INTRODUCTION TO DATA STRUCTURES and ALGORITHMS

Rutgers University

ELEMENTARY SORTS

- *rules of the game*
- *selection sort*
- *insertion sort*
- *shuffling*
- *comparators*



http://ds.cs.rutgers.edu

Slides adopted and modified from Sedgewick/Wayne

Elementary Sorts

- *rules of the game*
- *selection sort*
- *insertion sort*
- *shuffling*
- *comparators*

Sorting problem

- the number of compares and exchanges that they use for various types of inputs, and
- the amount of extra memory that they use

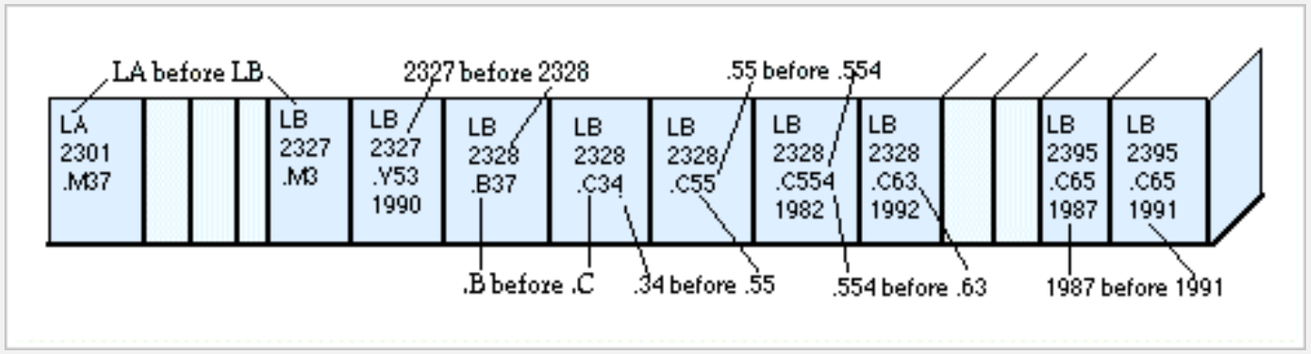
Ex. Student records in a university.

	Chen	3	A	(991) 878-4944	308 Blair
	Rohde	2	A	(232) 343-5555	343 Forbes
	Gazsi	4	B	(800) 867-5309	101 Brown
item →	Furia	1	A	(766) 093-9873	101 Brown
	Kanaga	3	B	(898) 122-9643	22 Brown
	Andrews	3	A	(664) 480-0023	097 Little
key →	Battle	4	C	(874) 088-1212	121 Whitman

Sort. Rearrange array of n items in ascending order by key.

Andrews	3	A	(664) 480-0023	097 Little
Battle	4	C	(874) 088-1212	121 Whitman
Chen	3	A	(991) 878-4944	308 Blair
Furia	1	A	(766) 093-9873	101 Brown
Gazsi	4	B	(800) 867-5309	101 Brown
Kanaga	3	B	(898) 122-9643	22 Brown
Rohde	2	A	(232) 343-5555	343 Forbes

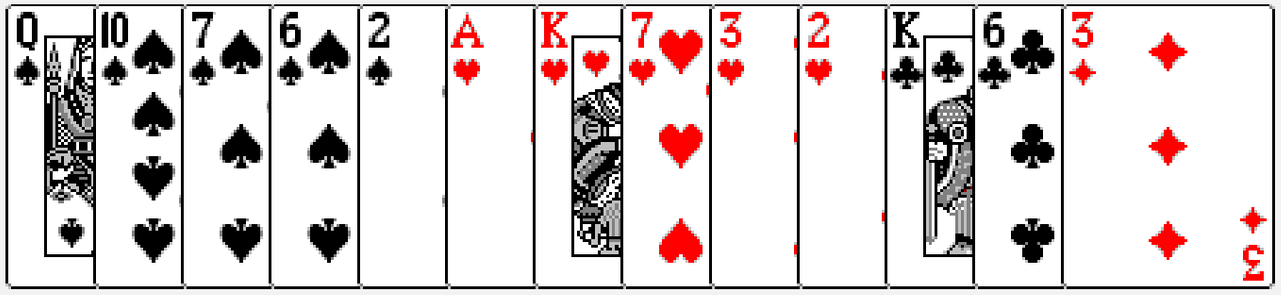
Sorting applications



Library of Congress numbers



FedEx packages



playing cards



contacts



Hogwarts houses

Sample sort clients

Goal. Sort any type of data. //must implement the Comparable Interface

Ex 1. Sort strings in alphabetical order.

```
public class StringSorter
{
   public static void main(String[] args)
   {
      String[] a = StdIn.readAllStrings();
      Insertion.sort(a);
      for (int i = 0; i < a.length; i++)
         StdOut.println(a[i]);
   }
}
```

```
% more words3.txt
bed bug dad yet zoo ... all bad yes

% java StringSorter < words3.txt
all bad bed bug dad ... yes yet zoo
[suppressing newlines]
```

Sample sort clients

Goal. Sort any type of data.

Ex 2. Sort random real numbers in ascending order.

seems artificial (stay tuned for an application)

```
public class Experiment
{
   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      Double[] a = new Double[n];
      for (int i = 0; i < n; i++)
         a[i] = StdRandom.uniform();
      Insertion.sort(a);
      for (int i = 0; i < n; i++)
         StdOut.println(a[i]);
   }
}
```

```
% java Experiment 10
0.08614716385210452
0.09054270895414829
0.1070874630489864 2
0.2116619007164681 8
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

Sample sort clients

Goal. Sort any type of data.

Ex 3. Sort the files in a given directory by filename.

```
import java.io.File;

public class FileSorter
{
   public static void main(String[] args)
   {
      File directory = new File(args[0]);
      File[] files = directory.listFiles();
      Insertion.sort(files);
      for (int i = 0; i < files.length; i++)
         StdOut.println(files[i].getName());
   }
}
```

