Introduction to Sorting

- Sorting Overview
- BubbleSort
- Selection Sort
- Insertion Sort

Why do we care so much about sorting?

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?

runtime
space
difficulty of implementation

What do we care about with sorting methods?

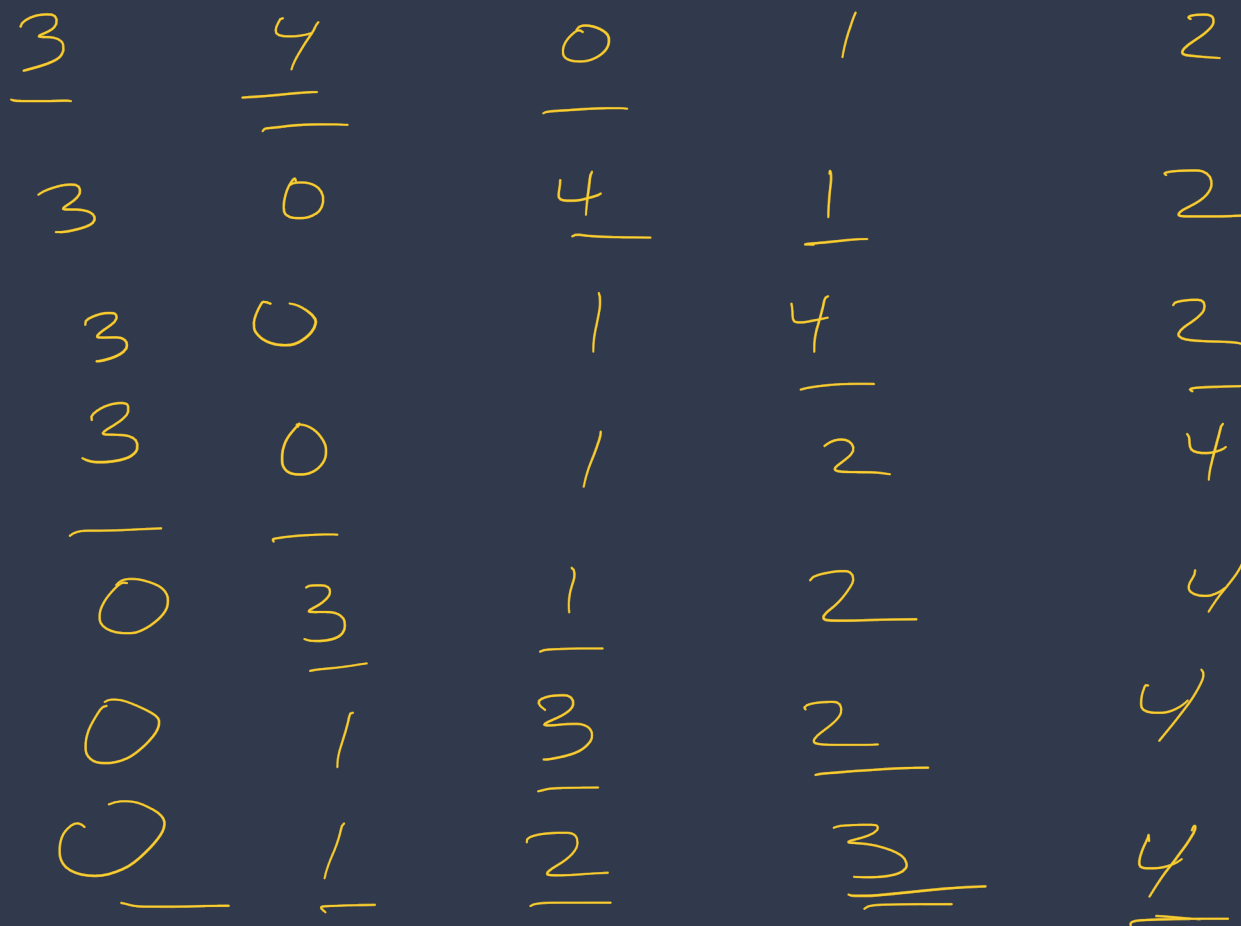
- **Certification:** guarantee that the result is always sorted
- **Runtime:**
  - number of comparisons or array accesses
  - why not exchanges?
- **Extra memory:**
  - in place (aka “in situ”)
  - extra copy of array
- **Types of data**: anything that can be ordered
- **Comparable/Comparator**: Java interfaces that make it easier to write more flexible sorting algorithms

