# Sorting

## CS143: lecture 12

**Byron Zhong, July 10**

# Makefile

# Makefile

- Makefile is comprised of rules.

name

dependencies

```
cat: cat.o readline.o
	clang -o cat cat.o readline.o
```

command

this is a TAB

- `make cat`
    - if cat.o and readline.o have changed, do the command
    - if not, do nothing
- But how do you make cat.o and readline.o?

# Makefile

- Rules can contain patterns.

```
%.o: %.c
  clang -o $@ -c $^
```

$^: the dependencies

$@: this rule's name

- To get anything .o, run the following command, depending on that .c
- `make readline.o`
  - if readline.c has changed, do the command; otherwise, do nothing
- One rule can depend on other rules; `make` will figure out the order

# Makefile

- You can also define variables in Makefile.

```
CC      = clang
CFLAGS  += -g -Wall -Wextra -Werror -pedantic -std=c11
LDFLAGS += -g

cat: cat.o readline.o
    $(CC) $(LDFLAGS) -o $@ $^

%.o: %.c
    $(CC) $(CFLAGS) -o $@ -c $^
```

$ to replace CC with the variable

# Separate Compilation

## How C is actually compiled


- removes comments
- execute the directives
  - finds the files to `#include`
  - replace the text with `#define`
  - `#if`

- Convert the C program into assembly
- Tokenize, parse, optimize, ....

- Convert the assembly code into machine-readable *object file*
- All functions are converted into binary

# Separate Compilation

## How C is actually compiled


- every .o file provides some functions
- linker links all the functions together
- finds main

# Separate Compilation

- Demo

# Sorting

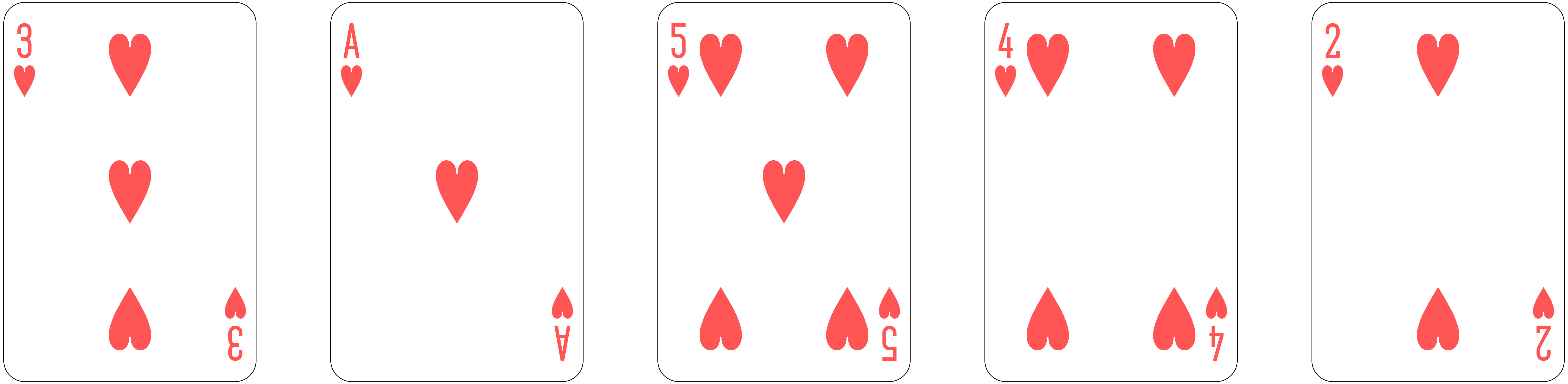
# Sorting

## Putting things in order

- What do we have:
  - A *list* of *n* elements
  - A comparison function: $\leq$
- What do we want:
  - The *list* has all the same elements as it started with
  - If $i \leq j$, `list[i]` $\leq$ `list[j]`

# Sorting

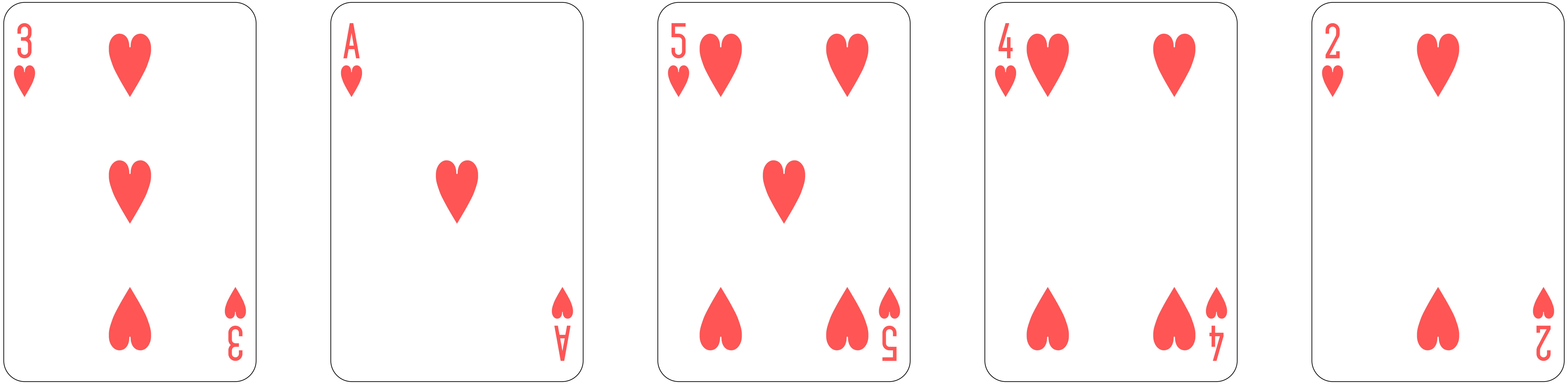
## Example



# Sorting

## Example 1 (how I do it)

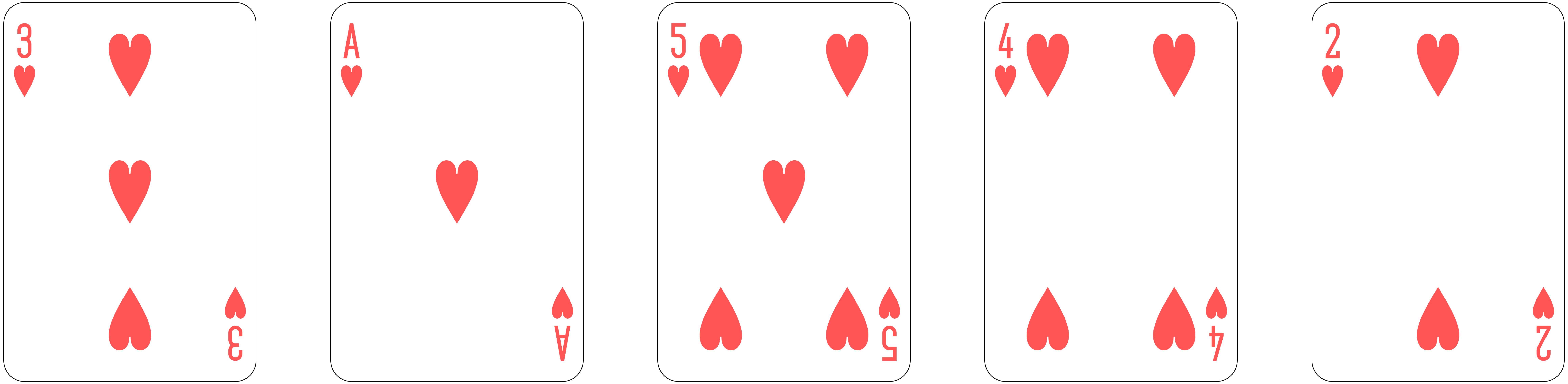
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

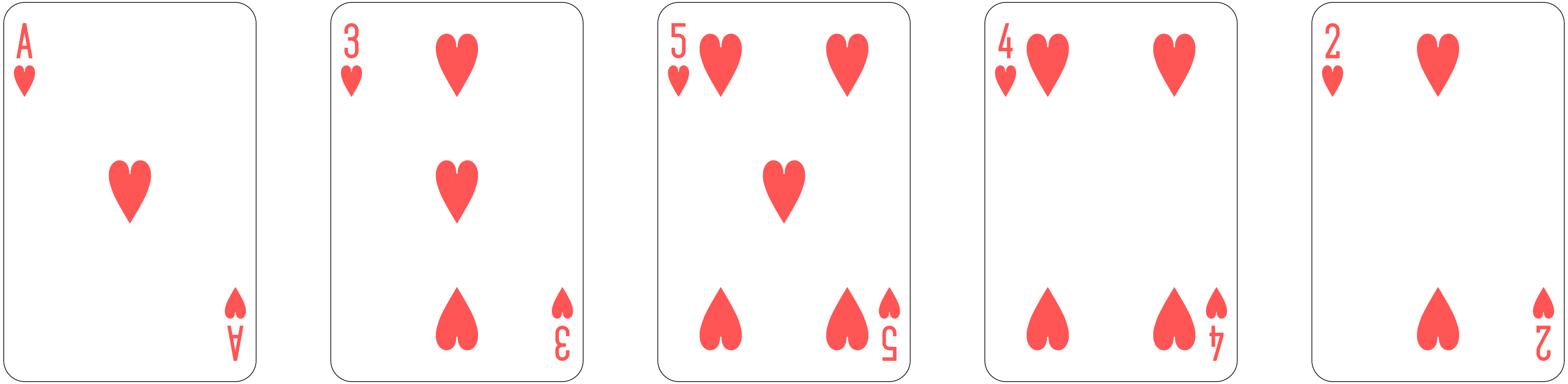
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

