Run Time?

At each level, total merge time is $O(n)$.

# levels? $O(\log n) = \# \text{times we can split list of size } n \text{ in } \frac{1}{2} \text{ until we get lists of size 1}$

Total: $O(n \log n)$. 
<Skip>

Our implementation (of Merge) not in-place.

Stable? Yes, ordering among equal elements is preserved.

Example of Divide-and-Conquer (and Combine) algorithm.

Divide: problem into smaller sub-problems (Split list in 1/2)
Conquer: (Solve) sub-problems (Merge)
Combine: (Apply to all sublists)
Quicksort: Divide-and-conquer-and-combine

Choose an element \( v \) (pivot), partition the list around \( v \):

\[ \ll v \rr \]

\[ \ll v \rr \]

Now recursively do the same on left and right sides.

Eventually, entire list will be sorted.

Quicksort (List A)

1. Divide - choose element \( v \) (pivot) and partition \( A \) into:
   - \( L \): elements \( \leq v \)
   - \( G \): \( \geq v \)

2. Conquer: Quicksort \( L \) and \( G \)

3. Combine: join \( L \) and \( G \)

Questions? (1) pivot? (2) partition?
Questions:

1. How to choose pivot?
2. How to partition?

How to Choose pivot?

worst-case: Pivot is always largest or smallest.

Each partition step partitions around only one element (pivot).

RunTime?
Choosing Pivot:
- smallest/largest element bad. Why?
  - only one element (i.e. the pivot) will get sorted
    in each partition (RunTime in this case?)
- random: expensive to generate
- median of 3: choose the median of leftmost, middle, rightmost elements.

ex: 20 5 37 61 15 11 59 12 48 1

\[ \text{MedOf3}(20, 15, 1) = 15 \]

Partition \((A, \text{first}, \text{last}, \text{pivot})\):
(1) swap pivot with last element
(2) \(i\) points to first element
(3) \(j\) "" element before pivot
(4) while \(i\) and \(j\) have not crossed:
   * move \(i\) right, move \(j\) left until
     \(A[i] > \text{pivot} \) and \(A[j] < \text{pivot} \)
   * swap \(A[i], A[j]\)
(5) swap pivot with \(A[i] \)
(6) return \(i\)
ex: 26 5 37 61 15 11 59 12 48 1

\[ \text{mof3}(26, 15, 1) = 15 \quad \text{(Move pivot out of the way)} \]

26 5 37 61 1 11 59 12 48 15

\[ i \quad j \leftarrow j \]

12 5 37 61 1 11 59 26 48 15

\[ i \leftarrow i \quad j \leftarrow j \]

12 5 11 61 1 37 59 26 48 15

\[ i \quad j \]

12 5 11 1 61 37 59 26 48 15

\[ j \quad i \]

Done since \( j < i \)

\text{swap pivot w/ A[i:j]}

\[ \underbrace{12 5 11 1 15 37 59 26 48 61} \]

\[ \text{Qsort}(\quad) \quad \text{Qsort}(\quad) \]
Partition $(A, \text{first}, \text{last}, \text{pivot})$  // Partitions $A$ around pivot

    swap pivot and $A[\text{last}]$
    $i = \text{first}$
    $j = \text{last} - 1$
    loop = true
    while (loop) {
        while ($A[i] \leq \text{pivot}$) $i++$
        while ($A[j] > \text{pivot}$) $j--$
        if ($i < j$)  // $i$ and $j$ have not crossed
            swap $A[i], A[j]$
        else  // $i$ and $j$ have crossed
            loop = false
        swap pivot with $A[i]$
    }

$T(n) = O(n)$

cutoff  - For small list ($\leq 3$) use insertion sort.

QuickSort $(A, \text{first}, \text{last})$  // Sorts $A$ with cutoff of 3
if ($\text{last} - \text{first} < 3$)  // $|A| \leq 3$
    InsertionSort ($A$)
else  // $|A| > 3$
    pivot = $\text{med3} (A, \text{first}, \text{last})$
    split_point = Partition $(A, \text{first}, \text{last}, \text{pivot})$
    QuickSort $(A, \text{first}, \text{split-point} - 1)$
    QuickSort $(A, \text{split-point} + 1, \text{last})$
RunTime Quicksort?

average-case: pivot always roughly halves the list

\(O(\log n)\) recursive calls, each takes \(O(n) \Rightarrow O(n \log n)\)

worst-case: If pivot is always smallest or largest value, \(O(n)\) recursive calls \(\Rightarrow O(n^2)\)

best-case: also \(O(n \log n)\) (sorted list, perfect pivot doesn't help).

cut-off: For small lists \((n \leq 3)\), use insertion sort.

Insertion, merge, quick sorts are examples of general sorting algorithms.

ex: \(A\) is list of \(n\) integers between 0 and \(m\):

\(0 \leq A[1], A[2], \ldots, A[n] < m\)

ex: \(M = 10\)

\[3\ 1\ 4\ 1\ 5\ 9\ 2\ 5\]

\[0\ 2\ 1\ 1\ 1\ 2\ 0\ 0\ 0\ 1\]

Sorted list: 1 1 2 3 4 5 5 9