SORTING

RECALL: SELECTION SORT

- go through list to find smallest value
- swap that value with value in first spot
- scan rest of list to find next smallest value
- swap that value with value in second spot
- continue with remaining spots for each position in the list


[1, 5, 0, 7, 3]
swap
[0, 5, 1, 7, 3]
after 1st pass
swap
[0, 1, 5, 7, 3]
after 2nd pass
swap
[0, 1, 3, 7, 5]
after 3rd pass
swap
[0, 1, 3, 5, 7]
after 4th pass

n = # of elements in the list

SELECTION SORT - COMPLEXITY

- Think about the work it does:
  - How many times does it go through the list?
  - For each pass, how many elements does it look at?

n - 1 passes

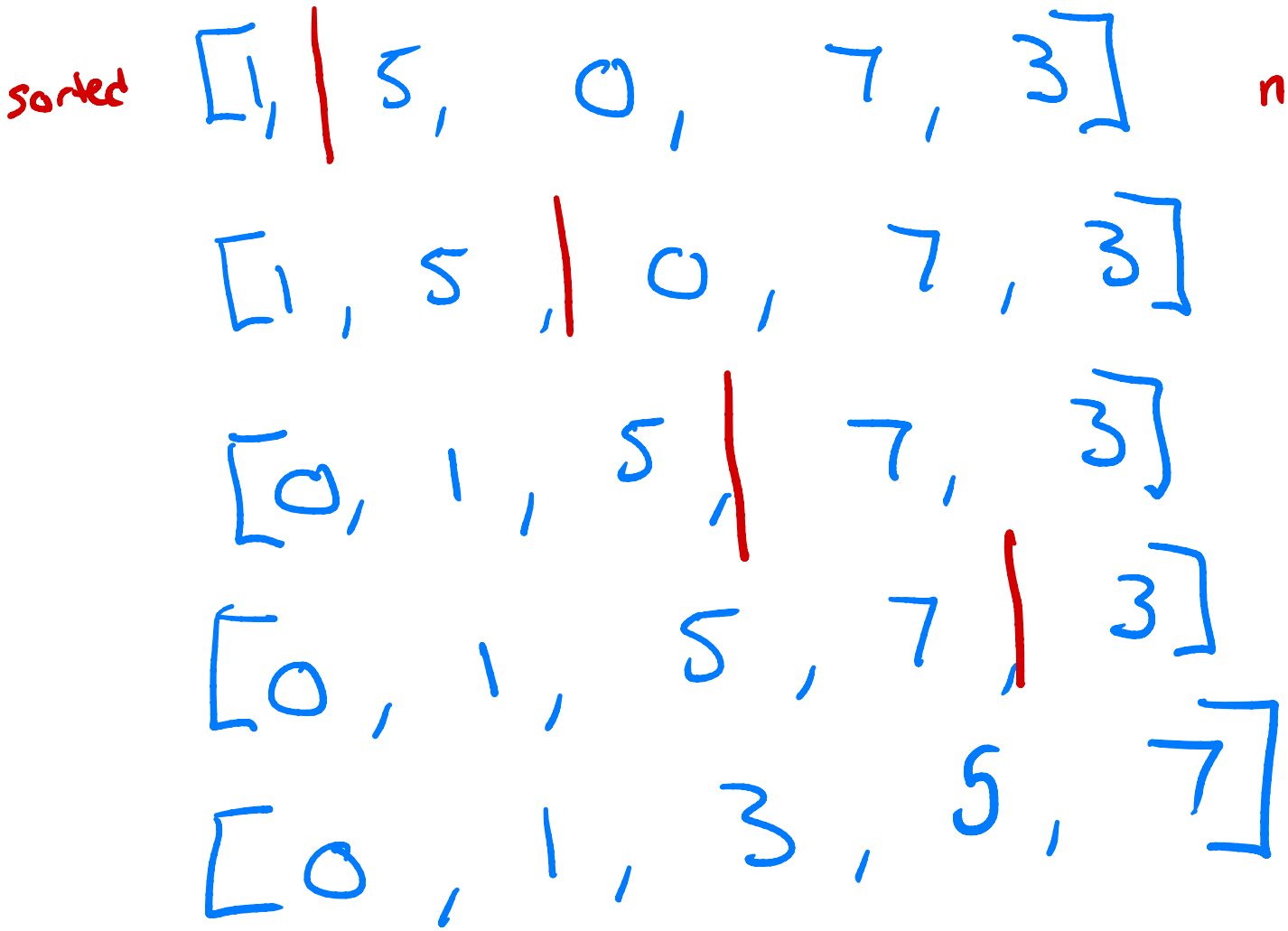
first pass: n
second pass: n - 1
third pass: n - 2
⋮

