HW 3 → Friday
P 3 → Monday
} 11:59 PM

Sorting I

- Sorting Overview
- Bubble Sort
- Selection Sort
- Insertion Sort

Why do we care so much about sorting?

- Sorting is sometimes a key ingredient in other algorithms.
- Sorting is a simple problem, so it is a good problem for understanding and appreciating differences in the algorithms.

What do we care about with sorting methods?

- **Certification:** verification of correctness
- **Runtime:**
  - number of comparisons or array accesses
  - ~~why not exchanges?~~
- **Extra memory:**
  - in situ (in place)
  - extra copy of array

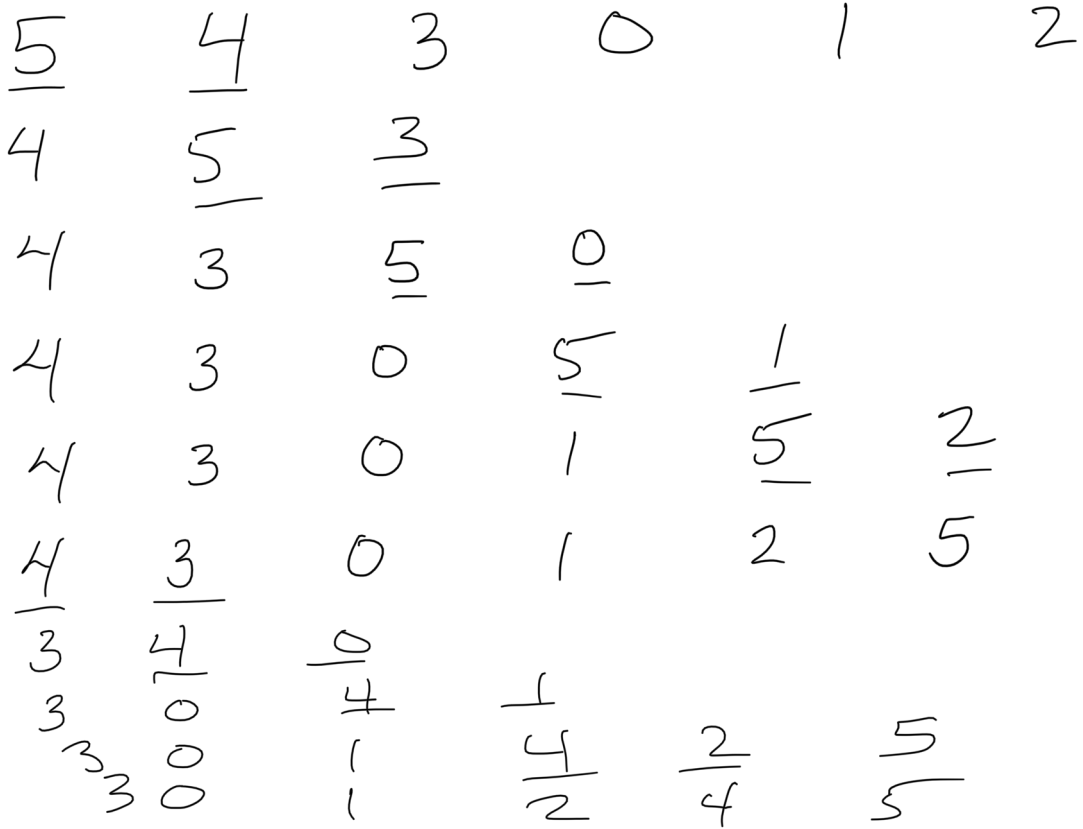
  } call stack (recursive algorithms)
- **Types of data**: anything that can be ordered
- **Comparable/Comparator**: Java interfaces that make it easier to write more flexible sorting algorithms

Visualization

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

https://www.youtube.com/watch?v=kPRA0W1kECg

Bubble Sort


5 4 3 0 1 2
4 5 3
4 3 5 0
4 3 0 5 1
4 3 0 1 5 2
4 3 0 1 2 5
3 4 0 1
3 0 4
3 3 0 1 4 2
3 0 1 2 5
4
5

