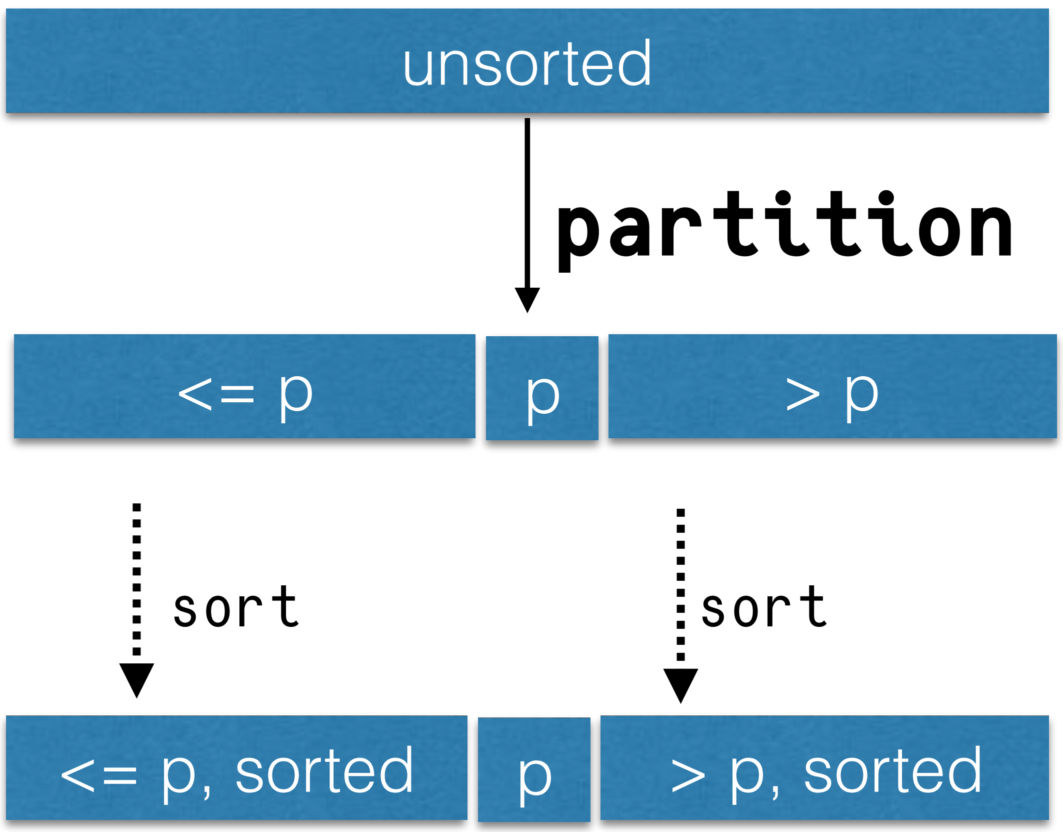
Quicksort

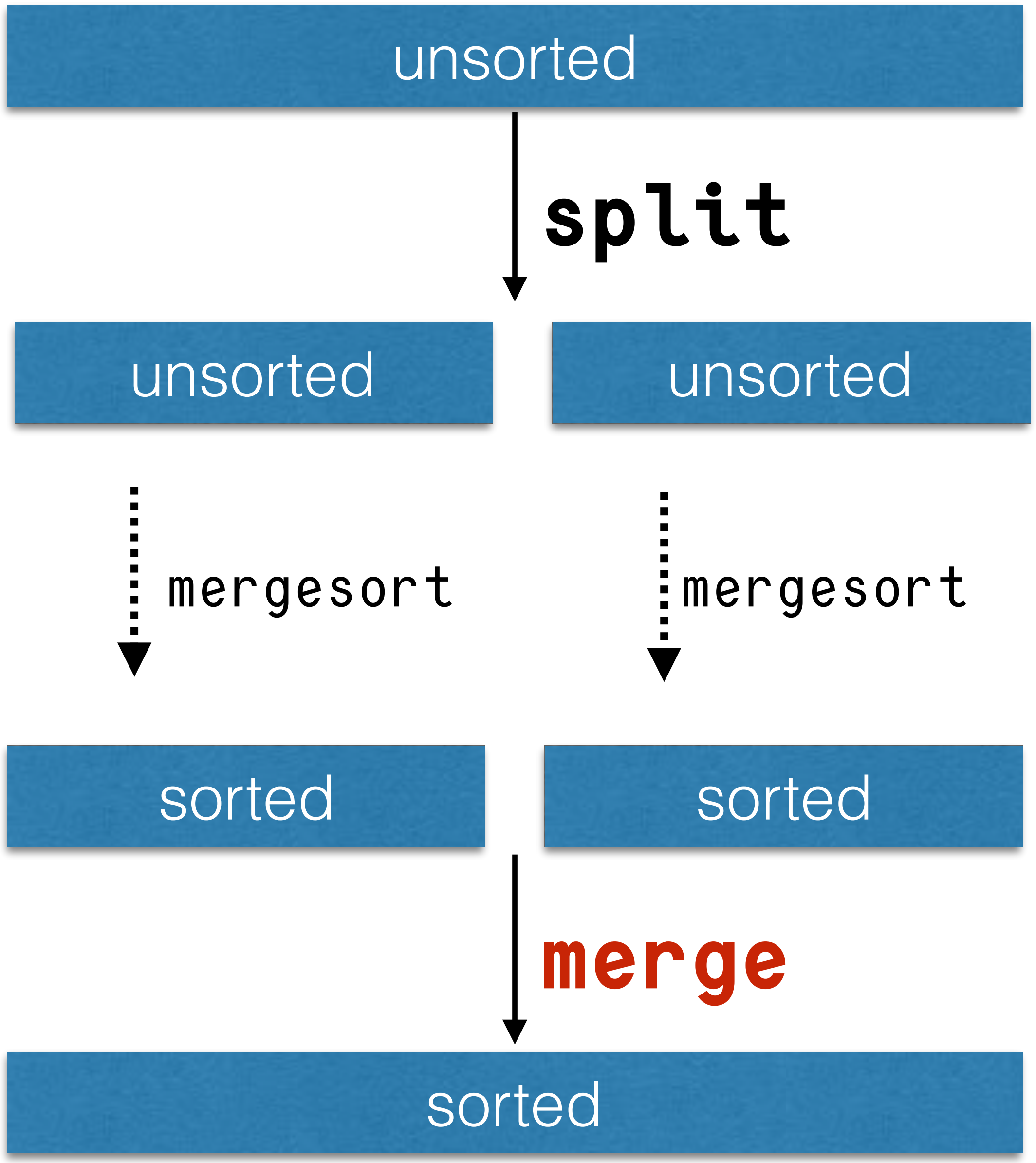
Data structure & Algorithms - CS 146 Spring 2017

