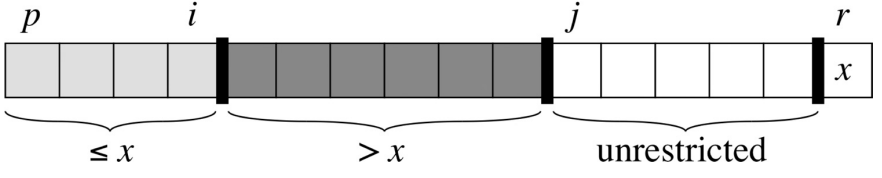
Quicksort

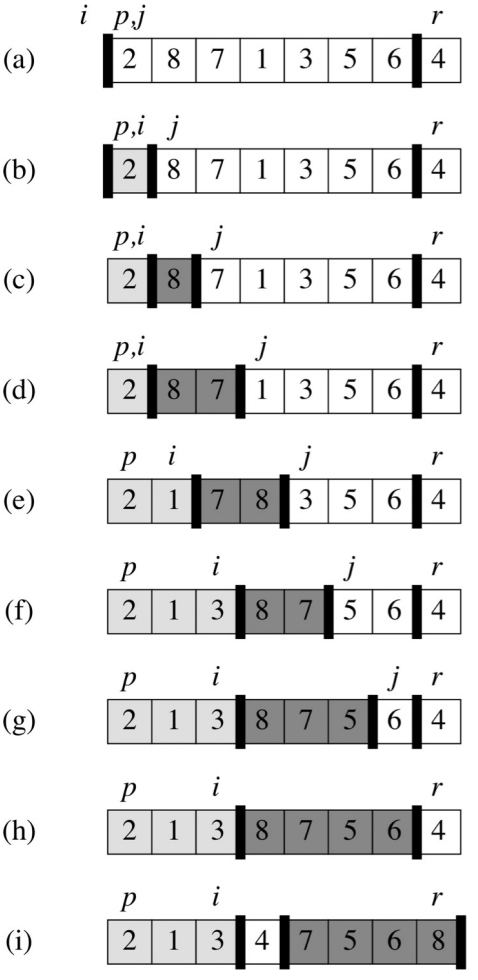
- Choose an element to be the pivot element, then partition the array so that the elements that are $\leq$ pivot are to the left of the pivot, and the elements $>$ pivot to the right of the pivot
- Recursively repeat the process on the subarrays to the left and right of the pivot
- After partitioning, the pivot will be in its final location

- Base cases
  - Subarray of size 1 - the one element must be the pivot in its proper place
  - Subarray of size 0 - nothing to do

```
PARTITION(A, p, r)
  x = A[r]
  i = p − 1
  for j = p to r − 1
      if A[j] ≤ x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1
```

$\Theta(n)$





Where is the work?

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

- The key contributor to the runtime is the loop in partition
- Each iteration of that loop is a comparison with the pivot
- Asyptotically, the runtime is based on the number of comparisons

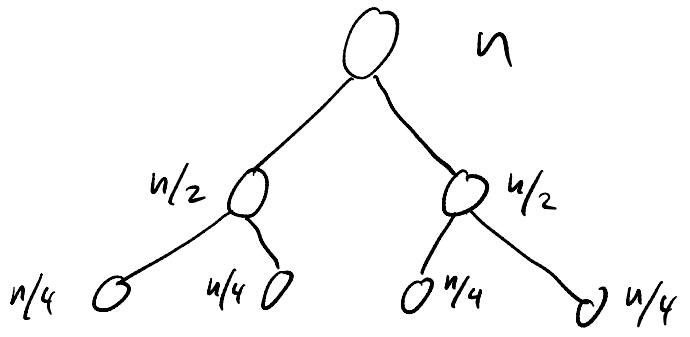
Best Case

- The loop in partition has $n-1$ iterations, which compare each element to the pivot, so $\Theta(n)$
- If the pivot produces the most even split possible, each recursive call will operate on no more than $n/2$ elements

