Lecture 07: QuickSort

COSC 311 *Algorithms*, Fall 2022

Announcements

1. Homework 1, Exercise 4
2. Collaboration and Exercise
3. Homework 2, Posted Sunday

Overview

1. QuickSort

Picture so Far:

SelectionSort. $O(n^2)$ operations

- $O(n^2)$ comparisons
- $O(n)$ swaps

BubbleSort and **InsertionSort.** $O(n^2)$ operations

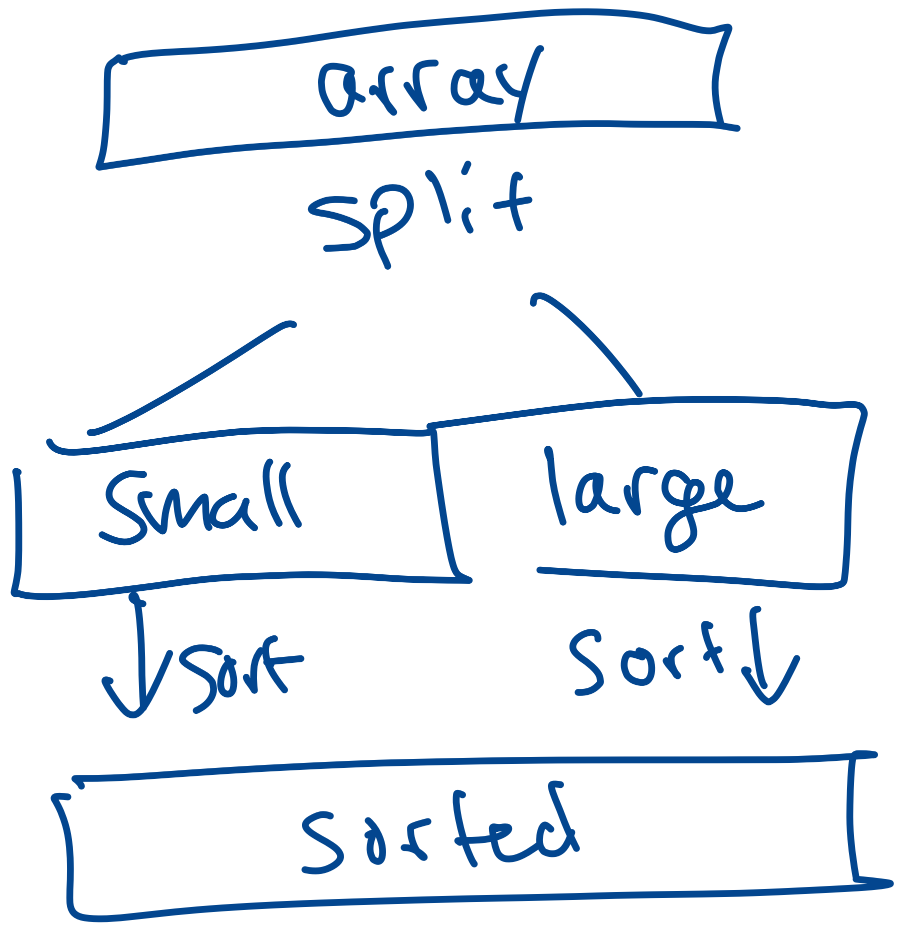
- $O(n^2)$ comparisons
- $O(n^2)$ swaps

MergeSort. $O(n \log n)$ operations

- $O(n \log n)$ comparisons
- $O(n \log n)$ modifications
- uses $O(n)$ space overhead