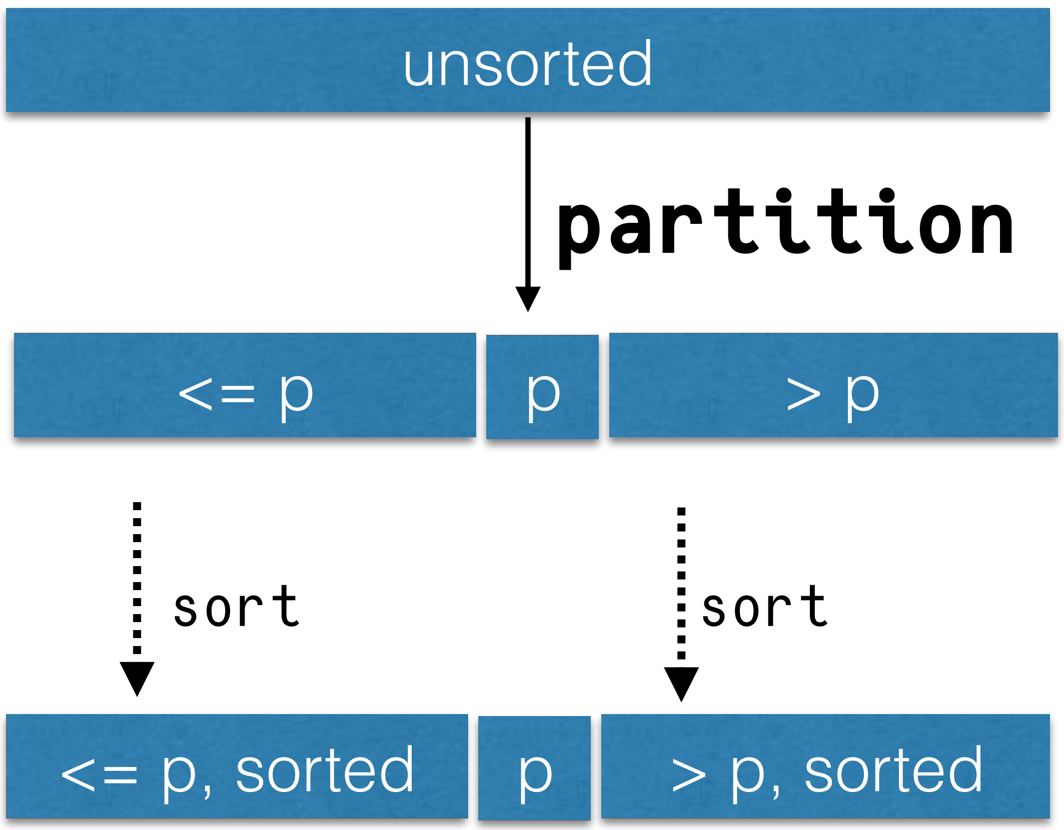
Recall the idea of quick sort



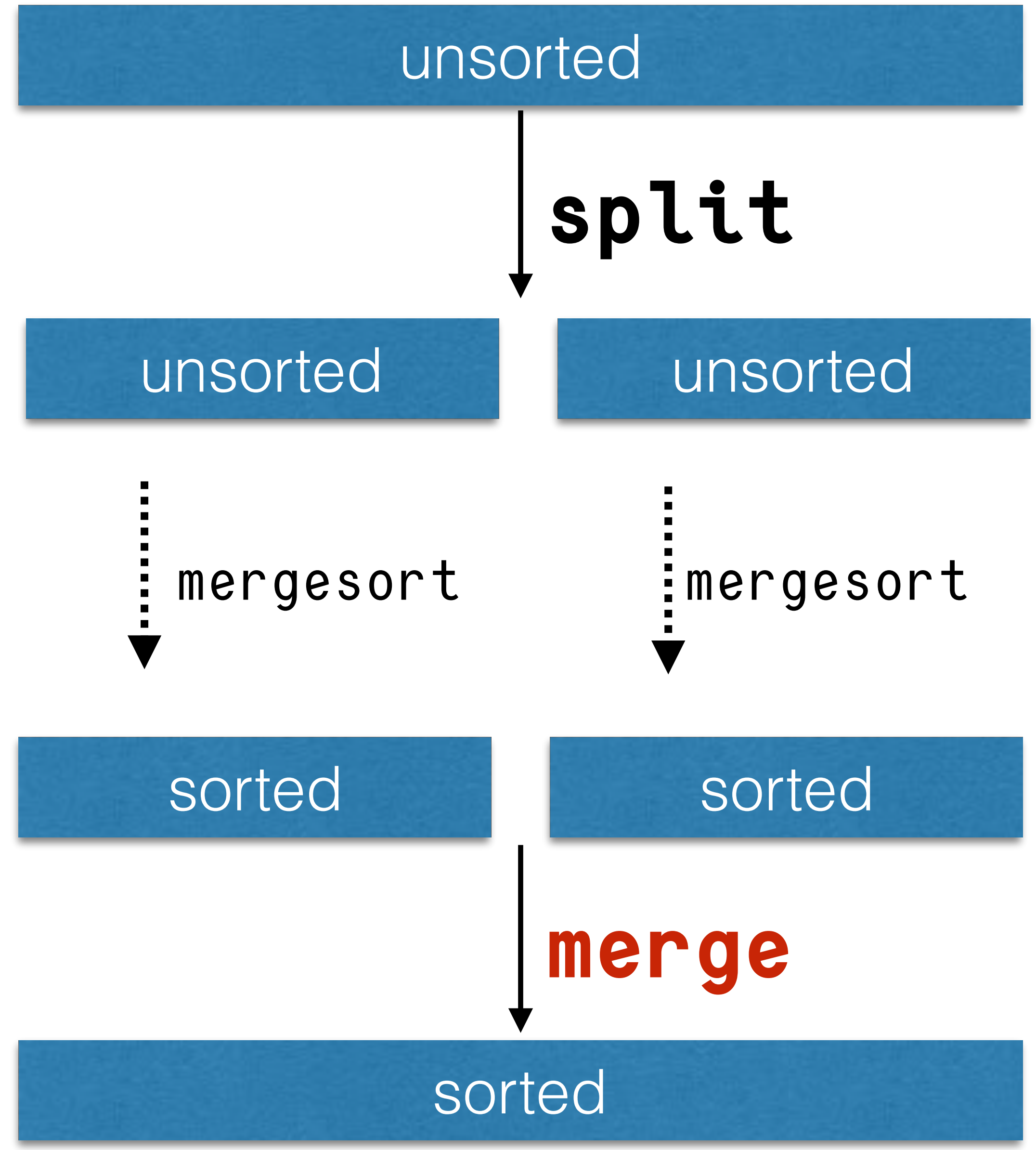

```
void mergesort(list) {

    if (length(list) <= 1) return;

    pick pivot,

    split list into sublist <= p, p ar
.                     sublist > p

    quicksort(sublist <= p);

    quicksort(sublist > p);

}
```

```
void mergesort(list) {

    if (length(list) <= 1) return;

    split list into left and
.                  right sublists

    mergesort(left);

    mergesort(right);

    merge left and right;

}
```





