Sorting Algorithms (ascending order)

Algorithm - set of steps that solve a problem

Insertion sort - most intuitive, simplest, how most of us would sort deck of cards

Insertion-Sort (L[0..n-1]) // Sorts list L

For (i = 1 .. n-1)

move L[i] left until (in correct sorted position)

L[i] is before all elements that are > L[i]

ex: 2 6 5 3 7 1 6 1 1 5 9

5 2 6 3 7 1 6 1 1 5 9

5 2 6 3 7 1 6 1 1 5 9

5 2 6 3 7 1 6 1 1 5 9

5 2 6 3 7 1 6 1 1 5 9

5 2 6 3 7 1 6 1 1 5 9

1 5 2 6 3 7 1 6 1 1 5 9

1 5 2 6 3 7 1 6 1 1 5 9

1 5 2 6 3 7 1 6 1 1 5 9

1 5 1 2 6 3 7 6 1 1 5 9

1 5 1 2 6 3 7 6 1 1 5 9

1 5 1 2 6 3 7 6 1 1 5 9

1 5 1 2 6 3 7 6 1 1 5 9

1 5 1 2 6 3 7 6 1 1 5 9

1 5 1 2 6 3 7 6 1 1 5 9
Run Times:

**Worst-case?** For each element, must compare with all previous elements \( \Rightarrow O(n^2) \).

When? reverse sorted list.

**Average-case:** Also \( O(n^2) \). Roughly \( n \) comparisons for each element.

**Best-case?** already sorted list: \( O(n) \) for each element; first comparison always fails.

If nearly sorted, very fast.

Also useful for small lists.

---

*SKIP*

In-place sorting - uses constant amount of extra space

Stable sort - preserves input order of equal elements.

**Merge-Sort**

1. Split lists into small sublists
2. Merge small sublists to create larger (sorted) sublists
3. Repeat (2) until entire list is sorted
The bulk of the algorithm is merging. How to merge 2 sorted lists?

1 element sublists:

\[
\begin{array}{c}
37 & 26 & 15 \\
\end{array}
\]

Just compare every 2!

\[
\begin{array}{c}
26 & 37 & 18 & 15 \\
\end{array}
\]

Now, how to merge lists of size 2?

(Compare A to B, place smaller in new list C,)

increment corresponding ptr

Another sublist D:

\[
\begin{array}{c}
11 & 15 & 69 & 61 & 65 & 100 \\
\end{array}
\]

\[
\begin{array}{c}
15 & 26 & 37 & 59 & 61 & 65 & 100 \\
\end{array}
\]

What if one sublist larger than the other?

Just copy remaining items of larger sublist.
Merge(A, B)

Ap = ptr to list A, Bp = ptr to list B

while (Ap < A.length and Bp < B.length)

Compare A element to B element

Place smaller element in C, increment corresponding ptr

if Ap < A.length copy A elements into C.
else if Bp < B.length "B"
return C

Runtime: O(n)

// Sorts list L [start, end]

MergeSort(L, start, end) recursive calls

Notice: Base case: if start = end (list of size 1, do nothing)

if (start < end) // Similar to Binary Search

mid = (start + end)/2

mergeSort(L, start, mid)
mergeSort(L, mid+1, end)

L = merge(L[start..mid], L[mid+1..end])

Runtime:

Each merge takes at most O(n)

How many merges? in terms of n? logn

Total: O(nlogn)
Merge

MS(L, 0, 7)

1, 5, 11, 15, 26, 37, 59, 61 \[= [26 \ldots 15]\]

\[\frac{n}{2} \Rightarrow (\frac{n}{2})(2)\]

= n

MS(L, 0, 3) MS(L, 4, 7)

= \frac{n}{4} \Rightarrow (\frac{n}{4})(2)

\[= n\]

[1, 5, 26, 37] \[\leq [26 \ldots 1]\] [50 \ldots 15]

\[\leq\]

MS(L, 0, 1) MS(L, 2, 3)

[5, 26] \[\leq [26, 5]\] [37, 17]

\[\leq\]

\[\leq\]

MS(L, 0, 0) MS(L, 1, 1)

[26] [5] [37] [11, 61, 111, 59, 15]

14=1 no recursing (splitting)

Runtime:

Each merge step takes \(O(n)\).

\(\log(n)\) merges \(\Rightarrow \) # merges = # times can split list of size \(n\) in \(1/2\) until we get lists of size 1.

Total: \(O(n \log n)\).
Our implementation (of Merge) not in-place.
Stable? ordering among equal elements is preserved.

Example of Divide-and-Conquer (and Combine) Algorithm

Divide problem into smaller sub-problem
Conquer (solve) sub-problems
Combine solutions