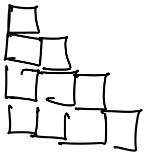
Quadratic-time sorts

To sort an array A of length n, indexed from 0 to n-1:

67	33	21	84	49	50	75
0	1	2	3	4	5	6

Selection Sort

Idea: Mentally divide the input array into two parts – the left part, which is sorted, and the right part, which is unsorted. Initially, the left part is empty (nothing is sorted), and the right part is the entire starting array. We find the smallest element in the right (unsorted) part, and swap it with the leftmost item in the unsorted part, thereby extending the sorted portion of the array by one element. This then moves the boundary between the sorted portion and the unsorted portion one spot to the right.



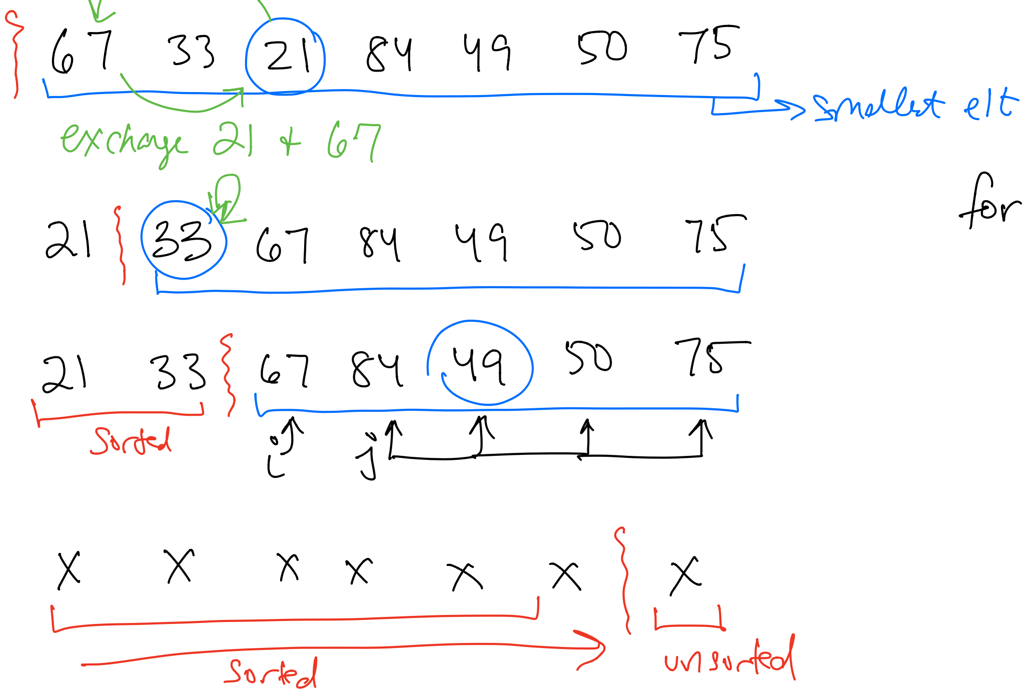
leftmost index of unsorted portion

```
for i = 0 to n-2 inclusive:
        // Find the smallest element in sublist A[i]...A[n-1]
        set smallpos = i
        for j = i+1 to n-1 inclusive:
                if A[j] < A[smallpos]:
                        set smallpos = j

        // and move that element into its proper position
        swap A[i] and A[smallpos]
```

Find the smallest elt & swap w/ i

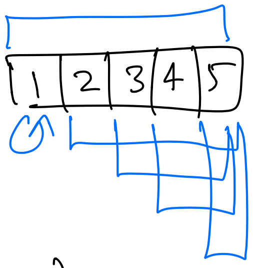
```
// Note that selection sort does not need to run for i=n-1 because this would
// correspond to finding the smallest element in a 1-item sublist.  In other
// words, by the time we get finish the i=n-2 loop, the array will be sorted.
```



for i in range(0, len(A)):

Big-oh : $O(n^2)$ → worst case

Best case?