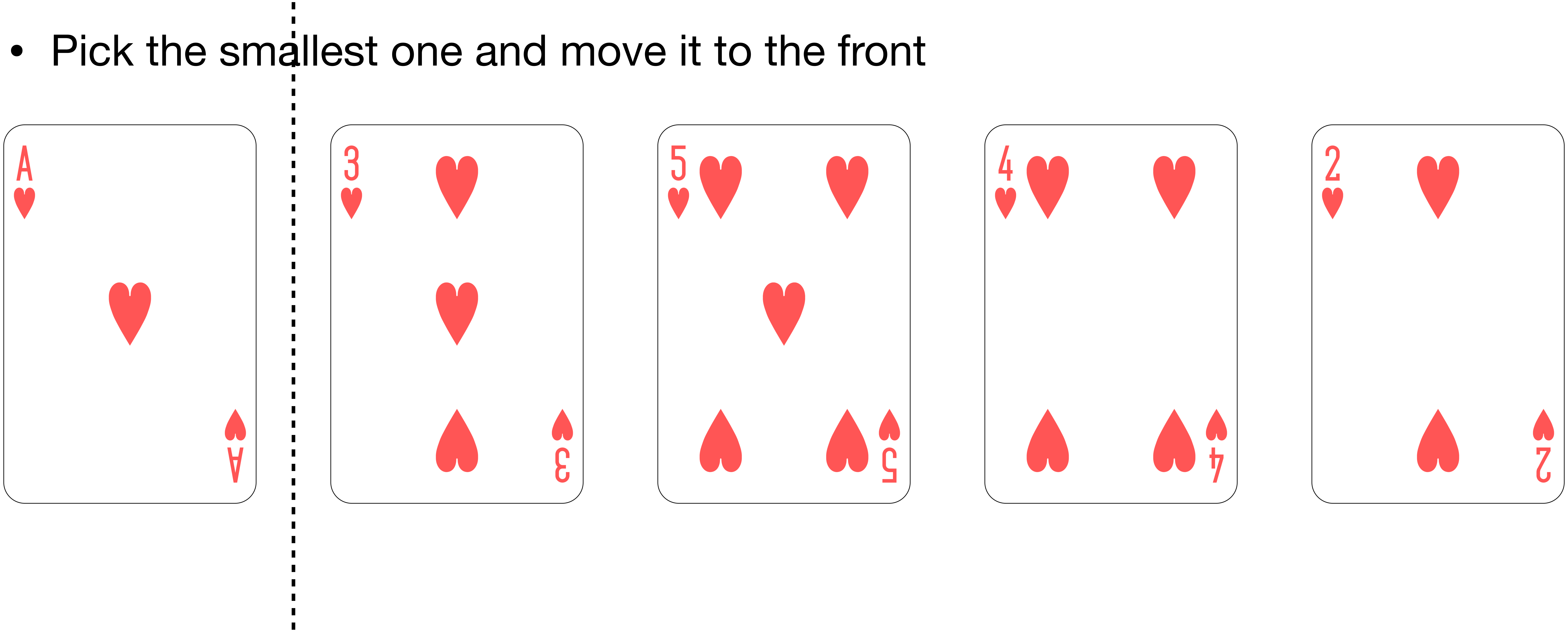
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

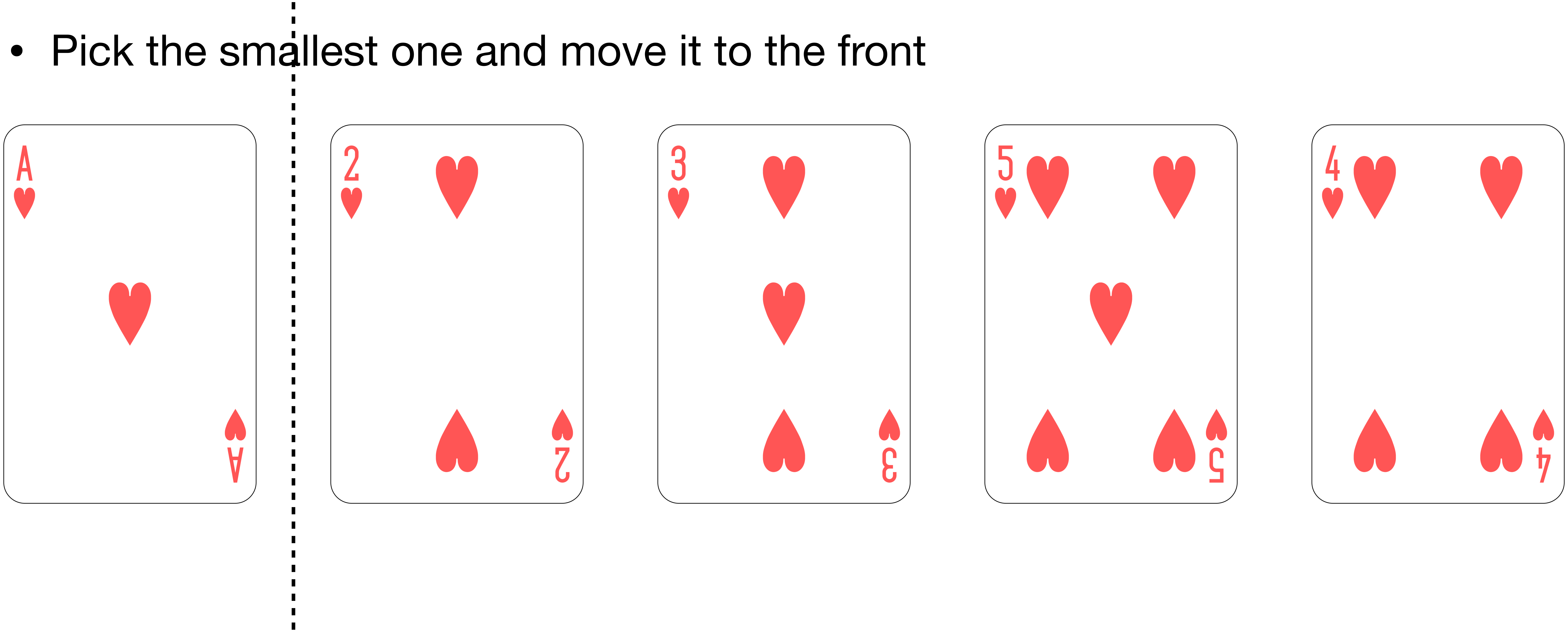
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

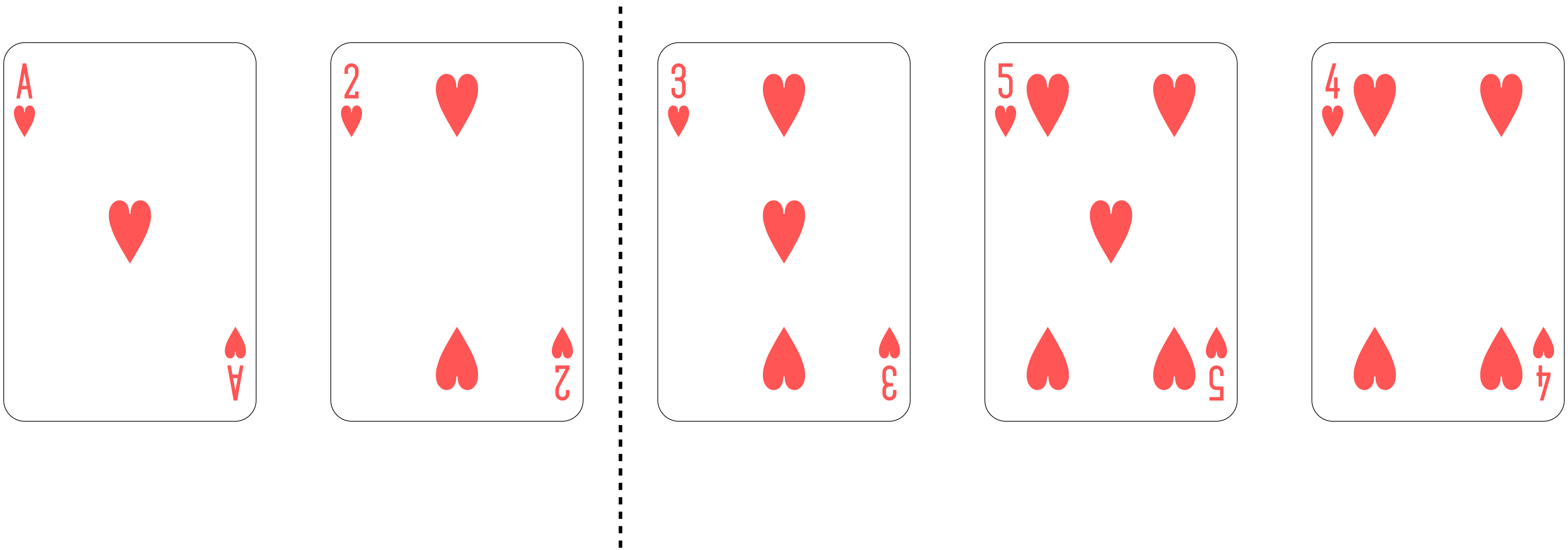
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

