Sorting Algorithms I

Mark Matthews PhD

Summary

- Why sorting is important
- Sorting in java
- Selection Sort
- Types of sorting algorithms
- Insertion Sort (maybe next time)

Why is sorting important?

- Sorted data powers almost everything
- More efficient to work with a sorted dataset
- More human-friendly also
- Enables much more efficient search algorithms (which we'll see soon!)

Why is sorting important?

Life without sorting

Find ‘1’.

[38, 2, 8, 48, 27, 43, 23, 3, 17, 33, 1]

Worse case scenario we have to look through the whole list

Time complexity: O(N) or linear

What do we mean by sorting?

Organising a set of similar items in a collection by some property

Items need to be homogenous

e.g., of the same type

Can be in increasing or decreasing order

e.g., 1,2,3,4 or 4,3,2,1

Sort by shared property

e.g., lexicographical (alphabetical),

numerical order, date etc.

Sorting problem

Ex. Student records in a university.

item →

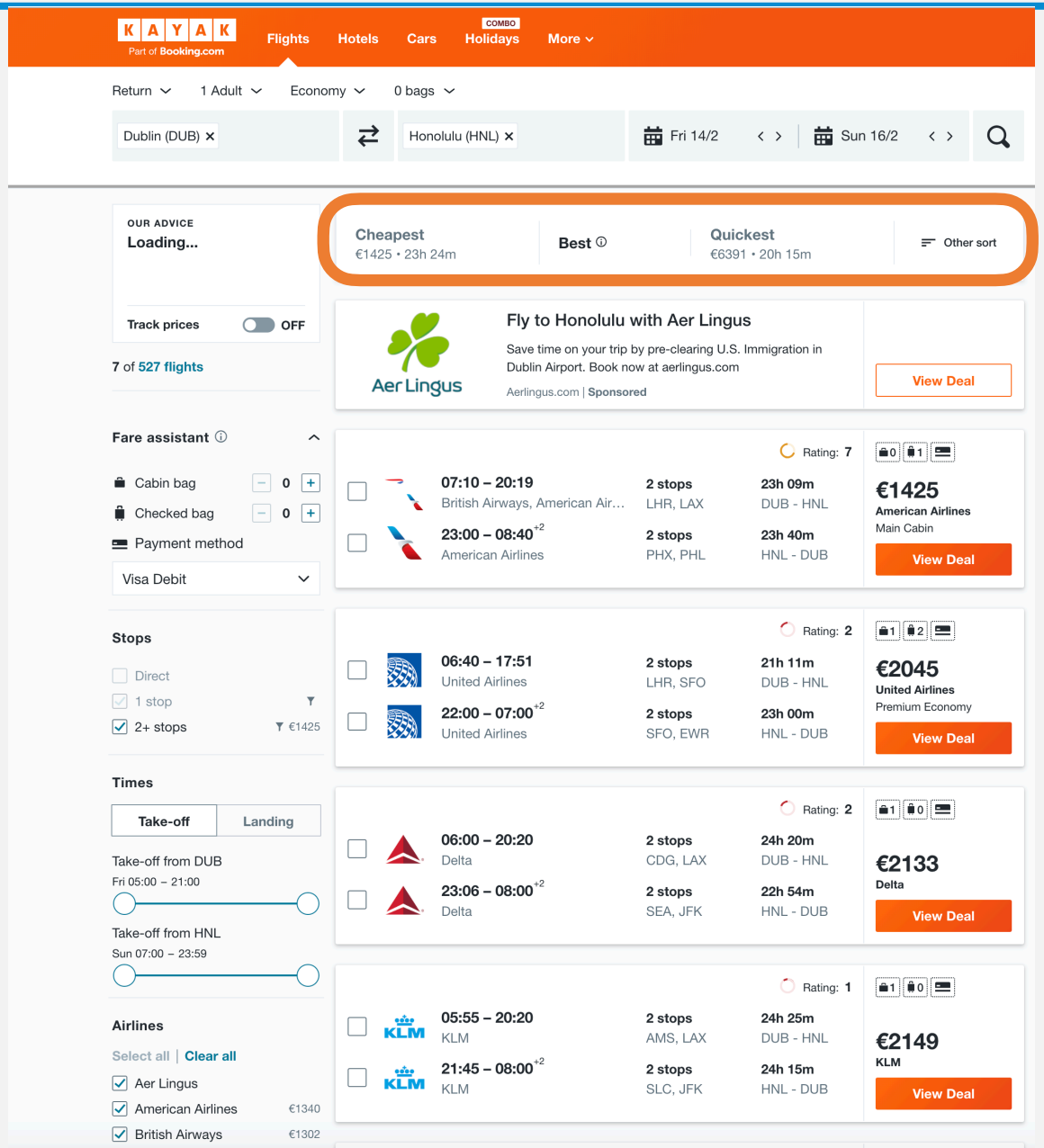
Chen	3	A	991-878-4944	308 Blair
Rohde	2	A	232-343-5555	343 Forbes
Gazsi	4	B	766-093-9873	101 Brown
Furia	**1**	**A**	**766-093-9873**	**101 Brown**
Kanaga	3	B	898-122-9643	22 Brown
Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman

key →

Sort. Rearrange array of N items into ascending order.

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	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

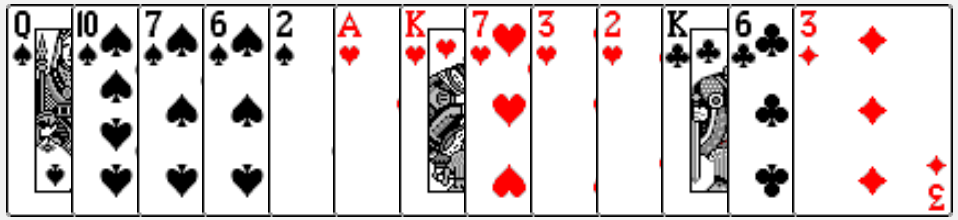
Sorting applications



Sorting applications

Postal packages

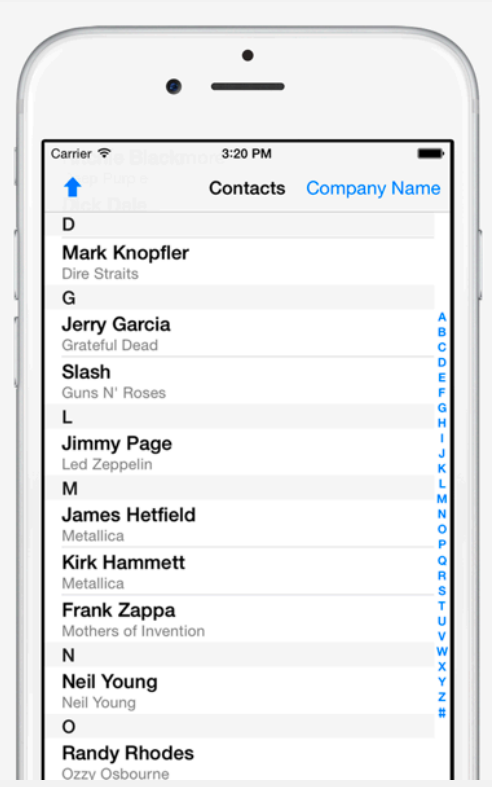
Hogwarts houses



playing cards

Databases, large datasets

Contacts



Sorting in Java

Sample sort client 1

Goal. Sort any type of data.

Ex 1. 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
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

Sample sort client 2

Goal. Sort any type of data.

Ex 2. 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 client 3

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
```

Sorting in Java

Goal. Sort any type of data (for which sorting is well defined).

Q. How can `sort()` know how to compare data of type `Double`, `String`, and `java.io.File` without any information about the type of an item's key?

Callback = reference to executable code.

- Client passes array of objects to `sort()` function.
- The `sort()` function calls object's `compareTo()` method as needed.

Implementing callbacks.

- Java: interfaces.
- C: function pointers.
- C++: class-type functors.
- C#: delegates.
- Python, Perl, ML, Javascript: first-class functions.

Callbacks: roadmap

client

```
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]);
   }
}
```

data-type implementation

```
public class String
implements Comparable<String>
{
   ...
   public int compareTo(String b)
   {
      ...
      return -1;
      ...
      return +1;
      ...
      return 0;
   }
}
```

Comparable interface (built in to Java)

```
public interface Comparable<Item>
{
   public int compareTo(Item that);
}
```

sort implementation

