SPRING 2023

Information Technology Research

Yi Han

Department of Information Management
National Sun Yat-sen University

Lecture slides are based on the supplemental materials of the textbook: https://algs4.cs.princeton.edu

http://algs4.cs.princeton.edu

2.1 ELEMENTARY SORTS

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

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Sorting problem

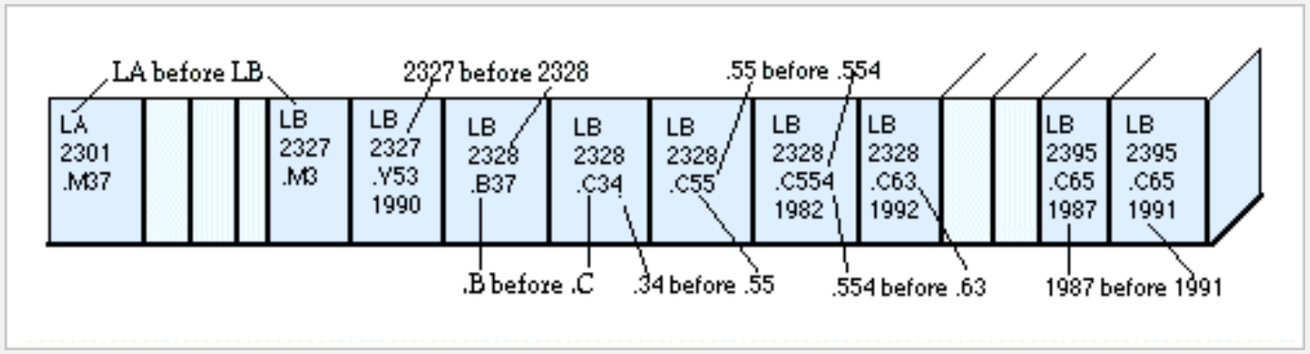
Ex. Student records in a university.

	Chen	3	A	991-878-4944	308 Blair
	Rohde	2	A	232-343-5555	343 Forbes
	Gazsi	4	B	766-093-9873	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 into ascending order.

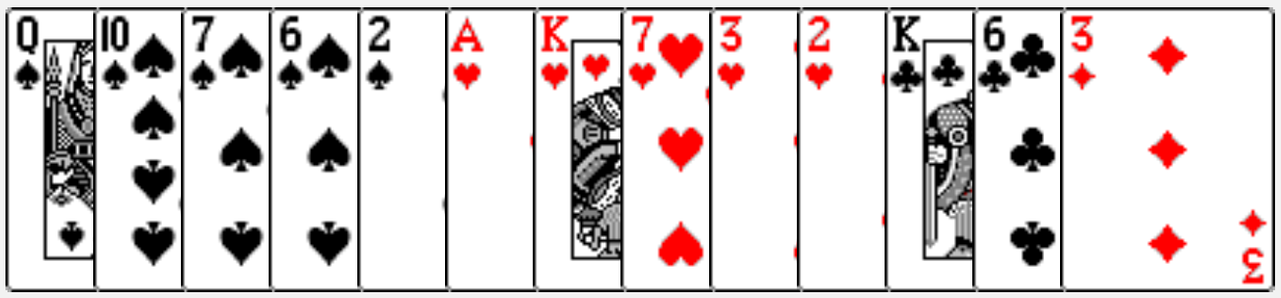
Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

Sorting applications

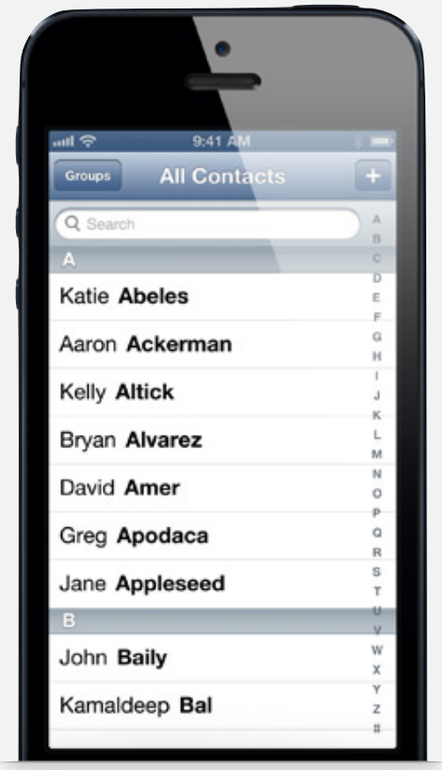


Library of Congress numbers

FedEx packages



playing cards



contacts

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
0.534002631135008 7
0.721612979370349 6
0.900350035441144 3
0.929399490884568 6
```

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
```

Total order

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

A total order is a binary relation $\leq$ that satisfies:

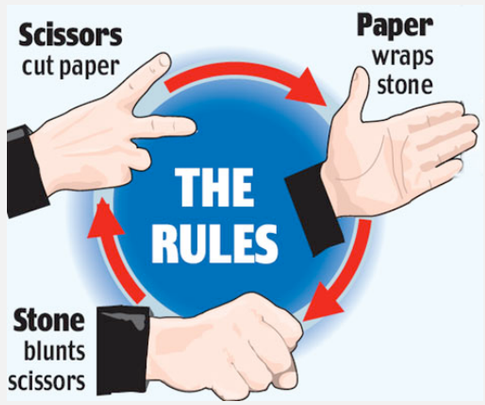
- Antisymmetry: if both $v \leq w$ and $w \leq v$, then $v = w$.
- Transitivity: if both $v \leq w$ and $w \leq x$, then $v \leq x$.
- Totality: either $v \leq w$ or $w \leq v$ or both.

Ex.

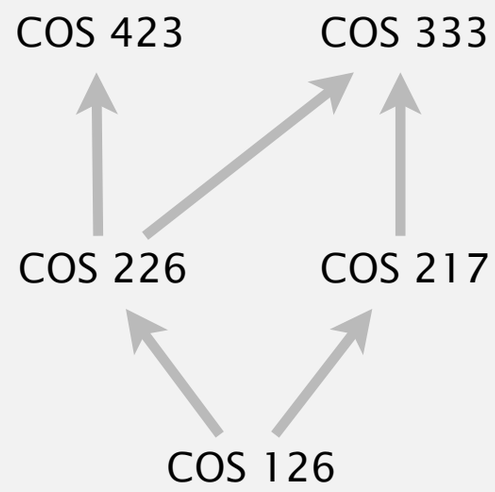
- Standard order for natural and real numbers.
- Chronological order for dates or times.
- Alphabetical order for strings.

No transitivity. Rock-paper-scissors.

No totality. PU course prerequisites.



violates transitivity



violates totality

Callbacks

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.

