COP3502 3/28/24

Quick Sort

Partition

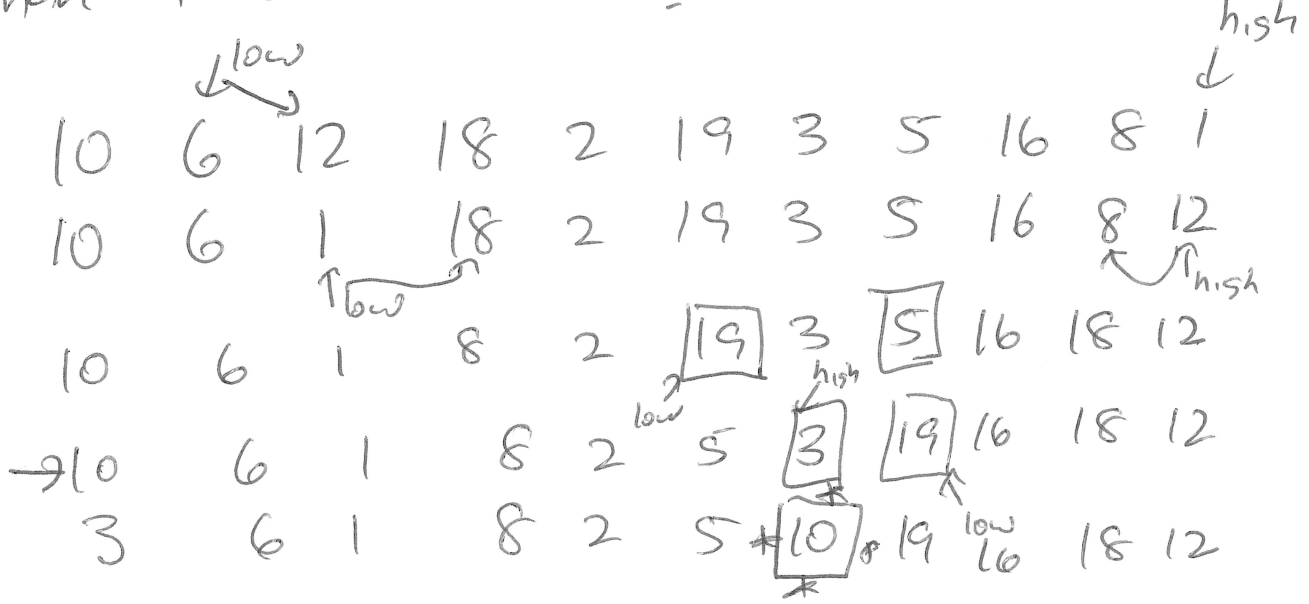
10	6	12	18	2	19	3	5	16	8	1

↑ partition element

methods to pick partition

1) pick leftmost item (some inputs consistently bad)
2) pick random item swap into leftmost place
3) pick 3 rand elem, take median of those + swap into leftmost
4) pick 5 = = , do same.

prob of getting a "more even" split increases BUT we have to do more work!



Quick Sort Pseudocode

```
qsort(int* arr, int low, int high) {
    if (low >= high) return;
    int partIndex = partition(arr, low, high);
    qsort(arr, low, partIndex-1);
    qsort(arr, partIndex+1, high);
}
```

Run time: Worst Case Split 0 + n-1 every time.

$$T(n) = T(n-1) + O(n) \Rightarrow T(n) = O(n^2)$$

Best Case: Split evenly roughly

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \Rightarrow T(n) = O(n \log n)$$

Avg Case Run Time

$$T(n) = \sum_{k=0}^{n-1} \frac{1}{n}\left(T(k) + T(n-1-k)\right) + O(n)$$

$\frac{1}{n}$	0	n-1
$\frac{1}{n}$	1	n-2
$\frac{1}{n}$	2	n-3
⋮		
$\frac{1}{n}$	n-1	0

$\sim$ $T(n) = O(n \log n)$

Merge Sort v Quick Sort

Merge Sort	Quick Sort
~~FD~~ new extra memory	in place
Worst Case $O(n \log n)$	Worst Case $O(n^2)$
In practice on avg data this is a bit slower (mallocs + double copy)	In practice, on avg data this is faster
theoretically better split BUT cost extra overhead	theoretically worse but less overhead
In place	Not in place!