```
public static void sort(Comparable[] a)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (a[j].compareTo(a[j-1]) < 0)
            exch(a, j, j-1);
         else break;
}
```

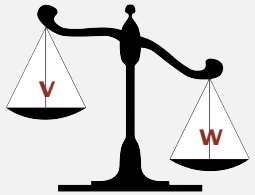
key point: no dependence on `String` data type

Comparable API

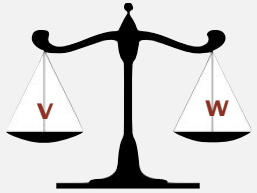
Implement compareTo() so that v.compareTo(w)

- Defines an order (total order)
- Returns a negative integer, zero, or positive integer

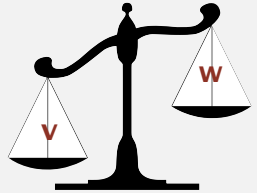
  if v is less than, equal to, or greater than w, respectively.
- Throws an exception if incompatible types (or either is null).



less than (return −1)



equal to (return 0)



greater than (return +1)

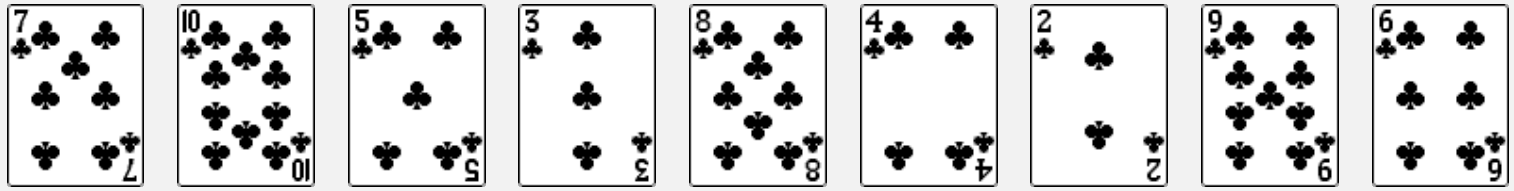
Built-in comparable types. Integer, Double, String, Date, File, ...

User-defined comparable types. Implement the Comparable interface.

Selection Sort

Selection sort demo

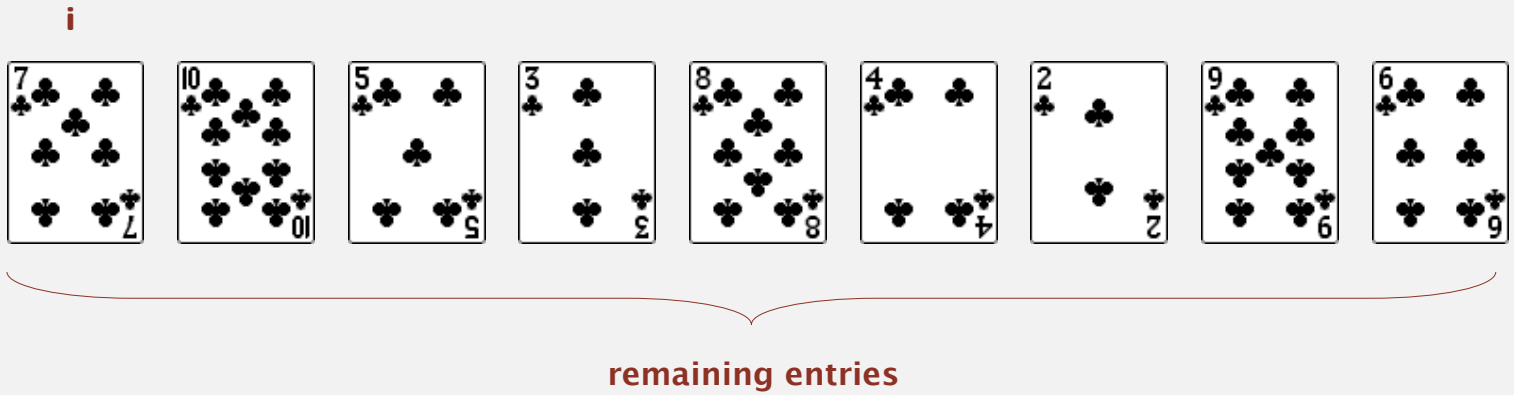
- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



initial

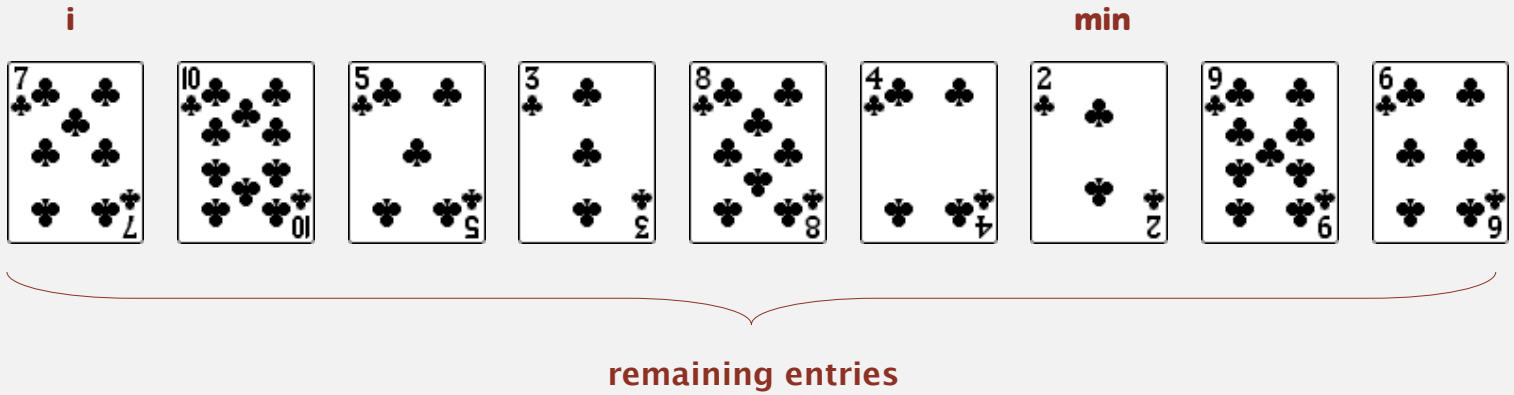
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



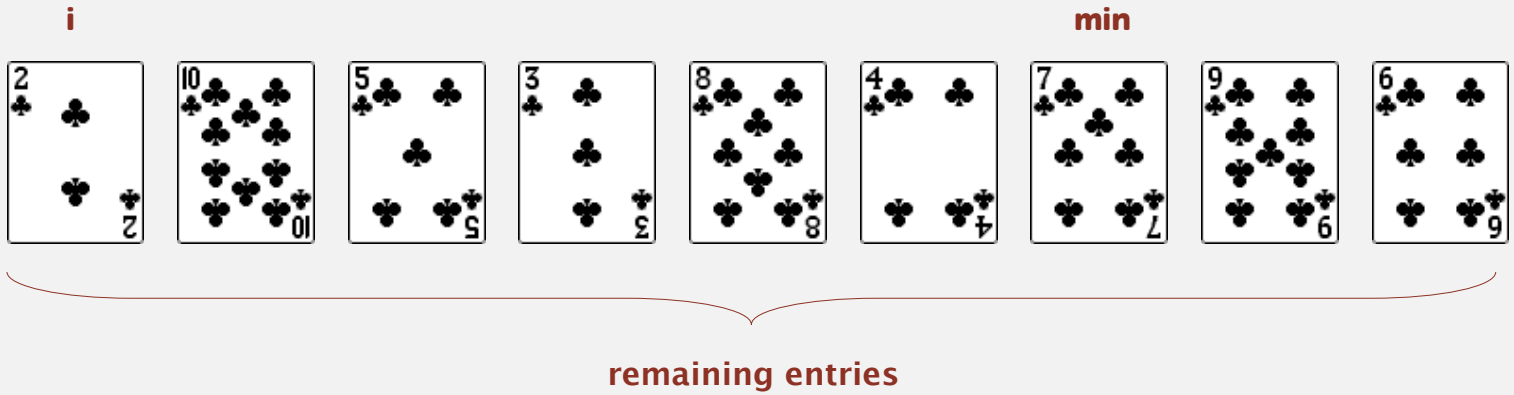
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



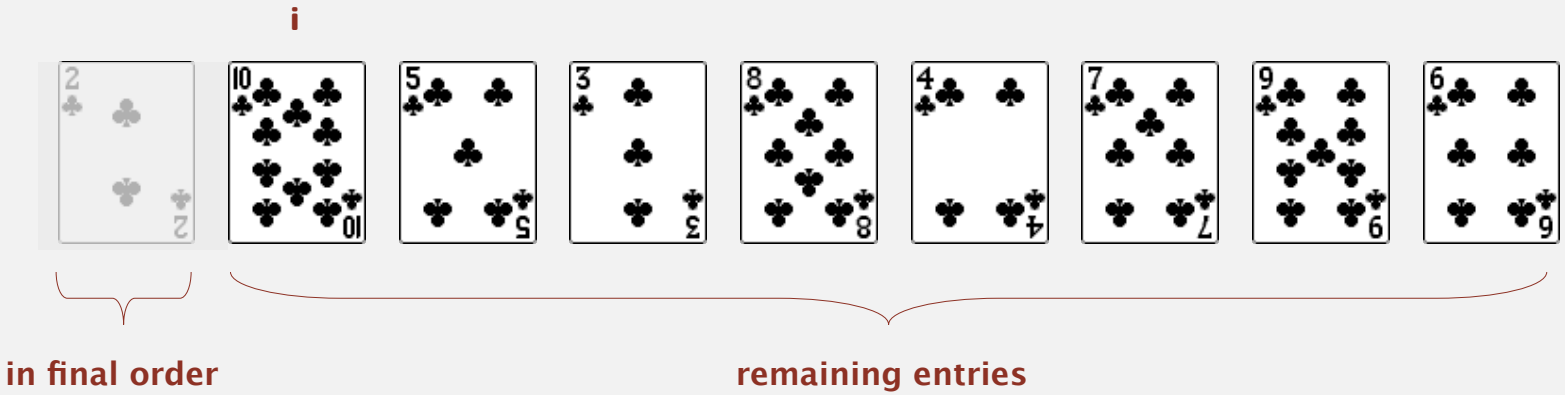
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



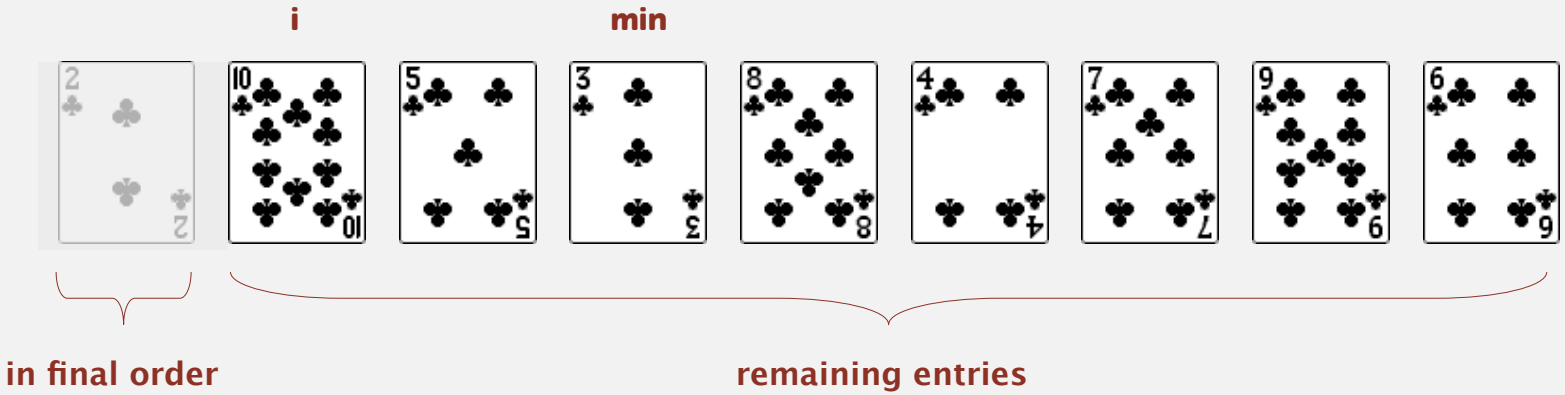
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