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

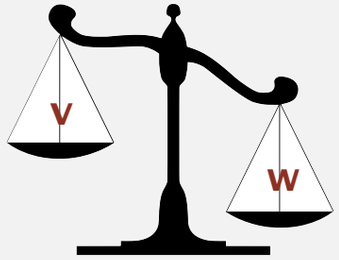
```
public static void sort(Comparable[] a)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (a[j].compareTo(a[j-1]) < 0)
            exch(a, j, j-1);
         else break;
}
```

key point: no dependence on String data type

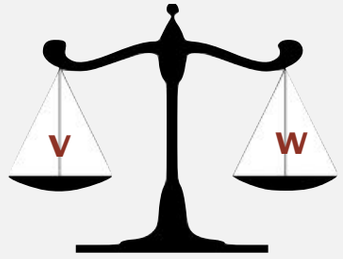
Comparable API

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

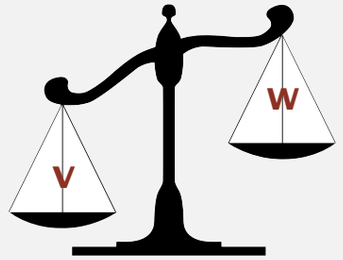
- Defines a total order.
- Returns a negative integer, zero, or positive integer
  if v is less than, equal to, or greater than w, respectively.
- Throws an exception if incompatible types (or either is `null`).



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.

Implementing the Comparable interface

Date data type. Simplified version of `java.util.Date`.

```
public class Date implements Comparable<Date>
{
   private final int month, day, year;

   public Date(int m, int d, int y)
   {
      month = m;
      day   = d;
      year  = y;
   }

   public int compareTo(Date that)
   {
      if (this.year  < that.year ) return -1;
      if (this.year  > that.year ) return +1;
      if (this.month < that.month) return -1;
      if (this.month > that.month) return +1;
      if (this.day   < that.day  ) return -1;
      if (this.day   > that.day  ) return +1;
      return 0;
   }
}
```

only compare dates to other dates

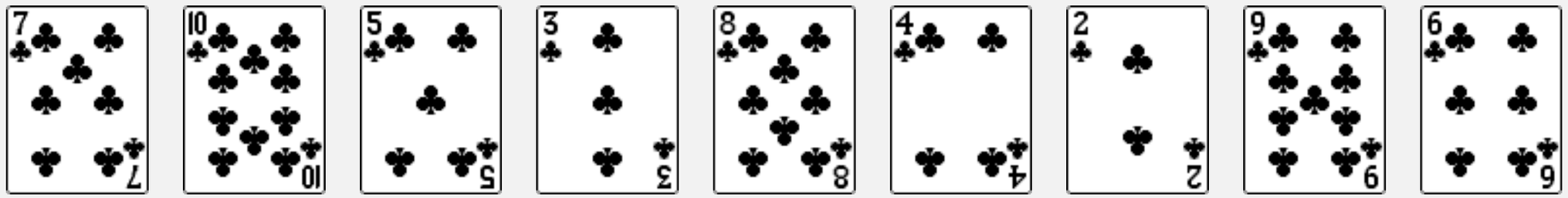
http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Selection sort demo

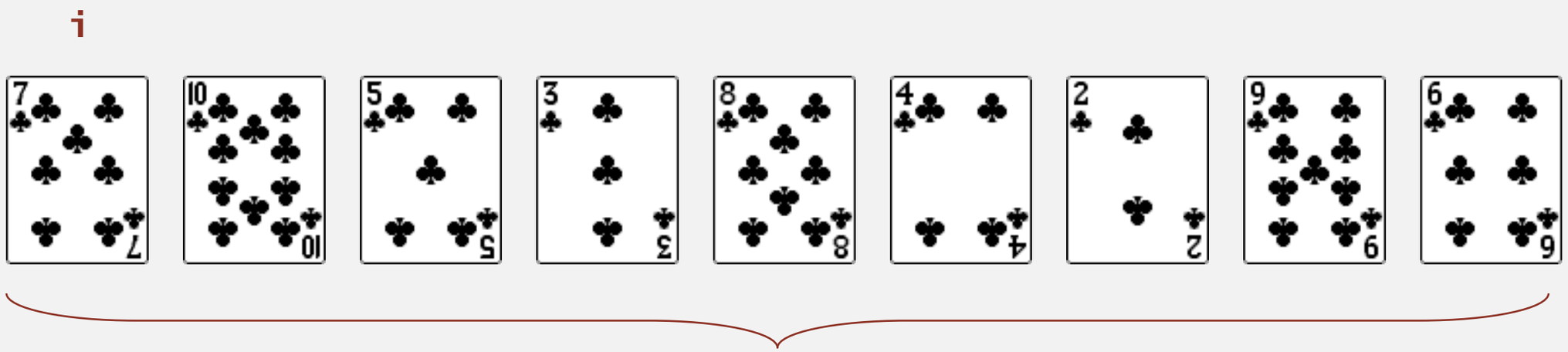
- In iteration `i`, find index `min` of smallest remaining entry.
- Swap `a[i]` and `a[min]`.