Today

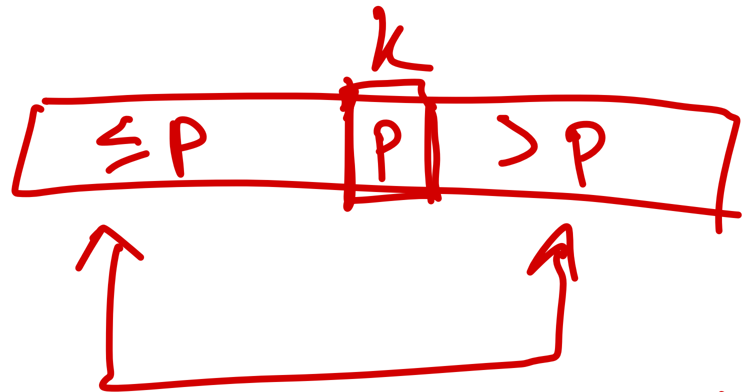
QuickSort: Divide by Value



QuickSort: Another D&C Sort

Idea. Divide array a by *value*

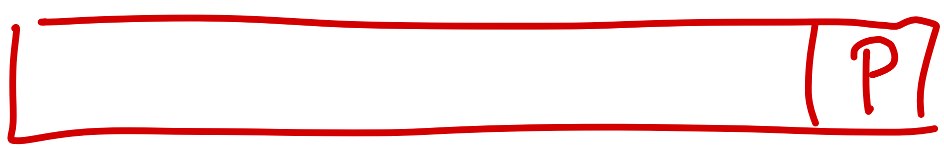
- choose a value p from a, the **pivot**
- arrage values of a such that:
  - p is at index k
  - values $\leq p$ are at indices $i \leq k$
  - values $> p$ are at indices $j > k$
- recursively sort indices $i < k$
- recursively sort indice $j > k$



