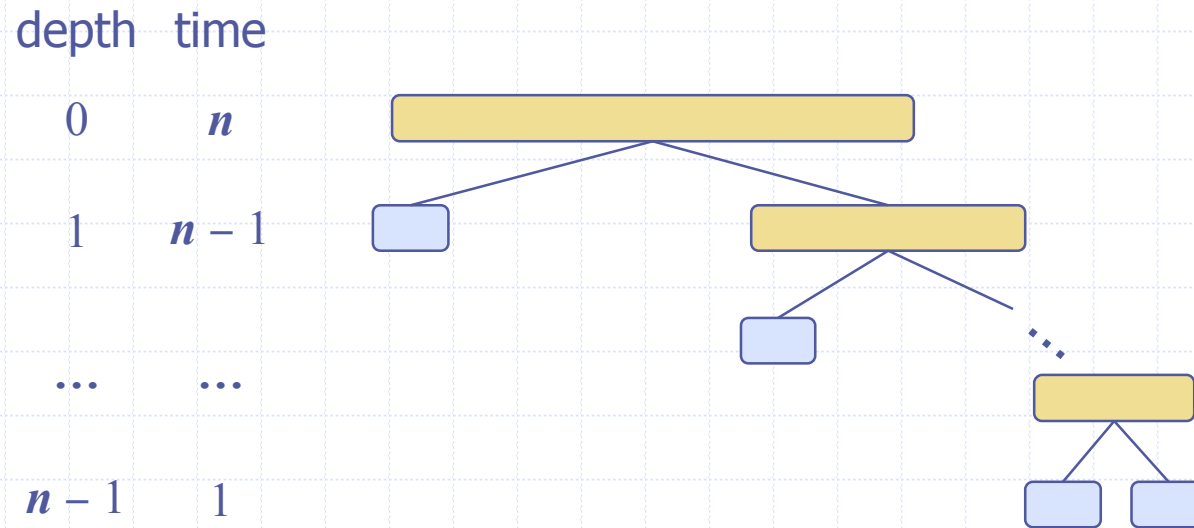


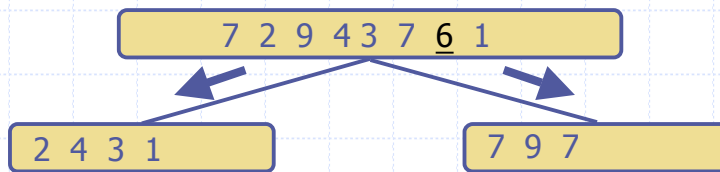
# Worst-case Running Time

- ◆ The worst case for quick-sort occurs when the pivot is the unique minimum or maximum element
- ◆ One of  $L$  and  $G$  has size  $n - 1$  and the other has size 0
- ◆ The running time is proportional to the sum
$$n + (n - 1) + \dots + 2 + 1$$
- ◆ Thus, the worst-case running time of quick-sort is  $O(n^2)$

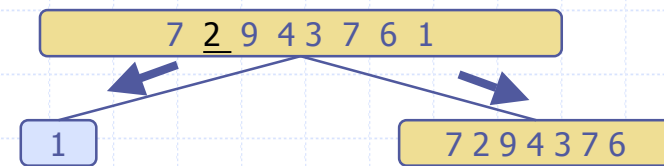


# Expected Running Time

- ◆ Consider a recursive call of quick-sort on a sequence of size  $s$ 
  - **Good call:** the sizes of  $L$  and  $G$  are each less than  $3s/4$
  - **Bad call:** one of  $L$  and  $G$  has size greater than  $3s/4$