initial

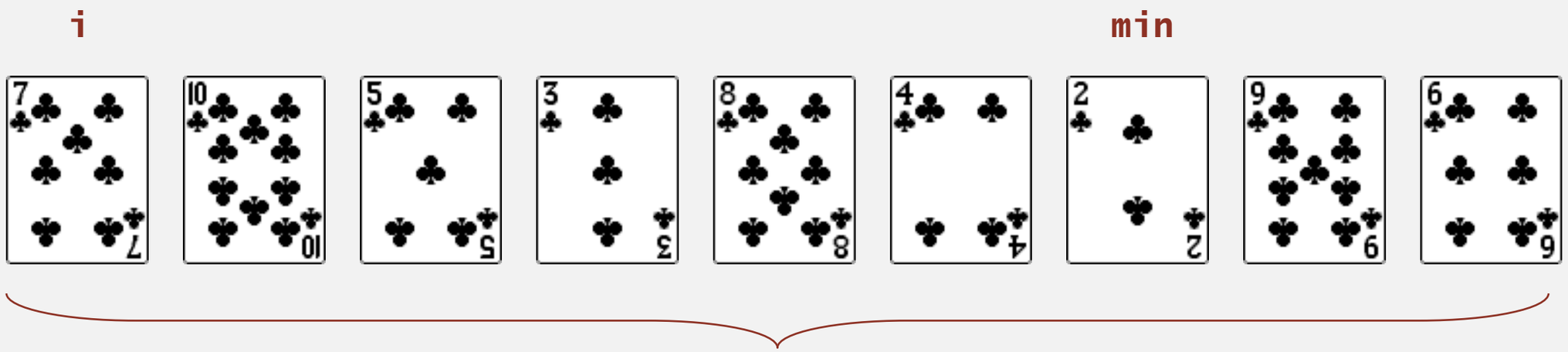
Selection sort demo

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



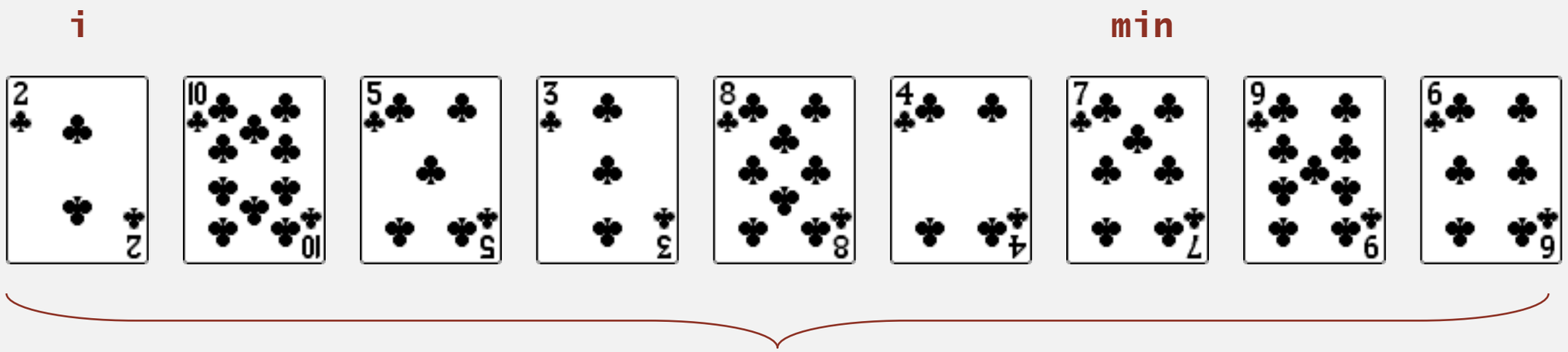
Selection sort demo

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



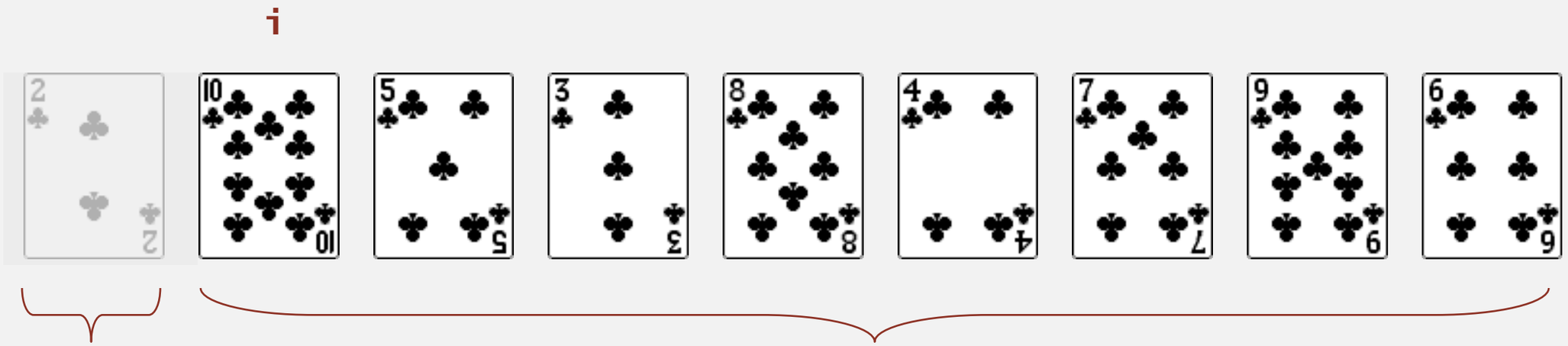
Selection sort demo

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



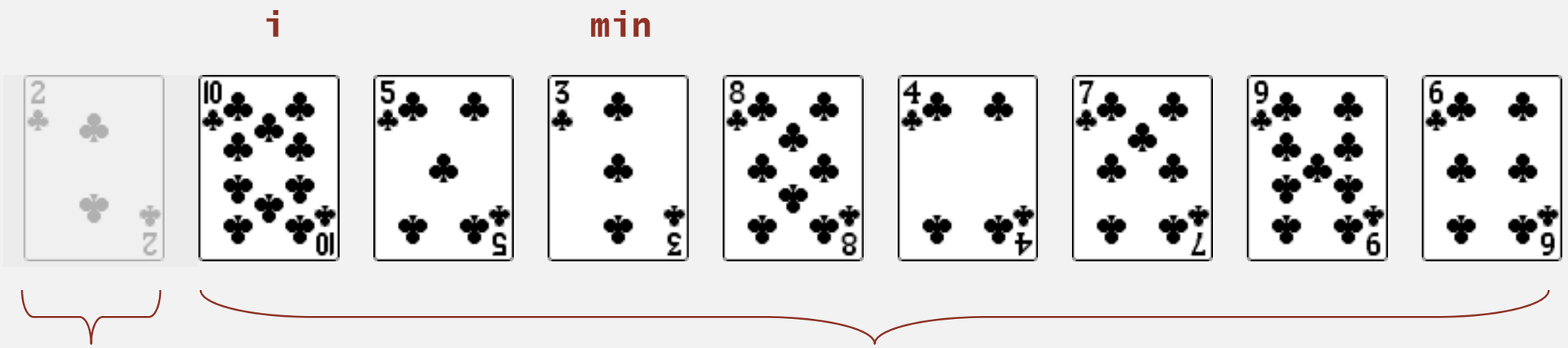
Selection sort demo

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



