2018-04-26 Quicksort, Radix Sort, and Others

Thursday, April 26, 2018 8:38 AM

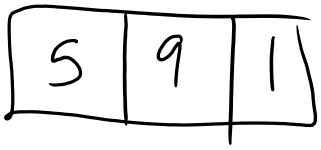
Quicksort

- General idea: find a "pivot" and put smaller values to the right of the pivot and larger value to the left of the pivot
- Recursively do this until we reach a sufficiently small size
  - In our example, either 1 or 2
  - In practice usually no smaller than 30

The Pivot

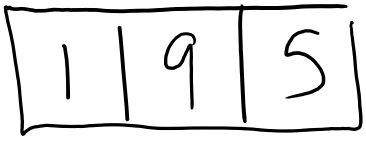
- Ideally, we want the pivot to perfectly split the data (i.e. find the median value)
  - Computationally, this is too expensive. We need a shortcut.
- Old-timey approaches:
  - Pick first value in array, last value in array, random value in array
- Best "bang for the buck" approach:
  - Best of three - pick median of three values
    - Chosen from "first" element, "middle" element, and "last" element

The "simplest" example of quicksort (array size 3)

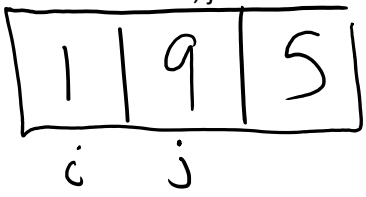


Pivot = 5

1. Swap pivot with last element in array



2. Declare i = start, j = end - 1

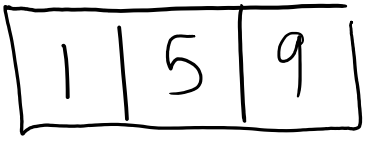


3. WHILE numbers[i] < pivot AND i < j
   a. i++
4. WHILE numbers[j] > pivot AND i < j
   a. j--
5. IF i < j, swap numbers[i], numbers[j]
   a. Go to #3
6. (i must be at least equal to j)
7. Swap numbers[end], numbers[i]
8. Recursively do the same on:
   a. numbers[start], numbers[i]
   b. numbers[i + 1], numbers[end]

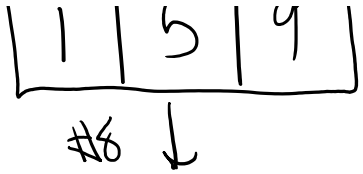
#3 6

i
j

#7



7. ~~Swap numbers[end], numbers[i]~~
8. Recursively do the same on:
    a. numbers[start], numbers[i]
    b. numbers[i + 1], numbers[end]



Recursive bit omitted (base case reached)

Example #2

4	7	6	9	1	3	1	2	5
0	1	2	3	4	5	6	7	8

Pivot options: 4, 5, 1

5	7	6	9	1	3	1	2	4
0	1	2	3	4	5	6	7	8

i j

- I and J don't move. I != J, so swap values at I and J

2	7	6	9	1	3	1	5	4
0	1	2	3	4	5	6	7	8

i → i j ← j

2	1	6	9	1	3	7	5	4
0	1	2	3	4	5	6	7	8

i → i j ← j

2	1	3	9	1	6	7	5	4
0	1	2	3	4	5	6	7	8

i → i j ← j

2	1	3	1	9	6	7	5	4
0	1	2	3	4	5	6	7	8

i → j
i

I and equals J, so swap pivot with index I

2	1	3	1	4	6	7	5	9

0	1	2	3	4	5	6	7	8

- Recursively do the same thing on subsegments

2	1	3	1
0	1	2	3

Pivot choices: 2, 1, 1 → 1, 3, 2

2	1	3	1
0	1	2	3

i j ← j

1	2	3	1
0	1	2	3

i → j
i

1	1	3	2
0	1	2	3

1	1	2	3
0	1	2	3

6	7	5	9
5	6	7	8

Pivot choices: 6, 7, 9 → note should be 6, 5, 9

6	9	5	7
5	6	7	8

i → i j

6	5	9	7
5	6	7	8

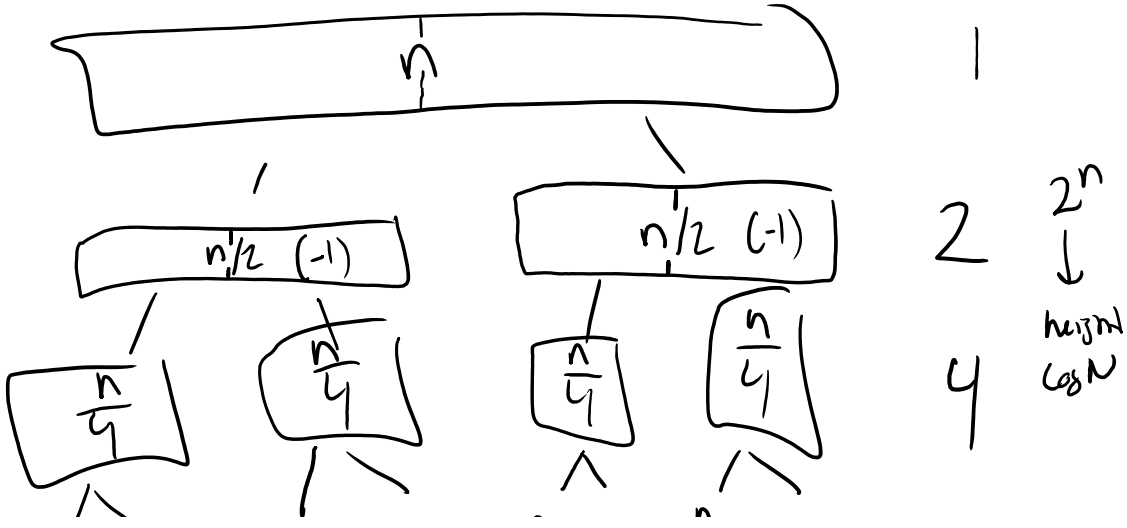
i → j

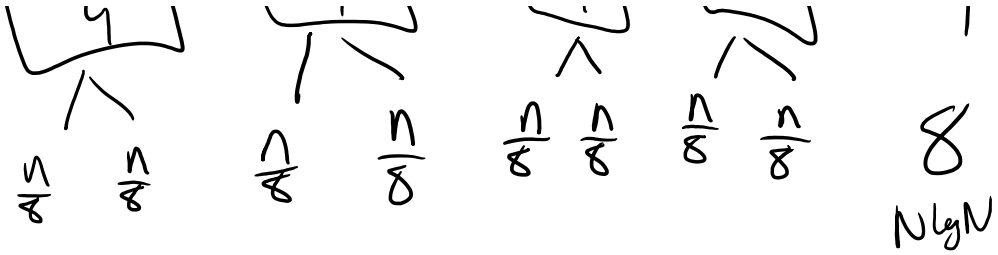
6	5	7	9
5	6	7	8

5	6	7	9
5	6	7	8

Algorithmic Efficiencies

- Consider the case in which we always choose the best pivot possible. How much new information does each iteration give us?





- Worst case: worst pivot always selected