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$
$$= \Theta(n \lg n)$$

- Every element is only ever compared with the pivot, so elements from different subarrays will never be compared after the split

Recursion tree



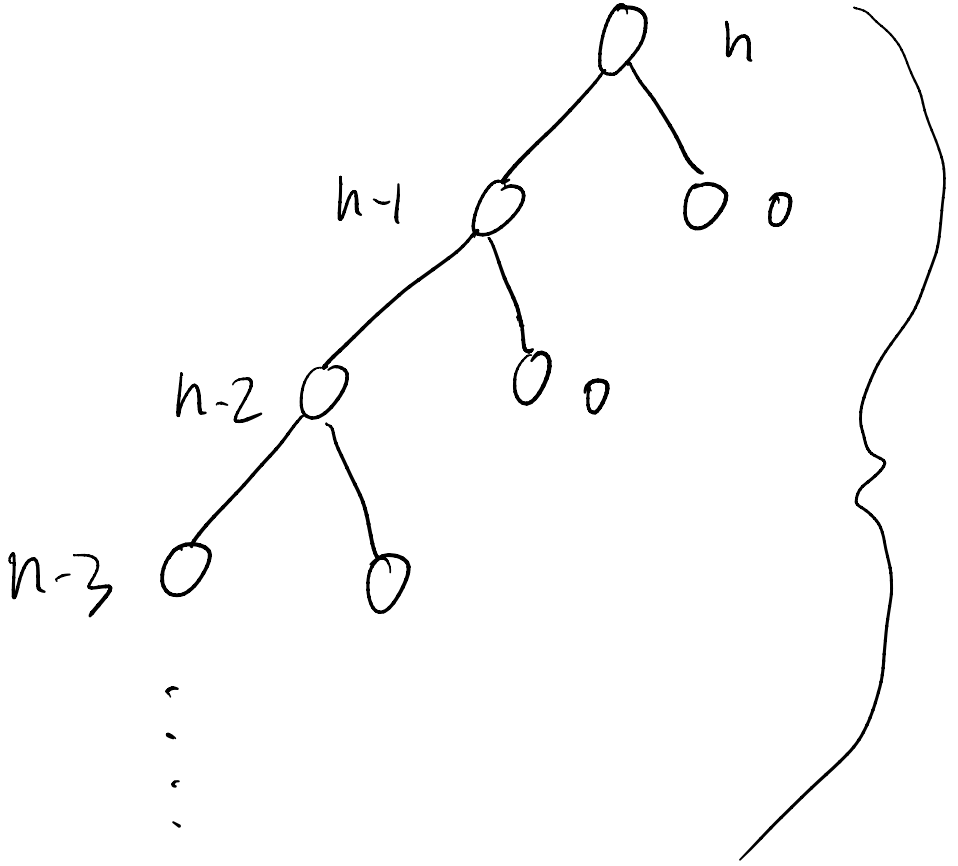
height is
$\Theta(\log_2(n))$

$\Theta(n)$ comparisons
at each level

$\Theta(n \text{ by } n)$

Worst Case

- The pivot is always either the largest or smallest element
- After partitioning, the new subarrays have sizes $n-1$ and 0
- We lose the advantage of avoiding future comparisons across the pivot
- Occurs when the list is already sorted, or if all elements are equal
    - Hoare's partition scheme avoids the problem of duplicates (see problem 7-1)



height is

$\Theta(n)$

# of comparisons is

$n + (n-1) + (n-2) + \cdots$

$= \frac{n(n+1)}{2} = \Theta(n^2)$

Randomizing

- What if we randomly select the pivot?
- Selecting the worst pivot every time is extremely unlikely
- Occasionally selecting the worst pivot is not so bad
- As long as there is at least one element on either side of the split we get by behavior, but possibly with a base smaller than 2
- Expected running time of randomized quicksort is $O(n \lg n)$

- In practice
  - Insertion sort is faster for small arrays
  - Common practice is to switch to insertion sort for subarrays smaller than some threshold