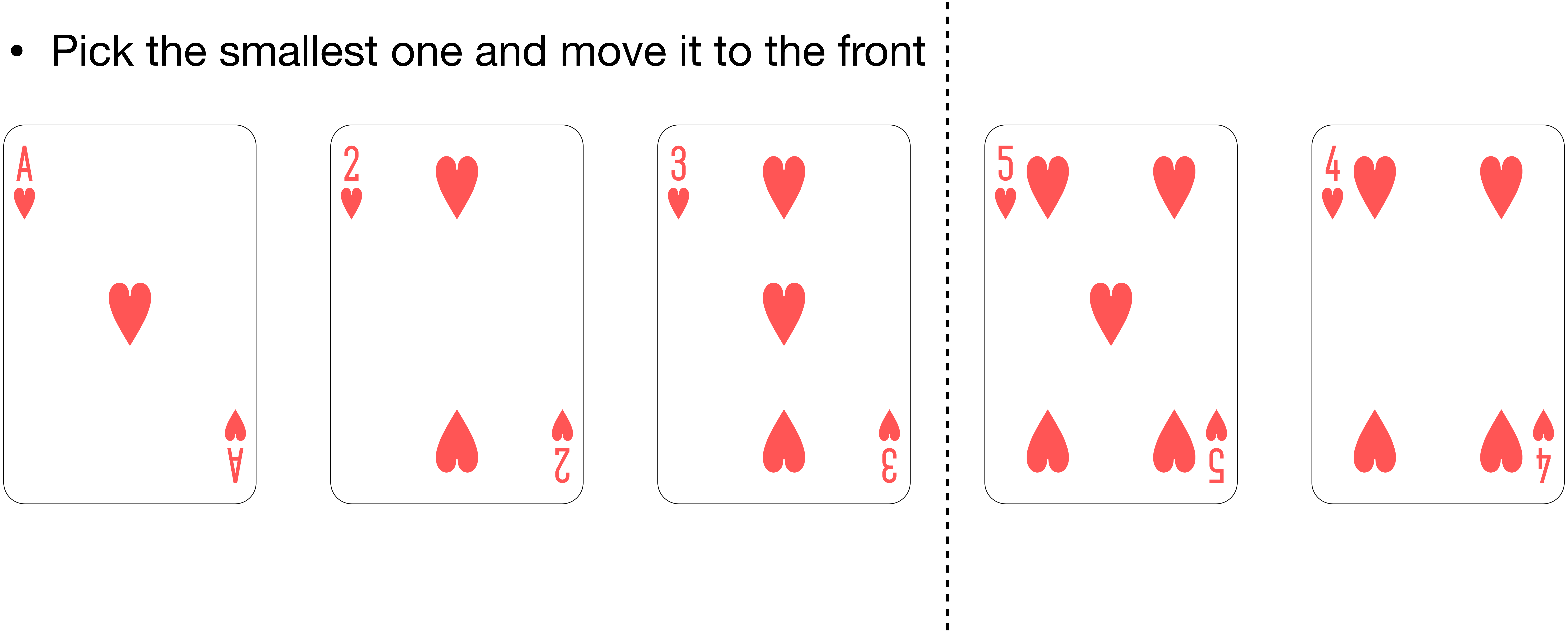
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

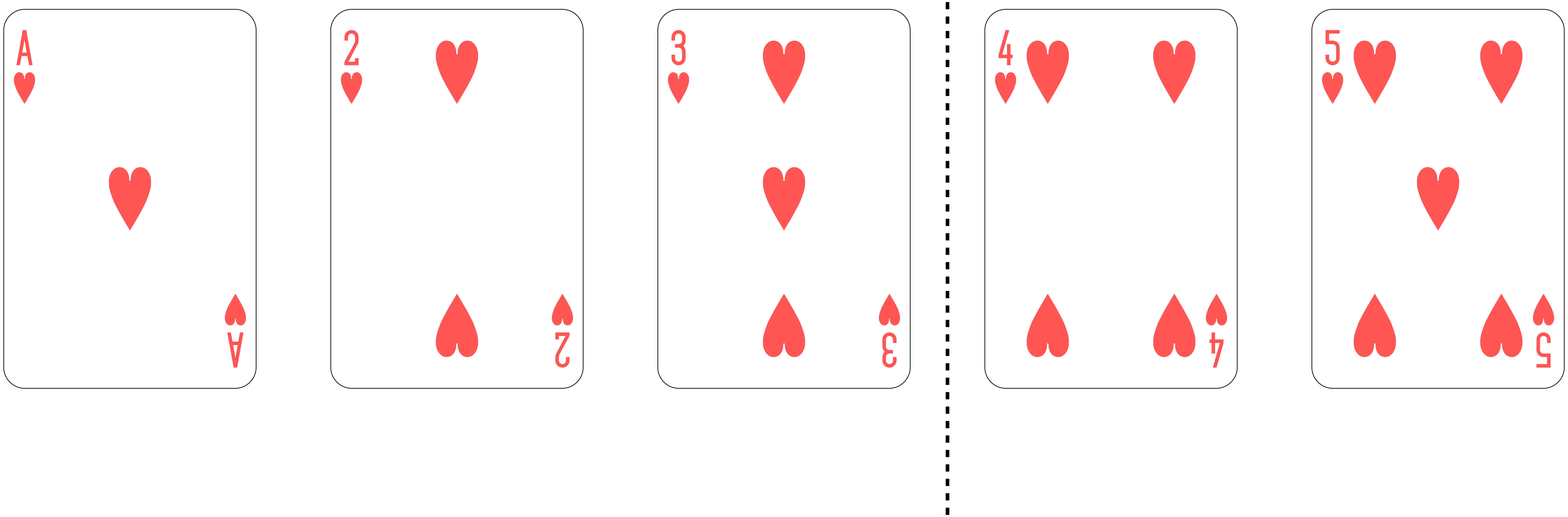
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

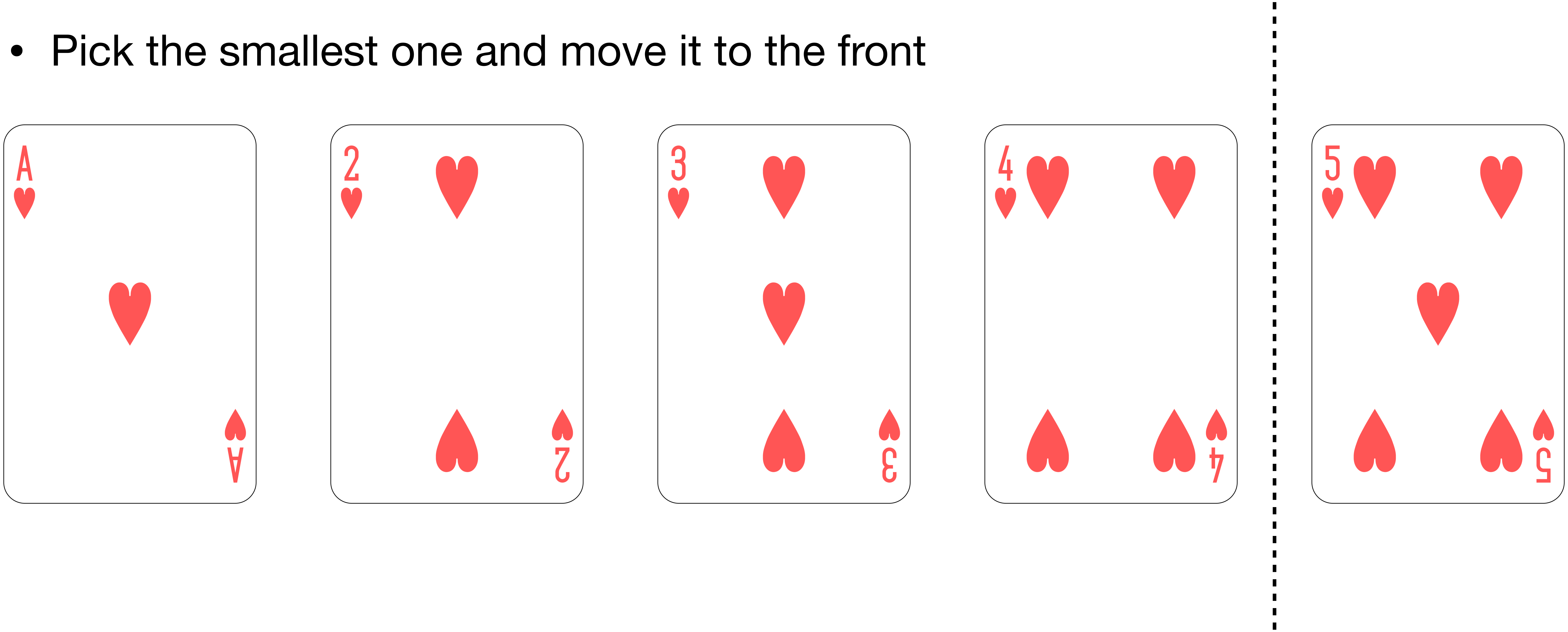
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

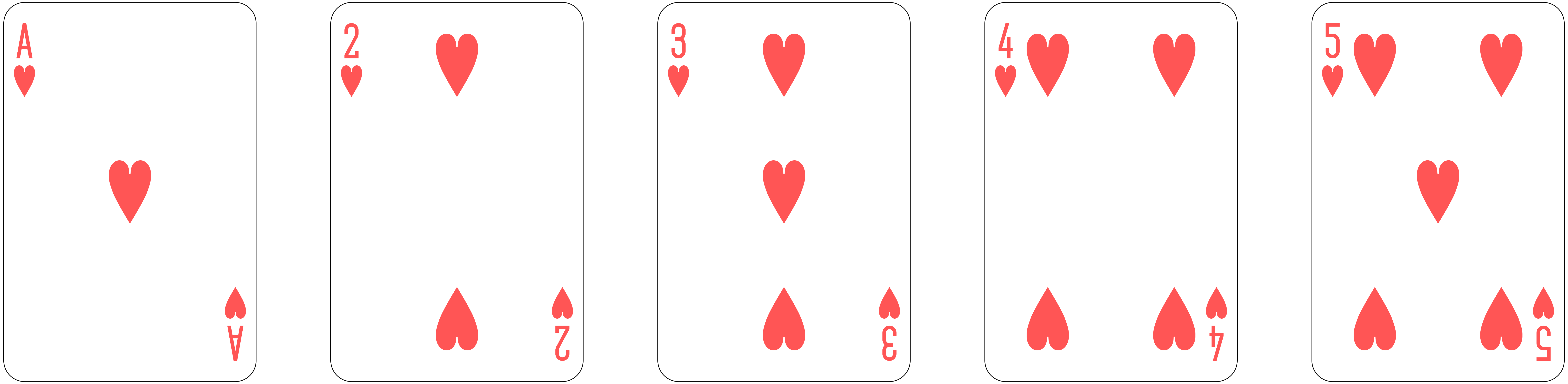
- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it)

- Pick the smallest one and move it to the front



# Sorting

## Example 1 (how I do it): Algorithm

```
For i = 1 to n - 1:
    min = index of smallest in A[i + 1 : n]
    swap A[min] and A[i]
```

- Why swap instead of pushing things over?
    - It's more efficient and we don't care about the order of the unsorted part
- This is called *selection sort* -- we *select* the one we want repeatedly.

