```
def median(L):
    L.sort()
    return L[len(L)//2]
```

) can we find medians more efficiently?

Why?

- Find good pivots in divide + conquer algorithms such as quicksort (either median of whole input or of sample)
- Statistics — find central/representative point for cloud of (1-d) data points

if points are true value + normally distributed noise (Bell curve) — use average

if a small number of points could be very far from main cloud of points (outliers) — median more robust will still be accurate as long as #outliers < 50%

"breakdown point" of median = 0.5

- Data whose ordering is meaningful but whose numerical value is not

Movie ratings by # of stars

Meaningful: you gave one movie more stars than another

Not meaningful: this movie is one star better than this other one

(diff. 4☆ − 3☆ ≠ 2☆ − 1☆)

Not meaningful: average of star ratings

Meaningful: median

- Facility location
  - Where along this road should I place my store to minimize average customer driving time? (ignore price, zoning, ...)
  - given n points on a line x_i
    median = point y that minimizes $\frac{1}{n}\sum_{i=0}^{n-1}|x_i - y|$

Proof: $\frac{d}{dx}\left(\frac{1}{n}\sum|x_i - y|\right)$

← each x_i contributes $\pm\frac{1}{n}$

$=$ (fraction of points $x_i > y$) $-$ (fraction of points $x_i < y$)

O(n) – time median computation

more general problem: select kth smallest
(k=0 ⇒ min k=n-1 ⇒ max k=n/2 ⇒ median)

Deterministic – complicated, tricky analysis, impractical

Randomized, based on quicksort – simple and practical

Basic idea –
interleave quicksort recursion with selection problem
instead of doing whole quicksort algorithm first –
skipping some irrelevant recursive calls ⇐ faster

```
def quickselect(L, k):
    if len(L) <= 1: return L[0]   # after checking that k==0
    choose random pivot p
    partition L into
        X: elements < p
        p itself
        Y: elements > p
    if k < len(X):
        return quickselect(X, k)
    else if k != len(X):
        return p
    return quickselect(Y, k-len(X)-1)
```

adjust k to account for removing X, p

Analysis of quickselect

Option 1: like quicksort,

$$E[\#\text{compares}] = \sum_{\text{pair } x,y} \Pr[x, y \text{ are compared}]$$

Option 2: recurrence

$$E[\text{Time}(n)] = O(n) + \sum_{\substack{\text{subproblem} \\ \text{size } s}} \Pr(s) \times \text{Time}(s)$$

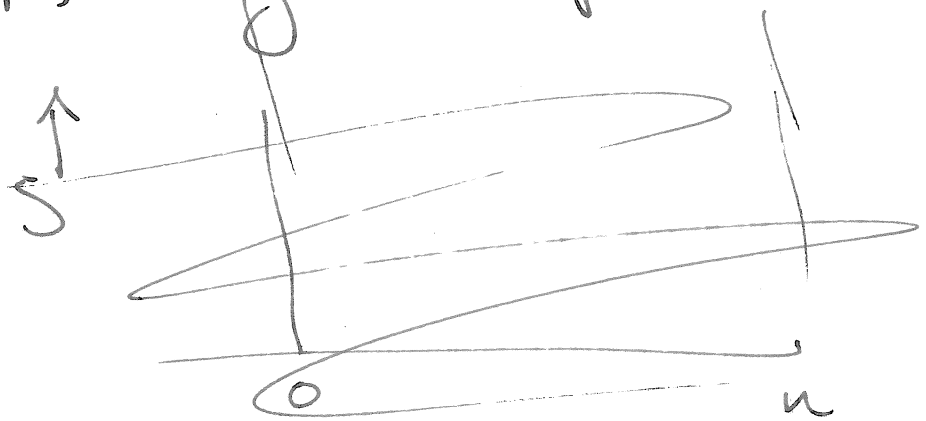
$O(n)$ at top level

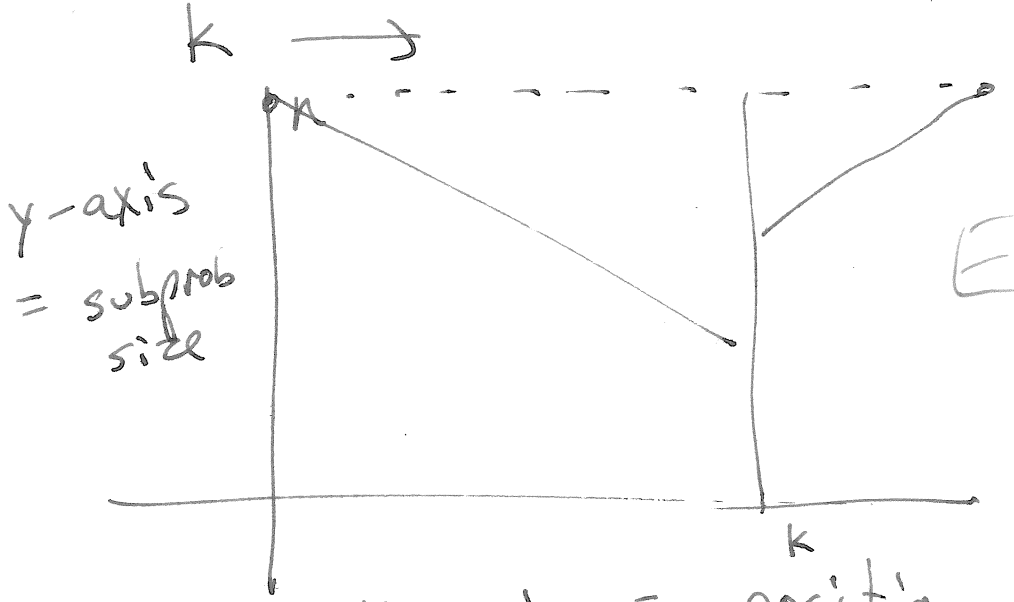
"=" $O(n) + T(\text{average subprob. size})$

(not really)

swapping $\geq$ and $T()$ only works if T already known to be linear

Average subproblem size?





x-axis = position of pivot in sorted list

Claim:
for worst case choice of k,
$E[\text{subprob size}] \leq \frac{3n}{4}$

$$T(n) = O(n) + \frac{3}{4} T(n)$$

$$= O(n)$$

plug into induction proof

$T(n')$ $n' < n$

linear by induct. hyp.

$\Rightarrow$ justifies $\leq / T$ swap