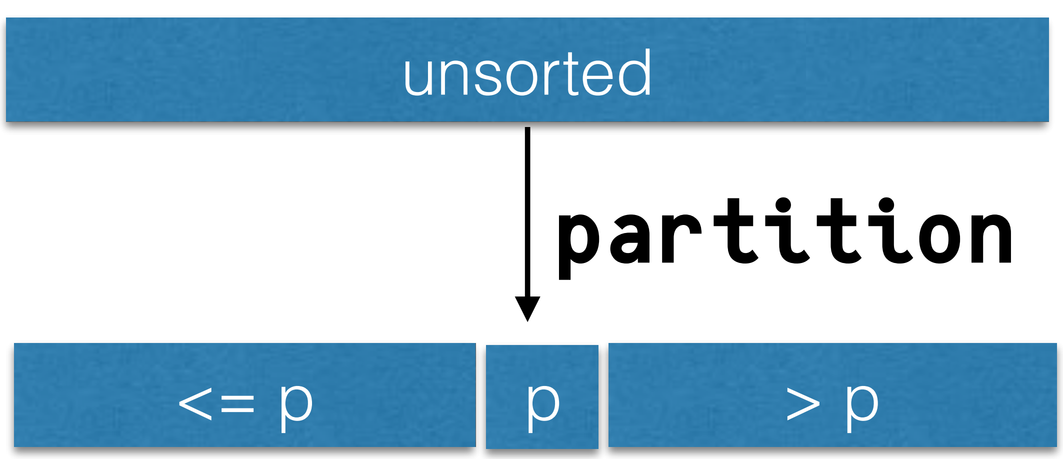
quick sort



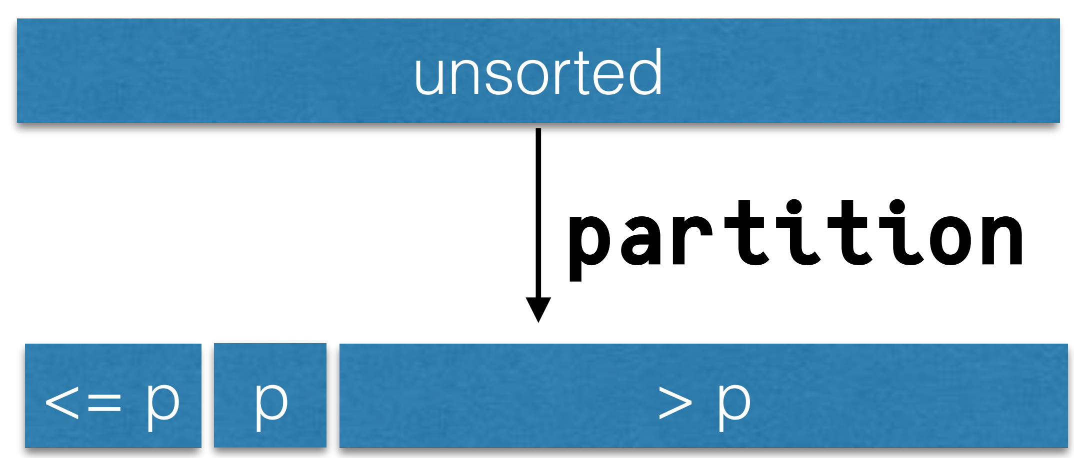
mergesort

Time complexity (1)

