2] Given a number
Can we find its index in the sorted array?

$\boxed{9}$	3	2	8	7	6	11	12	22	1
	>	>	>	>	>	<	<	<	>

Find # that are less than 9 : 6 of them
Find # that are greater than 9 : 3 of them

=> $\boxed{9}$ should be at index 6 if array is sorted

Given an index, can we find what number should end up there?

3]

$$\underbrace{\overset{k}{\boxed{\le A[P] \;\big|\; \big| \ge A[P]}}}_{\uparrow\, A[P]}$$

input: $A[\,* * * \dots * \overset{p}{\underset{= X}{A[p]}} * * \dots *\,]$

after: $A[\,* \dots * \overset{q}{X} * \dots *\,]$

$\le X$ $> X$

- p may not equal q
- partitioning is done "in-place"
  - a bit tricky to implement

5] input: A[7 3 1 9 2 6 4 11 12 5 22]

$p = 5$, $A[p] = 6$

line 1: swap pivot to front

swap

A[6 | 3 1 $\cancel{9}$ 5 2 $\cancel{7}$ 4 $\cancel{4}$ 7 11 12 $\cancel{5}$ 9 22]

i j i j

1st while loop: find item > pivot from left } Keep swapping
2nd while loop: find item ≤ pivot from right } until i & j cross

	0	1	2	3	4	5	6	7	8	9	10
A[	4	3	1	5	2	6	7	11	12	9	22]

$\underbrace{4\ 3\ 1\ 5\ 2}_{L}\ \underset{i}{6}\ \underbrace{7\ 11\ 12\ 9\ 22}_{R}$

6] Quick-select: choose pivot

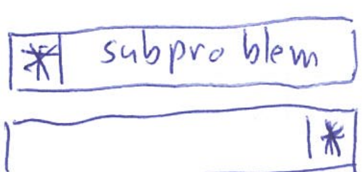
given k

i = partition

figure out how to proceed

1. if $i = k$ found element at k
2. if $i > k$, recurse, find k in L
3. if $i < k$, recurse, find $(k - i - 1)$ in R

worst case:



• one subproblem of size n-1 to recurse on

...

worst case:

$$T(n) = T(n-1) + cn$$
$$= [T(n-2) + c(n-1)] + cn$$
$$= T(n-3) + c(n-2) + c(n-1) + cn$$
$$\cdots$$
$$= d + c2 + c3 + \ldots + c(n-1) + cn$$
$$= d + c\left[(n+2)\left(\frac{n-1}{2}\right)\right] \in \Theta(n^2)$$

Best case: we're lucky - find item in the first pass
- still partition once

$\in \Theta(n)$

Average case:

```
fool(k)
  for i = 1 to k
    printf("\uD83\uDCA9")
```

#calls to printf in average?

$1 \le k \le 6$

$$\frac{1}{6}\sum_{i=1}^{6} i = \frac{21}{6} = \frac{7}{2}$$

- consider all inputs
  - all inputs are "equally likely"
- take average

- Assumption
  - each input is equally likely - "uniform distribution"

Need to consider
- all inputs of a certain size, then take average
  - sum up all runtimes
  - divide by # of inputs

Assumption
- no items are repeated in the array
- behaviour of alg depends on relative ordering of key
  - not actual value

eg [2 4 6 8] & [11 12 13 14] both worst case

=> may assume keys are $1, 2, \ldots, n$

=> need to consider all orderings: $n!$

* we'll count # of comparisons

Remember: n possible locations for pivot
- each one has $(n-1)!$ permutions of non-pivot elements
- each pivot location is "equally likely"
  => divide by n