Computer Science & Engineering 120
Learning to Code

Organizing Data I – Searching & Sorting

Christopher M. Bourke
cbourke@cse.unl.edu

Topic Overview

- Searching
- Sorting

Part I: Searching

Searching

- Given a collection (array) of elements, we wish to search it for a particular element
- Variations: finding the minimum, maximum, etc.
- Test for equality can be complex (achieved with a *callback*)
- Two basic algorithms: Linear Search, Binary Search

Linear Search

- Given an array, iterate through it, testing for equality on each element
- If found, stop, if not found, return some sort of *flag* to indicate failure
- Demonstration: `http://cs.armstrong.edu/liang/animation/web/LinearSearch.html`
- Best case: could get lucky and find it at the first index
- Worst case: may find it at the last index or not find it at all
- On average, we perform about $\frac{n}{2}$ comparisons/operations on an array of size n
- Amount of "work" is linear with respect to the size of the array

Binary Search

- Assume the array is *sorted*, we wish to search for k
- Examine the middle element, m:
  - If $m = k$: we've found it
  - If $m < k$: k must lie in the upper half of the array
  - If $k < m$: k must lie in the lower half of the array
- Demonstration: `http://cs.armstrong.edu/liang/animation/web/BinarySearch.html`
- In general, only requires $\approx \log(n)$ comparisons/operations

Comparison

- Suppose we have a size $n = 10^9$ (1 billion) array
- Linear search requires

$$\frac{10^9}{2} = 500,000,000$$

operations
- Binary search requires

$$\log(10^9) \approx 30$$

operations
- Another perspective: doubling the array size, $n \to 2n$
- Linear search requires *twice* as many operations
- Binary search requires only *one more* comparison!

$$\log(2n) = \log(n) + 1$$

Searching in JavaScript

- ES5: `indexOf()` – limited, works only for numbers and strings
- ES6: `find()` – takes a callback
- Binary Search: no version supports, but can be added with a *shim*

Part II: Sorting

Sorting

- Given an array, we want to reorganize it so that elements are *in order*
- Ascending or descending
- Ordering numbers & strings
- Ordering objects
- Example: students: by name? GPA? Class?
- Many algorithms exist

Selection Sort

- Iterate through the array and find the *smallest* element
- Swap it with the first element
- Repeat this process on the remaining $n - 1$ elements until sorted
- Demonstration: `http://cs.armstrong.edu/liang/animation/web/SelectionSort.html`
- Requires about n^2 operations

Other Sorting Algorithms

- Insertion Sort, Quick Sort, Merge Sort, Tim Sort, etc.
- "Slow" algorithms take about n^2 operations
- Doubling the size of the array *quadruples* the execution time!

$$(2n)^2 = 4n^2$$

- "Fast" algorithms require about $n \log(n)$ operations
- Doubling the size of the array requires (roughly) only *twice* as many operations
- Fast algorithms *scale*

Sorting the Right Way

- In Software Development its rarely good to "reinvent the wheel"
- Use built-in sorting and searching functions
- JavaScript: `arr.sort()`

Sorting in JavaScript

- Problem: the default behavior is to sort lexicographically.
- For strings: this is fine
- For numbers: it comes out wrong
- Demonstration

Demonstration

```
var names = ["Jolene", "Irene", "Roxanne", "Cecilia", "Lola"
names.sort();
names;

var nums = [8, 2, 9, 4, 100, 3];
nums.sort();
nums;
```

Use a Comparator

- Solution: use a callback to define the ordering!
- `sort()` knows how to sort, but not how to order
- We use a callback that takes two elements `a, b` and returns a number indicating their order:
  - < 0 if $a < b$
  - 0 if $a = b$
  - > 0 if $a > b$
- Such a function is called a *comparator*
- Demonstration

Demonstration

```
var nums = [8, 2, 9, 4, 100, 3];
nums.sort(function(a, b) {
  return (b - a);
});
nums;
```