Bubble Sort

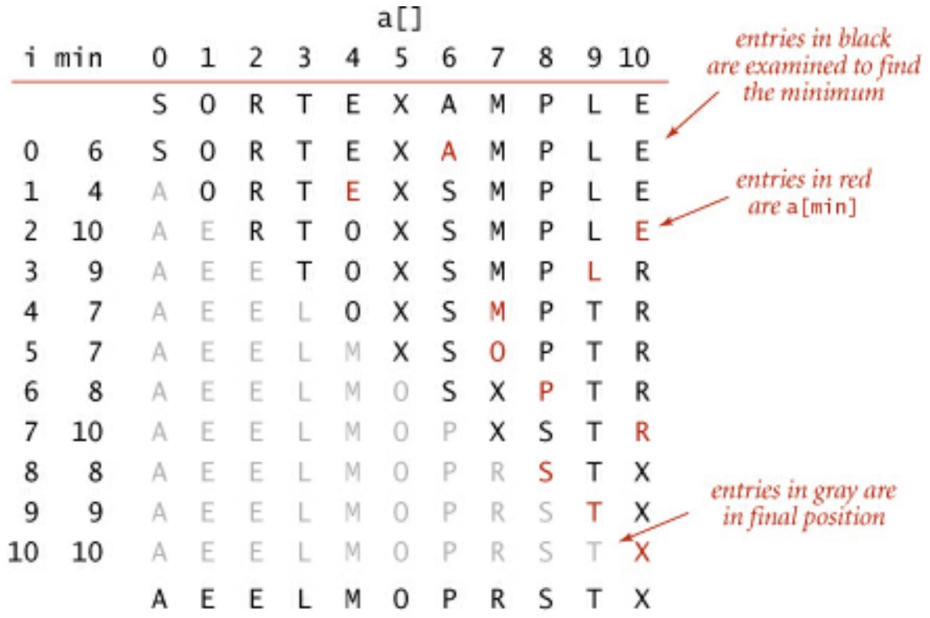
Input: An array A of size N
Output: Sorted A

best: $O(N^2)$
worst: $O(N^2)$

```
for j from 1 to N-1
    i = 0
    while i+1 < N:
        if A[i] > A[i+1]
            swap A[i] and A[i+1]
        i++
    end while
end for
```