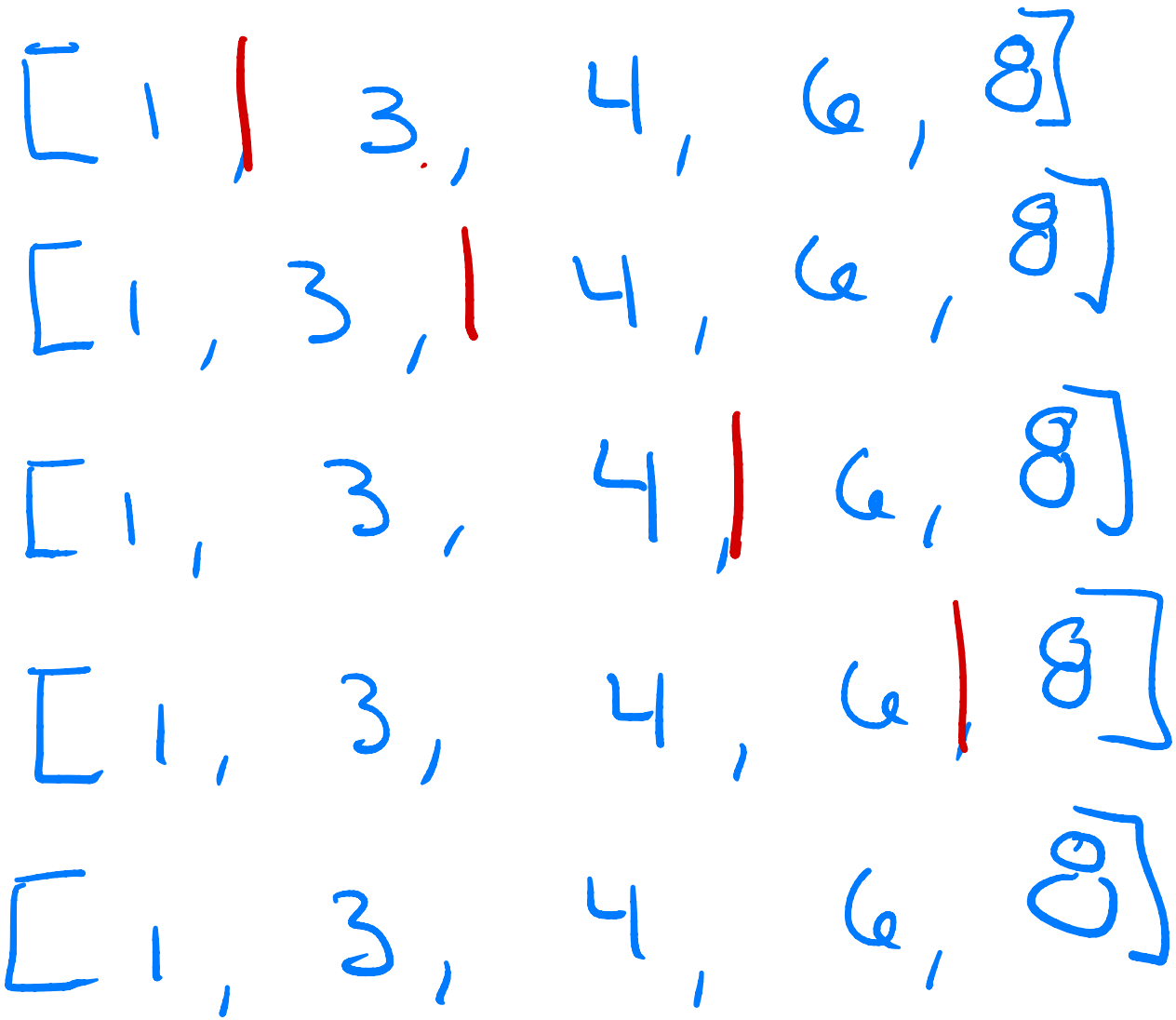
$n + n-1 + n-2 + \dots + 2 = n(n+1)/2 - 1 \rightarrow n^2$

$k + k-1 + \dots + 1 = k(k+1)/2$

RECALL: INSERTION SORT

- Breaks list into ~~to~~ two parts (sorted and not sorted)
- Goes through values
- For each value, inserts into the right spot in the sorted list
- Move onto next value, repeating process


sorted
[1, 5, 0, 7, 3]
not sorted
[1, 5, 0, 7, 3]
[0, 1, 5, 7, 3]
[0, 1, 5, 7, 3]
[0, 1, 3, 5, 7]


[1 | 3, 4, 6, 8]
[1, 3, | 4, 6, 8]
[1, 3, 4 | 6, 8]
[1, 3, 4, 6 | 8]
[1, 3, 4, 6, 8]