# Sorting

## Example 1 (how I do it): Algorithm

```
For i = 1 to n - 1:
    min = index of smallest in A[i + 1 : n]
    swap A[min] and A[i]
```

- How many comparisons do we need to do?
  - $(n-1) + (n-2) + \ldots + 1 = n(n-1)/2 = O(n^2)$
- How many swaps?
  - $n - 1$

# Sorting

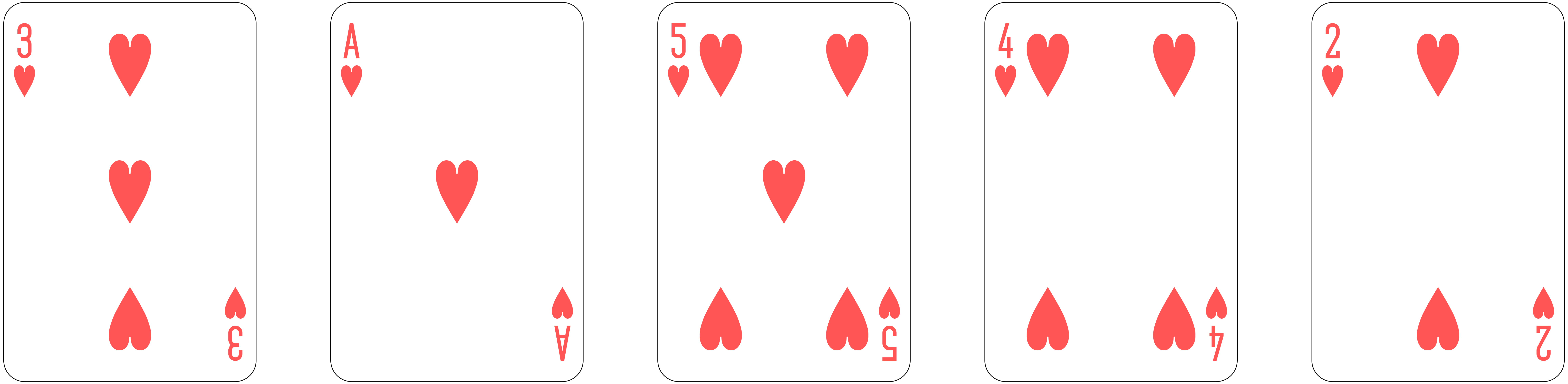
## Example 1 (how I do it): Algorithm

```
For i = 1 to n - 1:

    min = index of smallest in A[i + 1 : n]

    swap A[min] and A[i]
```

- Everything left of the line is sorted.
- Scanning the right (unsorted) part, and putting it to the end of the left (sorted) part.

# Sorting

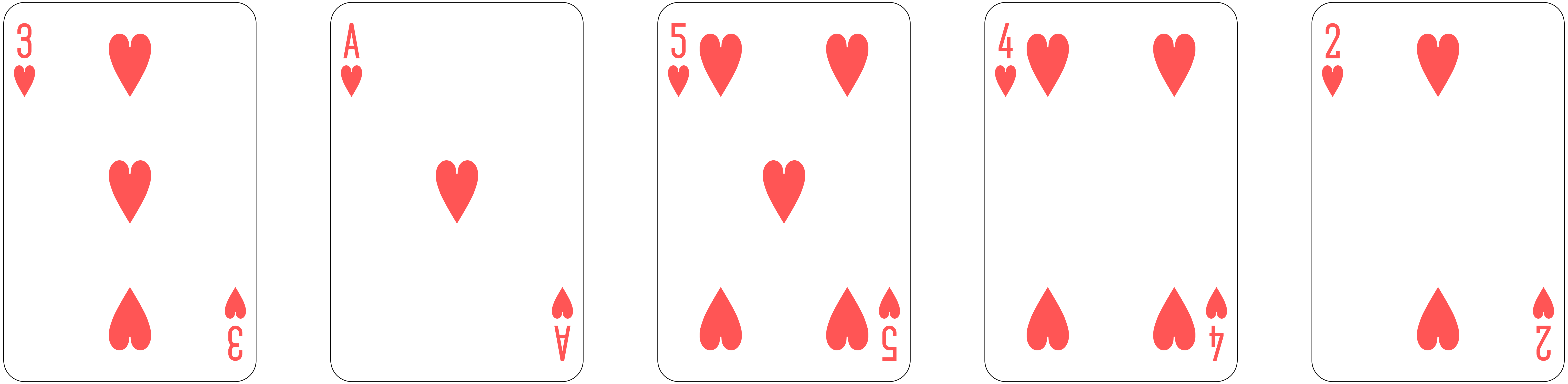
## Example 2



# Sorting

## Example 2

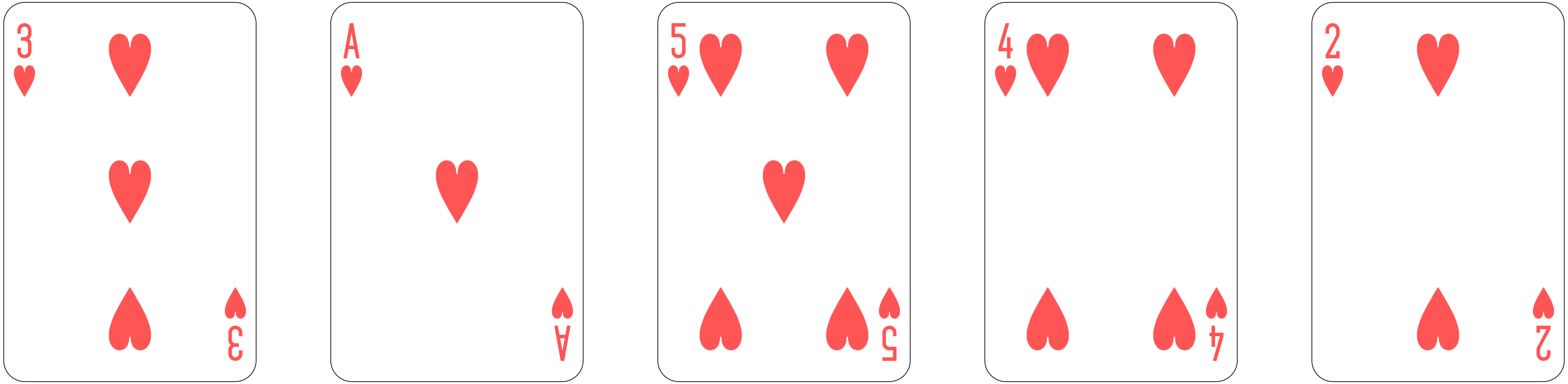
- Pick the first unsorted and insert it into the right place



# Sorting

## Example 2

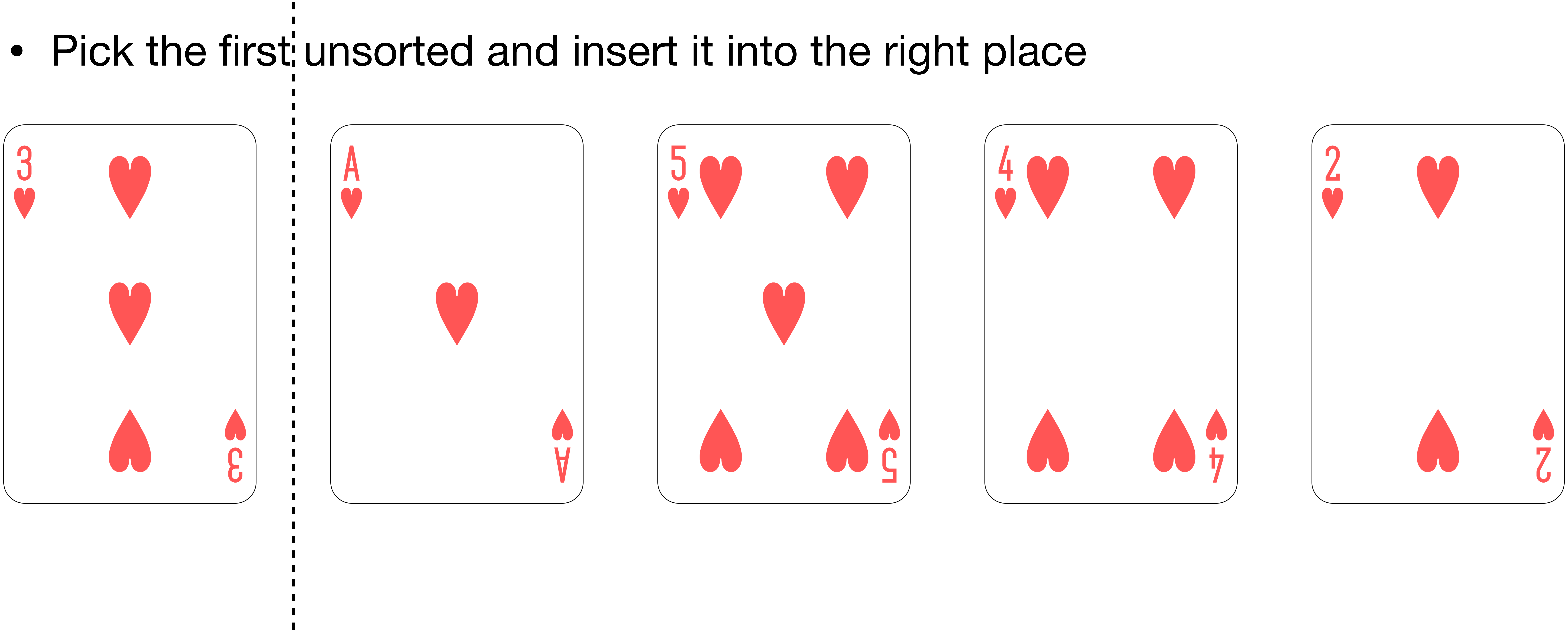
- Pick the first unsorted and insert it into the right place



