Efficiency of selection sort and insertion sort

Want $T(n)$: "time" taken on inputs of size n

Counting comparisons

Selection Sort

Extract minimum

Insertion Sort

Insert into a sorted list

$T(n) = T(n-1) + \underbrace{\text{Set up / Combination cost}}_{\text{"approx" } n \text{ steps}}$

Worst Case Analysis

$O(n)$ is "approximately" n

$f(n) = O(g(n))$ if $\exists$ constant k

s.t. $\forall n \geq 0 \quad f(n) \leq k \cdot g(n)$

?

$f(n) = 6n$ $g(n) = n$

$f(n) = O(g(n))$, choose $k \geq 6$

$f(n) = 6n^2 + 3n + 821$ $g(n) = n^2$

$k = 6 + 3 + 821$ $k \cdot g(n) = 6n^2 + 3n^2 + 821n^2$

Ignore n, constant term in $f(n) = an^2 + bn + c$

Insertion sort & Selection Sort

$$T(n) = T(n-1) + O(\cancel{n}) \quad n$$
$$T(0) = 1$$

Expand:

$$T(n) = \underline{T(n-1)} + n$$
$$= \underline{T(n-2) + (n-1)} + n$$
$$= T(0) + 1 + 2 + \dots + n$$
$$= 1 + 1 + 2 + \dots + n = 1 + \sum_{i=1}^{n} n$$
$$= 1 + \frac{n(n+1)}{2} = O(n^2)$$

$$\left.\begin{array}{ccccccc} 1 & 2 & 3 & \cdots & n-2 & n-1 & n \\ n & n-1 & n-2 & \cdots & 3 & 2 & 1 \end{array}\right\} \text{2 copies}$$

$$\underbrace{n+1 \quad n+1 \quad n+1 \quad \cdots \quad n+1}$$

$$= \frac{n(n+1)}{2}$$

Input of size $10 \rightarrow 100$ steps

$10^4 \rightarrow 10^8$ steps

$10^6 \rightarrow 10^{12}$ steps

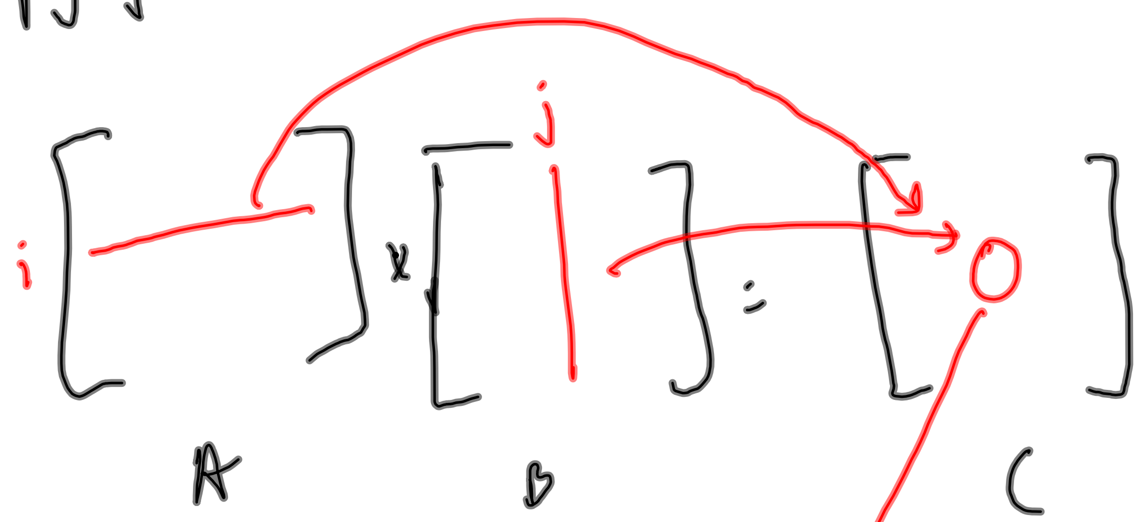
Any standard laptop or desktop

$\approx 10^{10}$ operations / sec

10^{12} operations $\longrightarrow$ 100 seconds

10^{7} input $\rightarrow$ 10^{14} 10000 seconds!

Multiplying 2 $n \times n$ matrices



n^2 values C_{ij}
each takes $O(n)$ steps

n^2 is not good enough

Better way to sort?