```
% java FileSorter .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
```

Elementary Sorts

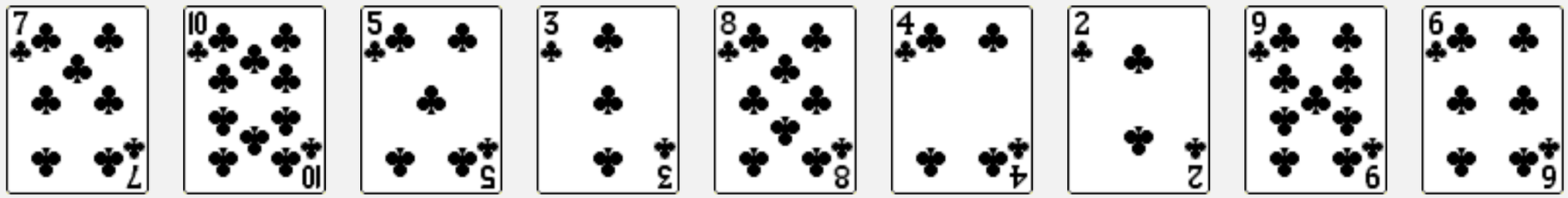
- *rules of the game*
- *selection sort*
- *insertion sort*
- *shuffling*
- *comparators*

Selection sort demo //in-place comparison sort

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min]. // O(1) additional memory space needed for swapping

Put the elements in a correct order starting from the first position of the array

initial



find the index of the smallest item and swap with i
(set element at index i as the SMALLEST initially)

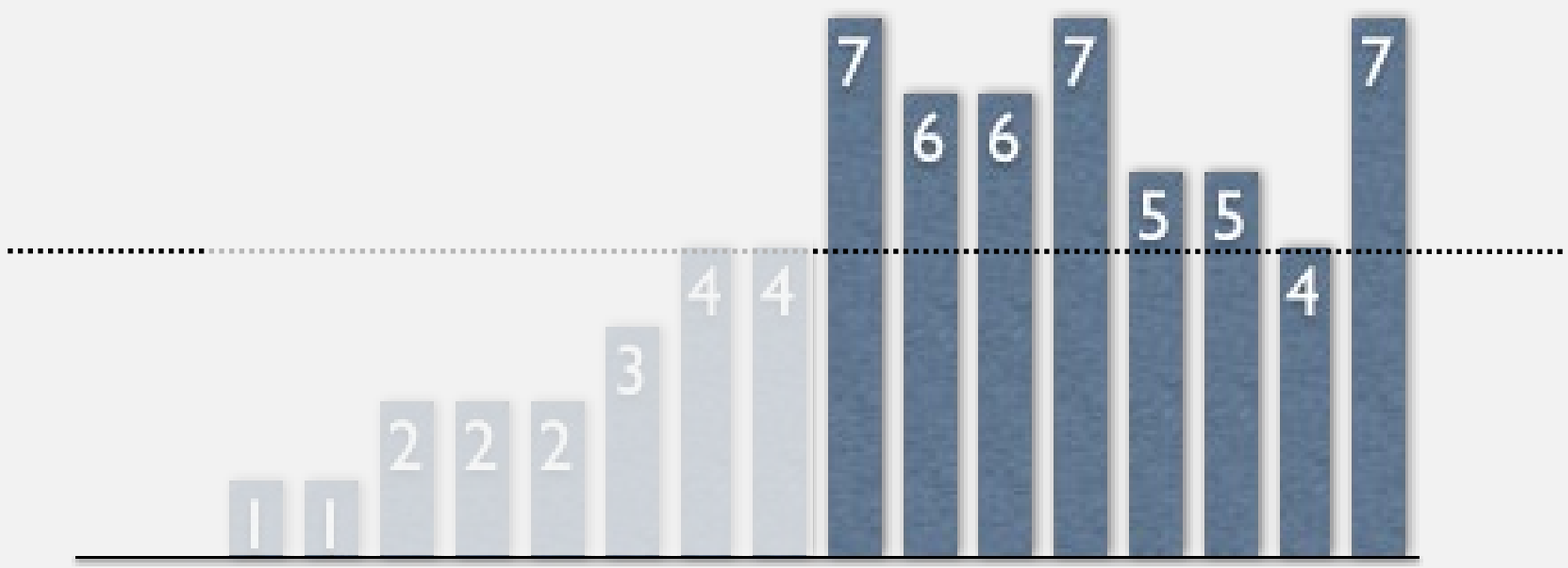
Selection sort

Algorithm. scans from left to right.

Invariants.

- Entries the left of *i* (including) fixed and in ascending order.
- No entry to right of *i* is smaller than any entry to the left of .

After ith iterations, top (left) i elements are in the correct positions (sorted)



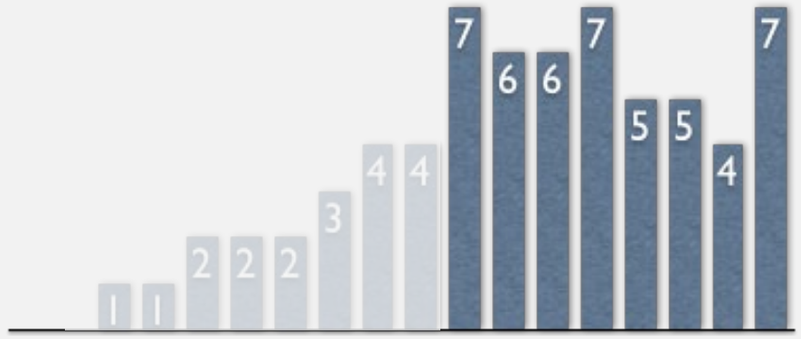
in final order

Selection sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

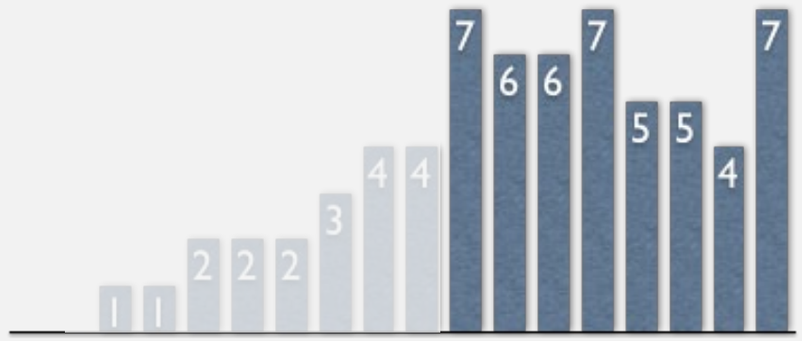
```
i++;
```



in final order

- Identify index of minimum entry on right.

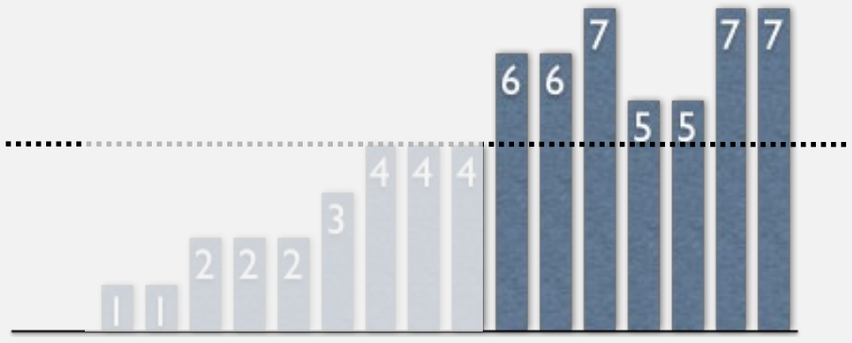
```
int min = i;  //keep the index of the smallest item
for (int j = i+1; j < n; j++)
   if (less(a[j], a[min]))
      min = j;  //keep track of the smallest
```



in final order

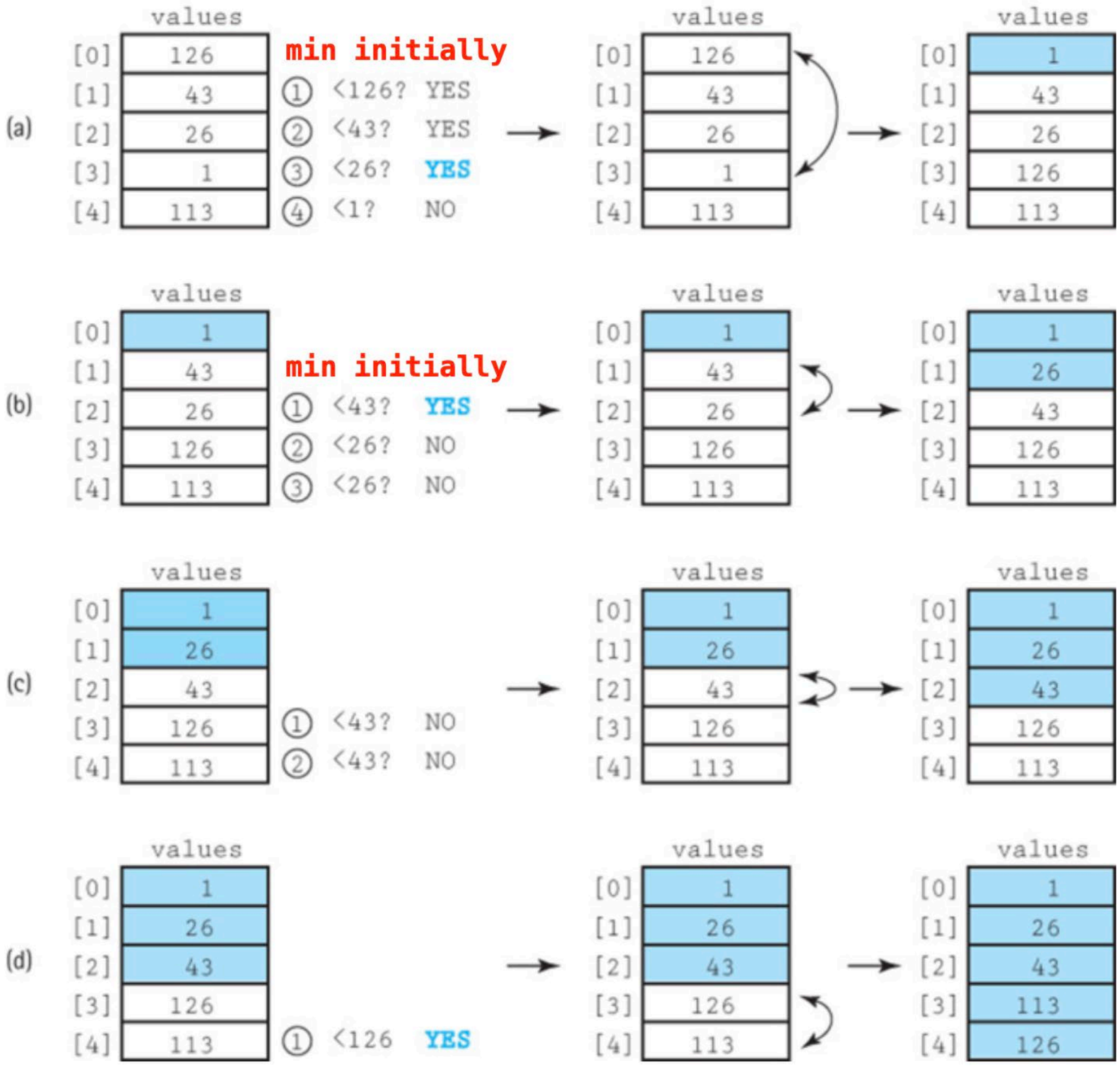
- Exchange into position.

