Topological Sorting
Solution Plan

➪ **Source vertex**

➪ **Brute force approach (?)**

- ✎ Perform DFS
- ✎ Reverse pop-order if DAG otherwise no solution

Exercise
Outline (identify?) a basic procedure to perform on each iteration.

➪ **Source removal algorithm**

- ✎ Remove a source: a basic procedure
- ✎ Solution design pattern (next)

Quiz
How does instance size change each iteration?

Decrease-and-Conquer

Long story short: iterate on input (inst) size

Source removal topological sorting is an example of a decrease-conquer algorithm.

➪ Use solution to smaller instance

- ✎ Classic example: factorial
- ✎ Binary search (later)
- ✎ Exponentiation 🕮

Definition calls for n–1 multiplications, which is not needed to get a result.

➪ Solution style: where to start?

Attempt a smaller instance on each iteration until you reach one small enough to evaluate.

- ✎ Top-down: from bigger instance
- ✎ Bottom-up (later)

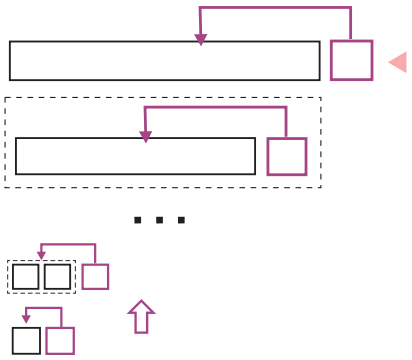
A Decrease-Conquer Sort

Only a simple procedure to insert key in sorted list is needed to relate instances n, n–1.

Strategy

Insert last element in remaining n–1 (hopefully) sorted sublist, repeat

May get lucky, complete sorting after just *O*(n) comparisons (when?). Generally will have to repeat on the n–1 sublist.



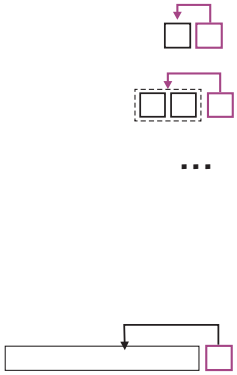
Quiz
What's the name of the returning phase of **recursion**?

Decrease-and-Conquer
Insertion Sort

➪ Bottom-up

Keep a sorted list at front of array, like a cards player would do in his hand as he draws cards from a pile! (Insert position highlighted.)

Pattern



89	**45**	68	90	29	34	17
45	**89**	**68**	90	29	34	17
45	68	89	**90**	29	34	17
45	68	89	90	**29**	34	17
29	**45**	68	89	90	**34**	17
29	34	45	68	89	90	**17**
17	29	34	45	68	89	90

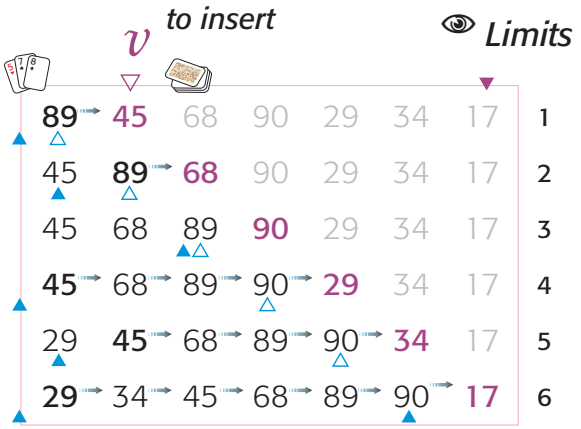
Decrease-and-Conquer Insertion Sort

Classic bottom-up

Algorithm *InsertionSort*
Input $\cdots\ A[0..n-1]\ \cdots$
Output

```
1: for i ← 1 to n − 1 do
2:     v ← A[i]
3:     j ← i − 1
4:     while j ≥ 0 and A[j] > v do
5:         A[j + 1] ← A[j]
6:         j ← j − 1
7:     A[j + 1] ← v
```

Exercise
Design a nonrecursive algorithm which keeps the sorted sublist near the end of the array (to the right of *v*). Print list after each round, compare to figures.



Quiz
How many key-comps are needed to insert in a sorted sublist (minor loop)? When would the worst and best cases happen? (Id an iteration number above to illustrate each).

Insertion Sort Performance

Quiz
Identify best and worst cases for insertion sort.

➪ Approach

✎ Worst-case, best-case

🕮 ✎ How well on average?

Quiz 🕮
Identify a scenario for a useful best-case insertion sort, how many key-comps needed? Why is it important?

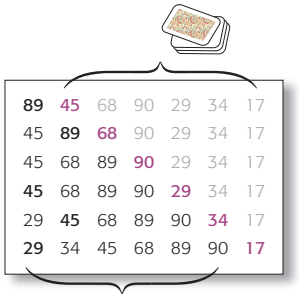
➪ Comparisons

89	45	68	90	29	34	17
45	**89**	68	90	29	34	17
45	68	**89**	90	29	34	17
45	68	89	90	29	34	17
29	**45**	68	89	90	34	17
29	34	45	68	89	90	17

89	45	68	90	29	34	**17**
17	45	68	90	**29**	34	89
17	29	68	90	45	**34**	89
17	29	34	90	**45**	68	89
17	29	34	45	90	**68**	89
17	29	34	45	68	90	**89**

Quiz
Compare key-comps patterns in insertion sort (above) and selection sort (below). Can we infer the worst-case C(n) of insertion sort? (Take a guess!).

Insertion Sort
Solution Pattern



Source removal produces a topological sorting of more verts by looking at a smaller instance each iteration, same as nonrecursive insertion sort.

Exercise
Write a pseudocode for a recursive insertion sort and code it to test. Print list after each insertion round. Hint: see Slide 3.

89 45 68 90 29 34 17
45 89 68 90 29 34 17
45 68 89 90 29 34 17
45 68 89 90 29 34 17
29 45 68 89 90 34 17
29 34 45 68 89 90 17

<u>Unsorted</u> list becomes smaller by 1 on each iteration, much like in topological sorting.