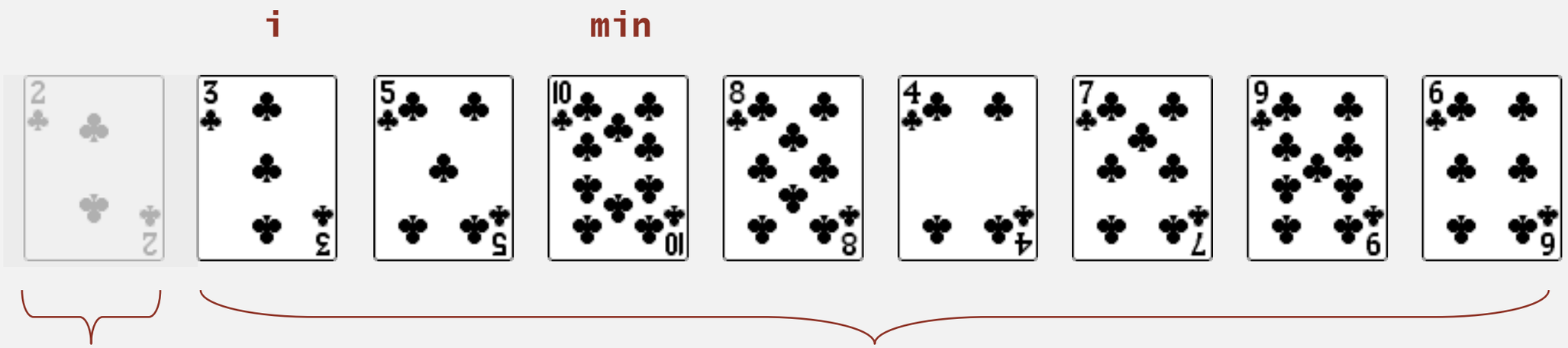
Selection sort demo

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



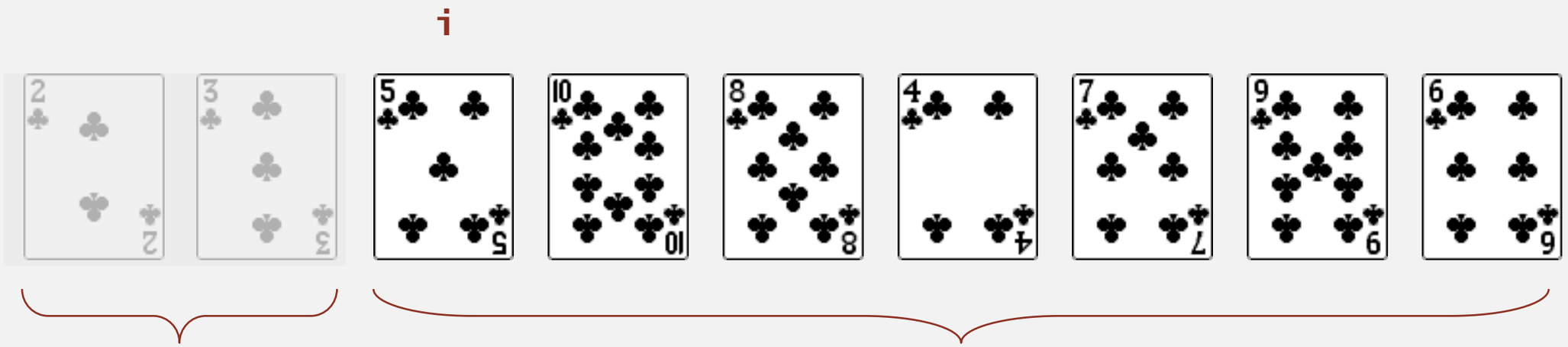
Selection sort demo

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



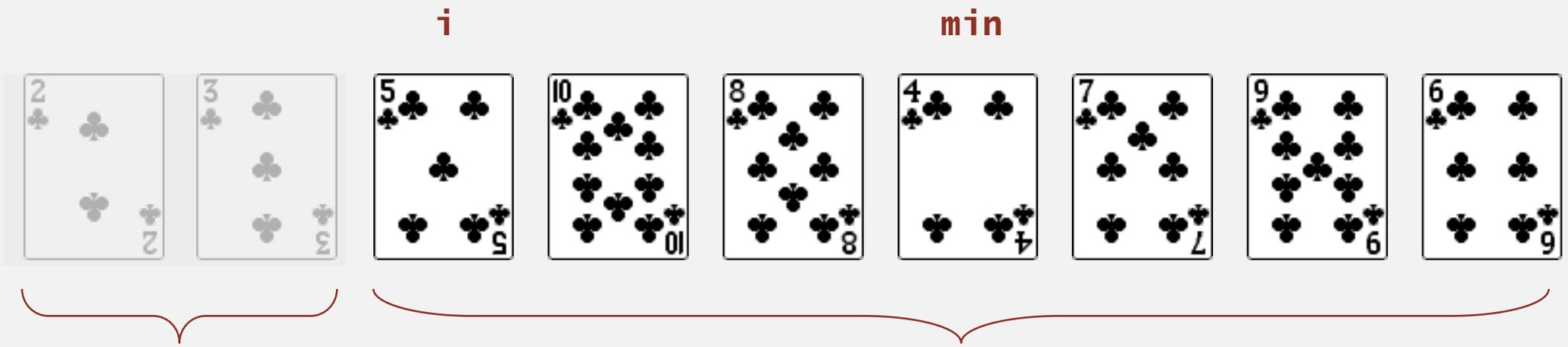
Selection sort demo

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



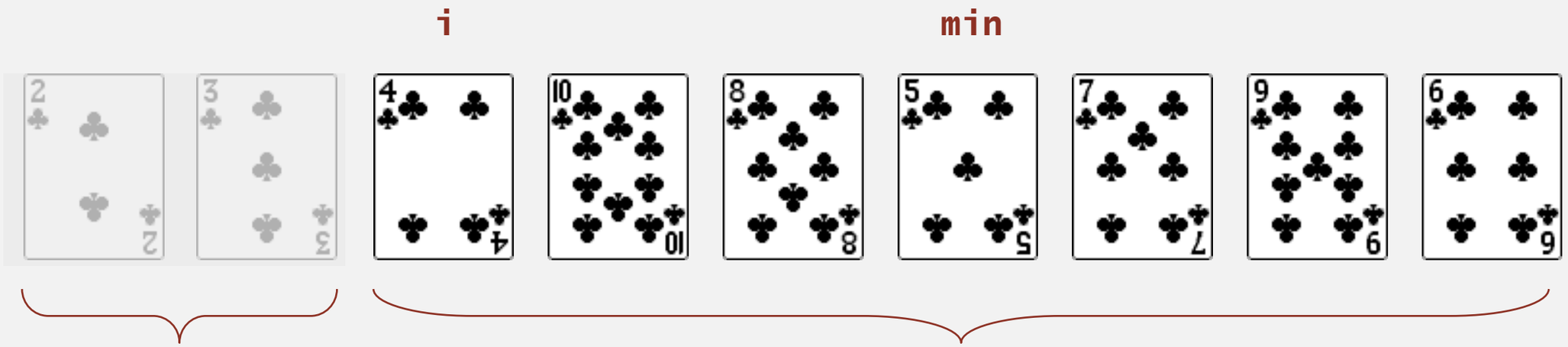
Selection sort demo

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