Bubble Sort

- "The easiest sorting algorithm to program"
- Algorithm:

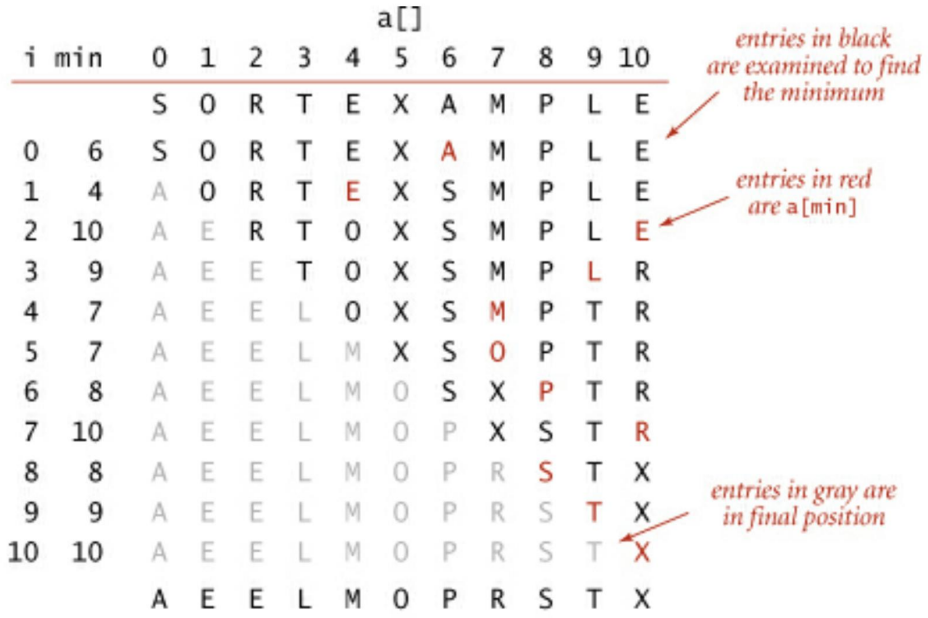
```
repeat
    for x = 1 to N-1
      if (A[x] > A[x+1]) then swap
    end
until (no more swaps)
```

Picture from [2]