"Divide and conquer"

Suppose I can split the list into halves
and sort each half

Can I combine them efficiently?

[9, 3, 7, 1, 5, | 2, 10, 6, 8, 4]

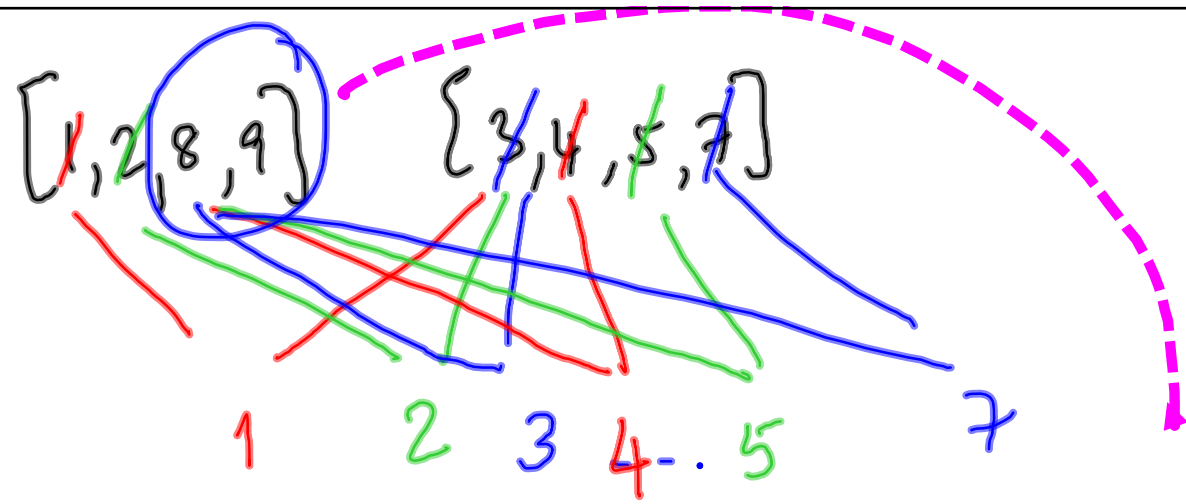
[~~1~~, ~~3~~, 5, 7, 9] [~~2~~, 4, 6, 8, 10]

?

Smallest overall must be the smallest of one of the subparts

[1, 2, 3 . . .

Need not alternate



"Merging" two sorted lists

Produce 1 value at a time

Each takes 1 comparison

n values to produce → $O(n)$ time

Merge sort

Split the list in two halves

Sort each half (inductively, use merge sort)

Merge the answers

Base Cases: $l = [\,]$ or $l = [x]$

Analysis

$$T(n) = \underline{2\,T(n/2)} + n$$

divide into 2 halves
+
merge answers

$$= 2\left(2\,T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= 2^2\,T\left(\frac{n}{2^2}\right) + 2n$$

$$T(n) = 2^2 T\left(\frac{n}{2^2}\right) + 2n$$

$$= 2^2 \left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

$$\vdots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + kn$$

$T(0) = T(1) = 1$

$\frac{n}{2^k} = 1 \longrightarrow T\left(\frac{n}{2^k}\right) = 1$

$\therefore$ if $k = \log_2 n$, $T\left(\frac{n}{2^k}\right) = 1$

After $\log_2 n$ expansions of $T(n)$

assume n was originally a power of 2

$T(n) = 2^{\overbrace{\log_2 n}^{k}} T(1) + (\underbrace{\log_2 n}_{k})\, n$

$= n \cdot 1 + n \cdot \log_2 n = O(n \log_2 n)$

$\therefore$ Mergesort takes time $O(n \log n)$ In C.S. base Is always 2

Useful shortcut:

$2^{10} \approx 1024$, $\log 1000 \approx 10$

	n^2	$n \log n$
$n = 1000$	10^6	10^4
$n = 10^6$	10^{12}	$10^6 \cdot (10+10) = 2 \times 10^7$
$n = 10^9$	10^{18}	$10^9 \cdot (30) = 3 \times 10^{10}$

```
def mergesort(l):
    if len(l) <= 1:
        return(l)
    else:
        mid = len(l)//2 + 1
        left = mergesort(l[:mid])
        right = mergesort(l[mid:])
        return(merge(left, right))
```

for any list l & any position j

l == l[:j]+l[j:]

OR return(merge(mergesort(l[:mid]),mergesort(l[mid:])))