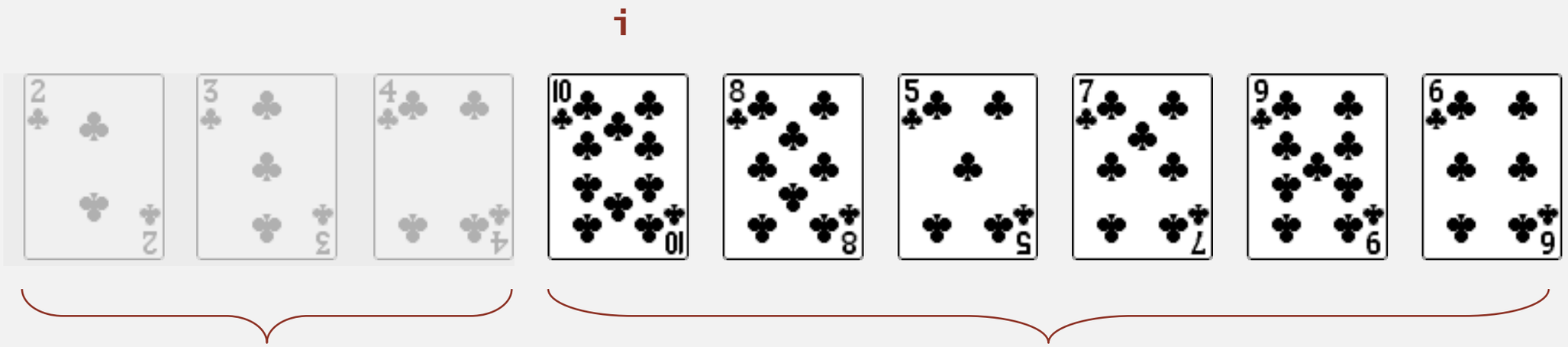
Selection sort demo

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



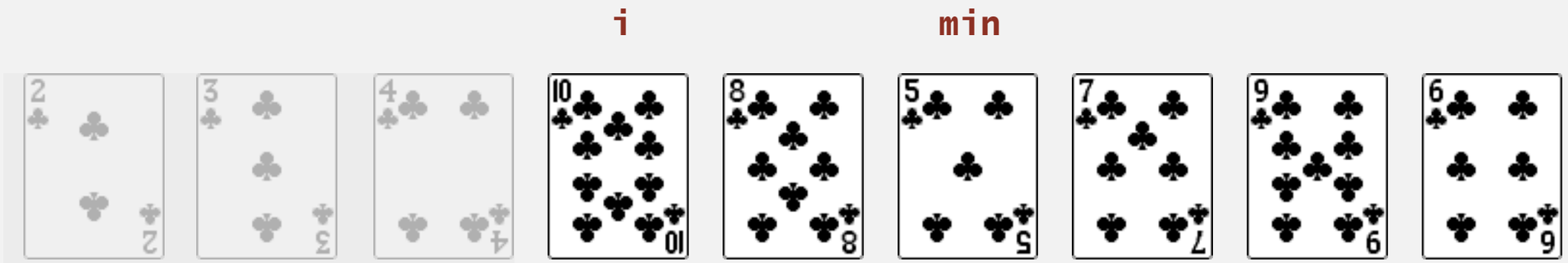
Selection sort demo

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



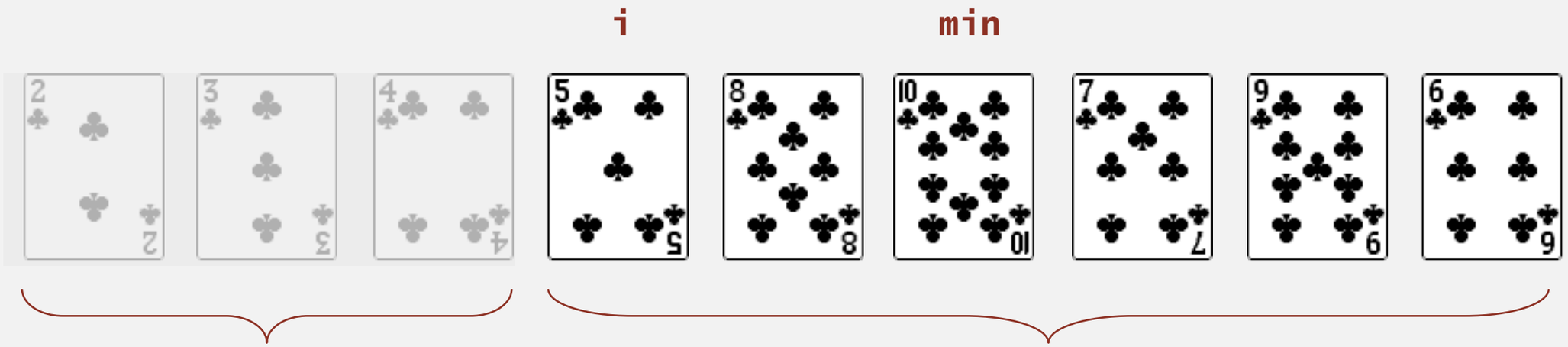
Selection sort demo

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



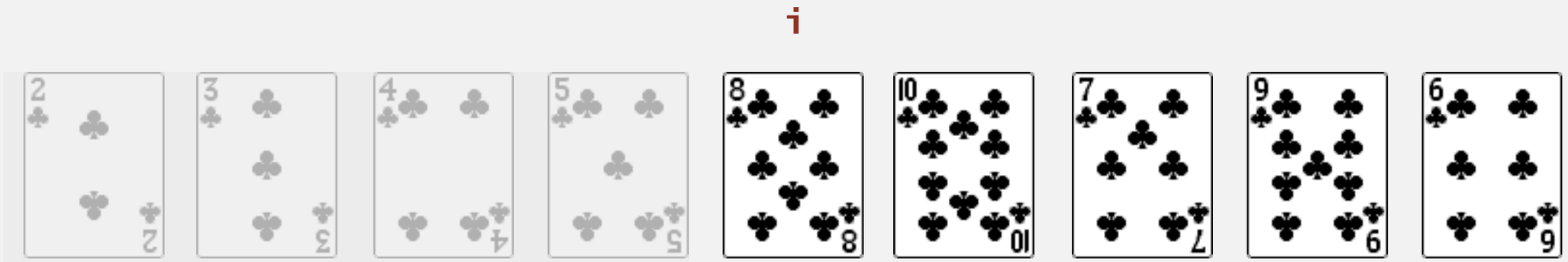
Selection sort demo