```
exch(a, i, min);
```



in final order

Selection Sort – more example



Two useful sorting abstractions

Helper functions. Refer to data only through compares and exchanges.

Less. Is item v less than w ?

```
private static boolean less(Comparable v, Comparable w)
{  return v.compareTo(w) < 0;  } //strictly smaller
```

Exchange. Swap item in array a[] at index i with the one at index j.

```
private static void exch(Object[] a, int i, int j)
{
   Object swap = a[i]; //the additional space needed for the swap
   a[i] = a[j];
   a[j] = swap;
}
```

Selection sort: Java implementation

```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int n = a.length;
      for (int i = 0; i < n; i++)
      {
         int min = i;
         for (int j = i+1; j < n; j++) //how many iterations?
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min); //how many exchanges?
      }
   }

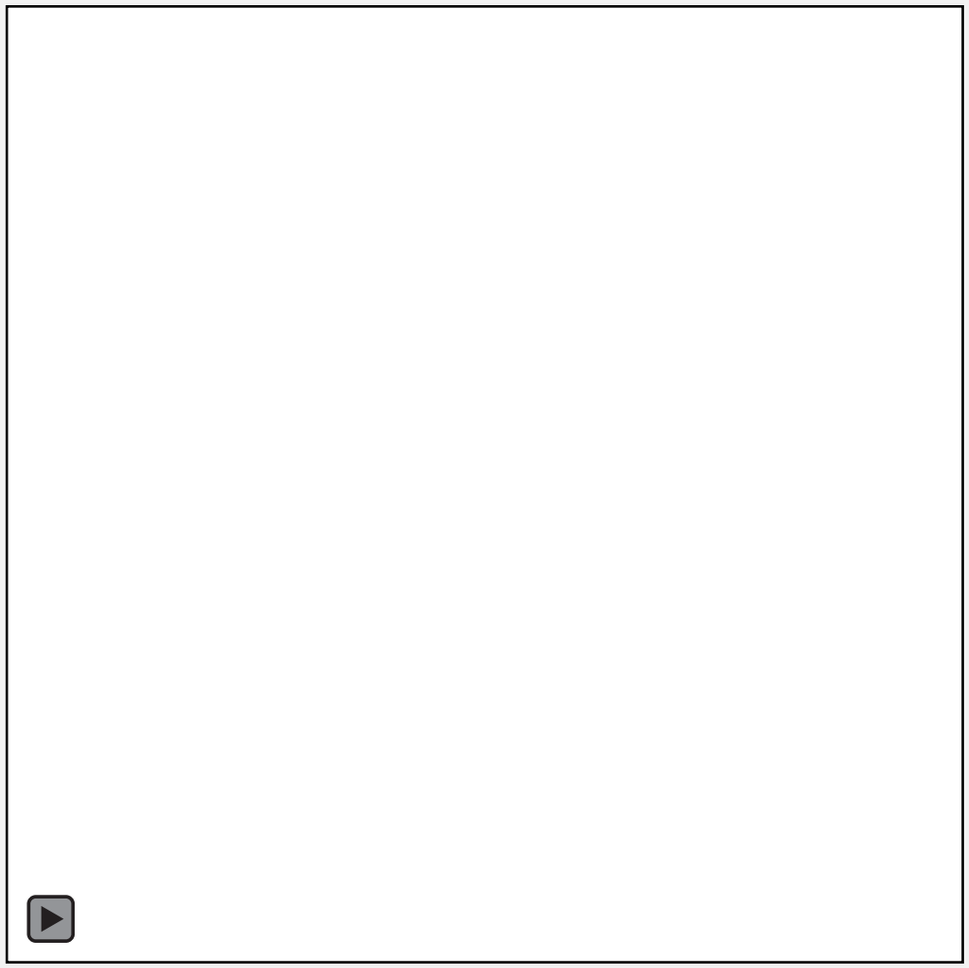
   private static boolean less(Comparable v, Comparable w)
   {  /* see previous slide */  }

