CS50

Binary Search

Overview

There are many different **algorithms** that can used to search through a given array. One option is **linear search**, but it can be a rather lengthy process. Luckily, there is a faster searching algorithm: **binary search**. You might recall that binary search is similar to the process of finding a name in a phonebook. This algorithm's speed can be leaps and bounds better than linear search, but not without a cost: binary search can only be used on data that is already sorted.

Key Terms

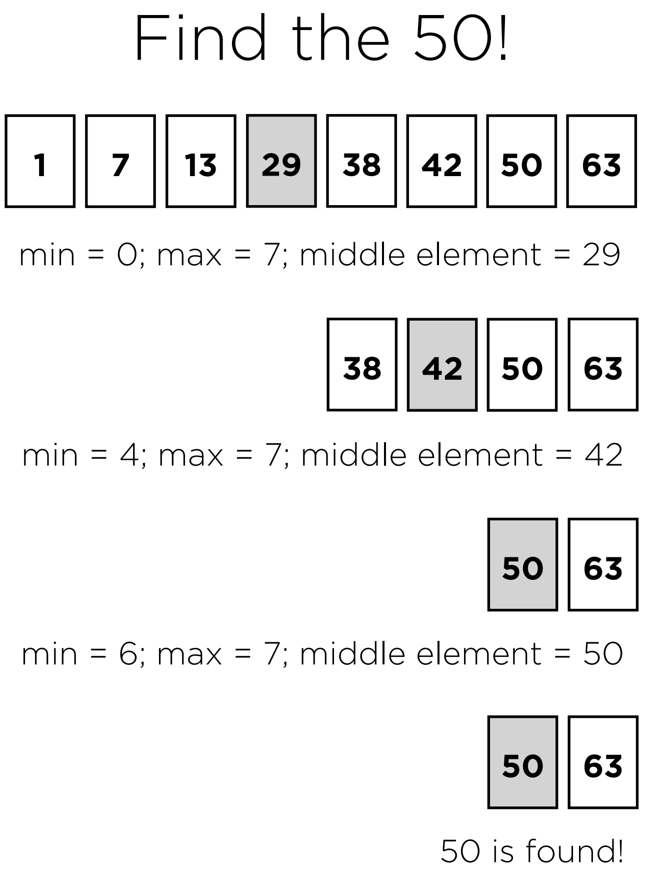
- algorithms
- linear search
- binary search
- pseudocode

The Binary Search Algorithm

The basis of binary search relies on the fact that the data we're searching is already sorted. Let's take a look at what the binary search algorithm looks like in **pseudocode**. In this example, we'll be looking for an element *k* in a sorted array with *n* elements. Here, `min` and `max` have been defined to be the array indices that we are searching between, marking the upper and lower bounds of our search.

```
1  set min = 0 and max = n - 1
2  find middle of array
3  if k is less than array[middle]
4      set max to middle - 1
5      go to line 2
6  else if k is greater than array[middle]
7      set min to middle + 1
8      go to line 2
9  else if k is equal to array[middle]
10     you found k in the array!
11 else
12     k is not in the array
```

Using the algorithm described above, let's search for the number `50` in the array on the right. First we set the `min = 0` and `max = 7`. Next, calculate the `middle` of the array. Well, how do we decide whether to pick the array index `3` or `4` as the `middle`? It actually does not matter, as long as the algorithm is consistent. For our algorithm, let's choose `3`. We've now determined that the `middle` element is `array[3]`, or `29`. Since `50` is larger than `29` and our array is sorted, we know that `50` will not be on the left of `29`. Therefore, there is no need to check those elements. To continue the search, `min` is set to `4`, `max` remains at `7`, and we repeat the process!


Find the 50!
1
7
13
29
38
42
50
63
min = 0; max = 7; middle element = 29
38
42
50
63
min = 4; max = 7; middle element = 42
50
63
min = 6; max = 7; middle element = 50
50
63
50 is found!

Binary Search vs. Linear Search

In computer science, it is a common theme that whenever we make some improvement, it is at the cost of another factor. In this case, we trade the speed of a searching algorithm for the time it takes to sort the array. In some cases, it would be faster to just use linear search rather than to sort the data and then use binary search. Nevertheless, binary search is useful if you plan on searching the array multiple times. In case an element does not exist in the array, linear search would iterate through the entire array – however long it may be – to know that the given element does not exist in the array. In binary search, we can be more efficient. Using the algorithm described in our pseudocode, if the searched element is less than the initial value at `min` or larger than the initial value at `max` (the first and last elements of the array, respectively), the algorithm knows the element is not in the array.