Selection Sort

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



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

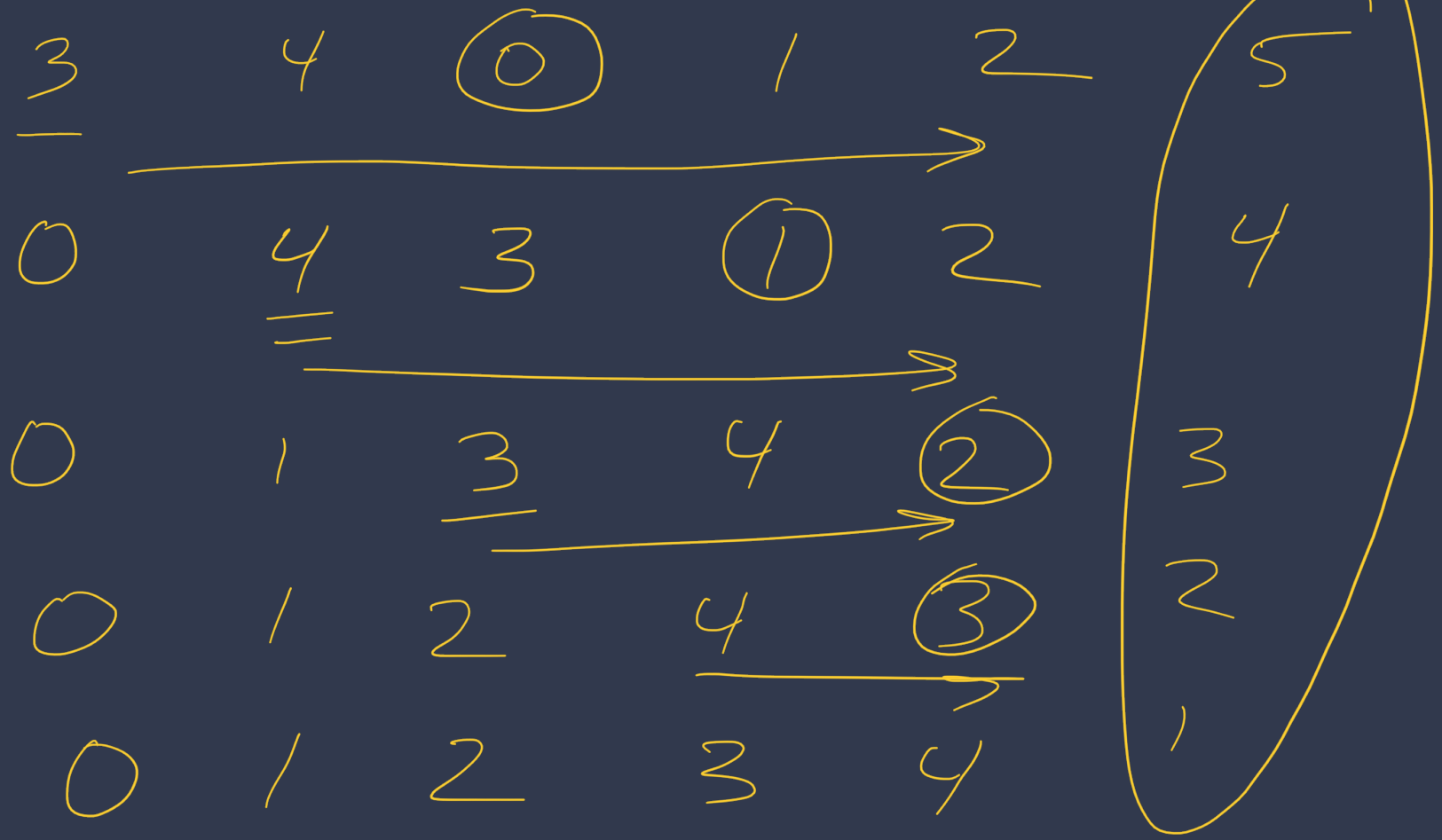
entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position

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

Picture from [1]


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

Pseudo code

A : array of size N

O(N) [

for i from 0 to N-1 :

m = findMin in A[i ... N-1] } O(N)

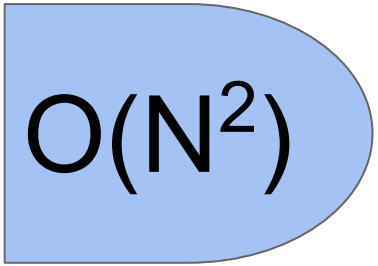
swap m and A[i]

end for

]