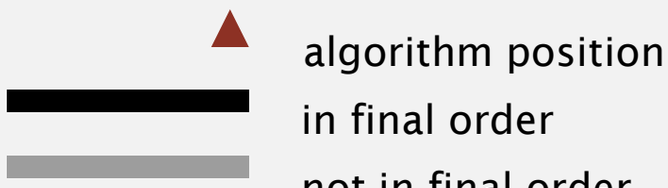
   private static void exch(Object[] a, int i, int j)
   {  /* see previous slide */  }
}
```

(N − 1) + (N − 2) + . . . + 2 + 1 + 0

Selection sort: animations

20 random items

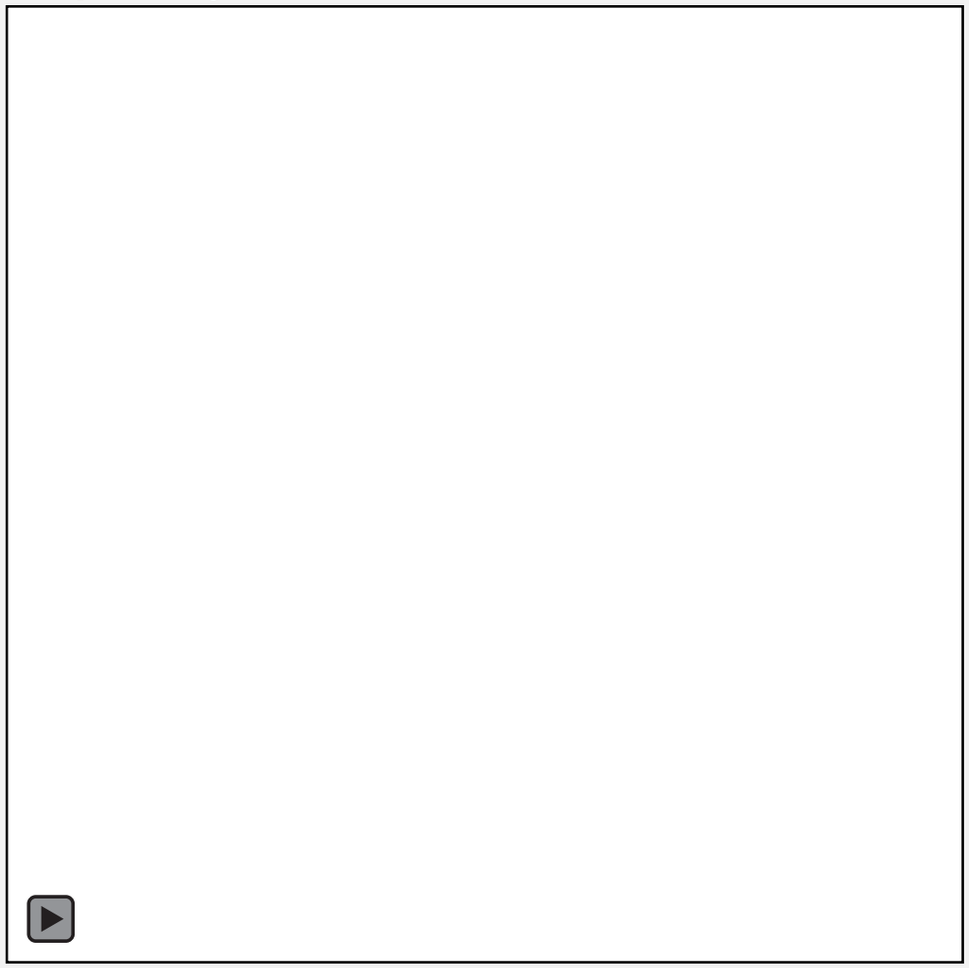


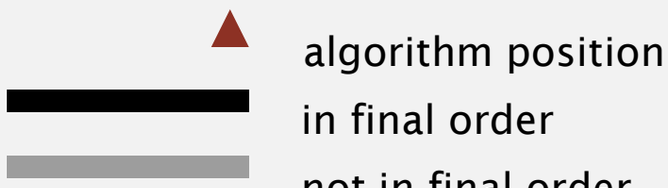


http://www.sorting-algorithms.com/selection-sort

Selection sort: animations

20 partially sorted items





http://www.sorting-algorithms.com/selection-sort

Elementary sorts: quiz 2

How many compares does selection sort make to sort an array of n distinct items in reverse order?

A. $\sim n$

B. $\sim 1/4 \ n^2$

C. $\sim 1/2 \ n^2$

D. $\sim n^2$

Selection sort: mathematical analysis

Proposition. Selection sort uses $(n-1) + (n-2) + \ldots + 1 + 0 \sim n^2 / 2$ compares and n exchanges to sort any array of n items.

		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

Running time insensitive to input. Quadratic time, even if input is sorted.

Data movement is minimal. Linear number of exchanges—exactly n.



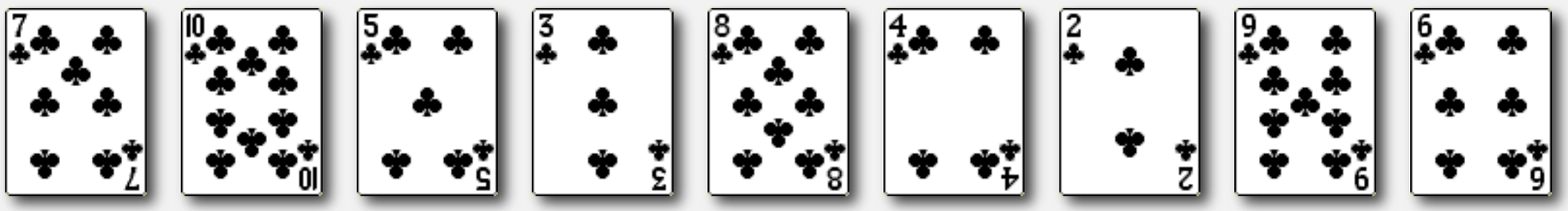
http://algs4.cs.princeton.edu

2.1 Elementary Sorts

- *rules of the game*
- *selection sort*
- *insertion sort*
- *shuffling*
- *comparators*

Insertion sort demo // in-place comparison sort