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



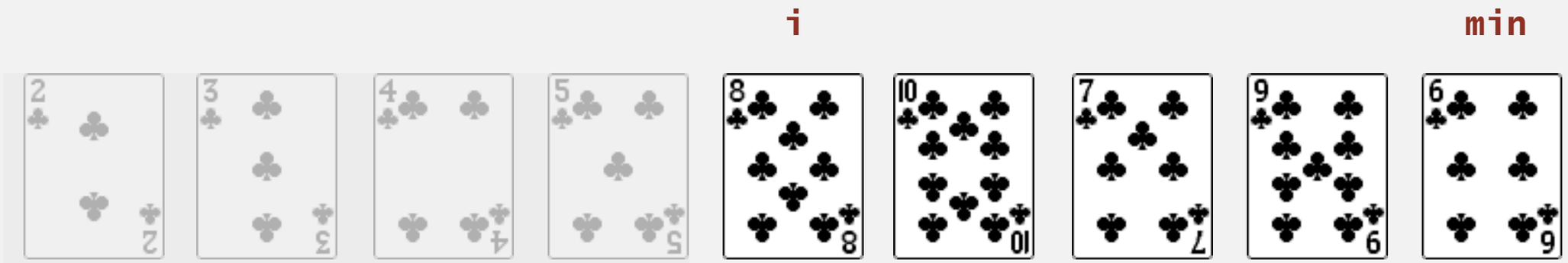
Selection sort demo

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



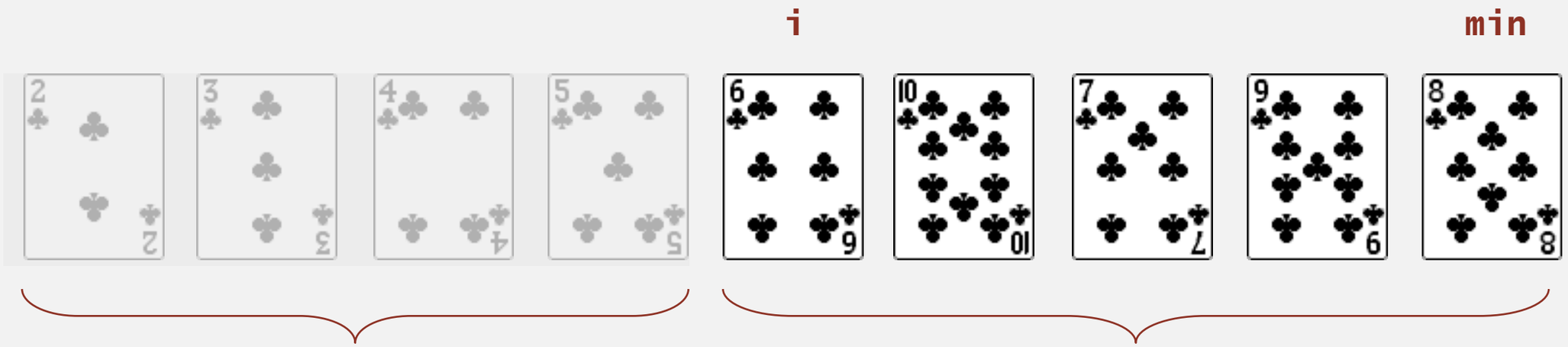
Selection sort demo

- 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

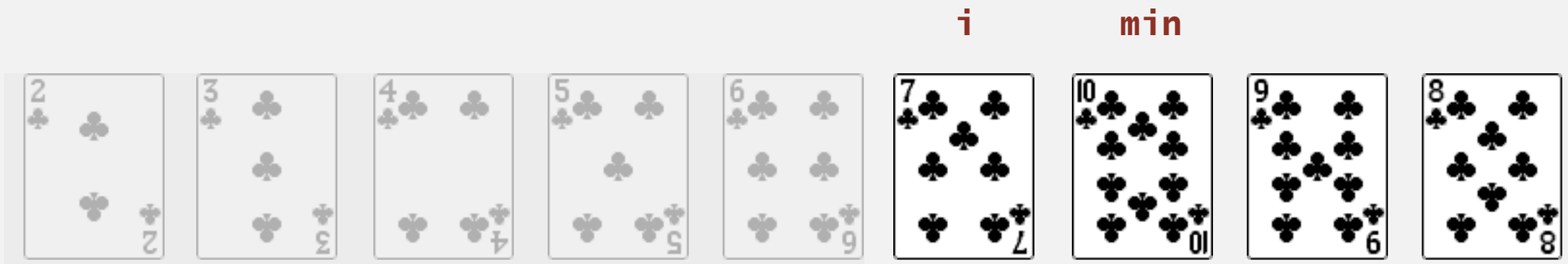
- In iteration `i`, find index `min` of smallest remaining entry.
- Swap `a[i]` and `a[min]`.

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]`.