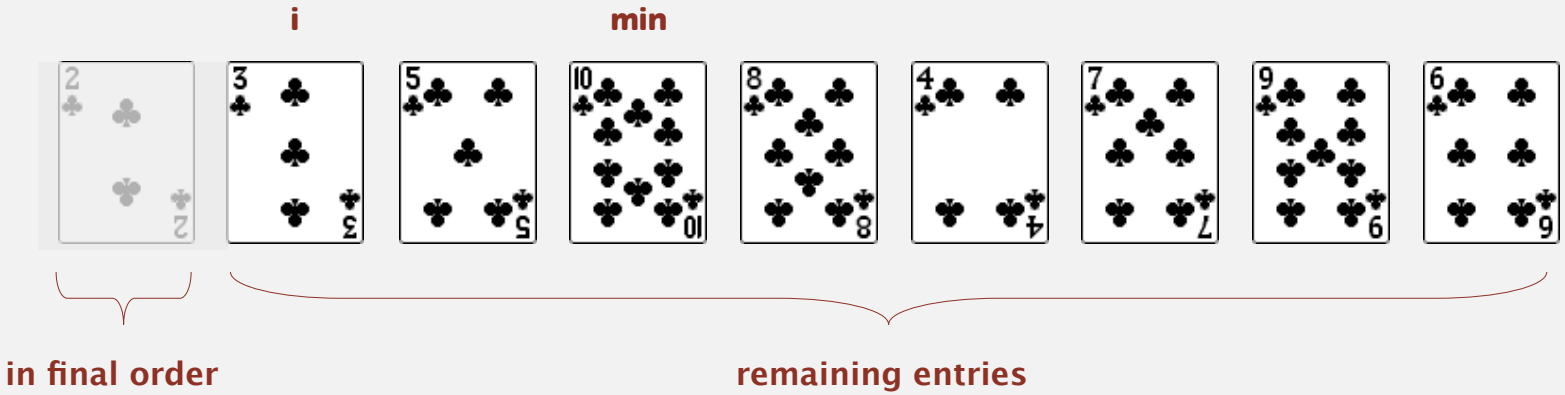
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



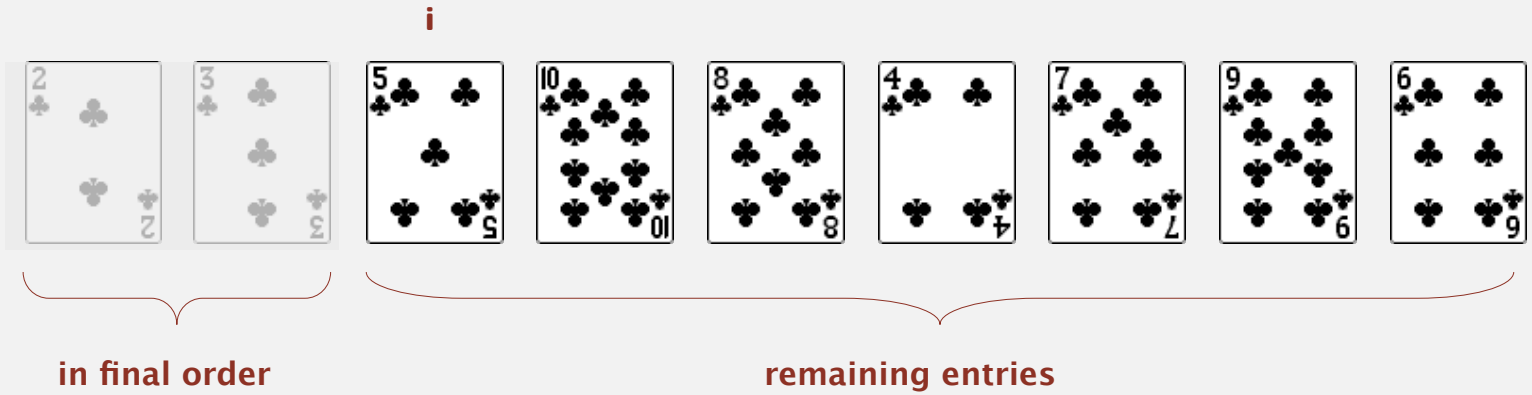
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



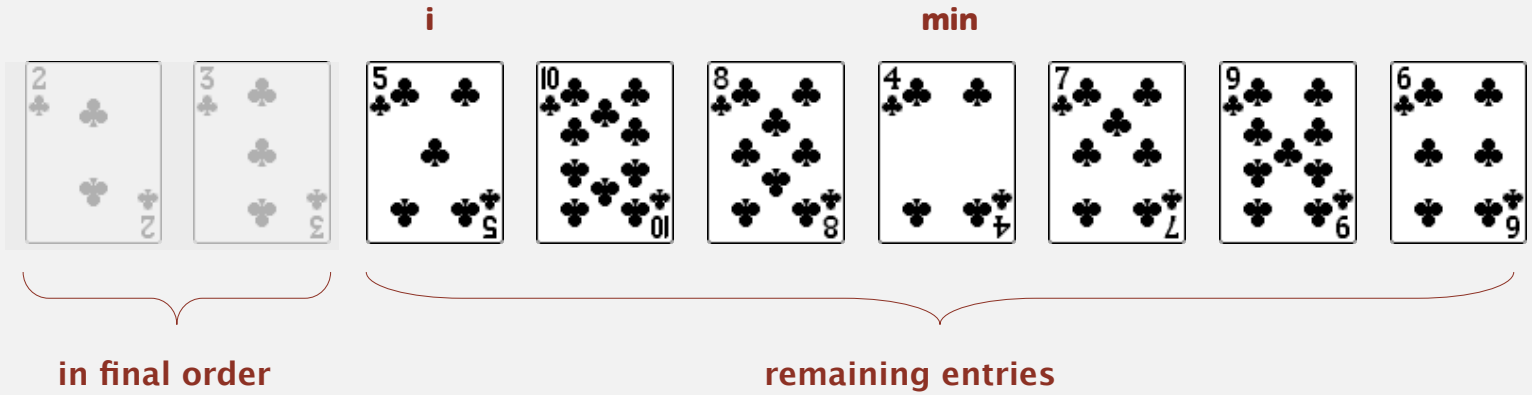
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



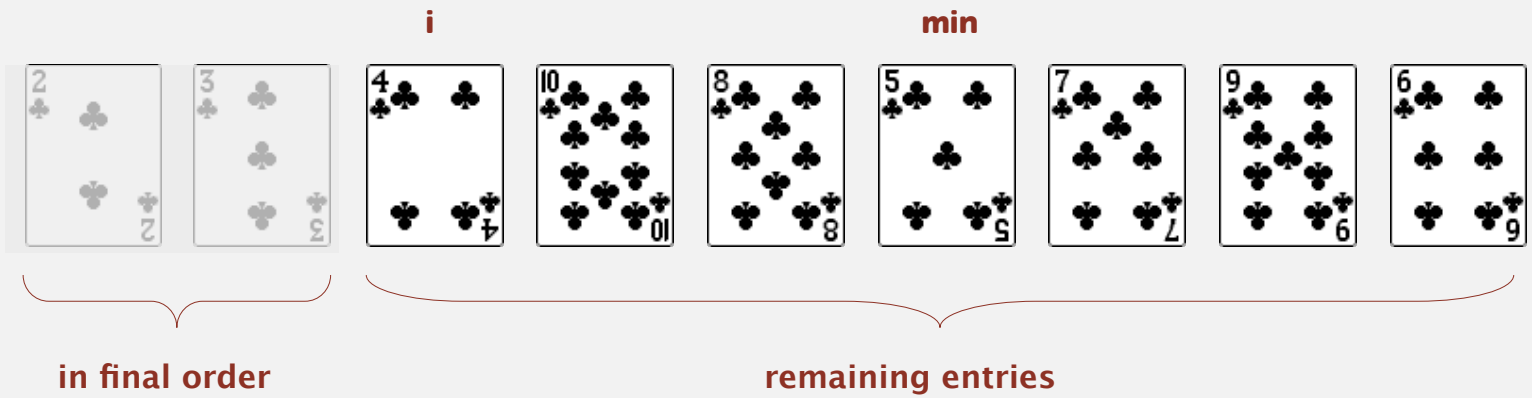
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



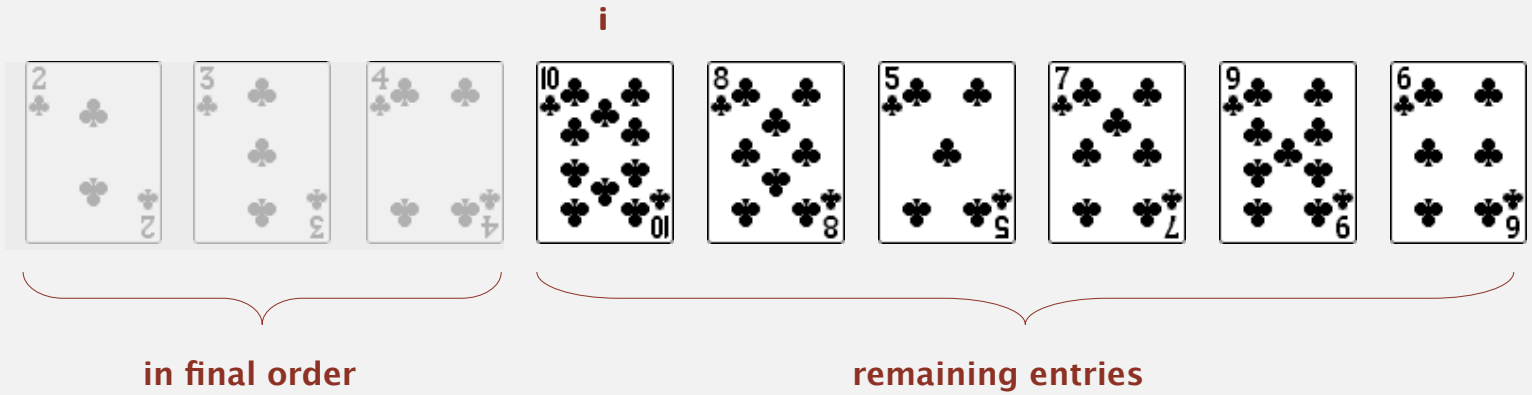
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



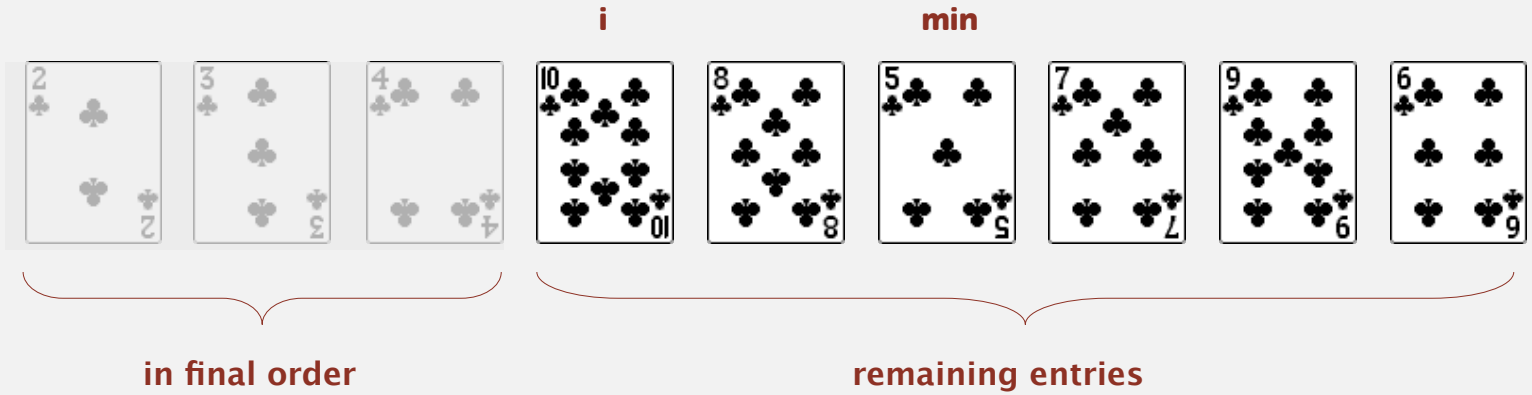
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



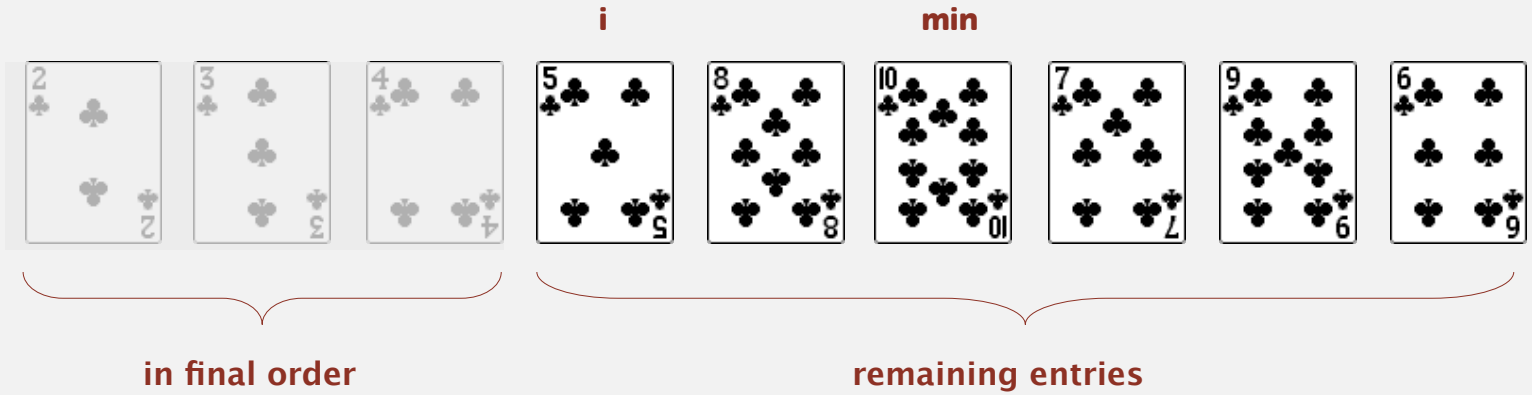
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



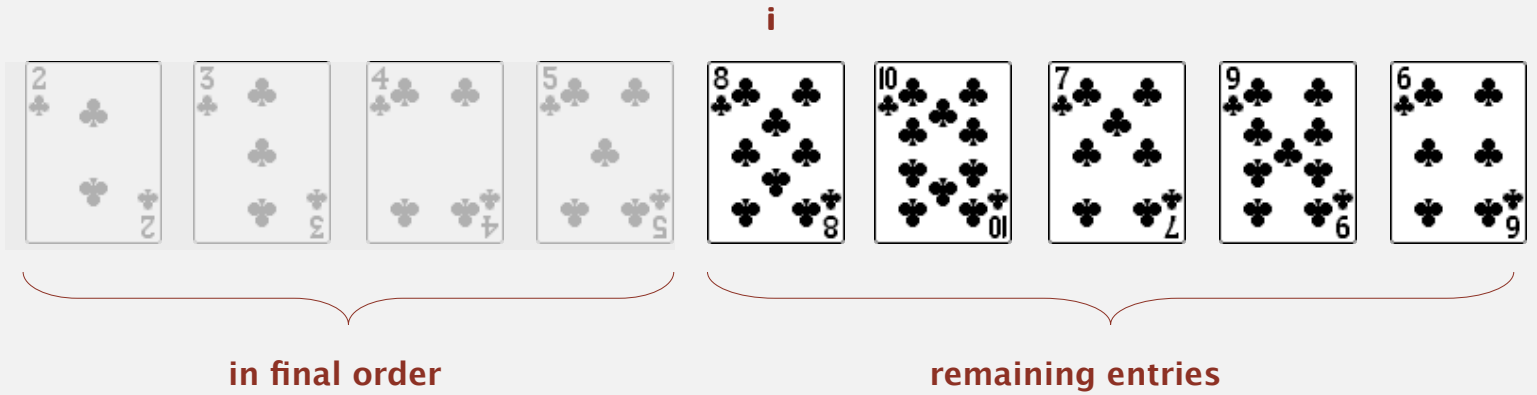