# Sorting

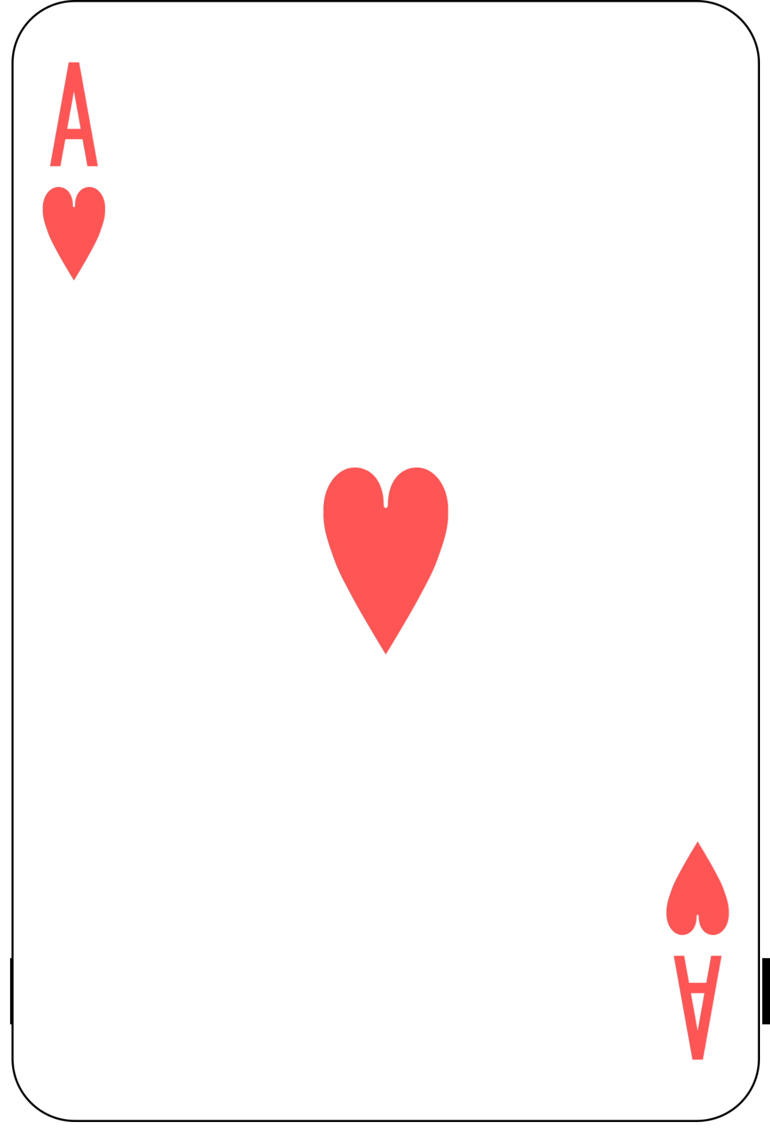
## Example 2

- Pick the first unsorted and insert it into the right place

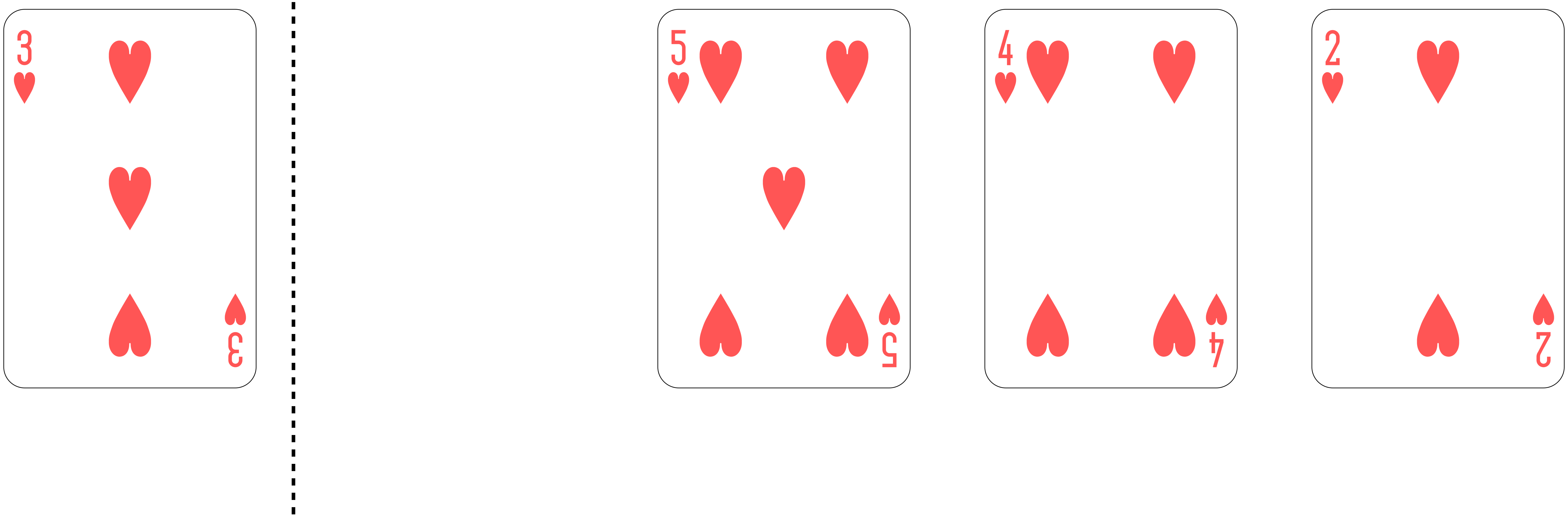


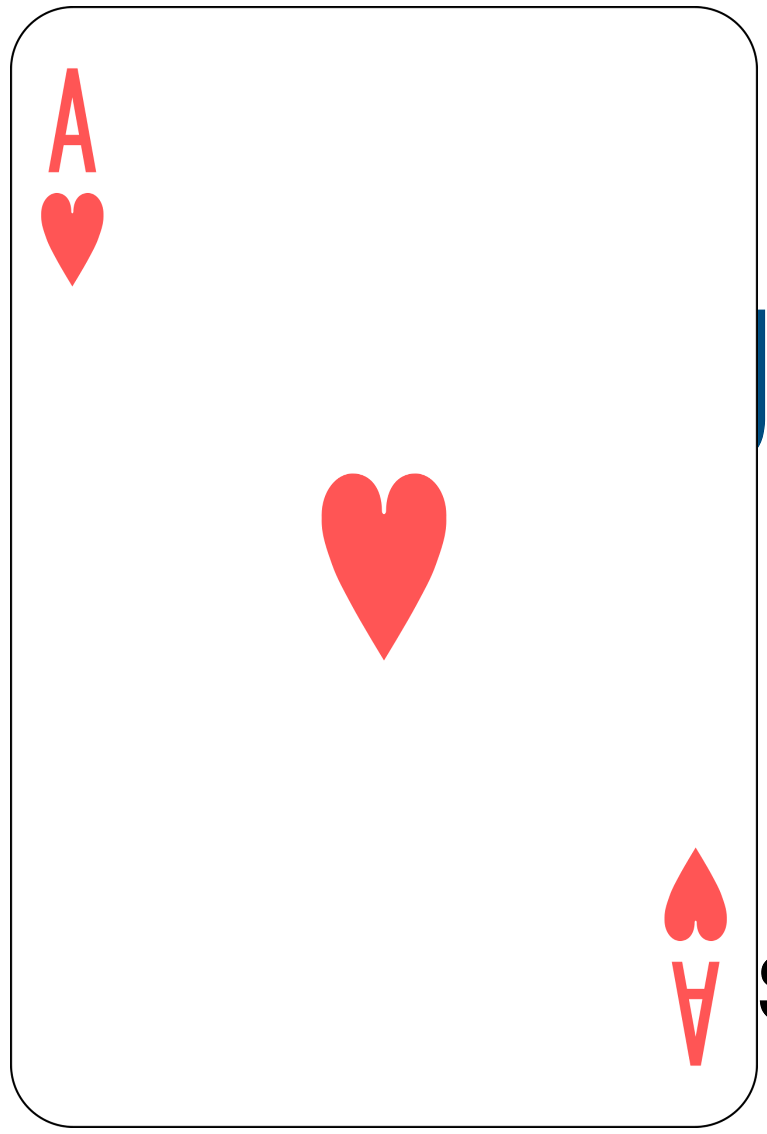