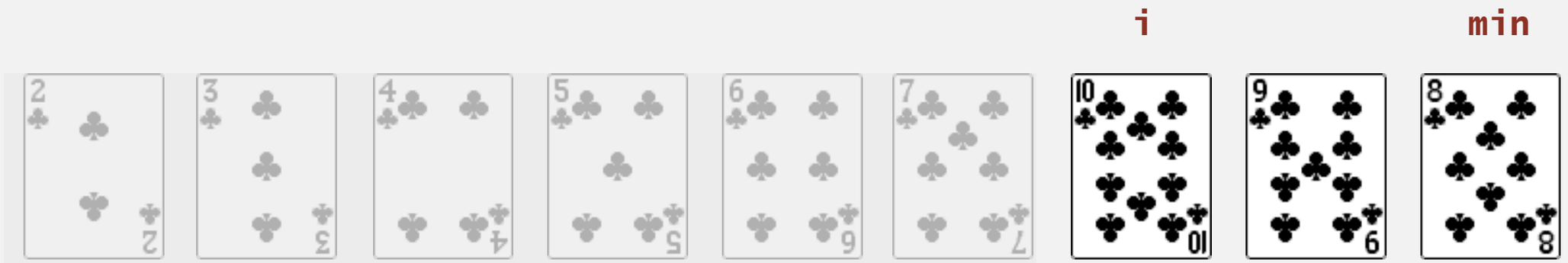


Selection sort demo

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

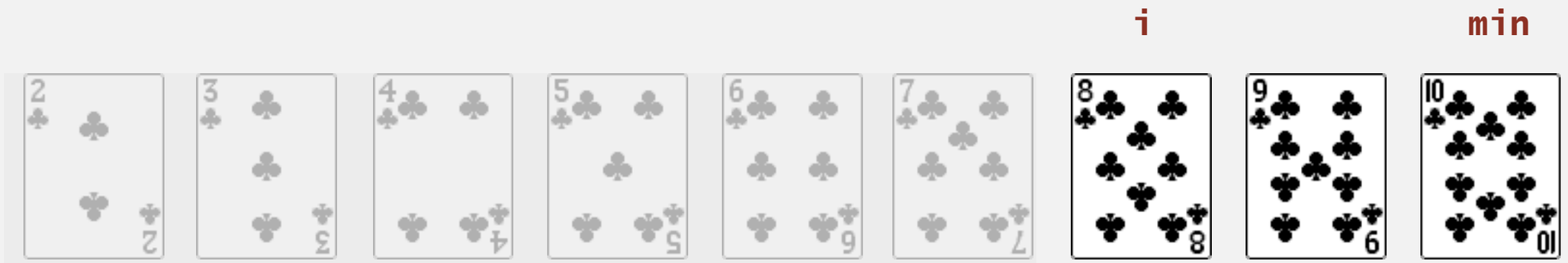
Selection sort demo

- 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

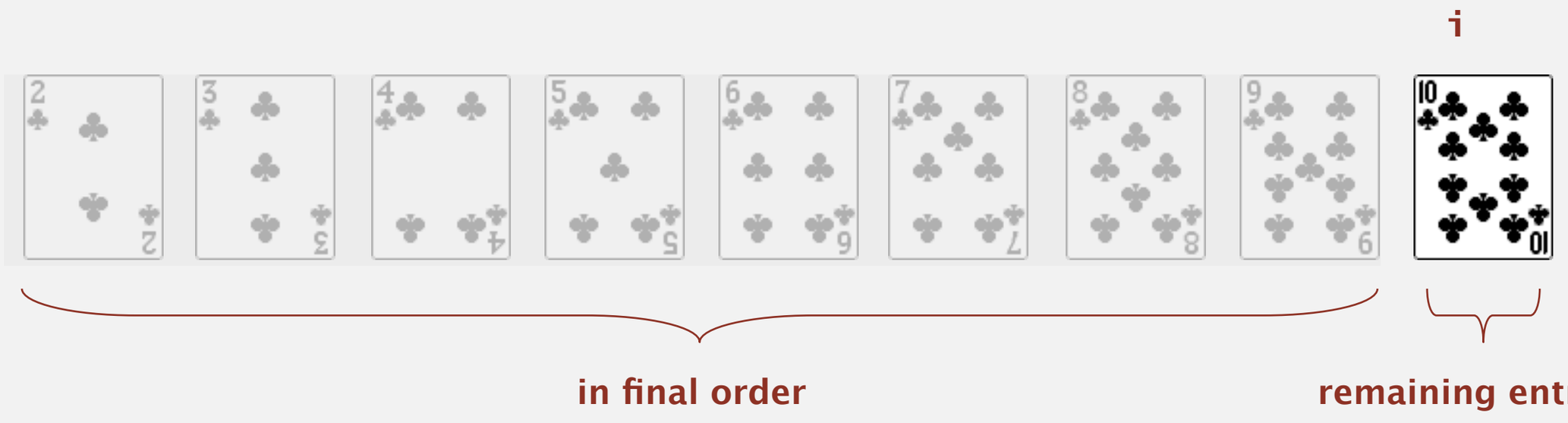
- In iteration `i`, find index `min` of smallest remaining entry.
- Swap `a[i]` and `a[min]`.

Selection sort demo

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

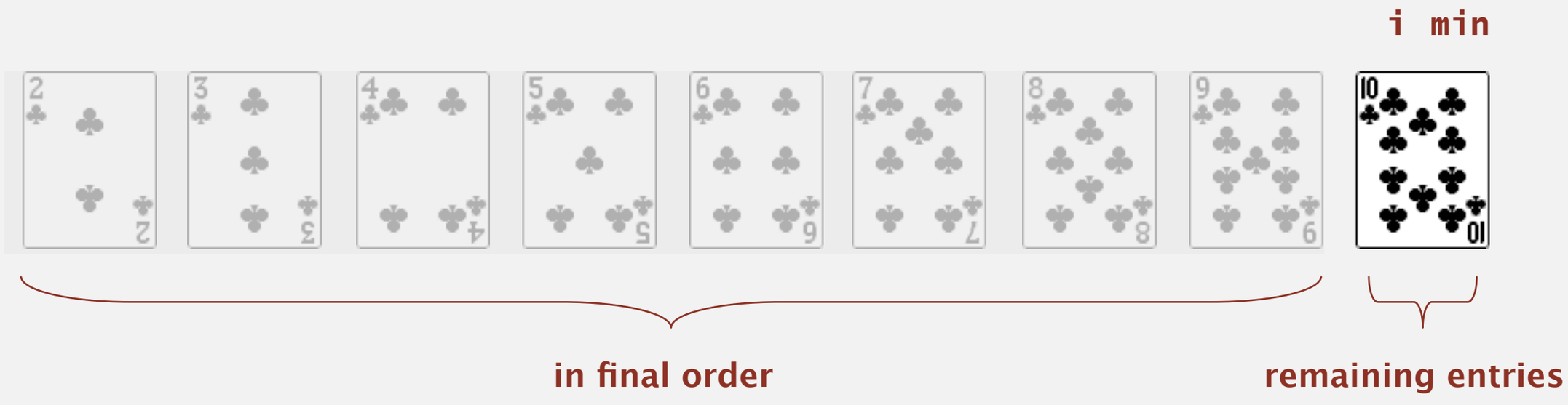
Selection sort demo

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



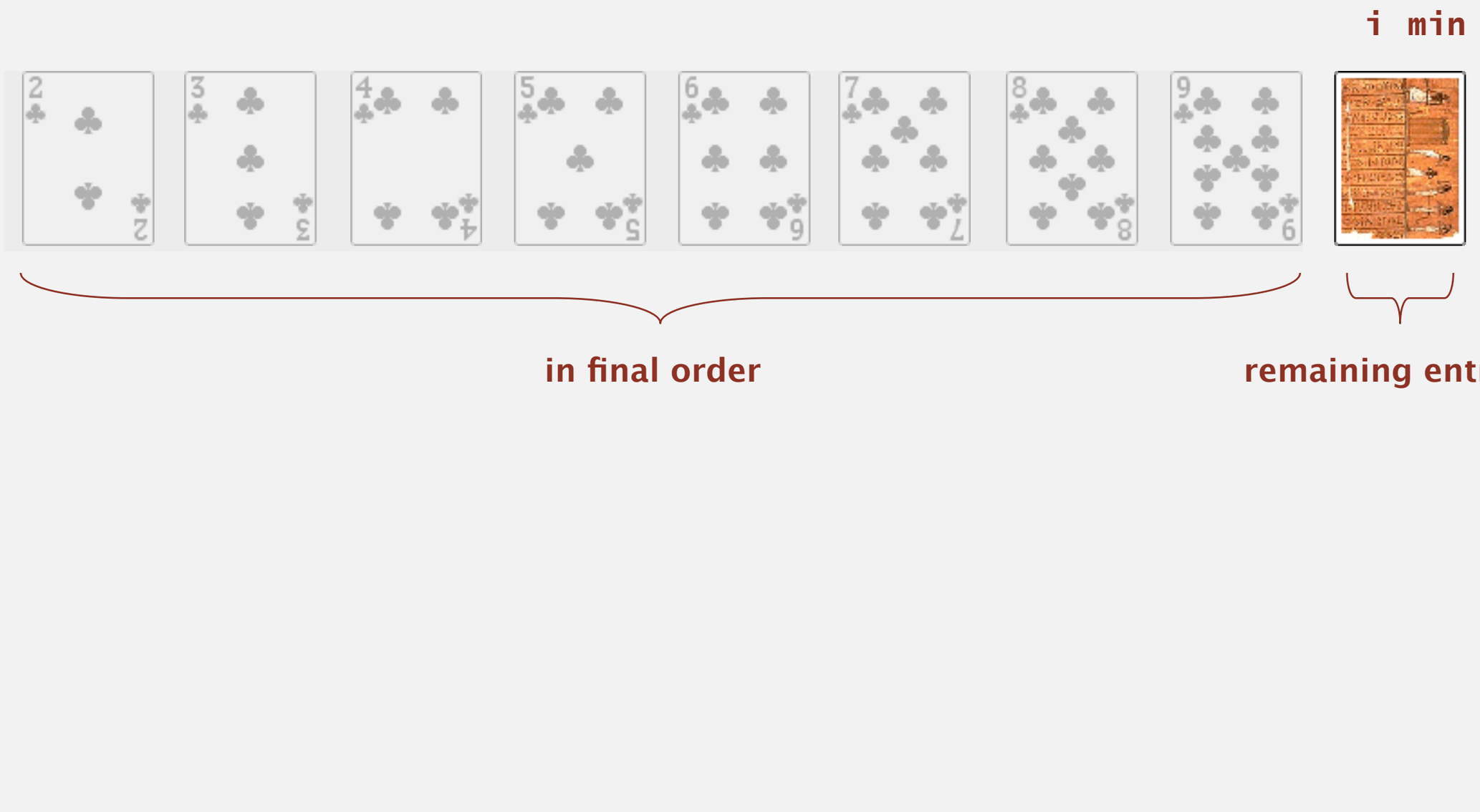
Selection sort demo

- 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]`.