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



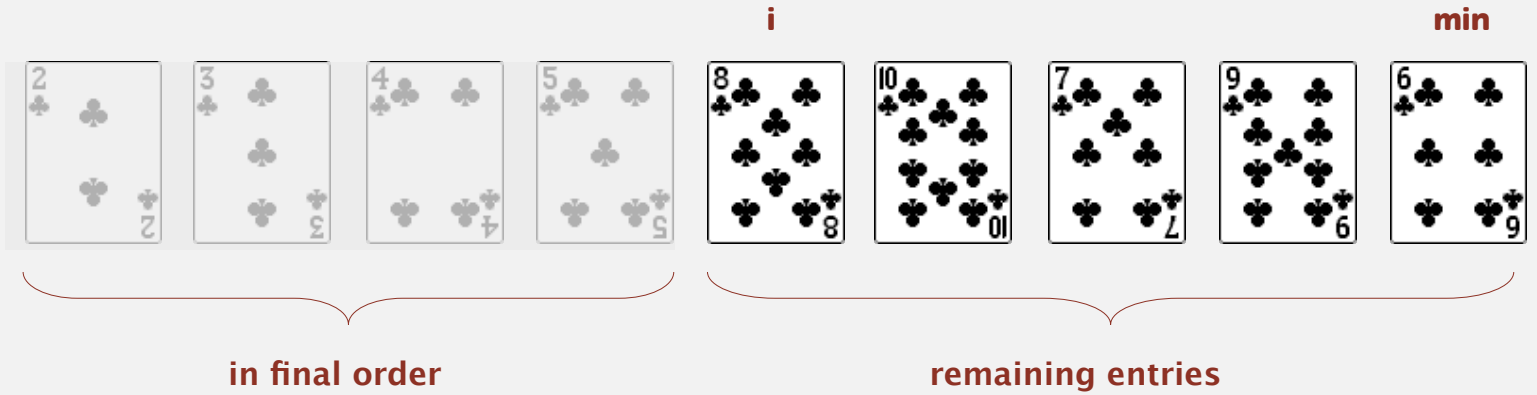
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



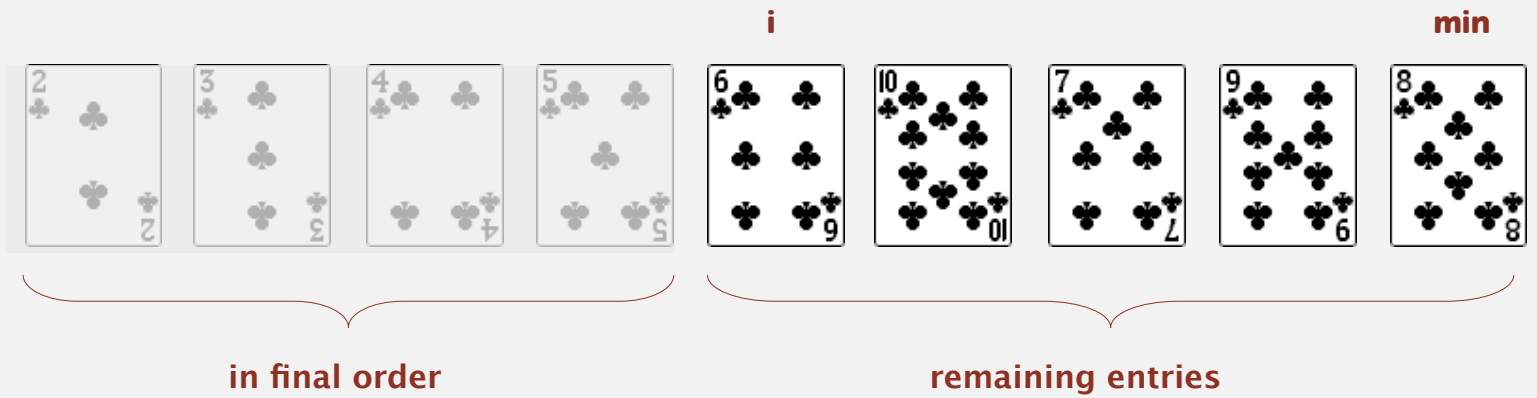
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



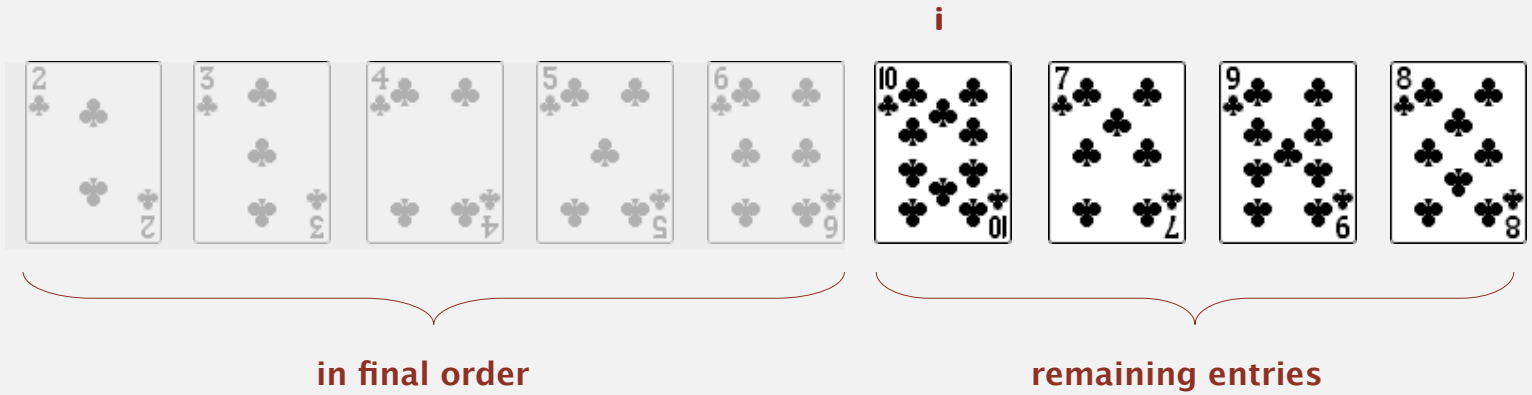
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



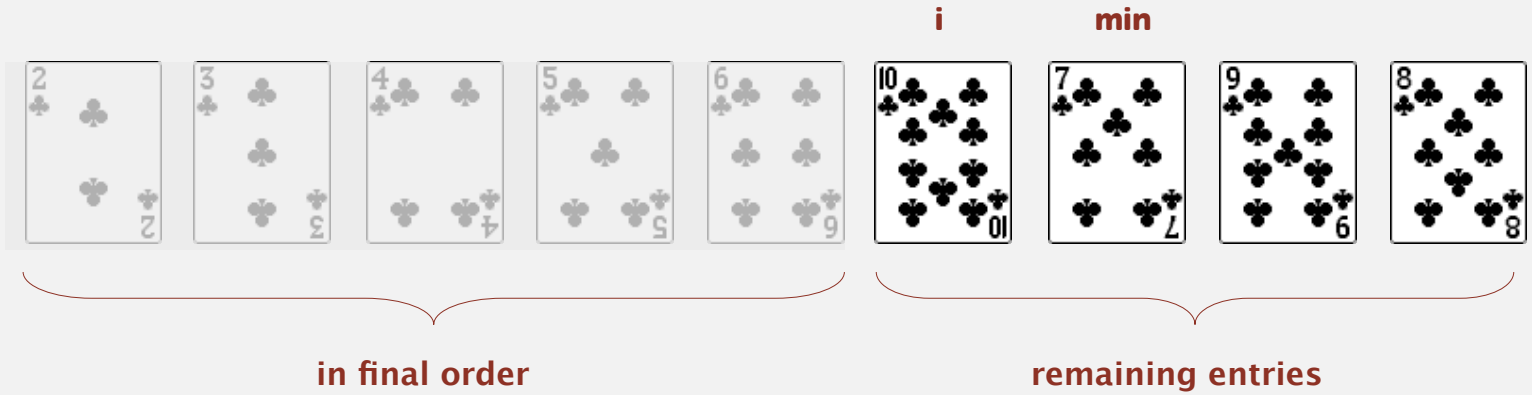
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



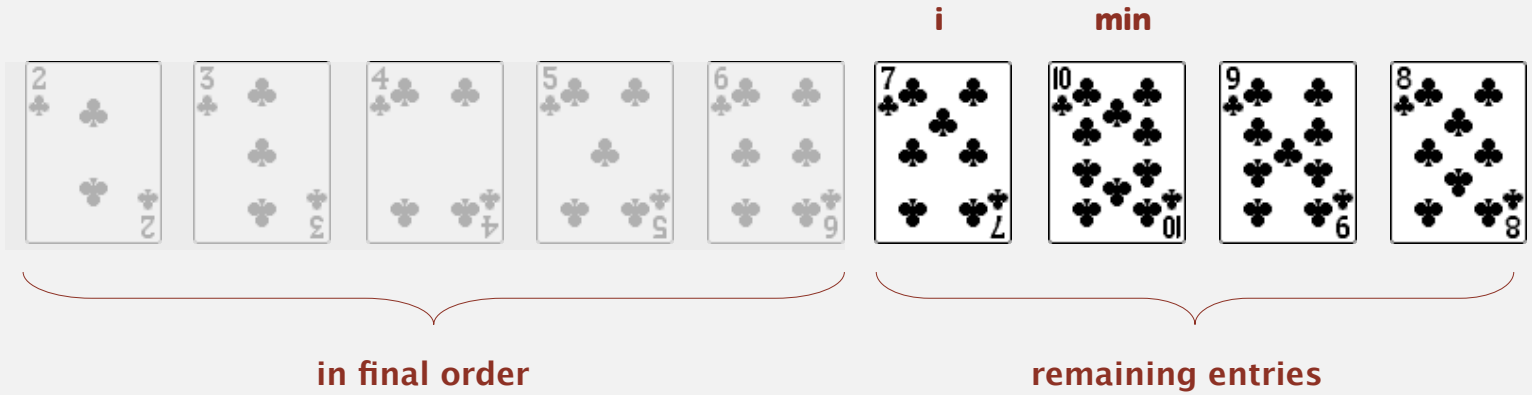
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



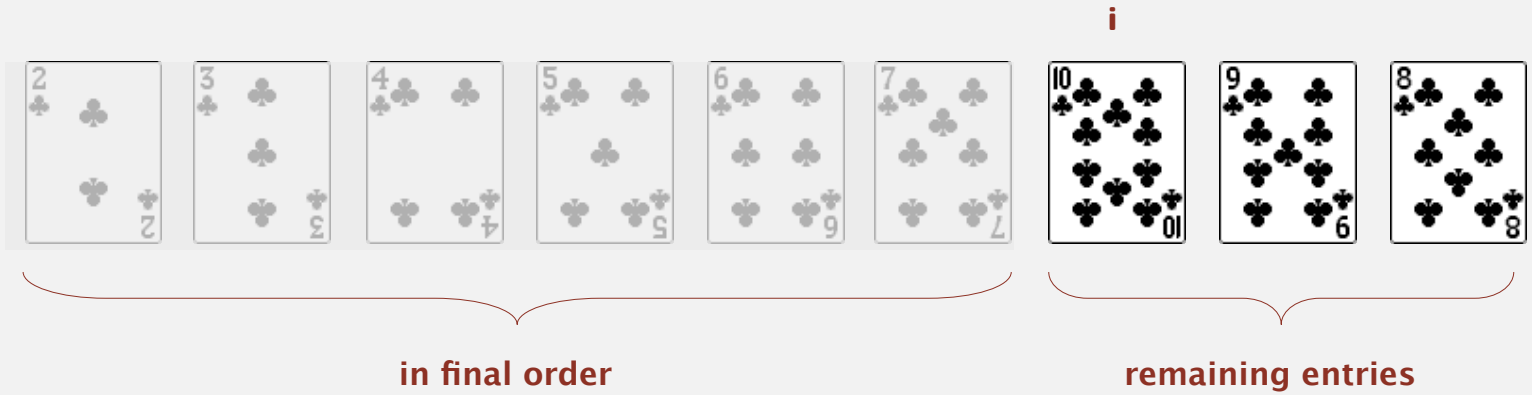
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



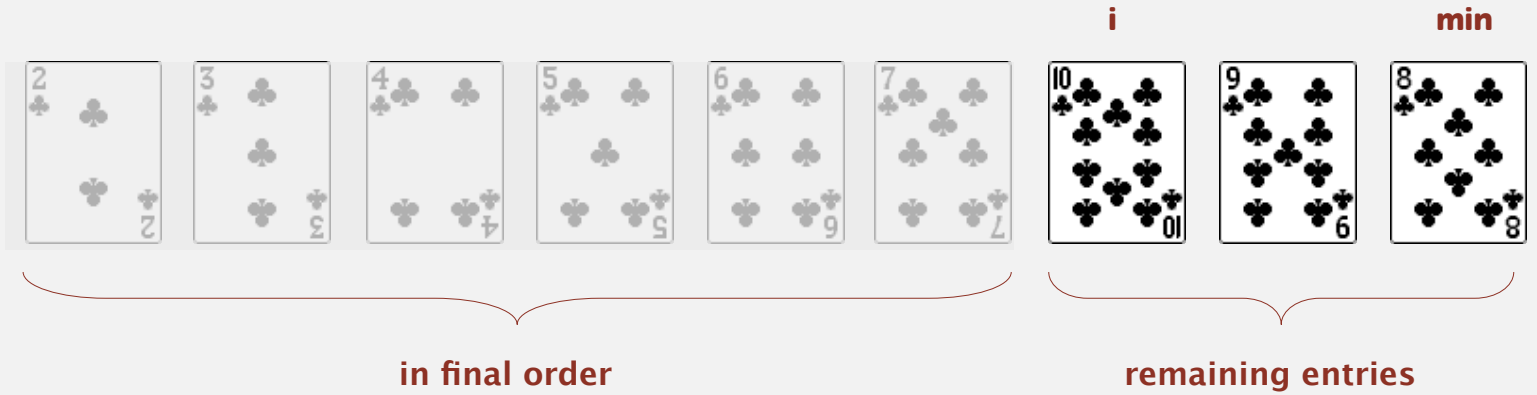