sort recursively

if p is max value

nothing



QuickSort Illustration

$\boxed{3\ 1\ 6\ 8\ 4\ 5\ 7\ 2}$

3 1 6 8 4 5 7 2

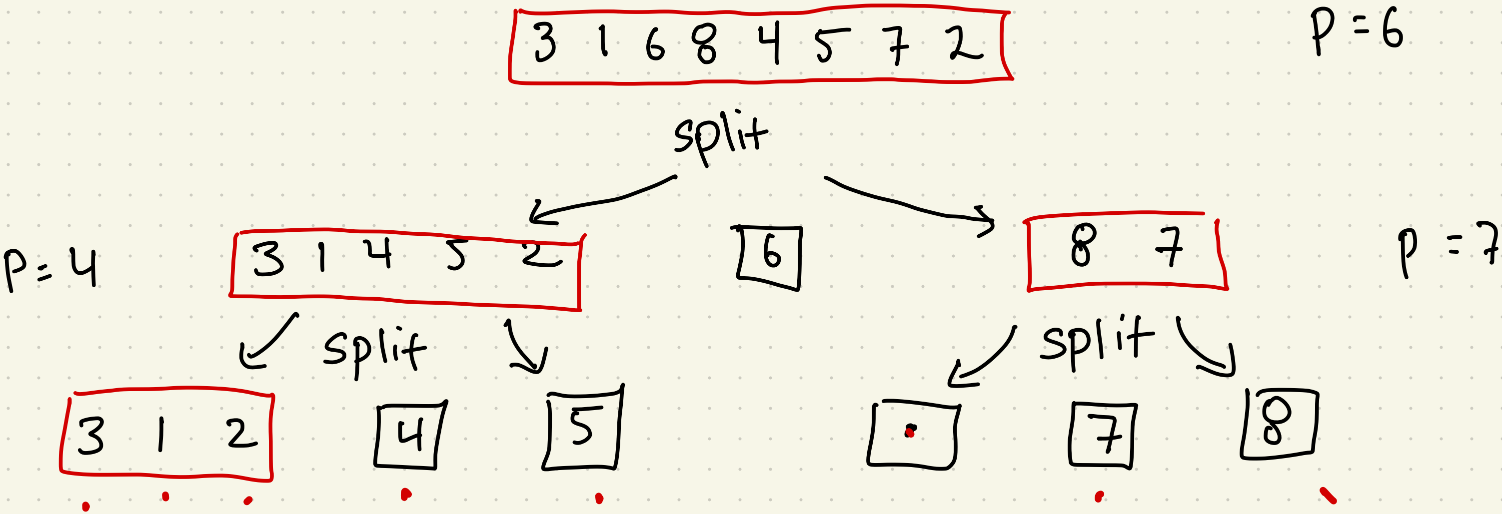
$p = 6$

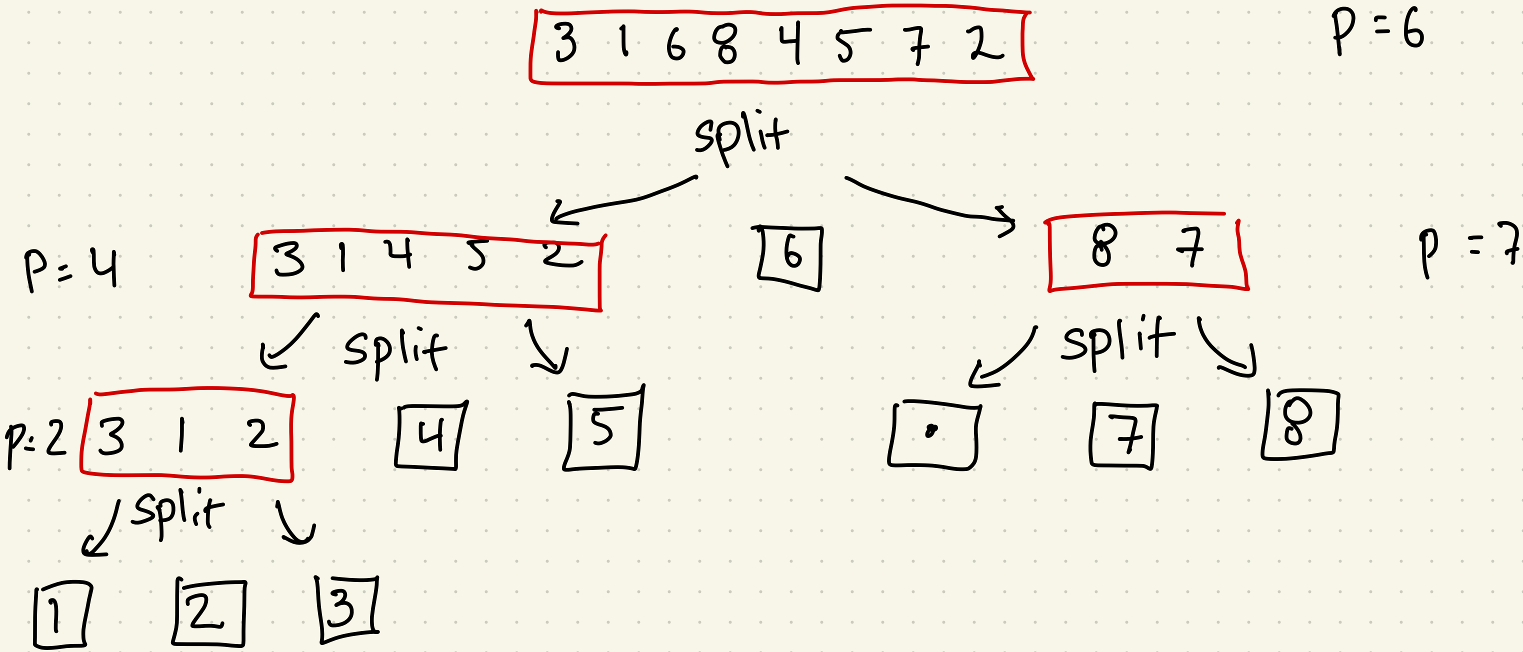
split

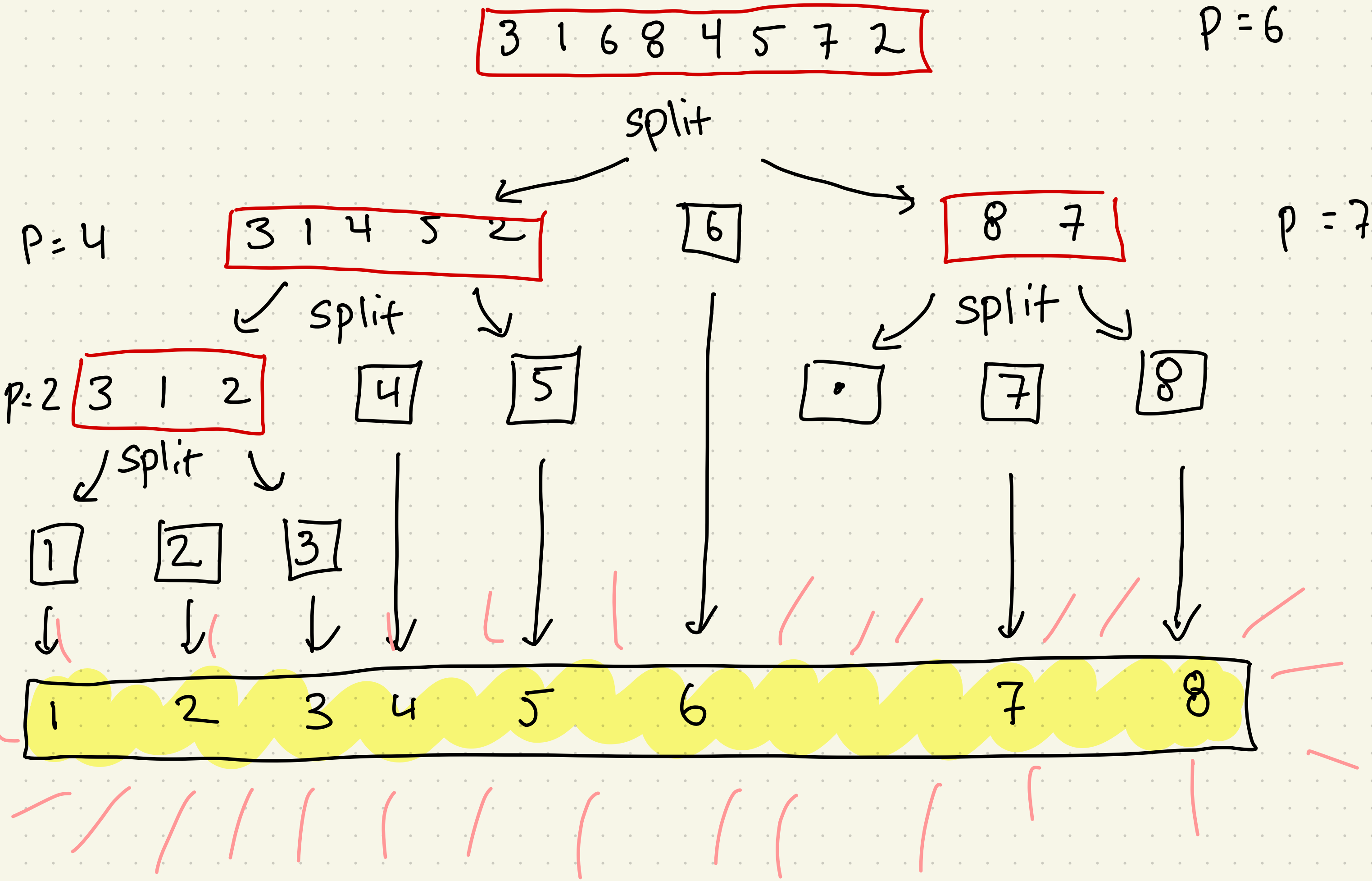
3 1 4 5 2

$\boxed{6}$

8 7


3 1 6 8 4 5 7 2
p = 6
split
p = 4
3 1 4 5 2
6
8 7
p = 7
split
split
3 1 2
4
5
7
8


3 1 6 8 4 5 7 2
p = 6
split
p = 4
3 1 4 5 2
6
8 7
p = 7
split
split
p: 2
3 1 2
4
5
7
8
split
1
2
3


3 1 6 8 4 5 7 2
P=6
split
P=4
3 1 4 5 2
6
8 7
p=7
split
split
p:2
3 1 2
4
5
7
8
split
1
2
3
1 2 3 4 5 6 7 8

array

sort between indices i and j

QuickSort Pseudocode

```
QuickSort(a, i, j):
  if j - i <= 1 then
    return
  endif
  p <- GetPivot(a, i, j) # select a pivot
  k <- Split(a, i, j, p)
  QuickSort(a, i, k-1)
  QuickSort(a, k+1, j)
```