# Sorting

## Example 2

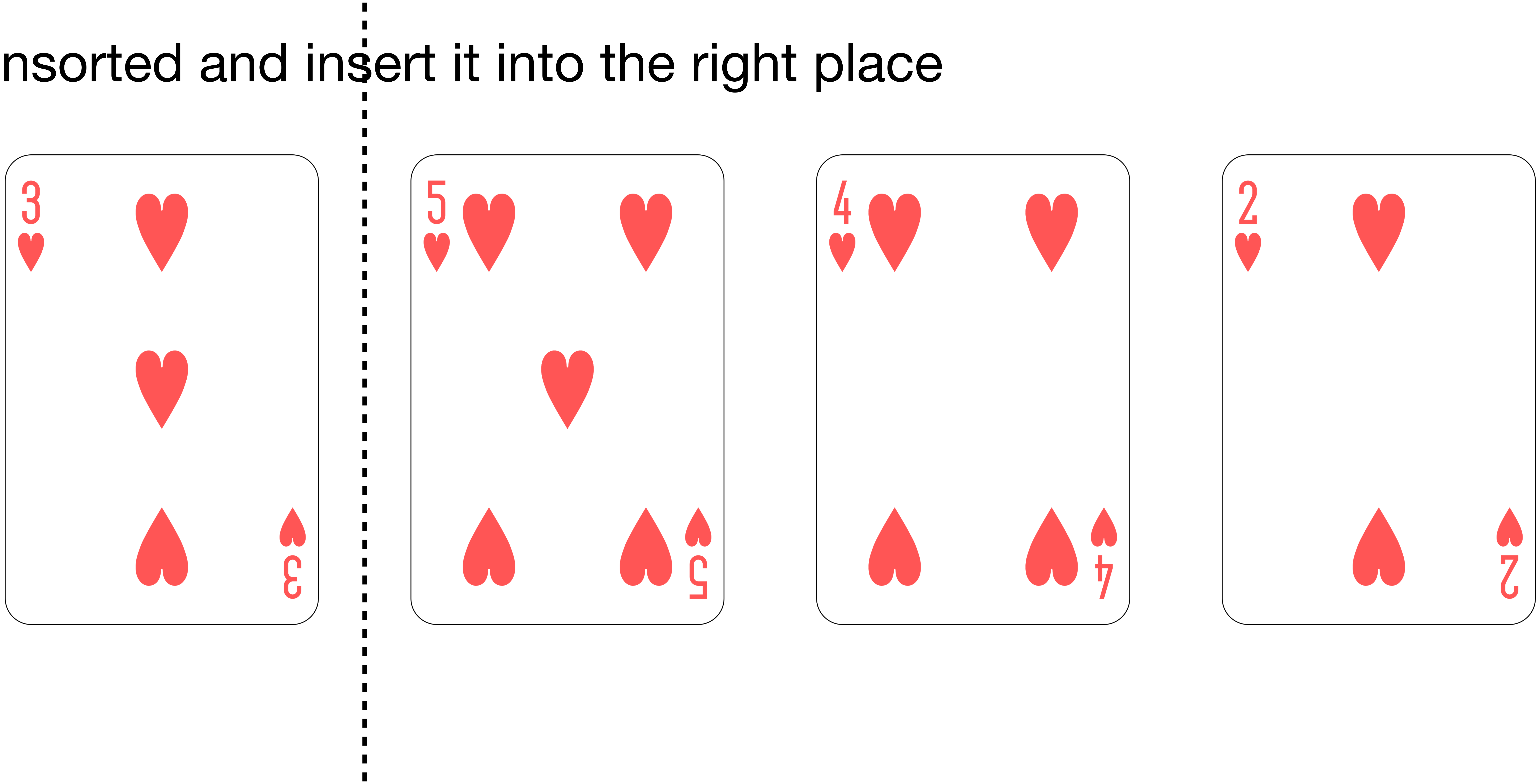



- Pick the first u nsert it into the right place





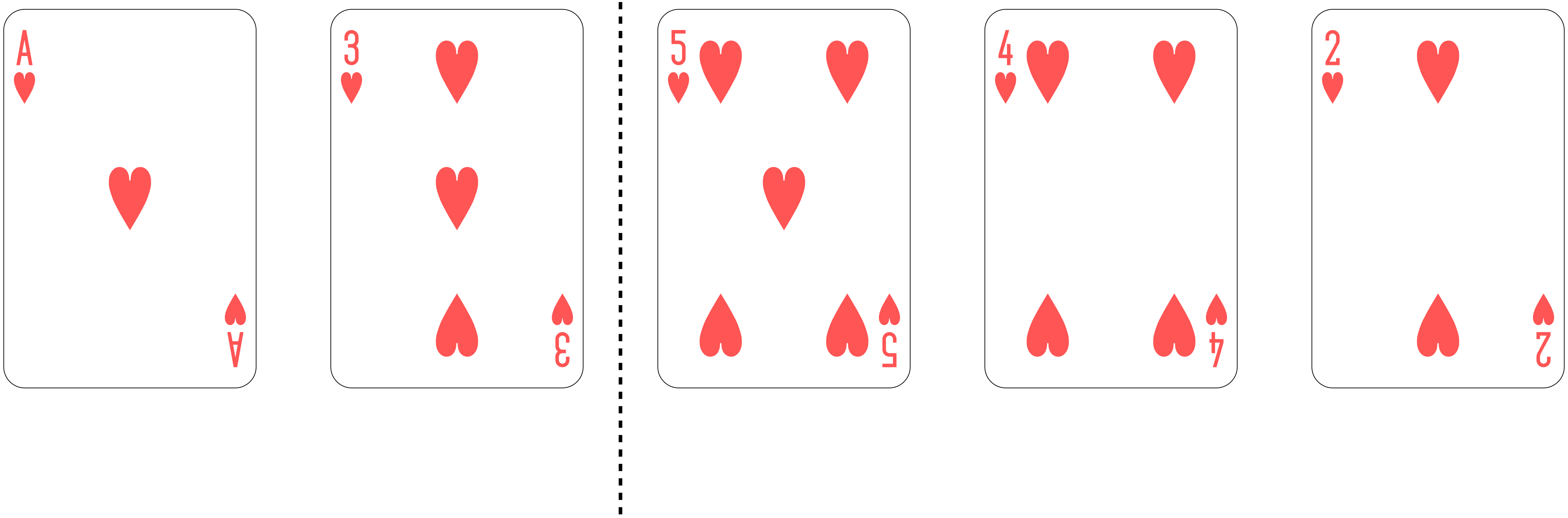
st unsorted and insert it into the right place