- partitioning takes O(n) time, n size of subarray
- depending on which pivot we choose, we can have a good or bad split



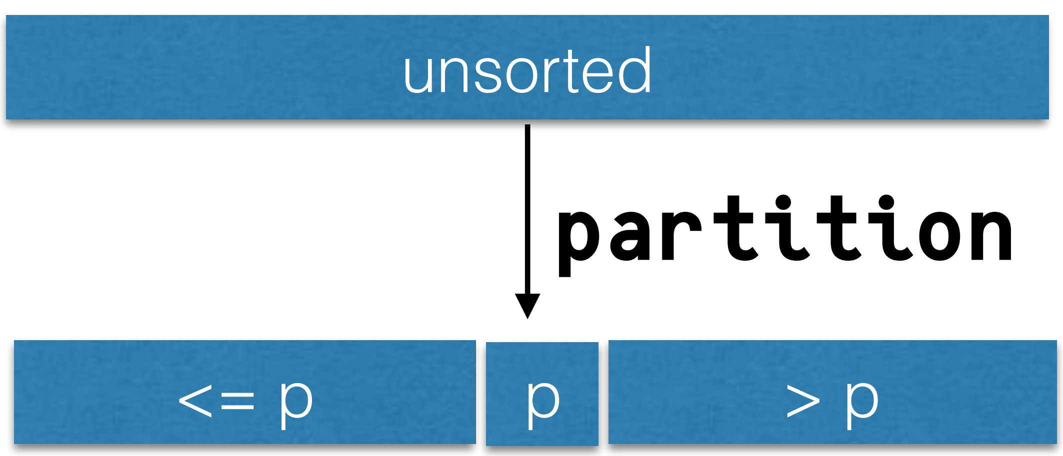
even split (good!)



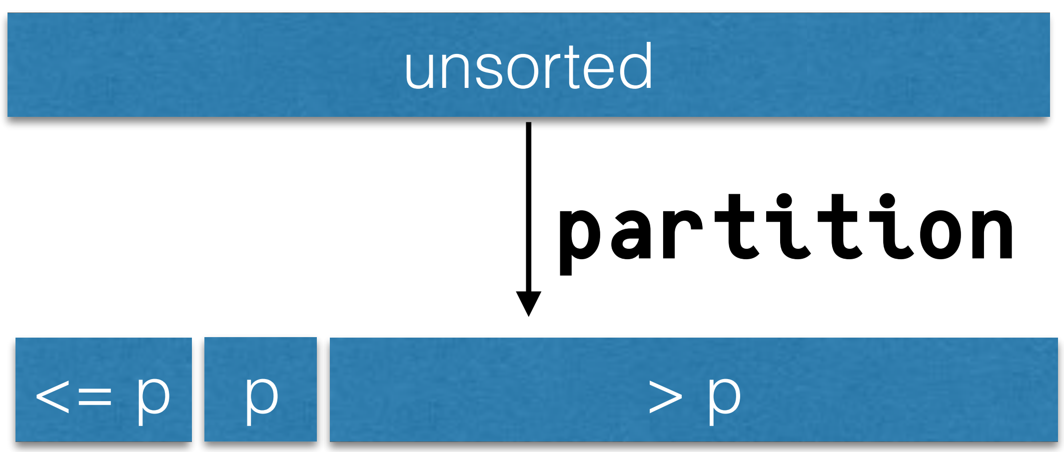
uneven split (bad!)

Time complexity (2)

- in the worst case, every split is uneven -> O(n^2) worst-case runtime.