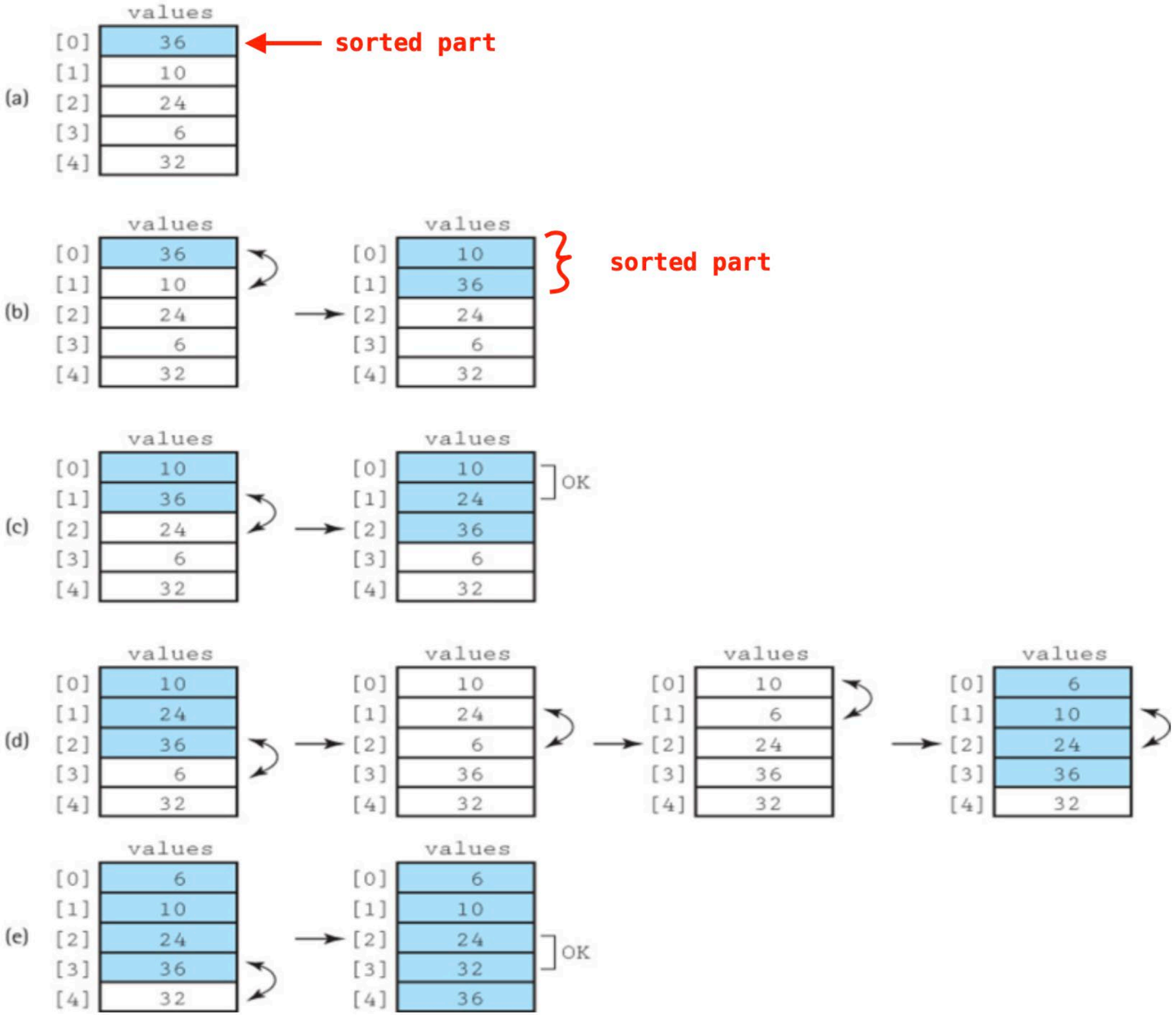
- In iteration i, swap a[i] with each larger entry to its left.
  - Initially, the sorted portion contains only one element→ the first element in the array.
  - Next, we take the second element in the array and put it into its correct place in the sorted part; that is, values[0] and values[1] are in order with respect to each other.
  - Next the value in values[2] is put into its proper place, so values[0]..values[2] are in order with respect to each other.
  - This process continues until all the elements have been sorted





initially, index 0 is the only part that is sorted,
next, to insert 10 to the sorted part, which contains only 7 in index 0
swap only when the left is larger, in this case, 7 < 10, no swapping is necessary.

Insertion Sort – more example



Insertion sort demo

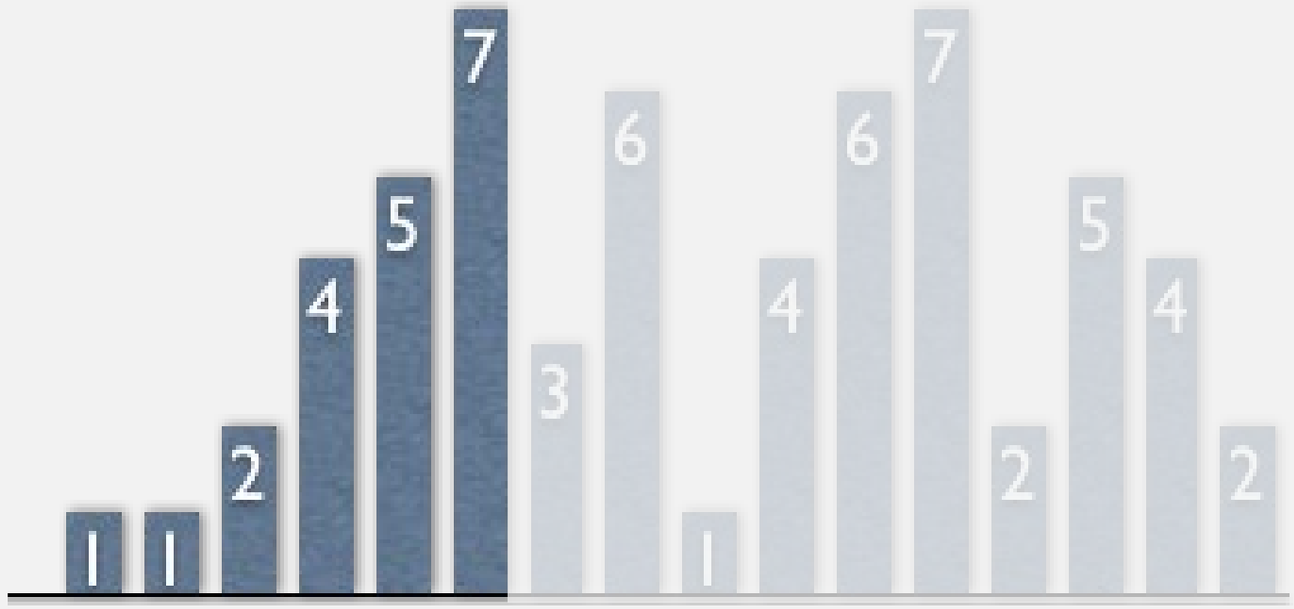
https://www.youtube.com/watch?v=ROalU379l3U

Insertion sort

Algorithm. scans from left to right.

Invariants.

- Entries to the left of *i* (including) are in ascending order.
- Entries to the right of *i* have not yet been seen.

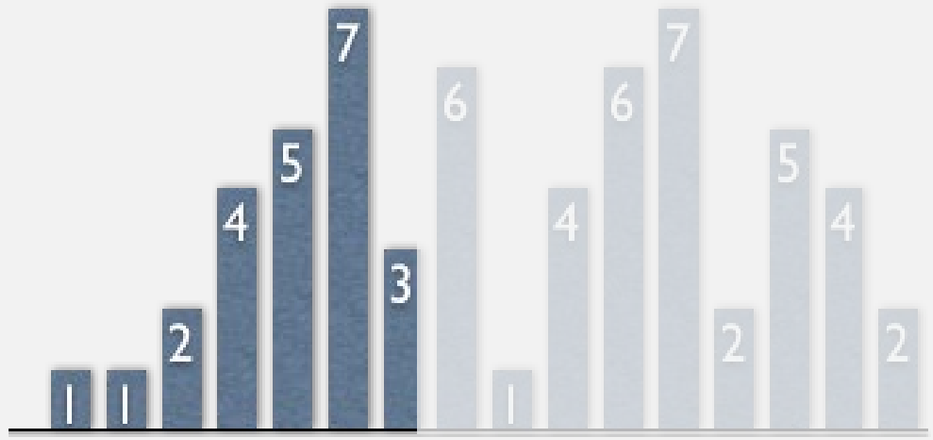


Insertion sort: inner loop

To maintain algorithm invariants:

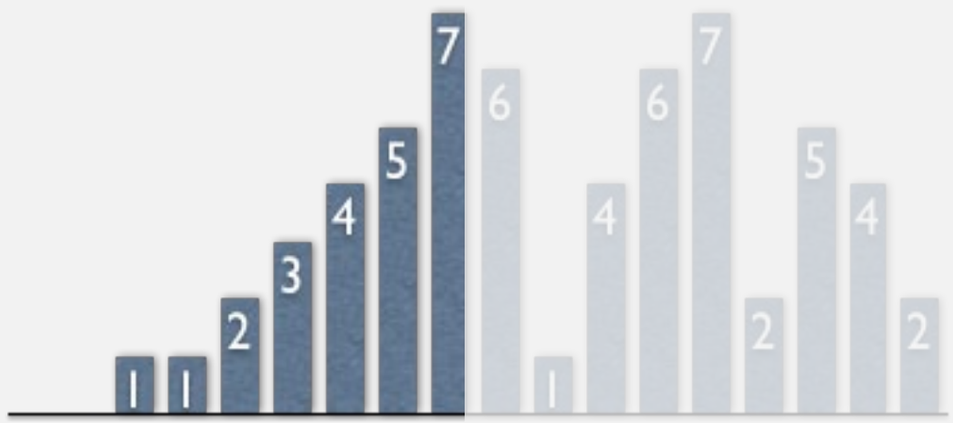
- Move the pointer to the right.