Selection Sort

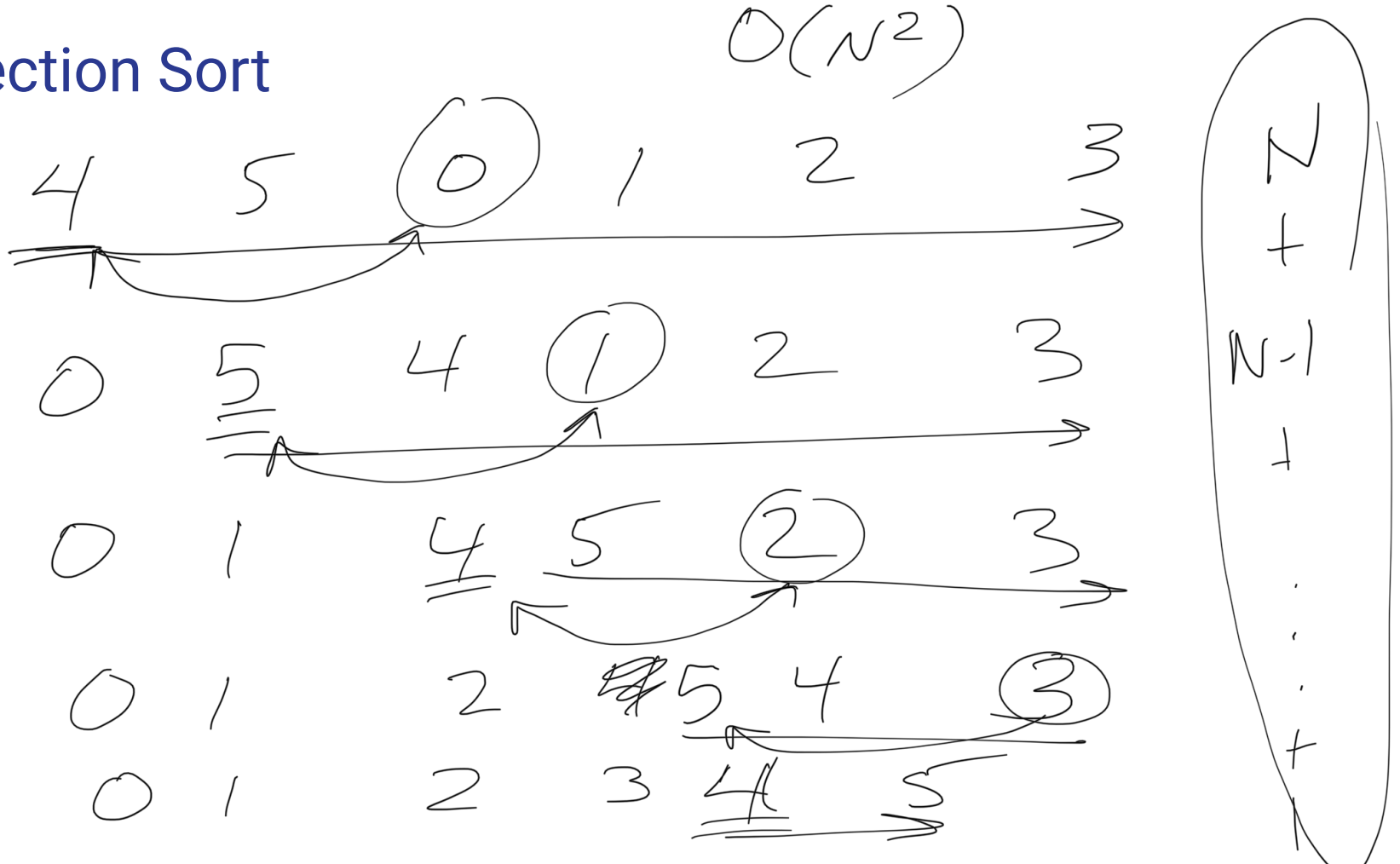
Repeatedly **select** the smallest item and put it into the next unsorted spot.



Trace of selection sort (array contents just after each exchange)

Picture from [1]

Selection Sort



Selection Sort

Input: an array A of size N

Output: A, sorted

for i from 0 to N-1:

→ min = find the min in i to N-1

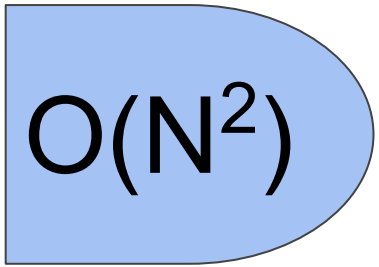
swap the min with A[i]