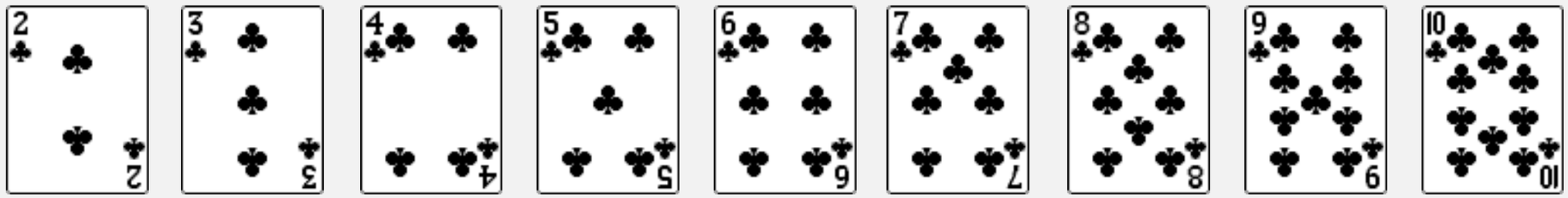
Selection sort demo

- 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]`.



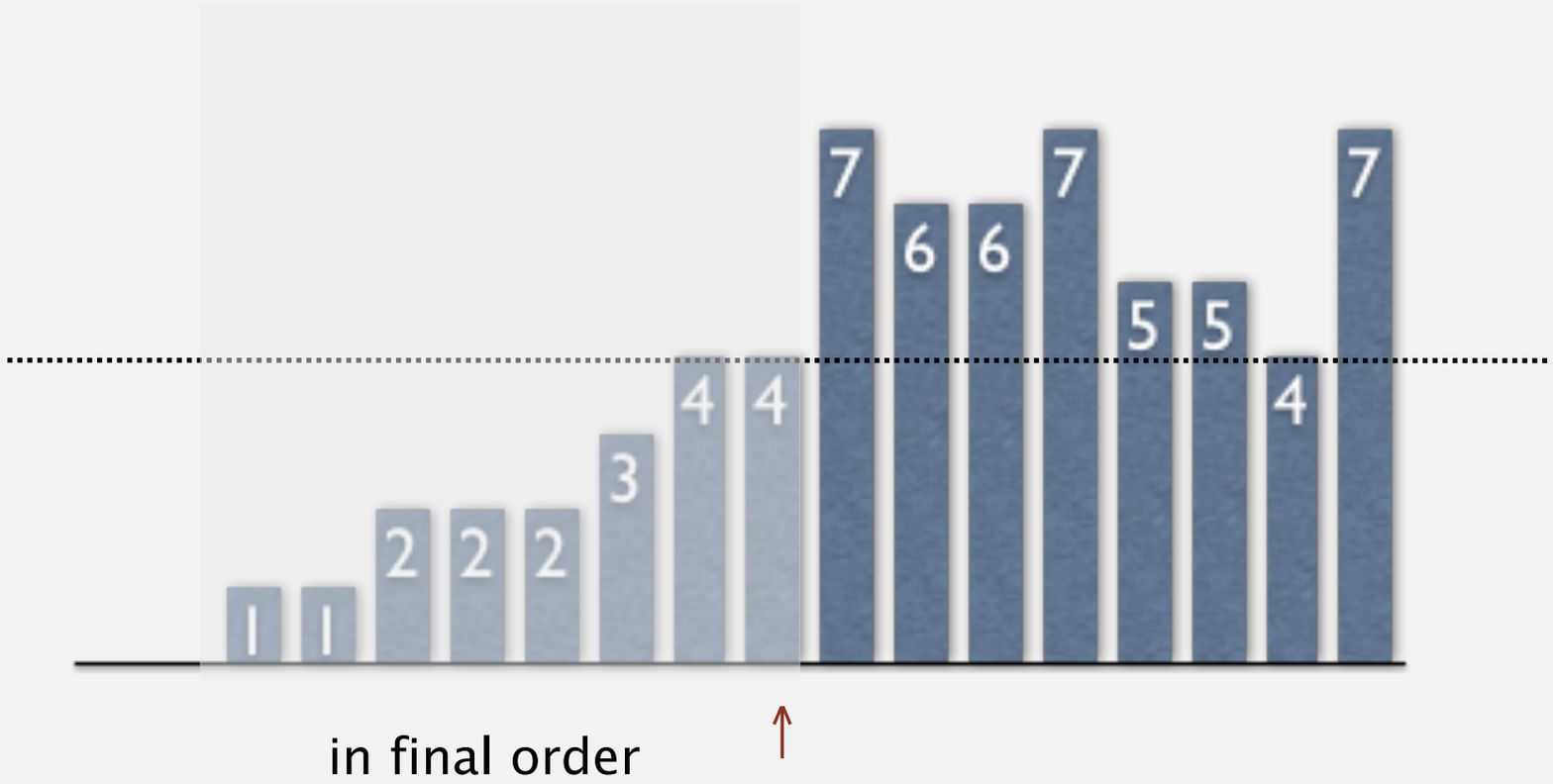
sorted

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 ↑.



Two useful sorting abstractions

Helper functions. Refer to data through compares and exchanges.

Less. Is item `v` less than `w` ?

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

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

```
private static void exch(Comparable[] a, int i, int j)
{
   Comparable swap = a[i];
   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;

         What to write here? 3 mins.
         Can use the following methods
          - less(Comparable v, Comparable w)
          - exch(Comparable[] a, int i, int j)

      }
   }

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

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

Selection sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

```
i++;
```

- Identify index of minimum entry on right.

```
int min = i;
for (int j = i+1; j < N; j++)
   if (less(a[j], a[min]))
      min = j;
```

- Exchange into position.

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

Selection sort: Java implementation

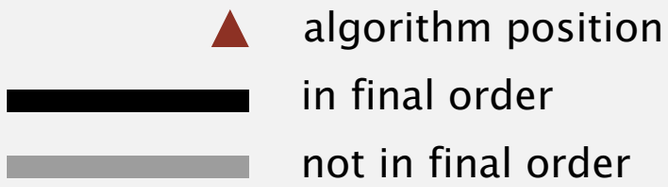
```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int min = i;
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
  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

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

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

Selection sort: animations

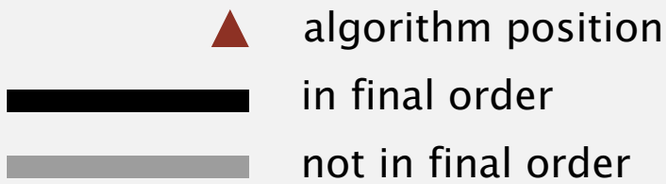
20 random items



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

Selection sort: animations

20 partially-sorted items



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

Selection sort: mathematical analysis

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