```
i++;
```



- Moving from right to left, exchange a[i] with each larger entry to its left.

```
for (int j = i; j > 0; j--)
   if (less(a[j], a[j-1]))
        exch(a, j, j-1);
   else break; //stop when item on left is smaller
```



Insertion sort: Java implementation

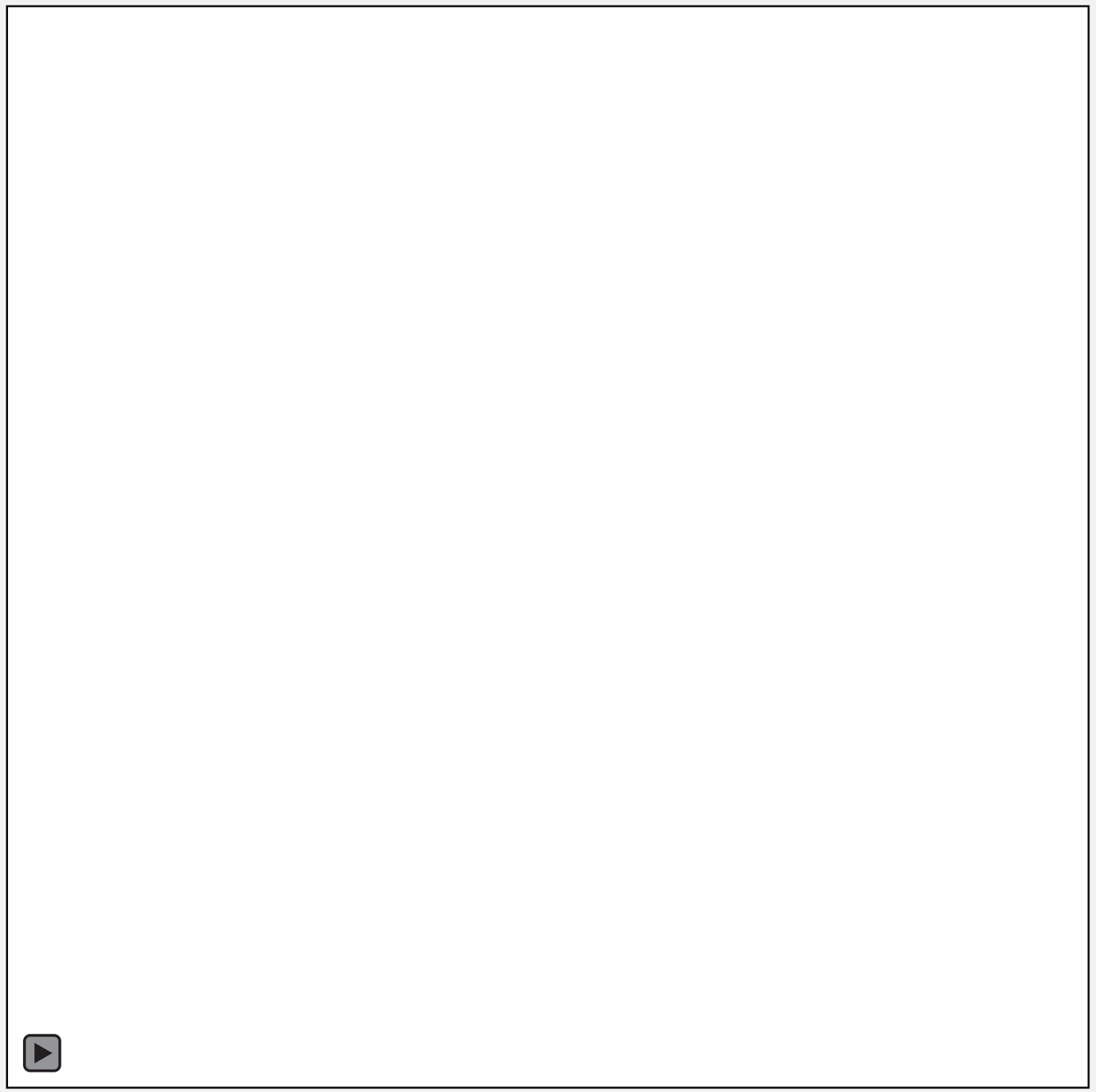
```
public class Insertion
{
   public static void sort(Comparable[] a)
   {
      int n = a.length;
      for (int i = 0; i < n; i++) //n iterations
         for (int j = i; j > 0; j--) //best case, 1 iteration
            if (less(a[j], a[j-1]))
               exch(a, j, j-1);
            else break; //if element on left is smaller, exit inner for loop
   }