end for

$N + N-1 + N-2 + \ldots + 1 \Longrightarrow O(N^2)$

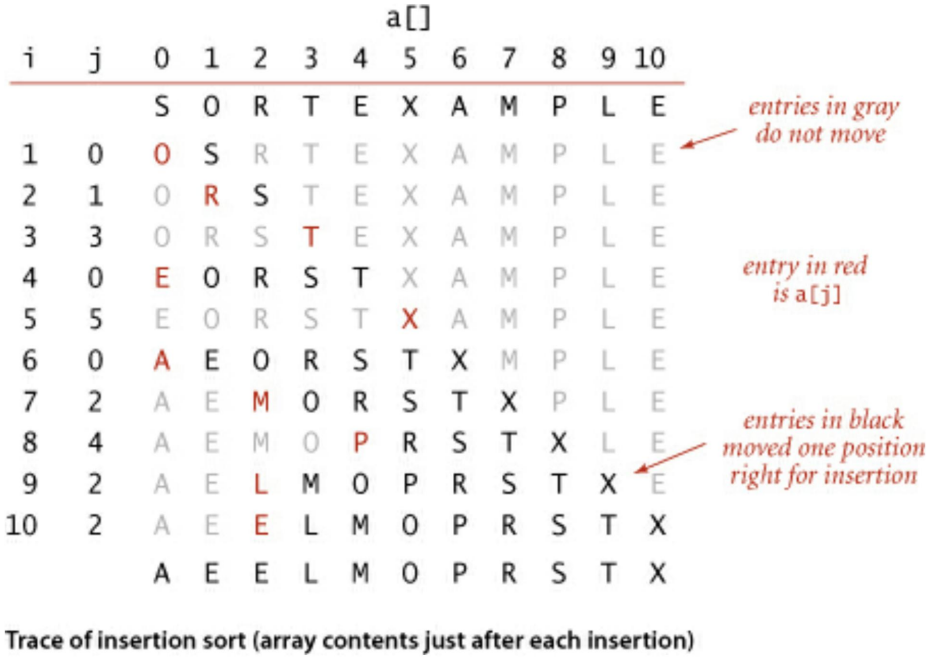
Properties of Selection Sort

- Running time is insensitive to input
  - Input array is in random order
  - Input array is already sorted
  - Input array is sorted in opposite direction
  - Input array is all the same number
- Data movement is minimal—O(N) exchanges
- Anything left of the current index is sorted and in final position



Insertion Sort

Insert current item into its proper place in the sorted part of the array (on the left).



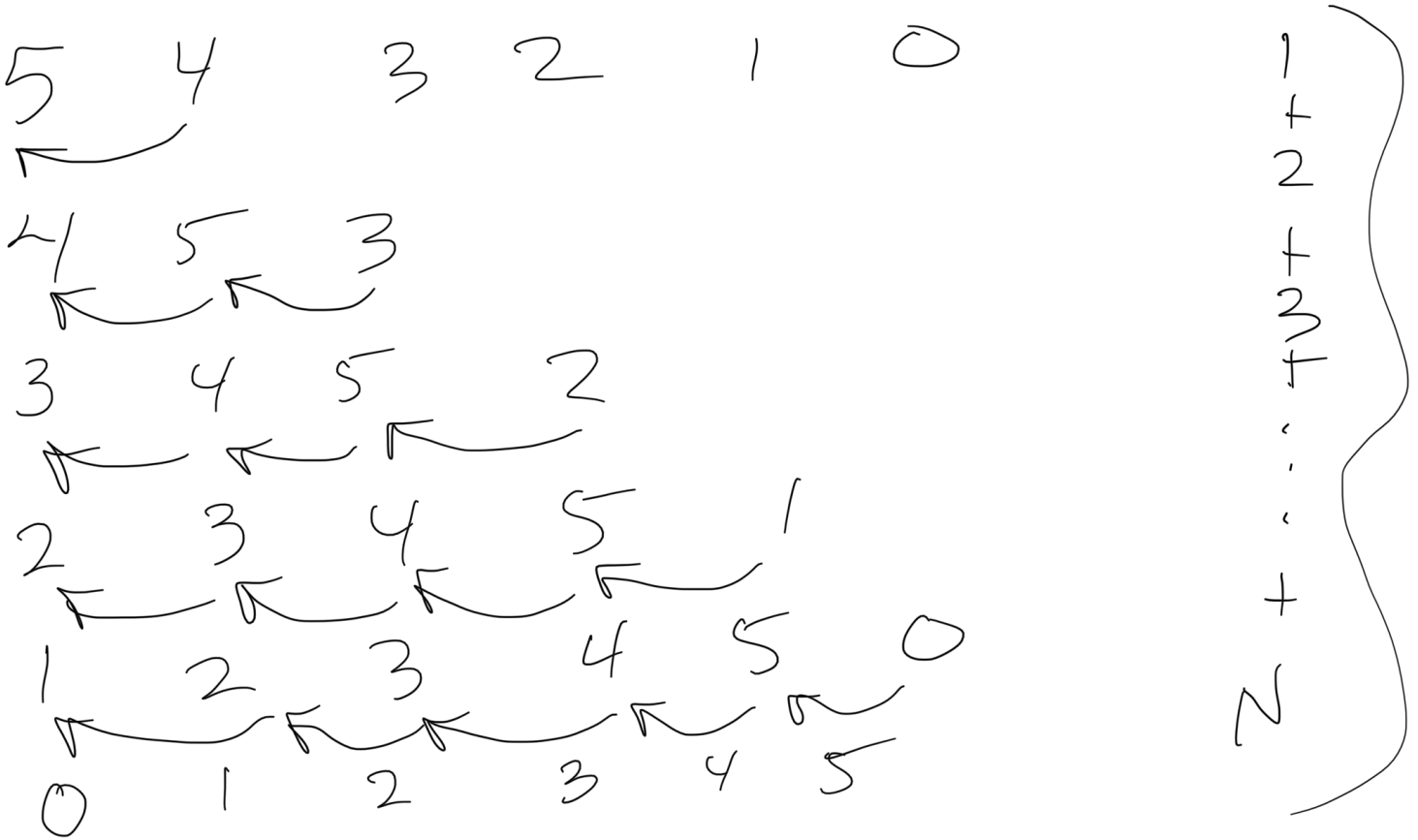
		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
1	0	O	S	R	T	E	X	A	M	P	L	E
2	1	O	R	S	T	E	X	A	M	P	L	E
3	3	O	R	S	T	E	X	A	M	P	L	E
4	0	E	O	R	S	T	X	A	M	P	L	E
5	5	E	O	R	S	T	X	A	M	P	L	E
6	0	A	E	O	R	S	T	X	M	P	L	E
7	2	A	E	M	O	R	S	T	X	P	L	E
8	4	A	E	M	O	P	R	S	T	X	L	E
9	2	A	E	L	M	O	P	R	S	T	X	E
10	2	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