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



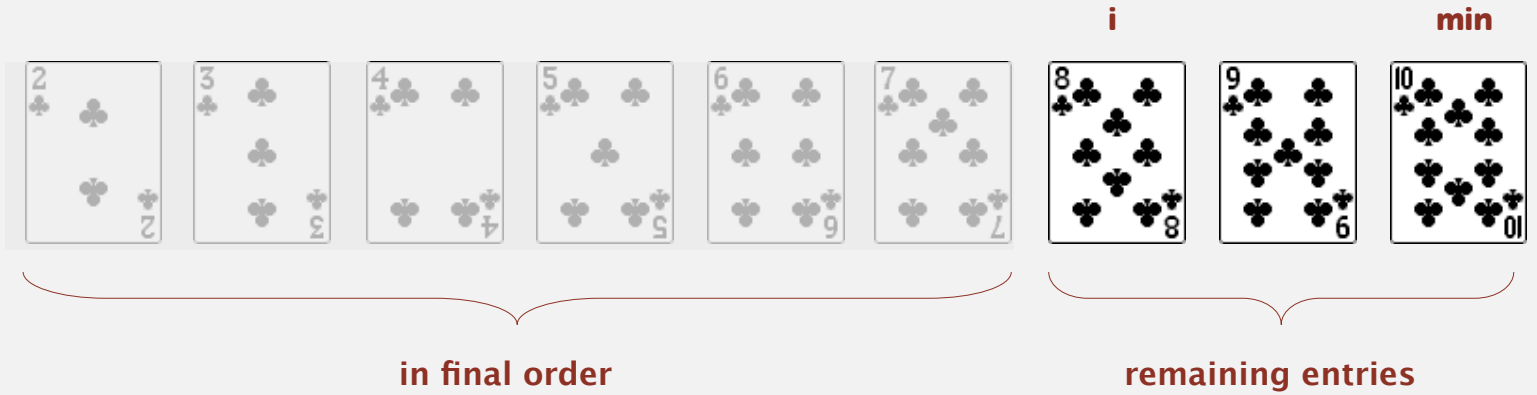
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



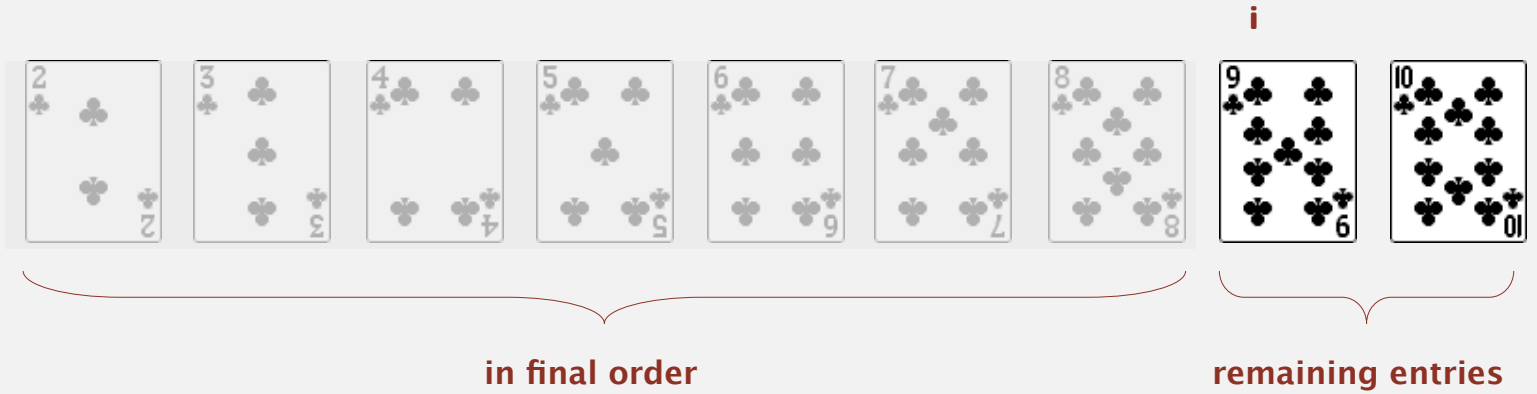
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



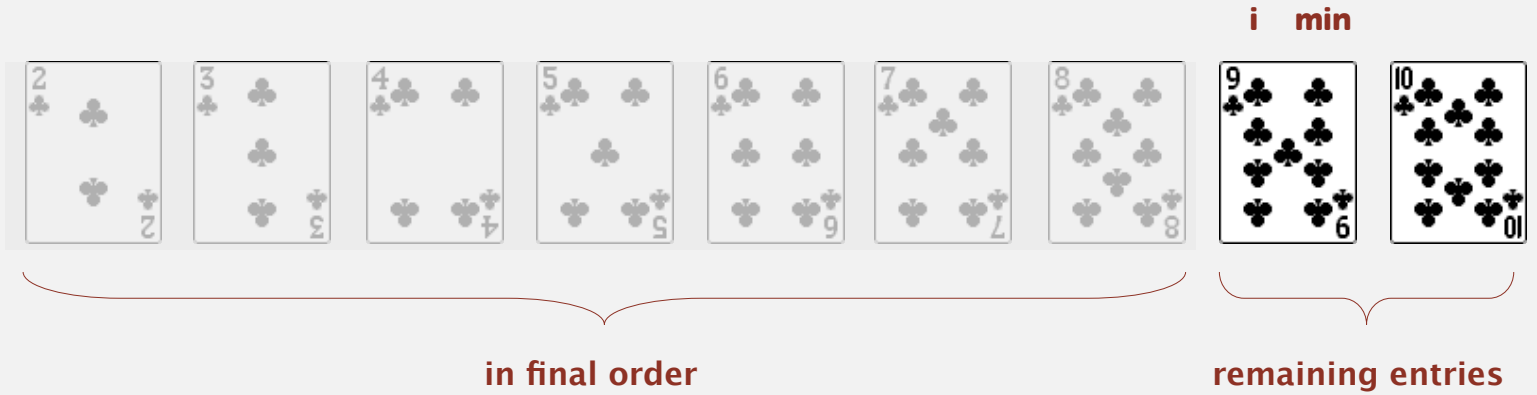
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



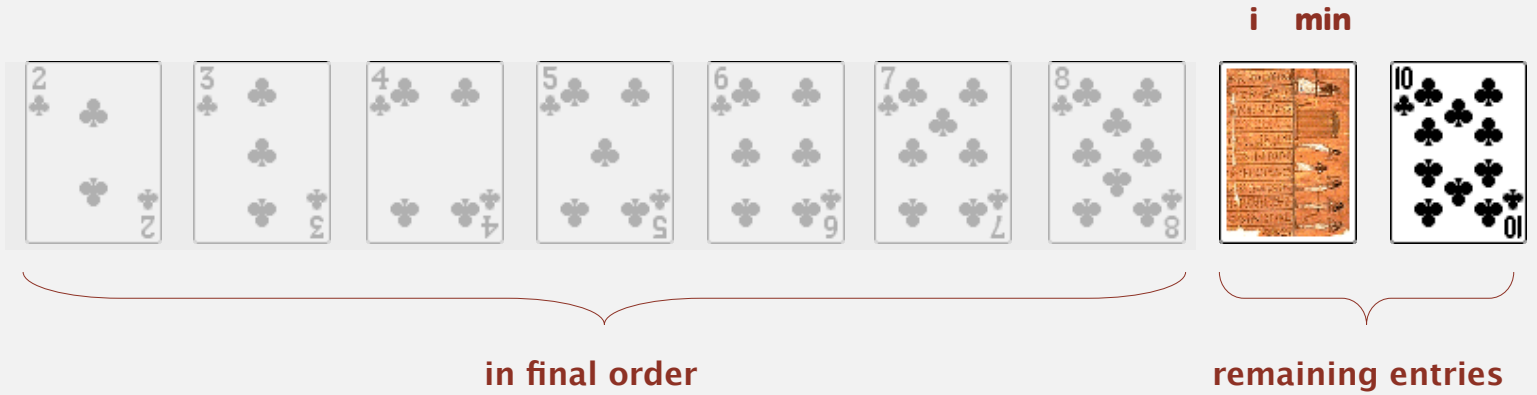
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].

Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].

Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].

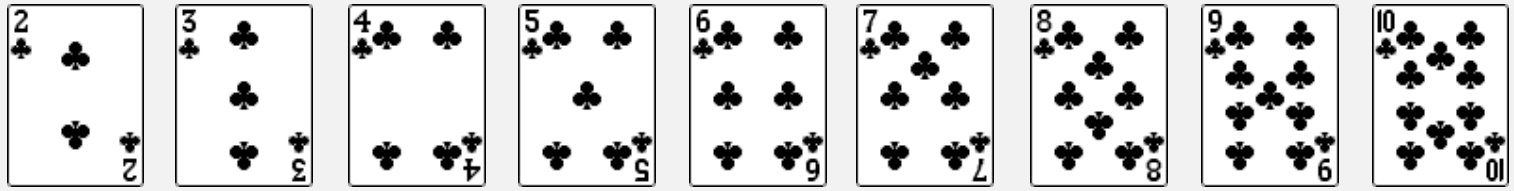
Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].

in final order

Selection sort demo

