Advanced Sorting Topics

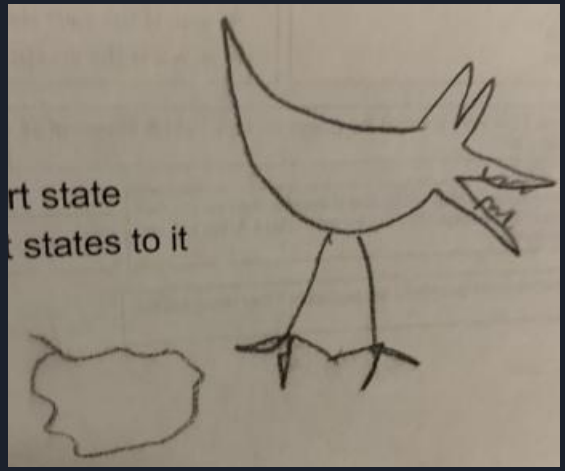
CS 240 Spring 2019

So far...

- We've seen sorts with $O(n^2)$ and O(nlgn) time complexities
  - Insertion sort - O(n) best case, $O(n^2)$ average and worst case
  - Merge sort - O(nlgn) all cases
  - Quick sort - O(nlgn) best and average cases, $O(n^2)$ worst case
- Can we do better?
- Yes
  - Maybe
    - It depends

The Pigeonhole Principle

- Given *n* slots and *n+1* items to insert
- At least one slot will have 2 entries
- Can we make a sort out of this?
  - Hell yeah



(Garrison, 2019)

Pigeonhole Sort — The Concept

- Given an array *A* to be sorted
- Find the minimum and maximum value in *A*
- Create a new array *S* of size *max - min + 1*
  - Initialize each value to 0
  - Index will act as a key, value will be an accumulator
  - We'll need an offset — the min value must be index 0
- Iterate over *A* again, this time incrementing each value's accumulator

Pigeonhole Sort — Pseudocode

```
def pigeonhole_sort(a):
    my_min = min(a)
    my_max = max(a)
    size = my_max - my_min + 1 # size of range of values in the list

    holes = [0] * size # our list of pigeonholes

    for x in a: # Populate the pigeonholes.
        holes[x - my_min] += 1

    i = 0
    for count in range(size): # Put the elements back into the array in order.
        while holes[count] > 0:
            holes[count] -= 1
            a[i] = count + my_min
            i += 1
```

Radix Sort — The Concept

- Two options — sort by least significant digit first (LSD) or most significant digit first (MSD)
- We'll do LSD
- Stably sort the array by least significant digit
- Then resort it by the next least significant digit
- Continue until we reach the most significant digit

Radix Sort — Pseudocode

```
def radixSort(arr):
    max1 = max(arr)

    exp = 1
    while max1/exp > 0:
        someStableSortByDigitPos(arr,exp)
        exp *= 10
```

TimSort — The Concept

- Know that insertion sort has a best case of O(n)
- Know that merge sort has a best and worst case of O(nlgn)
- TimSort combines these two for a best case of O(n) and a worst case of O(nlgn)
- Finds runs of increasing order in array using insertion sort-like approach
- Then zips them up like merge sort
- So it’s a hybrid sort because it combines aspects of insertion sort and merge sort
- It’s also a heuristic sort because it parses through the array looking for runs

TimSort's Performance

Array Type	Array Size	Insertion Sort (ms)	Merge Sort (ms)	TimSort (ms)	Quick Sort (ms)
Sorted	100	0.00186	0.06327	0.00788	0.13169
Sorted	10000	0.19880	10.04211	0.27717	857.48018
Random	100	0.04676	0.09062	0.06512	0.01968
Random	10000	219.45443	6.64250	17.57641	2.55195
Runs (5 x 20)	100	0.02236	0.04594	0.02050	0.04231
Runs (50 x 200)	10000	184.90110	5.87031	1.28531	88.83914

Let’s review

1. {1,2,3,5,4,6,7,8,9}
    a. Insertion Sort or Quick Sort?
2. {1,2,3,1,3,4,3,1,2,1,3,4,2,1,2,3,4,2,1,2,3}
    a. Pigeonhole Sort or Quick Sort?
3. {1,2,3,1,2,3,1,2,3,1,2,3}
    a. TimSort or Quick Sort?
4. {9365,862,395,606,3758,2,30,549,70,32,524,612,74}
    a. TimSort or Quick Sort?

Final Comments

- The data you're working with should determine which sorting algorithm you choose
- Don't be fooled by pure time complexities — sorting algorithms have a hidden constant *k* in their time complexities
  - Radix sort — O(k * n)
  - Quick sort — O(k * nlgn) average case
- Quick sort's k is much much lower than Radix sort's