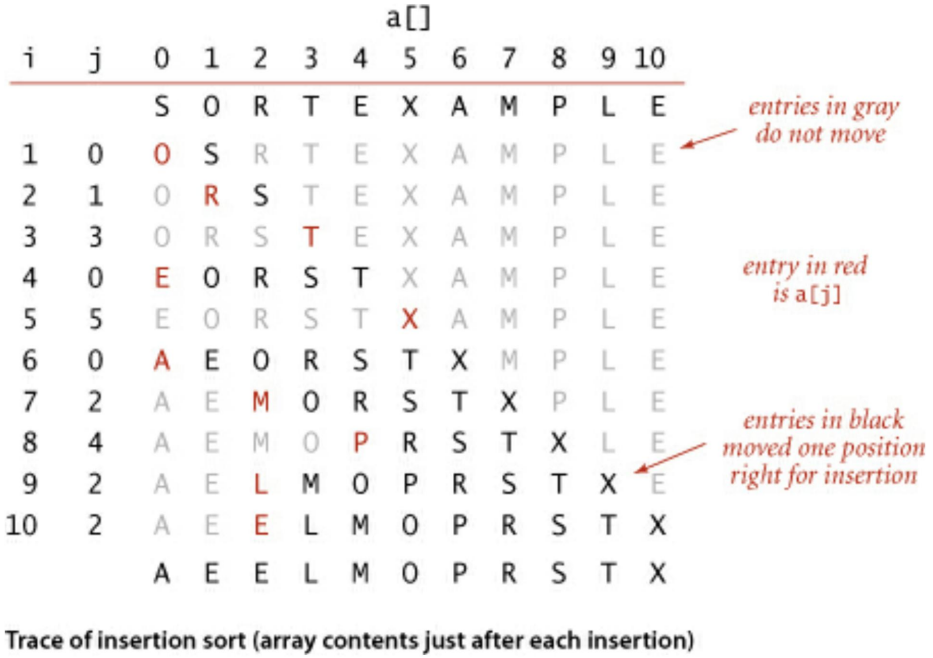
Properties of Selection Sort

- 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

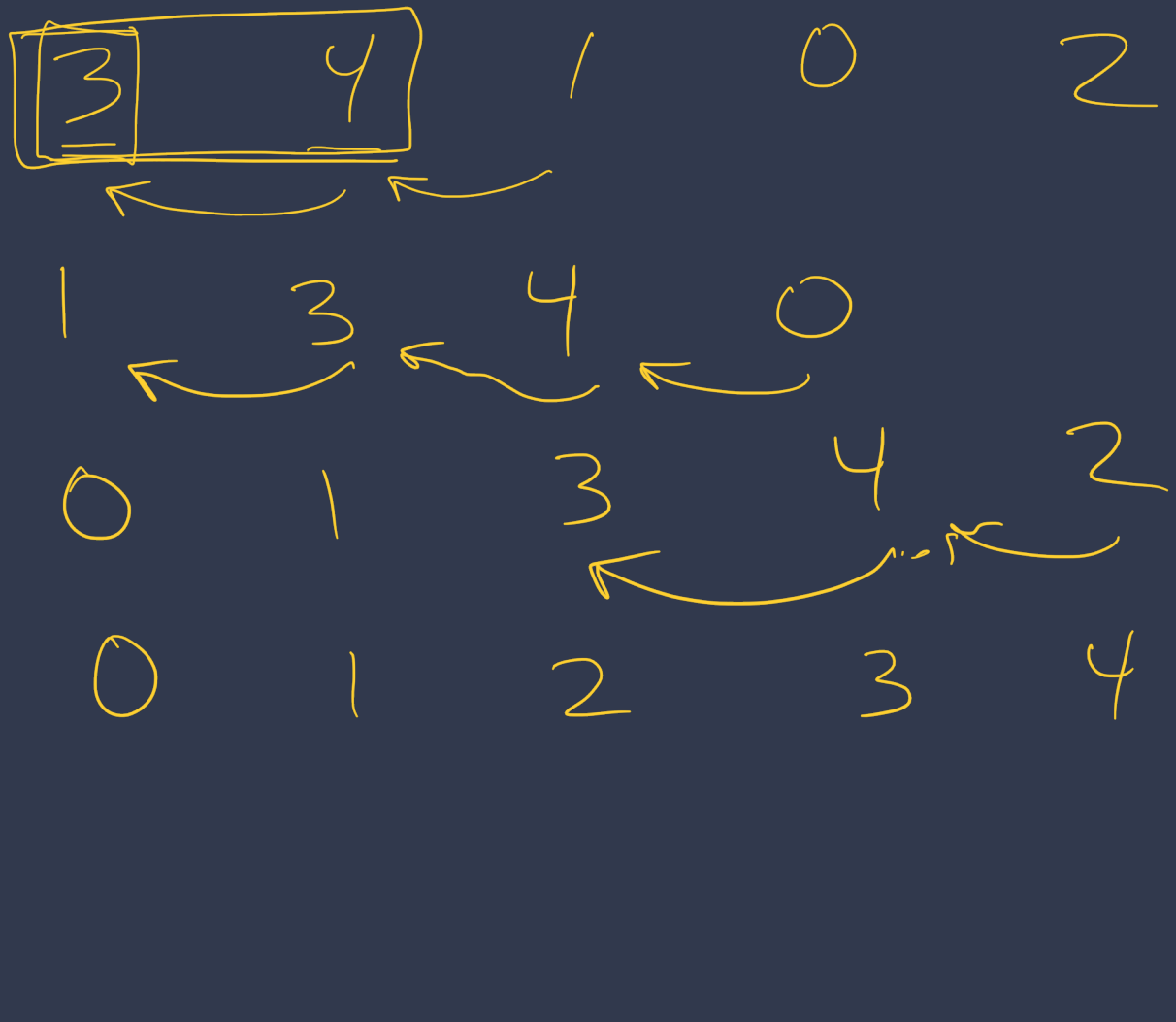
entries in gray do not move

entry in red is a[j]

entries in black moved one position right for insertion

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

Picture from [1]


