) increases as a function of the base-2 logarithm of the number of values.

Example: Binary searching a list of 64 elements takes at MOST $\log2(64)$ = 6 comparisons to complete.



Binary Search

The binary search algorithm efficiently finds a goal element in a sorted dataset. The algorithm repeatedly compares the goal with the value in the middle of a subset of the dataset. The process begins with the whole dataset; if the goal is smaller than the middle element, the algorithm repeats the process on the smaller (left) half of the dataset. If the goal is larger than the middle element, the algorithm repeats the process on the larger (right) half of the dataset. This continues until the goal is reached or there are no more values.

Print

codecademy