Then $L + [\text{pivots}] + R$ is sorted since each entry of L is less than the pivot and each entry of R is greater than the pivot.

Let's analyze worst case behavior by recurrence relations: Let $T(n)$ be the runtime of quicksort on a list of length n. Suppose we have the worst input. Then we have

$$T(n) = O(n) + T(n-1)$$

comparison in the first iteration (when the list has length n). In the next iteration we have

$$T(n-1) = O(n-1) + T(n-2),$$

and so on.

$$T(n) = O(n) + T(n-1) = O(n) + O(n-1) + T(n-2).$$
$$= \sum_{i=1}^{n} O(i). = O(n^2).$$

What if <u>most</u> inputs had good pivots — that is, pivots were always roughly the median? Then we would expect the following timing:

$$T(n) = n + 2 \cdot T(\tfrac{n}{2}).$$
$$= n + 2\left(\tfrac{n}{2} + 2T(\tfrac{n}{4})\right) = 2n + 4T(\tfrac{n}{4})$$

$$= 2n + 4\left(\tfrac{n}{4} + 2\cdot T\left(\tfrac{n}{8}\right)\right) = 3n + 8\cdot T\left(\tfrac{n}{8}\right).$$

If $n = 2^m$, then we expect:

$$T(n) = m\cdot n + 2^m \cdot T\left(\tfrac{n}{2^m}\right)$$
$$= (\log n)\cdot n + n\cdot T(1)$$
$$= n\log n + n$$

Thus, the algorithm would run in time

$$O(n\log n).$$

This is the average-case complexity, assuming a uniform distribution of the entries. We won't say more about this.

Merge Sort.

The average case of quick sort has the best complexity we've seen so far. What made it so successful? $\boxed{\text{Splitting into 2.}}$ ~~Mer~~ Merge Sort builds this into the core algorithm. (Note the pivot is not always the median.)

Let's forget the pivot and just chop the list into two (roughly) equal ~~list~~ pieces. We'll sort the pieces individually, like QS, but when we put them together, we need to be careful!!

We cannot assume that

L, R : two pieces from original.

$$SORT(L) + SORT(R)$$

would yield a sorted list because we're just cutting down the middle.

Ex: $A = [2, 4, 3, 1]$

$L = [2, 4]$, $R = [3, 1]$.

Not sorted!

$$SORT(L) + SORT(R) = [2, 4; 1, 3]$$

Instead we just merge two queues based on the values.

$SORT(L) = [2, 4]$ $SORT(R) = [1, 3]$.

check first entries: $1 < 2$

$$MERGED = [1]$$

Then we do it again with $[2, 4]$, $[3]$

Now $2 < 3$ so

$$MERGED = [1, 2].$$

Do it again with $[4]$, $[3]$.

Now $3 < 4$ so

$$MERGED = [1, 2, 3].$$

Now one of lists is empty, so we'll just extend by ~~the~~ nonempty list: $[4]$

$$MERGED = [1, 2, 3, 4].$$

This is merge sort! (code)

What does the worst input look like for MS?

Well, it will force the largest number of comparisons during the ~~ee~~ merging stage. Let's assume both L and ~~[crossed out]~~ R have length n. ~~4~~

- At best we have only n ~~cke~~ comparisons.

  Ex: every entry of L is less than every entry of R. More specifically L[-1] < R[0]

- At worst we have 2n-1 comparisons.

Both are $\Theta(n)$, so in terms of complexity we should ~~not~~ expect average-case and worst-case to ~~be~~ have ~~the~~ the same complexity.

So what is the worst case? Suppose now that the original list has length n. Then the time is

$$T(n) = \underbrace{2\left(\tfrac{n}{2}\right) - 1}_{\text{merging}} + \underbrace{2T\left(\tfrac{n}{2}\right)}_{\text{sorting both } L \text{ and } R}$$

$$= n - 1 + 2T\left(\tfrac{n}{2}\right)$$

$$= n - 1 + 2\left(\tfrac{n}{2} - 1 + 2T\left(\tfrac{n}{4}\right)\right)$$

$$= 2n - 3 + 4T\left(\tfrac{n}{4}\right)$$

$$= 2n - 3 + 4\left(\tfrac{n}{4} - 1 + 2T\left(\tfrac{n}{8}\right)\right)$$

$$= 3n - 7 + 8T\left(\tfrac{n}{8}\right).$$

If $n = 2^m$, we have:

$$T(n) = m \cdot n - 2^m + 1 + 2^m \cdot T\left(\frac{n}{2^m}\right)$$
$$= mn - 2^m + 1 + 2^m$$
$$= mn + 1 = O(m n) = O(n \log n).$$

What about when n is <u>not</u> a power of 2? Many ways to think about this.

<u>Ex:</u> $n=6$ $\qquad\qquad$ $n=8$

	$n=6$	$n=8$
n	[1, 2, 3, 4, 5, 6]	[1,2,3,4,5,6,7,8]
$n/2$	[1,2,3] [4,5,6]	[1,2,3,4] [5,6,7,8]
$n/4$	[1,2] [3] [4,5] [6]	[1,2] [3,4] [5,6] [7,8]
$n/8$	[1] [2] [4] [5]	[1] [2] [3] [4] [5] [6] [7] [8]

Since we're using big-O, we are only after upper-bounds. Thus let $m = \log_2 n$ and round up, so

$$n \leq 2^m$$

Then the runtime for lists of length n is

$$O(m \cdot 2^m) = O(n \log n).$$

To summarize

Selection Sort : $O(n^2)$
Bubble Sort : $O(n^2)$
Quick Sort : $O(n^2)$
Merge Sort : $O(n \log n)$

compare bubble and merge

40