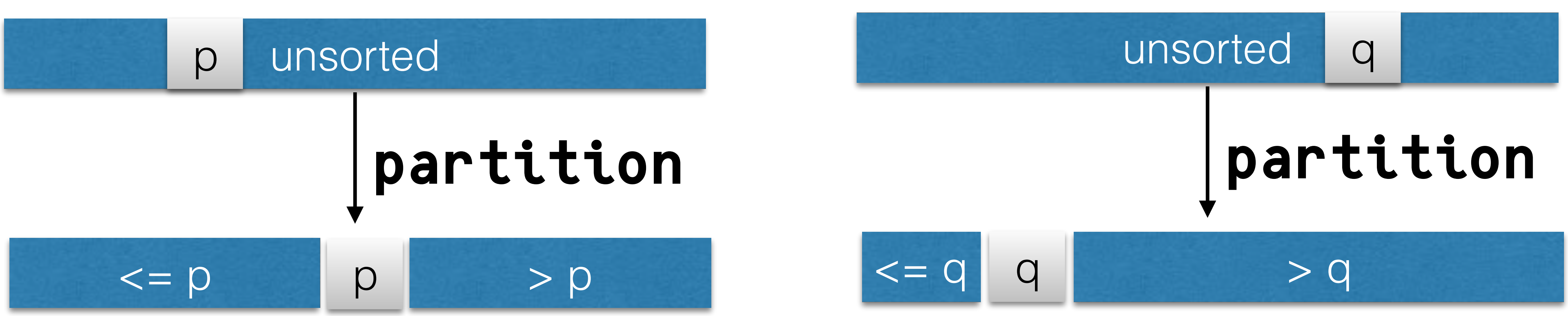
even split (good!)



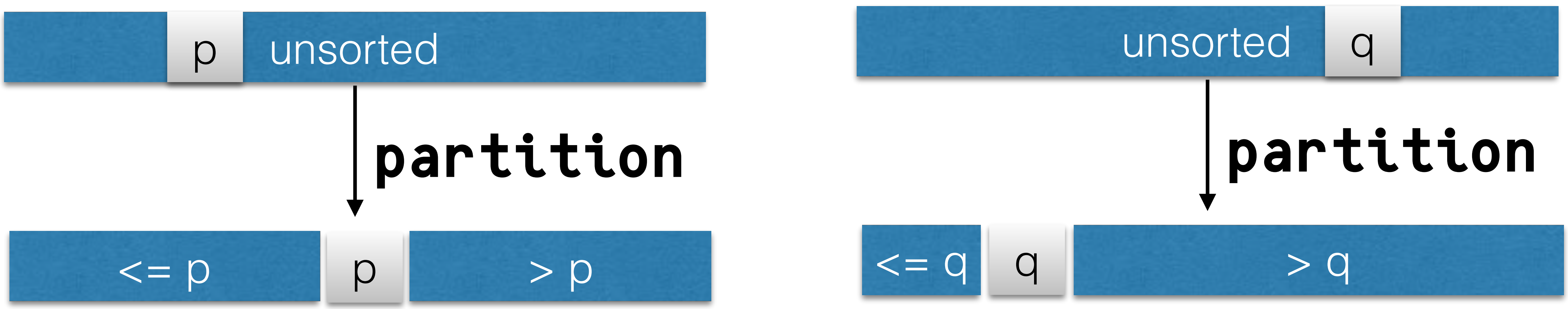
uneven split (bad!)

Randomized quick sort

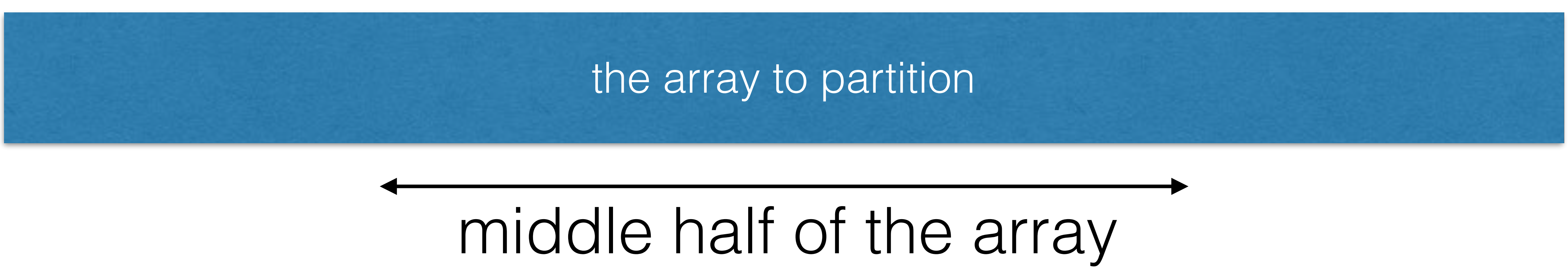
- instead of always choosing the first (or middle) element as the pivot,
- choose an element in a random position as pivot