Trace of insertion sort (array contents just after each insertion)

Picture from [1]

Insertion Sort

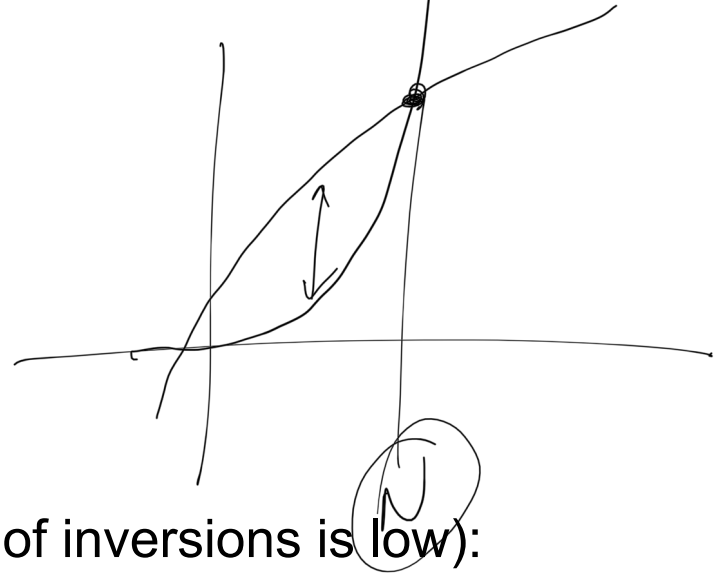
best: $O(N)$ $O(N^2)$



Insertion Sort

```
Input: Array A of size N
Output: A, sorted
for i from 1 to N-1
    j = i
    while j-1 ≥ 0:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
            j--
        end if
        else break;
    end while
```