# Sorting

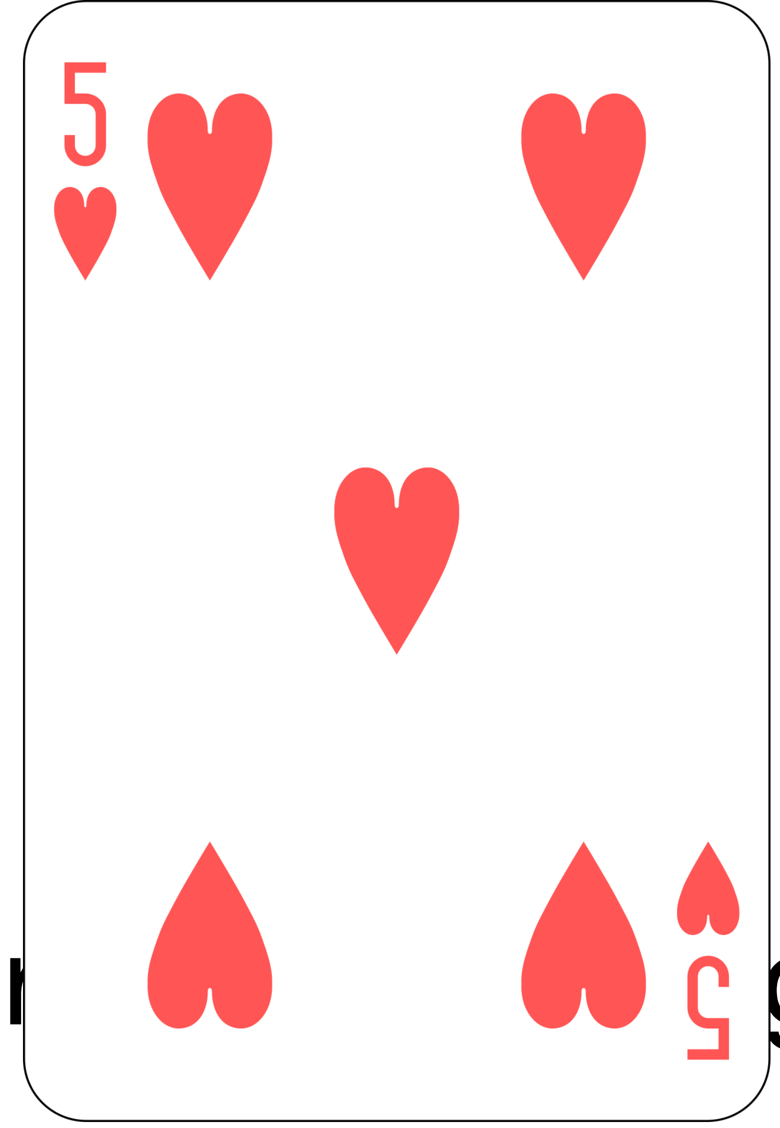
## Example 2

- Pick the first unsorted and insert it into the right place

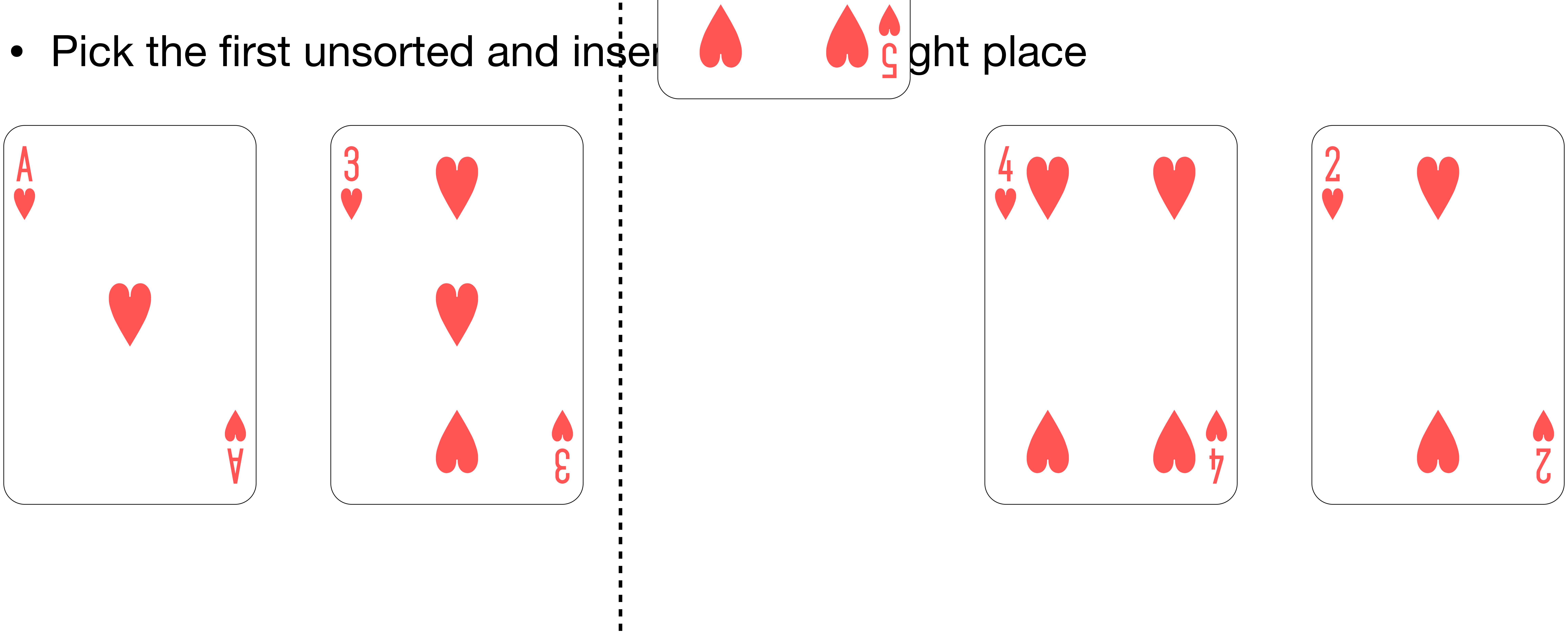


# Sorting

## Example 2

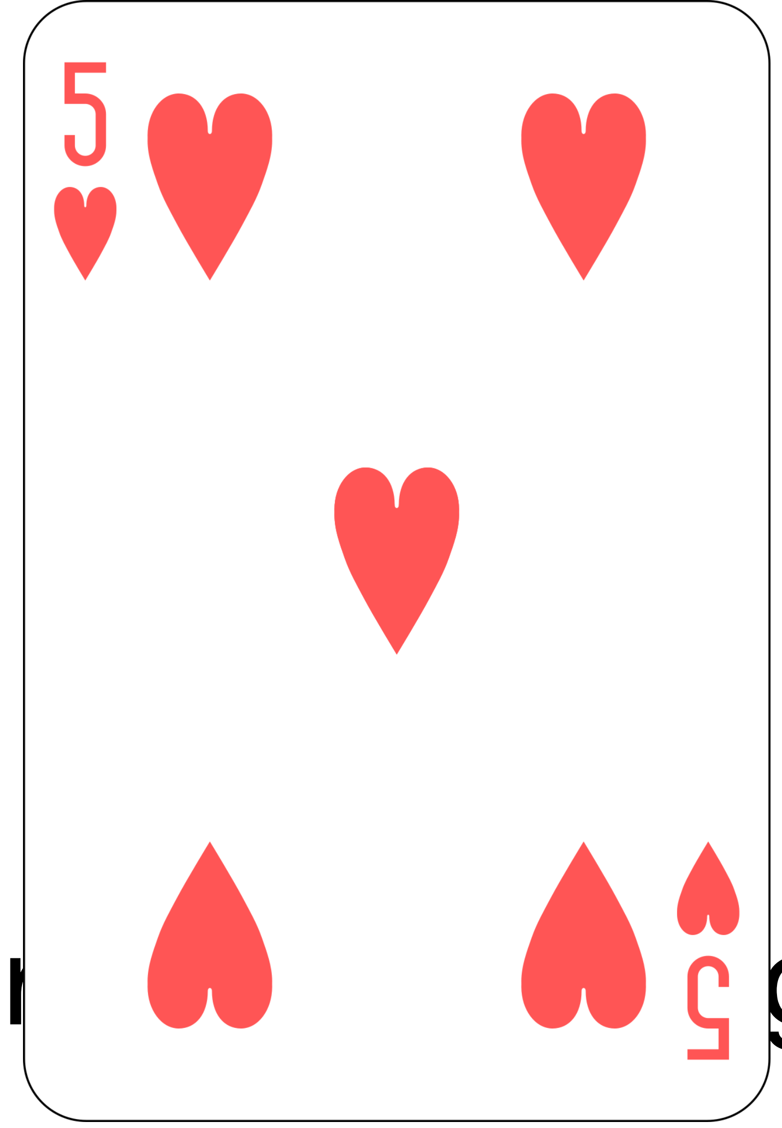



- Pick the first unsorted and inser ght place

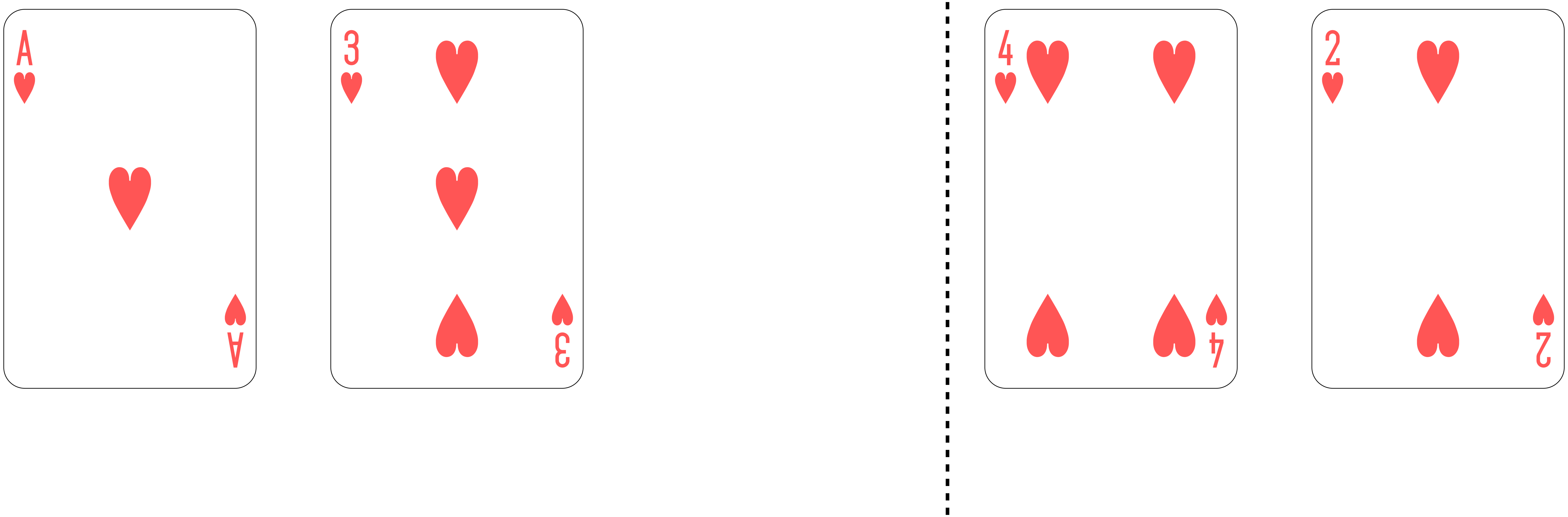


# Sorting

## Example 2



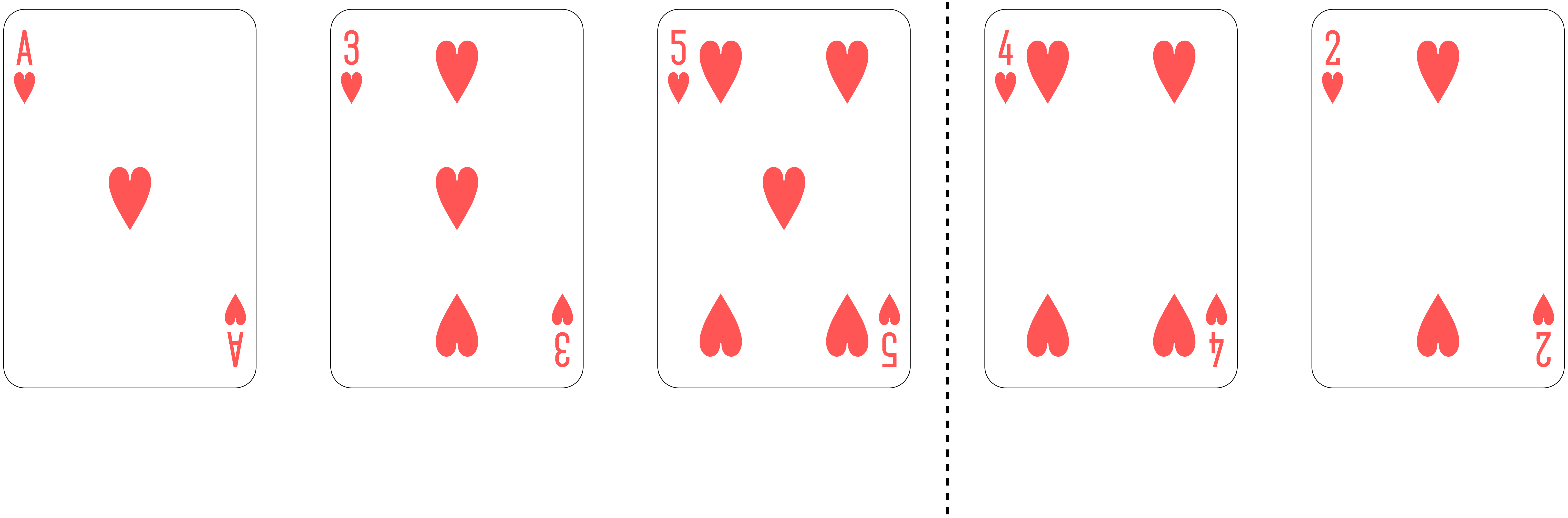