- In iteration i, find index min of smallest remaining entry.
- Swap a[i] and a[min].



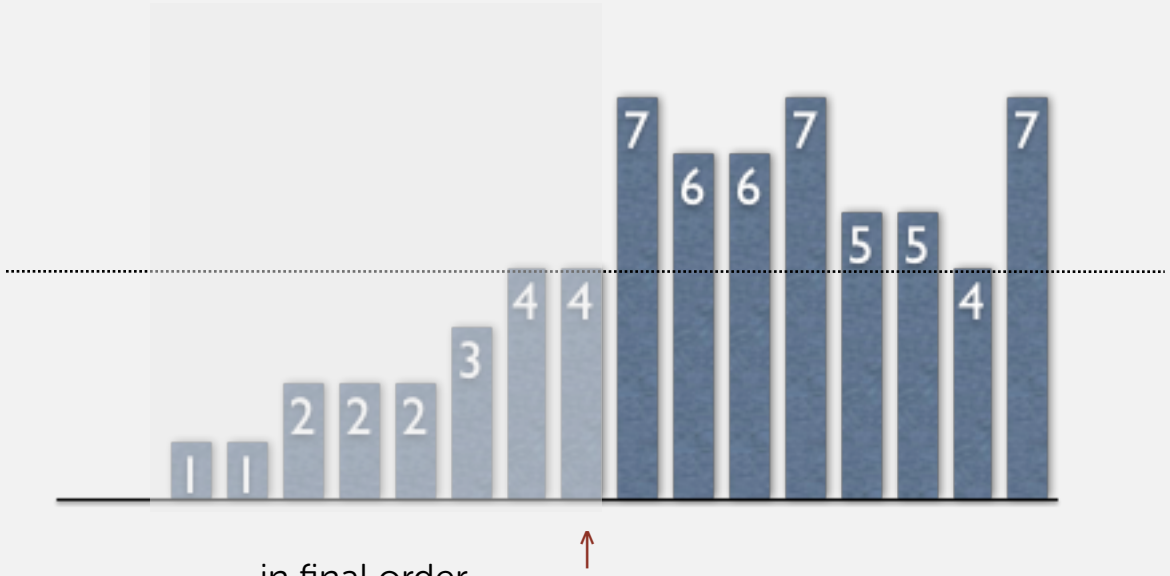
sorted

Selection sort

Algorithm. ↑ scans from left to right.

Invariants.

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

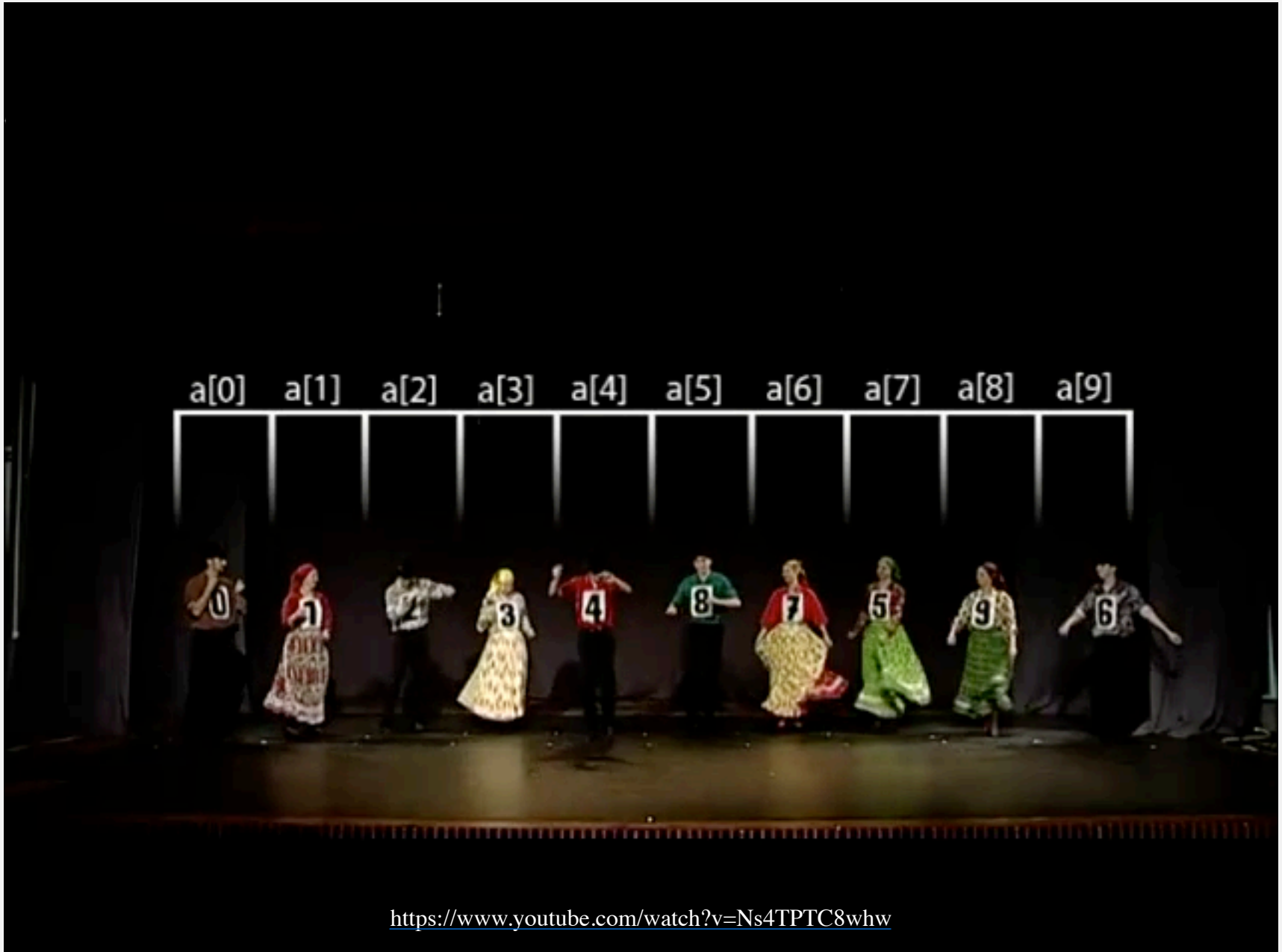


Human Selection Sort

1. Look in the array for the smallest element
2. Swap with the first element from your search
   *Note not always the first element in the array
3. Don't both with the last element, logically that will take care of itself

https://www.youtube.com/watch?v=Ns4TPTC8whw

Selection sort: Gypsy folk dance



https://www.youtube.com/watch?v=Ns4TPTC8whw

Selection sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