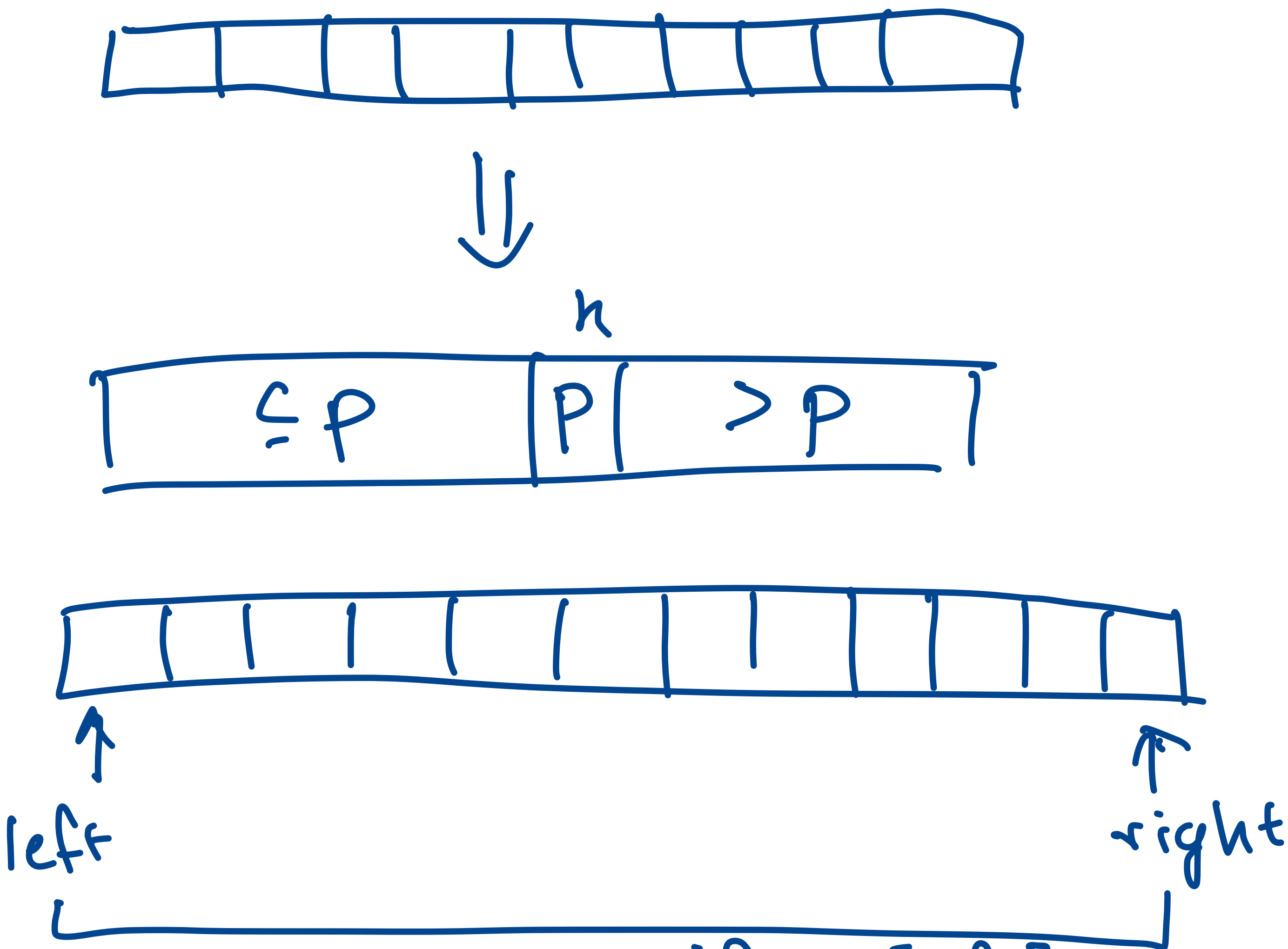
Base case

k

$\le p$ | p | $\ge p$

Split Illustration

p = pivot



swap if a[left] > p and a[right] ≤ p

otherwise · inc. left if small
· dec right if large

Split Method

```
Split(a, i, j, p):
  left <- i, right <- j
  while left < right do
    if a[left] > p and a[right] <= p then
       swap(a, left, right)
       left++, right--
    else
      if a[left] <= p then left++
      if a[right] > p then right--
    endif
  endwhile
  return right
```

both values out of place

O(1)

≤ k iterations = O(k)

What Is Split Running Time?