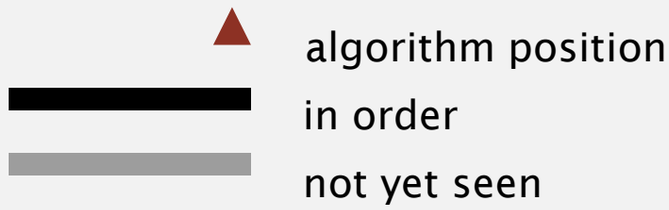
   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Object[] a, int i, int j)
   {  /* as before */  }
}
```

Insertion sort: animation

40 random items





http://www.sorting-algorithms.com/insertion-sort

Insertion sort: mathematical analysis

Proposition. To sort a randomly ordered array with distinct keys, insertion sort uses $\sim \frac{1}{4} n^2$ compares and $\sim \frac{1}{4} n^2$ exchanges on average.

Pf. Expect each entry to move halfway back.



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

Elementary sorts: quiz 3

Which is faster in practice to sort an array of n random items, selection sort or insertion sort?

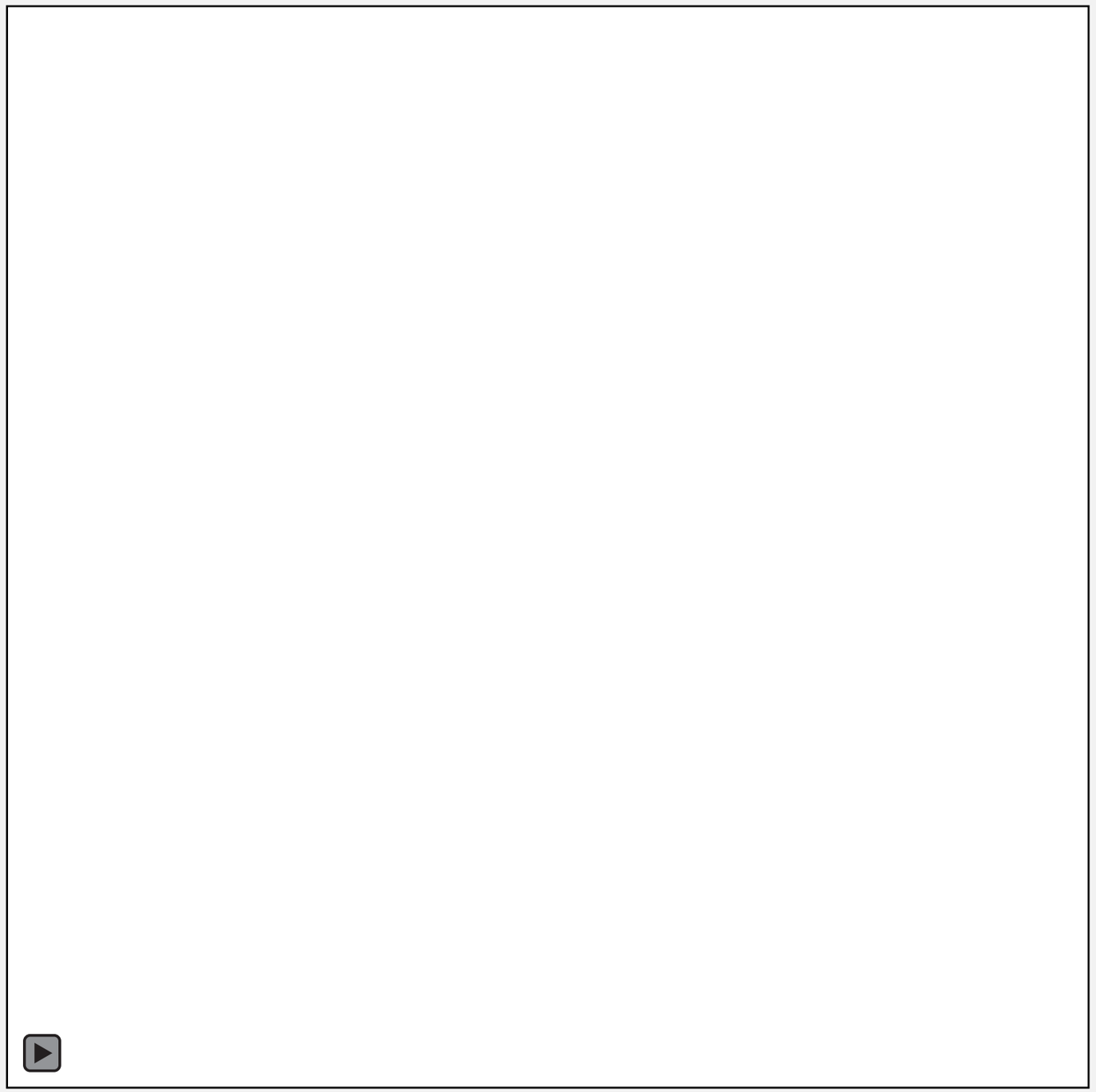
A. Selection sort.

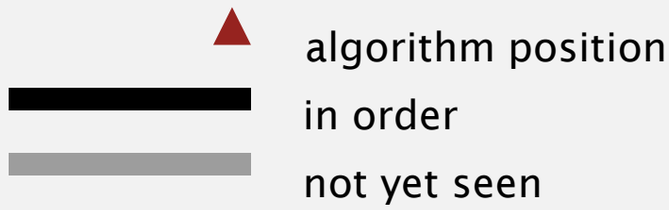
B. Insertion sort.

C. No significant difference.

Insertion sort: animation

40 reverse-sorted items





http://www.sorting-algorithms.com/insertion-sort

Insertion sort: analysis

Worst case. If the array is in descending order (and no duplicates), insertion sort makes $\sim \frac{1}{2} n^2$ compares and $\sim \frac{1}{2} n^2$ exchanges.

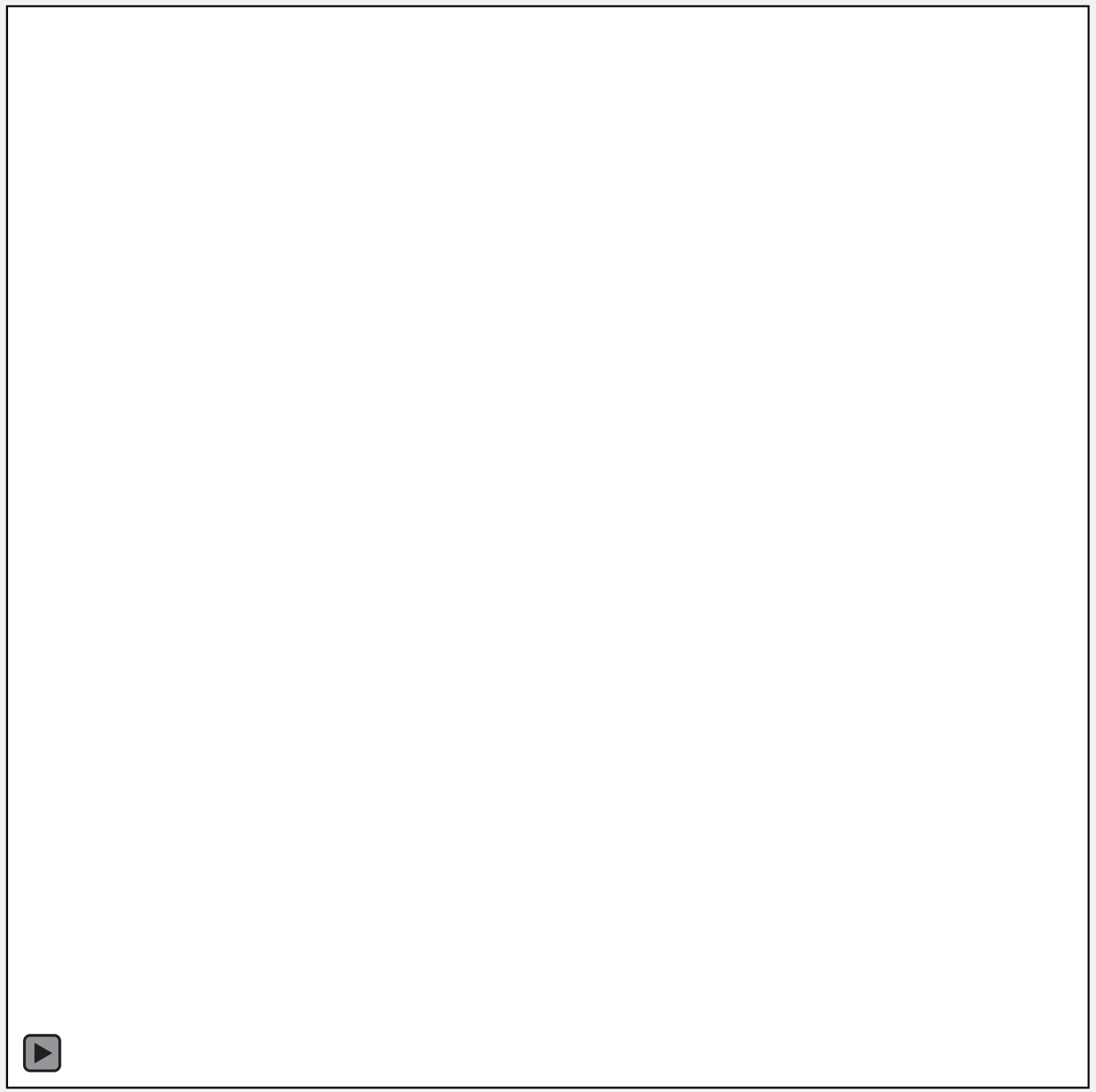
X T S R P O M L F E A

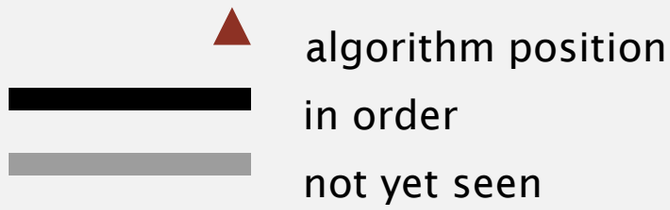
Best case. If the array is in ascending order, insertion sort makes $n - 1$ compares and 0 exchanges.

A E E L M O P R S T X

Insertion sort: animation

40 partially sorted items





http://www.sorting-algorithms.com/insertion-sort

Insertion sort: partially sorted arrays

Def. An inversion is a pair of keys that are out of order.

A E E L M O T R X P S

T-R T-P T-S R-P X-P X-S
(6 inversions)

Def. A family of arrays is partially sorted if the number of inversions is $\leq c\, n$.

- Ex 1. A sorted array has 0 inversions.
- Ex 2. A subarray of size 10 appended to a sorted subarray of size n.

Proposition. Insertion sort runs in linear time on partially sorted arrays.

Pf. Number of exchanges in insertion sort = number of inversions.