Randomized quick sort

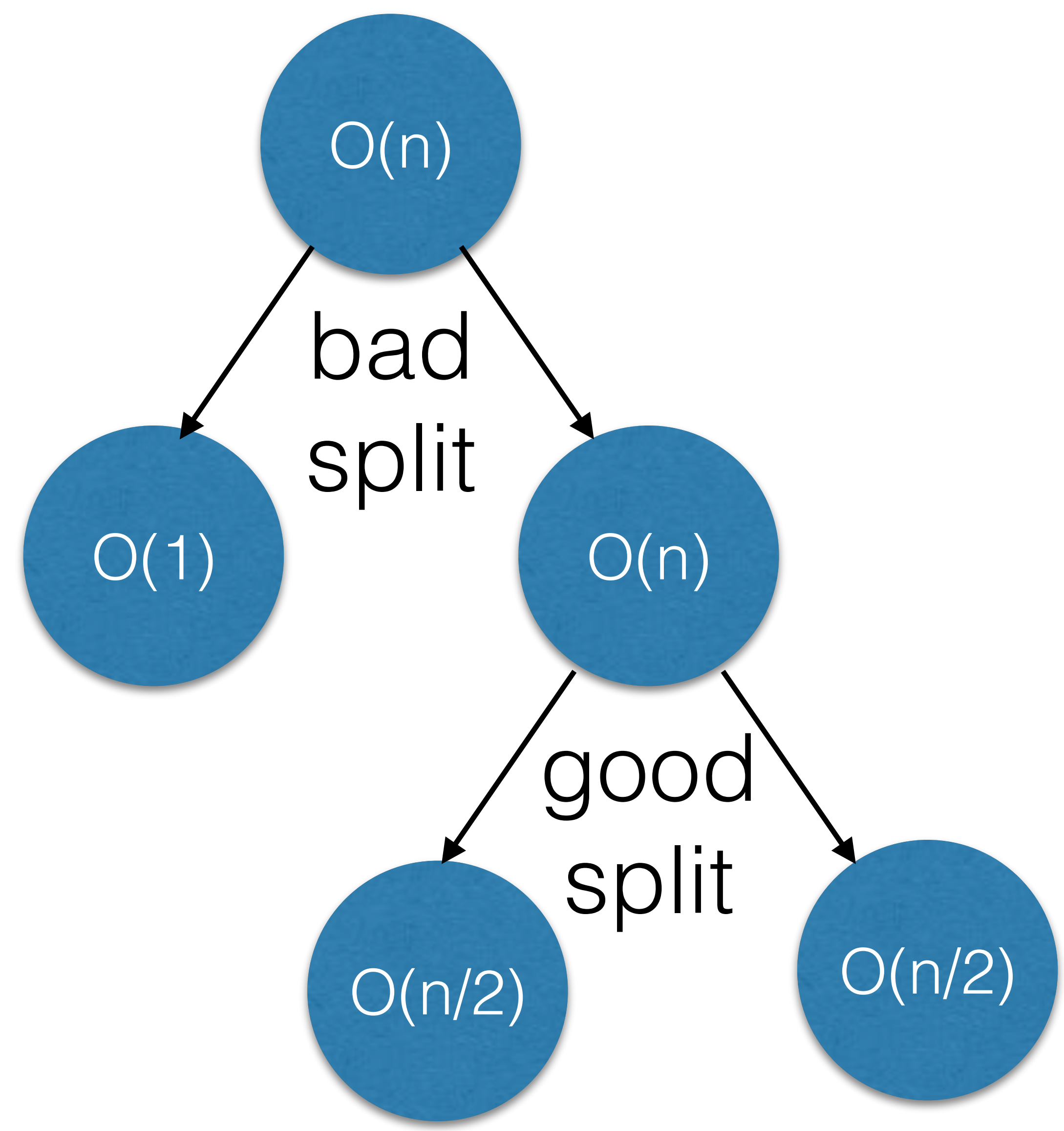


- with probability 1/2, a randomly chosen pivot will end up in the middle half of the array



How many good splits?

- with probability 1/2, partition() gives a good split
- with probability 1/2, partition() gives a bad split
- on expectation, after 2 partition(), we have a good split



Randomized quick sort analysis

- O(n) work per level
- 2 log(n) levels (twice because of the bad splits)
- total expected time:
- **2n log n**