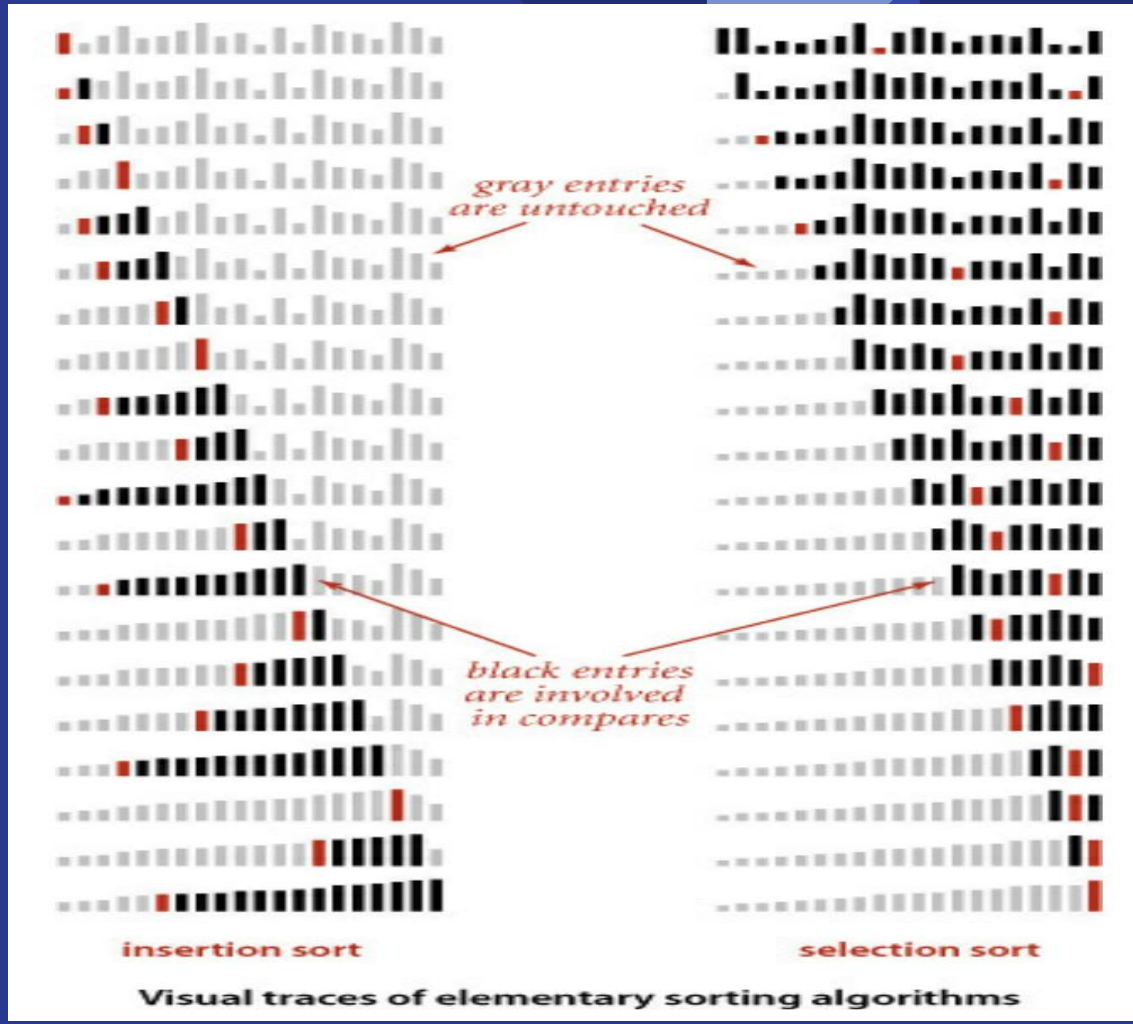
Properties of Insertion Sort



- Running time is sensitive to input
- Works well for:
  - Tiny arrays
  - Partially sorted arrays (number of inversions is low):
    - each entry is not far from its final position
    - small array appended to a large sorted array
    - array only has a few elements out of place
- Anything left of the current index is sorted but not necessarily in final position

Visual Representation of Sorting Methods:

- Shows sorting of bars by length
- Note:
  - For insertion sort, nothing right of the current index is touched
  - For selection sort, nothing left of the current index is touched.
  - Insertion sort tends to require fewer compares.



Picture from [1]

Analysis

Algorithm	Best	Average	Worst	Extra Space	Comments
Bubble Sort	O(N)	$O(N^2)$	$O(N^2)$	O(1)	• similar to Insertion Sort with worse average case • easy to implement
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$	O(1)	• minimal data movement (O(N) in the worst case) • does not take advantage of the input
Insertion Sort	O(N)	$O(N^2)$	$O(N^2)$	O(1)	• takes advantage of input • works well on small arrays • works well on partially sorted arrays

References

[1] *Algorithms, Fourth Edition;* Robert Sedgewick and Kevin Wayne (and associated slides)