number of compares ≤ exchanges + (n – 1)

Stability

A typical application. First, sort by name; then sort by section.

Selection.sort(a, Student.byNameOrder());

Andrews	3	A	(664) 480-0023	097 Little
Battle	4	C	(874) 088-1212	121 Whitman
Chen	3	A	(991) 878-4944	308 Blair
Fox	3	A	(884) 232-5341	11 Dickinson
Furia	1	A	(766) 093-9873	101 Brown
Gazsi	4	B	(800) 867-5309	101 Brown
Kanaga	3	B	(898) 122-9643	22 Brown
Rohde	2	A	(232) 343-5555	343 Forbes

Selection.sort(a, Student.bySectionOrder());

Furia	1	A	(766) 093-9873	101 Brown
Rohde	2	A	(232) 343-5555	343 Forbes
Chen	3	A	(991) 878-4944	308 Blair
Fox	3	A	(884) 232-5341	11 Dickinson
Andrews	3	A	(664) 480-0023	097 Little
Kanaga	3	B	(898) 122-9643	22 Brown
Gazsi	4	B	(800) 867-5309	101 Brown
Battle	4	C	(874) 088-1212	121 Whitman

@#%&@! Students in section 3 no longer sorted by name.

A stable sort preserves the relative order of items with equal keys.

Elementary sorts: quiz 5

Which sorting algorithm(s) are stable?

A. Selection sort.

B. Insertion sort.

C. Both A and B.

D. Neither A nor B.

Stability: insertion sort

Proposition. Insertion sort is stable.

```
public class Insertion
{
   public static void sort(Comparable[] a)
   {
      int n = a.length;
      for (int i = 0; i < n; i++)
         for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
            exch(a, j, j-1);
   }
}
```

i	j	0	1	2	3	4
0	0	B_1	A_1	A_2	A_3	B_2
1	0	A_1	B_1	A_2	A_3	B_2
2	1	A_1	A_2	B_1	A_3	B_2
3	2	A_1	A_2	A_3	B_1	B_2
4	4	A_1	A_2	A_3	B_1	B_2
		A_1	A_2	A_3	B_1	B_2

Pf. Equal items never move past each other.

Stability: selection sort

Proposition. Selection sort is not stable.

```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int n = a.length;
      for (int i = 0; i < n; i++)
      {
         int min = i;
         for (int j = i+1; j < n; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }
}
```

i	min	0	1	2
0	2	B_1	B_2	A
1	1	A	B_2	B_1
2	2	A	B_2	B_1
		A	B_2	B_1

Pf by counterexample. Long-distance exchange can move an equal item past another one.