```
i++;
```

- Identify index of minimum entry on right.

```
int min_index = i;
for (int j = i+1; j < arr.length; j++) {
        if (arr[min_index] > arr[j])
            min_index = j;
    }
```

- Exchange into position.

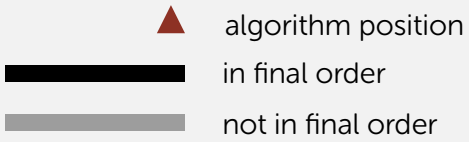
```
arr[i] = arr[min_index];
```

Selection sort: Java implementation

```
void sort(int arr[])
    {
        int temp;
        int min_index;

        // One by one move boundary of unsorted subarray
        for (int i = 0; i < arr.length-1; i++){
            min_index = i;

            // Find the minimum element in unsorted array
            for (int j = i+1; j < arr.length; j++) {
                if (arr[min_index] > arr[j])
                    min_index = j;
            }

            // Swap the found minimum element with the first element
            temp = arr[i];
            arr[i] = arr[min_index];
            arr[min_index] = temp;
        }
    }
```

Selection sort: animations

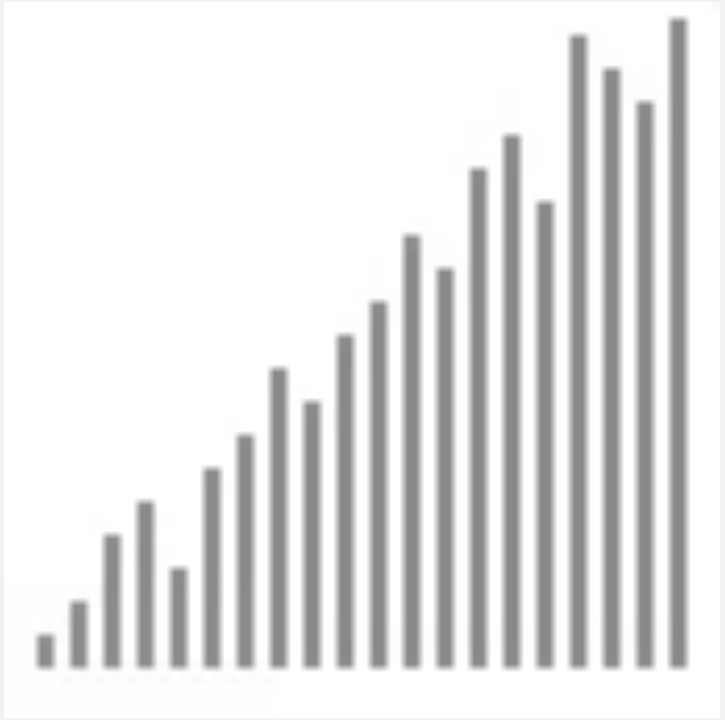
20 random items

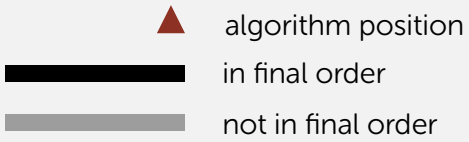


http://www.sorting-algorithms.com/selection-sort

Selection sort: animations

20 partially-sorted items





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

Selection sort: analysis

Proposition. Selection sort uses $(N-1) + (N-2) + \ldots + 1 + 0 \sim N^2 / 2$ compares and N exchanges.

		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)

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

Data movement is minimal. Linear number of exchanges.

Classifying Sorting Algorithms

1. Comparison vs. Non-comparison
   - Comparison sorts compare two items at a time
   - Non-comparison sorts simply ones that do NOT compare two items
2. Time complexity
   - How much time it takes the algorithm based on the size of the input
3. Space complexity (in-place vs. out-of-place)
   - How much memory / space the algorithm will need based on size of the input
4. Stability
   - Stable sorting algorithms preserve the existing relative order of elements when comparing equal keys
5. Internal vs. External
   - Internal sort is one where the items being sorted can be kept in main memory / RAM
   - External sort is one where the items being sorted need to use external memory
6. Recursive vs. Non-recursive
   - Recursive sorting uses a divide & conquer approach splitting up dataset into smaller inputs
   - Non-recursive simply does not use recursion / does not split the dataset up

Selection sort: summary

Comparison sort?	Comparison
Time Complexity	$O(N^2)$
Space Complexity	In place
Internal or External?	Internal
Recursive / Non-recursive?	Non-recursive