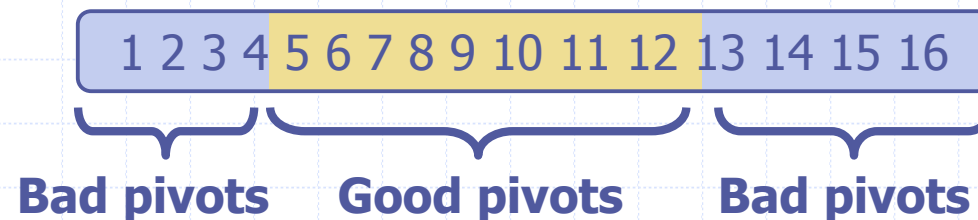


**Good call**



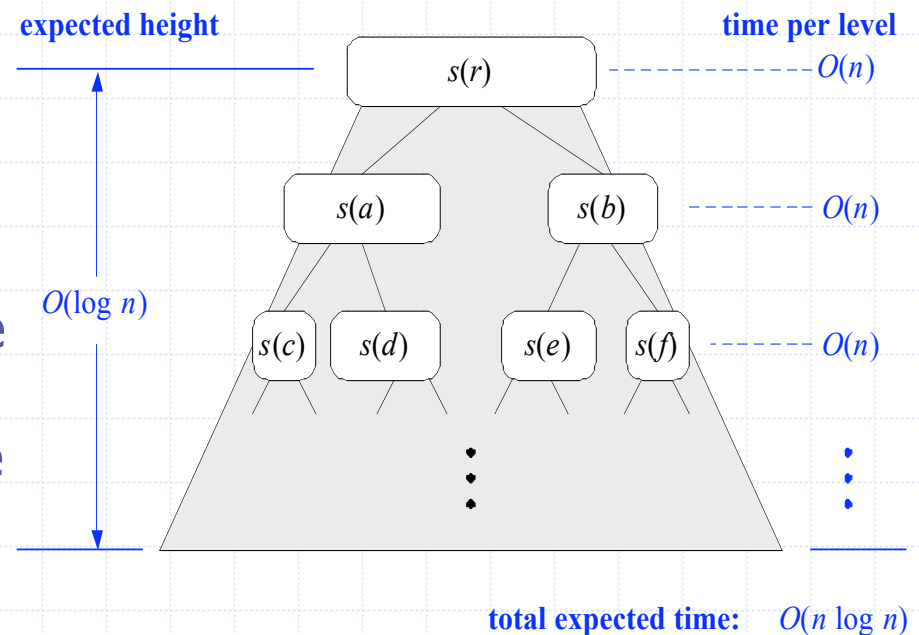
**Bad call**

- ◆ A call is **good** with probability  $1/2$ 
  - $1/2$  of the possible pivots cause good calls:



# Expected Running Time, Part 2

- ◆ **Probabilistic Fact:** The expected number of coin tosses required in order to get  $k$  heads is  $2k$
- ◆ For a node of depth  $i$ , we expect
  - $i/2$  ancestors are good calls
  - The size of the input sequence for the current call is at most  $(3/4)^{i/2}n$
- ◆ Therefore, we have
  - For a node of depth  $2\log_{4/3}n$ , the expected input size is one
  - The expected height of the quick-sort tree is  $O(\log n)$
- ◆ The amount of work done at the nodes of the same depth is  $O(n)$
- ◆ Thus, the expected running time of quick-sort is  $O(n \log n)$

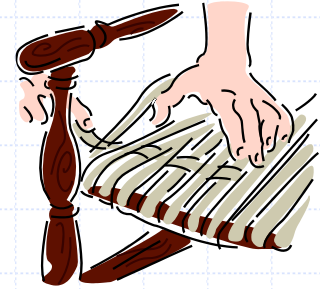


# In-Place Quick-Sort



- ◆ Quick-sort can be implemented to run in-place
- ◆ In the partition step, we divide into two non-empty parts (less than or equal to pivot and greater than or equal to pivot).
- ◆ Recursively divide each part into two smaller parts, etc.

# In-Place Partitioning



- ◆ Perform the partition using indices  $lo$  &  $hi$  to split  $S$  into two non-empty subsets  $\leq pivot$  and  $\geq pivot$ .

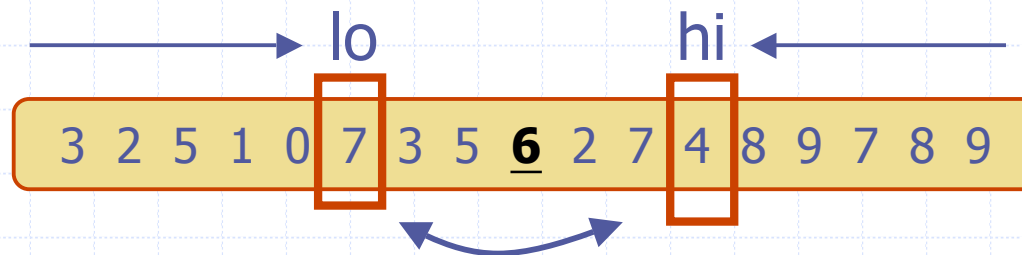
$lo$

$hi$

3 2 5 1 0 7 3 5 6 2 7 4 8 9 7 8 9

(pivot = 6)

- ◆ Repeat until  $j$  and  $k$  cross:
  - Scan  $lo$  to the right until find an element  $\geq x$ .
  - Scan  $hi$  to the left until find an element  $\leq x$ .
  - Swap elements at indices  $lo$  and  $hi$



# Java Code

```
static void qSort( int a[], int lo0, int hi0 ) {
    int lo = lo0;
    int hi = hi0;
    int mid = ( lo + hi ) / 2 ;
    int pivot = a[mid];

    // partition array in-place into low and high parts
    while( lo <= hi ) {
        while(( lo < hi0 )&&( a[lo] < pivot )) {
            lo++;
        }
        while(( hi > lo0 )&&( a[hi] > pivot )) {
            hi--;
        }
        if( lo <= hi ) {
            swap( a, lo, hi );
            lo++;
            hi --;
        }
    }
    // sort the left and right parts recursively
    if( hi > lo0 ) {
        qSort( a, lo0, hi );
    }
    if( lo < hi0 ) {
        qSort( a, lo, hi0 );
    }
}
```