Problem Solving - Divide and Conquer

Download Demo Code

Goals

- Describe what a divide and conquer strategy is
- Discuss when it's possible to use divide and conquer, and when not
- Explain how divide and conquer strategies improve time complexity

An Example: Searching Review

Linear Search

How is ***indexOf*** implemented in JavaScript?

- Loop through your array looking for the desired value
- If you find it, return the index
- If you exhaust the array, return -1
- Time complexity: O(n)

demo/linearSearch.js

```
// custom indexOf implementation
function indexOf(arr, val) {

  for (let i = 0; i < arr.length; i++) {
    // if you find the value, return the index
    if (arr[i] === val) return i;
  }

  // if you never find the value, return -1
  return -1;
}

indexOf([4, 8, 15, 16, 23, 42], 15); // 2
indexOf([4, 8, 15, 16, 23, 42], 42); // 5
indexOf([4, 8, 15, 16, 23, 42], 100); // -1
```

What if your array has 100 elements? 1,000,000,000?

Binary Search

If our array is *sorted*, we can look for values much more quickly!

- Look at the middle value in the array
- If the middle value is the one you're looking for, congratulations!
- If the middle value is too big, you can eliminate **every** value to the right!
- If the middle value is too small, you can eliminate **every** value to the left!
- Among all remaining values, pick the middle one, and repeat.

How Many Comparisons?

Array size	Linear search	Binary Search
3	Max 3 comparisons	Max 2 comparisons
7	Max 7 comparisons	Max 3 comparisons
15	Max 15 comparisons	Max 4 comparisons
31	Max 31 comparisons	Max 5 comparisons
1,023	Max 1,023 comparisons	Max 10 comparisons
1,048,575	Max 1,048,575 comparisons	Max 20 comparisons

Time Complexity: O(log n)

Code Example

demo/binarySearch.js

```
function binarySearch(arr, val) {

  let leftIdx = 0;
  let rightIdx = arr.length - 1;

  while (leftIdx <= rightIdx) {
    // find the middle value
    let middleIdx = Math.floor((leftIdx + rightIdx) / 2);
    let middleVal = arr[middleIdx];

    if (middleVal < val) {
      // middleVal is too small, look at the right half
      leftIdx = middleIdx + 1;
    } else if (middleVal > val) {
      // middleVal is too large, look at the left half
      rightIdx = middleIdx - 1;
    } else {
      // we found our value!
      return middleIdx;
    }
  }

  // left and right pointers crossed, val isn't in arr
  return -1;
}
```

The General Pattern

Divide and Conquer Description

- Given a data set, a divide and conquer algorithm **removes a large fraction of the data set** from consideration at each step.
- Typically, the data set must have some additional structure (e.g. be sorted).
- Significantly improves time complexity, when it's applicable (O(n) -> O(log n))

Why Logs?

A base-2 log (roughly) measures the number of times you can cut a value in half before the value is less than 1.

With divide and conquer, you're often cutting your data set in half until you can't anymore!

Array size	Binary Search	Number of times you can cut array size in half
3	Max 2 comparisons	2
7	Max 3 comparisons	3
15	Max 4 comparisons	4
31	Max 5 comparisons	5
1,023	Max 10 comparisons	10
1,048,575	Max 20 comparisons	20

Divide and Conquer Tips

General Tips

- When considering whether to use divide and conquer, make sure your data is properly structured!
  - For example, if your array isn't sorted, binary search won't work
- If you can think of a linear solution quickly, can you use a divide and conquer approach to improve the complexity?
- Watch out for off by one errors! Managing the left / right pointers can be tricky.