Insertion Sort and Merge Sort

- Why Sorting ?
- Insert Sort
- Merge sort (divide & conquer)
- Recurrence Solving

□ Why Sorting ?

- obvious: phone book
- Problems become easy once sorted

  e.g. Find a median

  $A[0:n] \longrightarrow B[0:n]$ median $= B[n/2]$

  Binary Search: look for specific item k

  $A[0:n] \rightarrow B[0:n]$, compare k to $B[0:n/2]$

- Not so obvious application

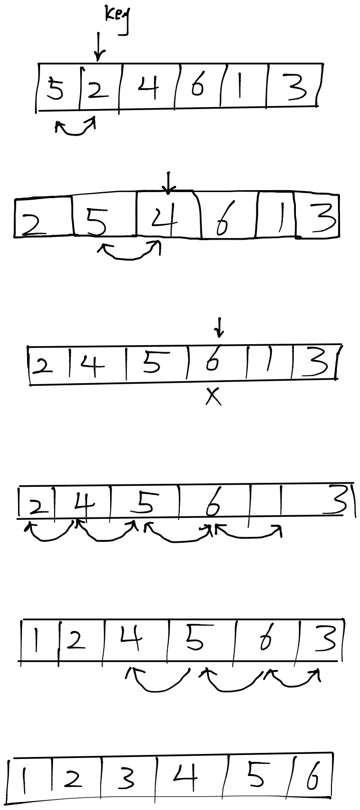
  data compression

  Computer Graphics: front to back rendering

Inscertion Sort

For $i = 1, 2, \ldots n$:

insert $A[i]$ into sorted array $A[0:i-1]$ by pairwise swaps down to the correct position e.g.



increasing order

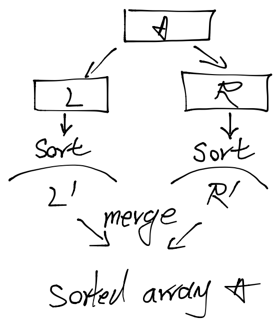
{ $\Theta(n)$ steps (key positions)
each step is $\Theta(n)$ swaps / compares

$\Rightarrow$ $\Theta(n^2)$ Algorithm if Comparision is as expensive as swaps

If Comparision much more expensive than swaps, do a binary seach on A[0:i-1] already sorted, then $\Theta(n \lg n)$ is the cost.

Binary search make over all insertion algorithm cost $\Theta(n \lg n)$ if doing compares, but still $\Theta(n^2)$ if doing swaps (have to insert, takes time)

□ Merge Sort (Divide & Conquer)



size n

2 arrays of size $n/2$

2 sorted arrays of size $n/2$

Sorted array of size n

Merge: Two <u>Sorted</u> arrays as input.

e.g.

L'	R'
20	12
13	11
7	9
2	1

two finger algo:

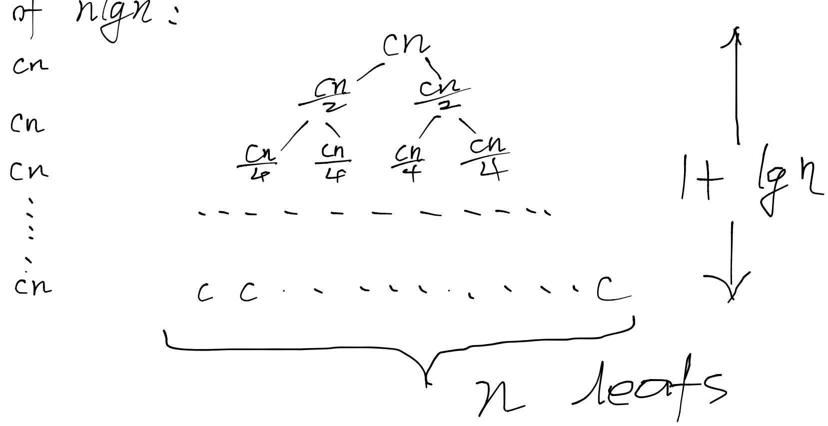
1 2 7 9 11 12 13 20

$\Theta(n)$ complexity

Complexity of merge sort:

$$T(n) = \underbrace{c_1}_{\text{divide}} + \underbrace{2T(n/2)}_{\text{recursion}} + \underbrace{c \cdot n}_{\text{merge}}$$

Proof of $n \lg n$:



$$T(n) = (1 + \lg n) \cdot cn = \Theta(n \lg n)$$

□ Compare Insertion and Merge Sort

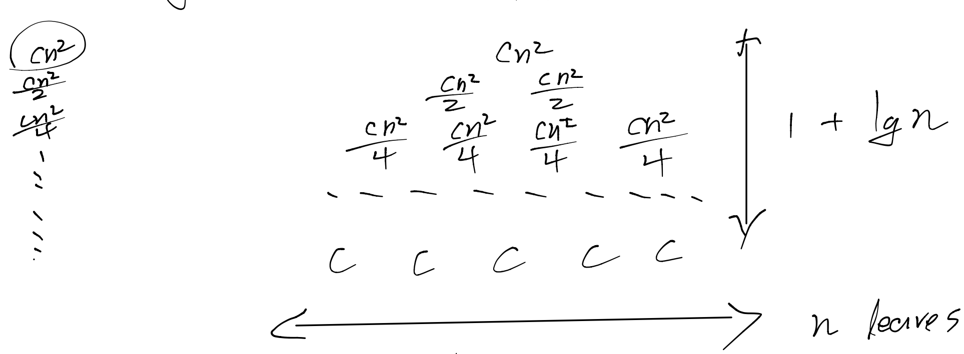
	Insertion Sort	Merge Sort
Cost	$\Theta(n^2)$	$\Theta(n \lg n)$
auxiliary memory	$\Theta(1)$ (in-place sort)	$\Theta(n)$ (copy elements)

* Merge sort in Python $= 2.2\, n \lg n$ (μs)
Insertion sort in python $= 0.2\, n^2$ μs
Insertion sort in C $= 0.01\, n^2$ μs

□ Recurrence Solving

recursion tree

e.g. $T(n) = 2T(n/2) + cn^2$



☆ All of the work done at the root !! (dominant)
$\Theta(n^2)$ Algorithm

e.g. $T(n) = 2T(n/2) + \Theta(1)$

$$
\begin{array}{ccccccccccccccc|c}
 & & & & & & & c & & & & & & & & c \\
 & & & c & & & & & & & & c & & & & 2c \\
 & c & & & & c & & & & c & & & & c & & 4c \\
 & \cdots & & & & & & & & & & & & \cdots & & \vdots \\
c & c & c & c & c & c & c & c & c & c & c & c & c & c & & nc
\end{array}
$$

dominates!

* all of the work done by the leaves