INSERTION SORT - COMPLEXITY

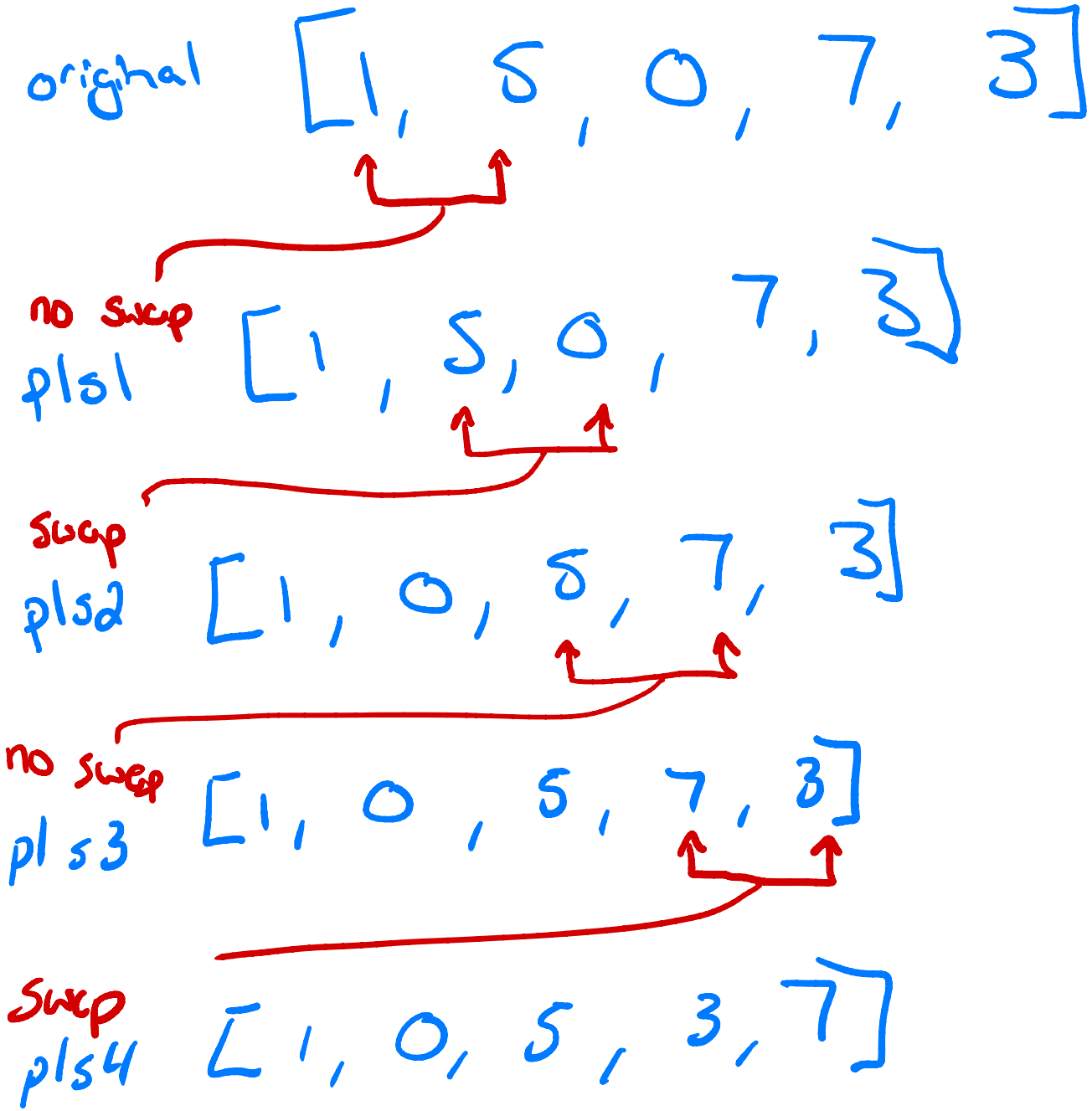
- How many times do we go look at a value?
- For each value, how much work does it take (how many spots do we consider) to find the right spot?

n^2 - general

n - already sorted list w/efficient implementation

BUBBLE SORT

- Make passes through list
- On each pass, swap out of order elements


original [1, 5, 0, 7, 3]
no swap
pls1 [1, 5, 0, 7, 3]
swap
pls2 [1, 0, 5, 7, 3]
no swap
pls3 [1, 0, 5, 7, 3]
swap
pls4 [1, 0, 5, 3, 7]

after pass 1 [1, 0, 5, 3, 7]

after pass 2 [0, 1, 3, 5, 7]
* *

we can see it's sorted

no swaps at all

after pass 3 [0, 1, 3, 5, 7]

entire pass with no swaps → it's sorted

BUBBLE SORT - COMPLEXITY

- How many passes through list are made?
- How much work does each pass take?
- Can we stop early if not all passes needed?

n - 1

1st = n - 1
2nd = n - 2
⋮

n^2 ⇒ general

best case ⇒ n
↳ already sorted list with efficient implementation
↳ stops after full pass with no swaps