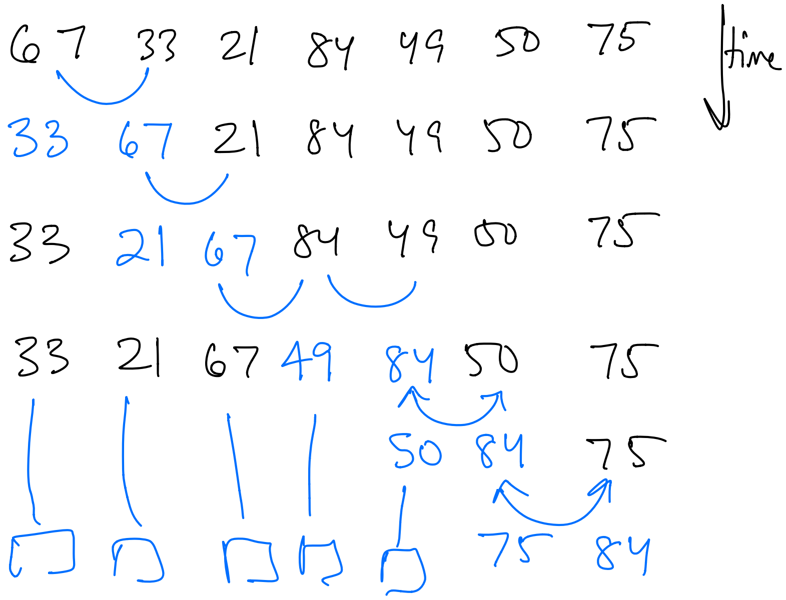
$O(n^2)$

Avg case $O(n^2)$

Bubble sort

Idea: Iterate through the array from front to back, swapping adjacent pairs of elements if they are out of order. Repeat this iteration until you complete an entire pass through the array resulting in zero swaps (meaning the array is sorted).

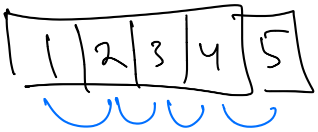
```
do
        swapped = false
        for i = 0 to n-2 inclusive:
                // if this pair is out of order
                if A[i] > A[i+1]
                        swap A[i] and A[i+1]
                        swapped = true
repeat until not swapped
```



Worst case $O(n^2)$

Best case? → $O(n)$

→ sorted array.

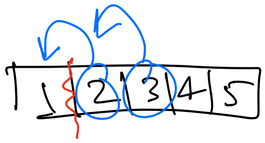


Insertion sort

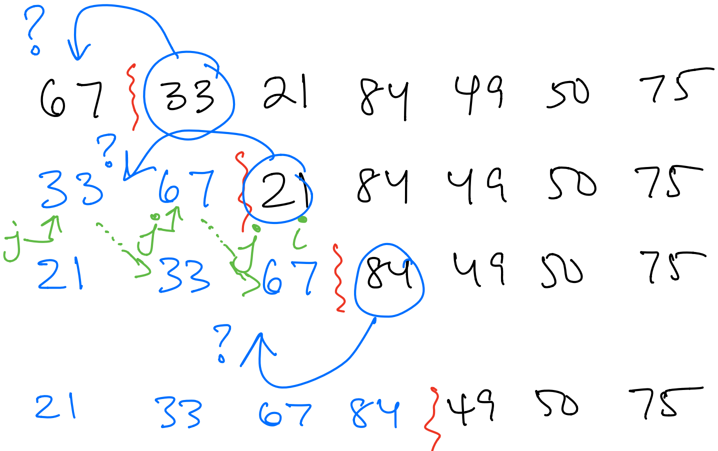
Big-oh - $O(n^2)$ - worst case

Best case? $O(n)$

└ sorted array.



temp = 21



Insertion Sort

Idea: Mentally divide the input array into two parts – the left part, which is sorted, and the right part, which is unsorted. Initially, the left (sorted) part is just one single element at index [0], and the right (unsorted) part is the everything else in the array. We take the leftmost item in the right (unsorted) part, and find the spot in the left (sorted) part of the array where this item should go. We slide the other items in the sorted section to the right to make room. This overwrites the leftmost item in the unsorted section, but of course that item is exactly the one being inserted, so this is OK. This insertion increases the size of the sorted portion, moving the boundary one spot to the right.

```
for i = 1 to n-1 inclusive:
        // A[0]...A[i-1] are the sorted portion.
        // move A[i] into its proper position within sorted section A[0]...A[i-1]
        temp = A[i]
        j = i
        while j > 0 and A[j-1] > temp:
                A[j] = A[j-1]             // slide A[j-1] to the right (into A[j]'s place)
                j--
        A[j] = temp
```

Sorting 3 7 4 9 2 6 1

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	7	4	9	2	6	1

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	7	4	9	2	6	1

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	7	4	9	2	6	1