Size of call is $k = j - i$

$$O(k).$$

QuickSort Pseudocode

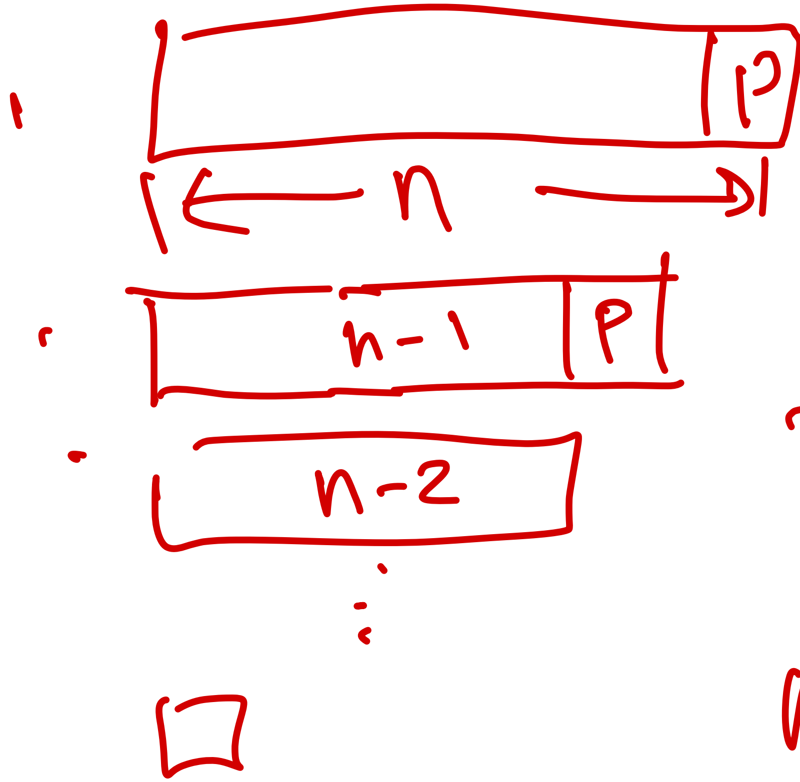
```
QuickSort(a, i, j):
  if j - i <= 1 then
    return
  endif
  p <- GetPivot(a, i, j) # select a pivot
  k <- Split(a, i, j, p)  ← O(j-i) time
  QuickSort(a, i, k-1)
  QuickSort(a, k+1, j)
```

What is Worst-Case QS Running Time?

Assume GetPivot(a, i, j) returns a value in $a[i..j]$

- as with MergeSort, total time at each depth of recursion is $O(n)$

Bad: pivot is always max elt.



Depth $\leq n$

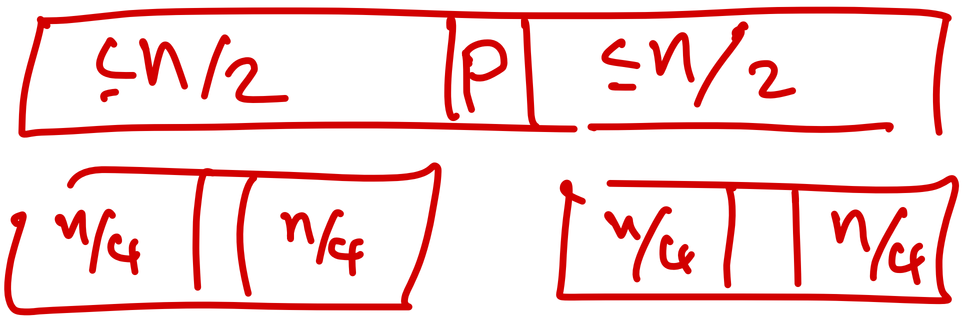
Depth * time per depth

$n \cdot O(n) = O(n^2)$

When is QS Running Time Better?

Best pivot selection?

What if p = median value



Same analysis as merge sort

$\Rightarrow$ running time $= \Theta(n \log n)$.

Acceptable Pivot Selection?

What if we can't get the best possible? What might still be acceptable?

→ consider rank of element

→ get pivot w/ rank between $\frac{n}{4}$ and $\frac{3n}{4}$

$\log(a^b) = b \log a$

A Heuristic

A pivot p is **good** if its rank is between $\frac{1}{4}k$ and $\frac{3}{4}k$

- $\implies$ larger recursive call has size at most $\frac{3}{4}k$

If all calls are good, what is depth of recursion?

0 • n original call

1 • $\frac{3}{4}n$

2 $\left(\frac{3}{4}\right)\square = \left(\frac{3}{4}\right)^2 \cdot n$

⋮

ℓ $\left(\frac{3}{4}\right)^\ell \cdot n$

Base case when = 1

$\left(\frac{3}{4}\right)^\ell \cdot n = 1$

$\iff n = \left(\frac{4}{3}\right)^\ell$

$\log n = \log\left(\left(\frac{4}{3}\right)^\ell\right)$

$\log n = \ell \log\left(\frac{4}{3}\right)$

$\boxed{\ell = \Theta(\log n)}$