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

```
A: Array of size N
for i from 1 to N-1:
    j = i
    while j > 0 && A[j] < A[j-1]:
        swap A[j] and A[j-1]
        j--
    end while
end for
```

worst: $O(N^2)$

best: $O(N)$

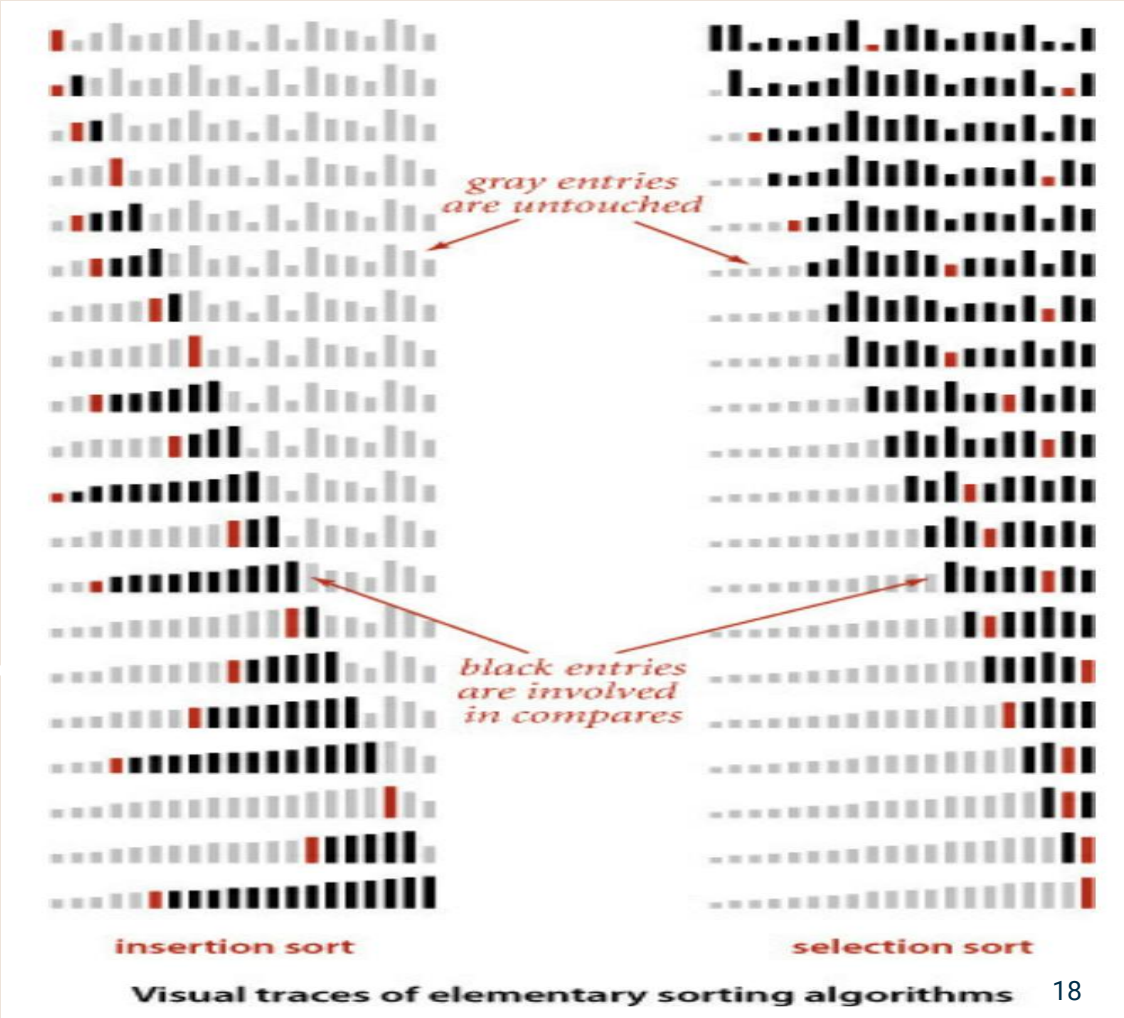
Properties of Insertion Sort

Tim 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.



Visual traces of elementary sorting algorithms

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)