- Pick the first unsorted and inser ght place



# Sorting

## Example 2

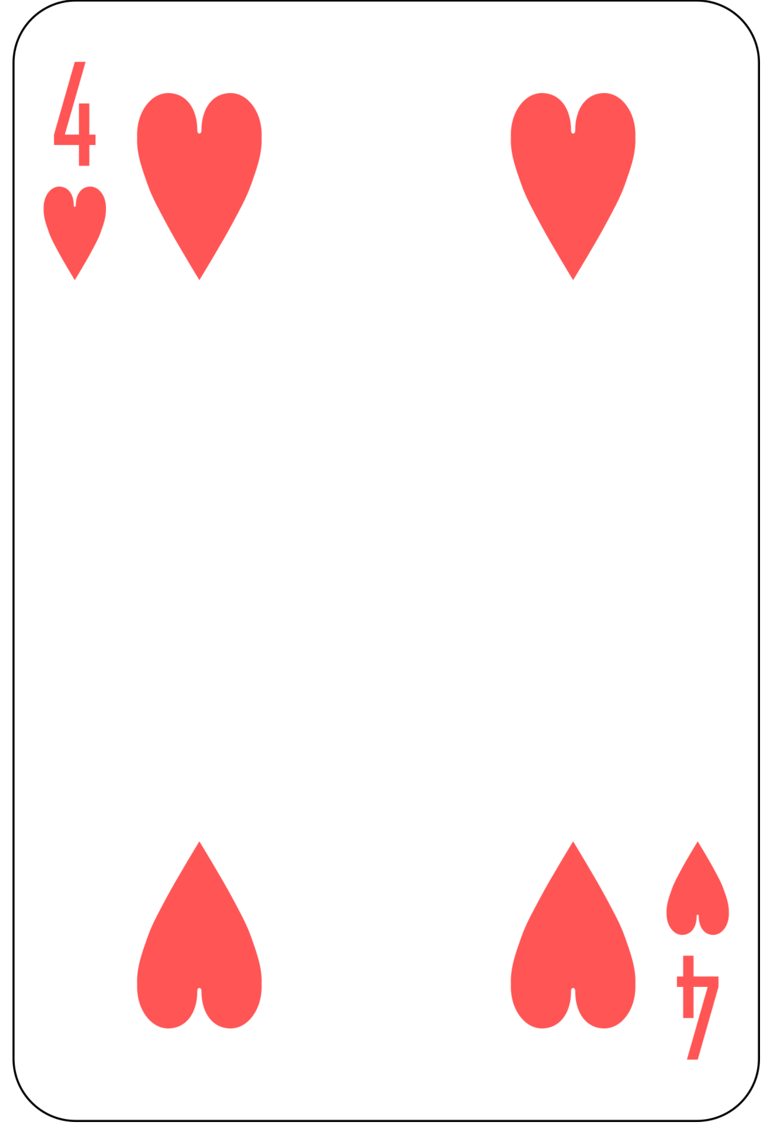
- Pick the first unsorted and insert it into the right place

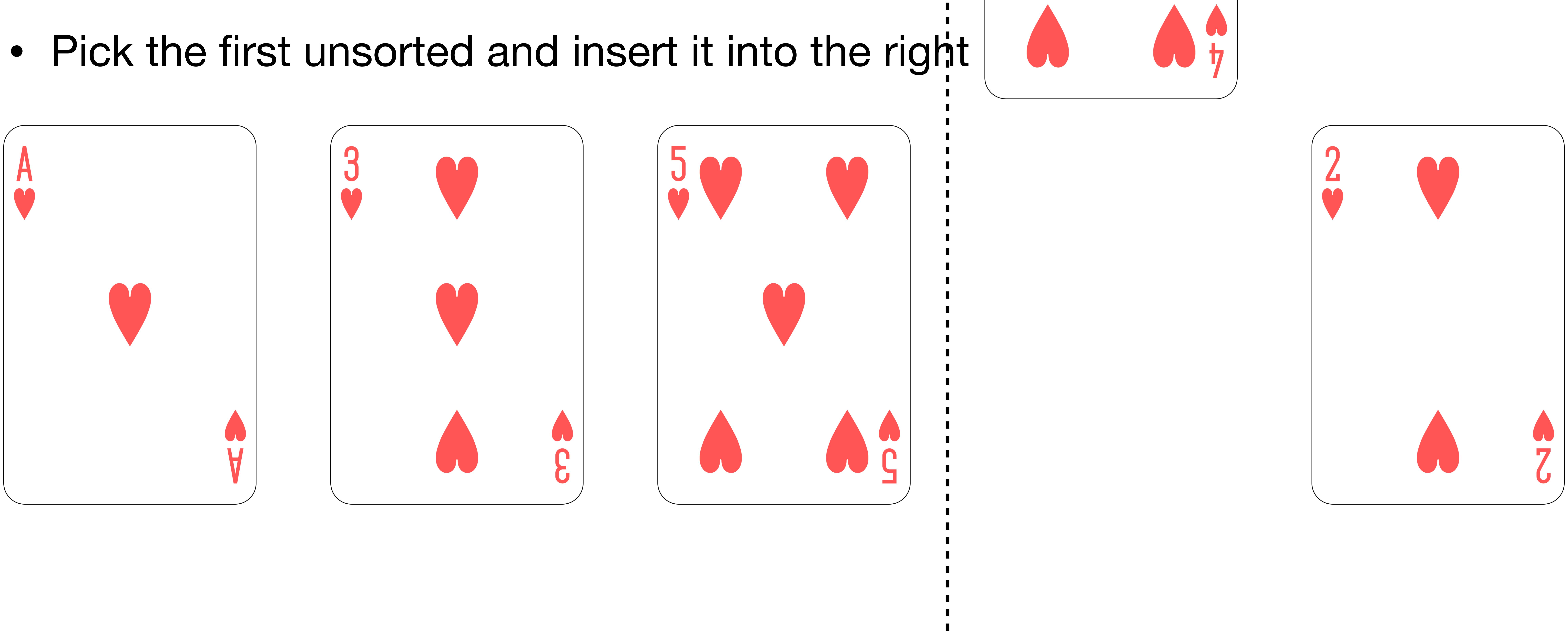


# Sorting

## Example 2

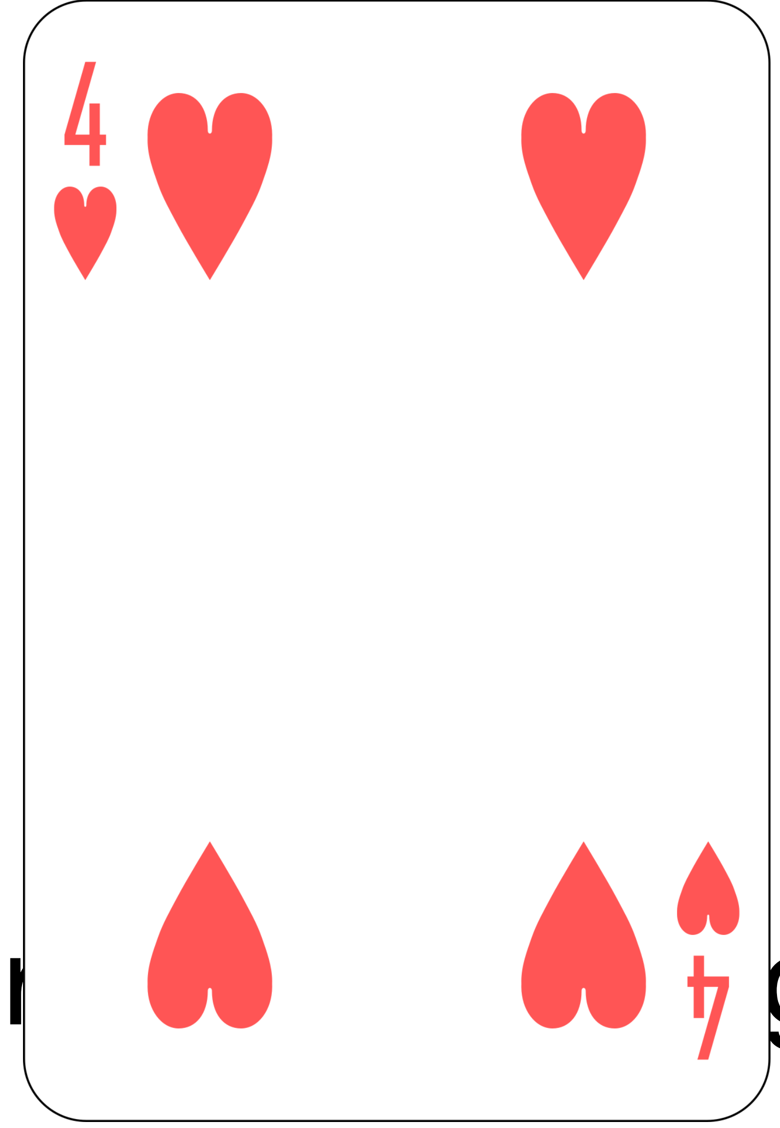
- Pick the first unsorted and insert it into the right



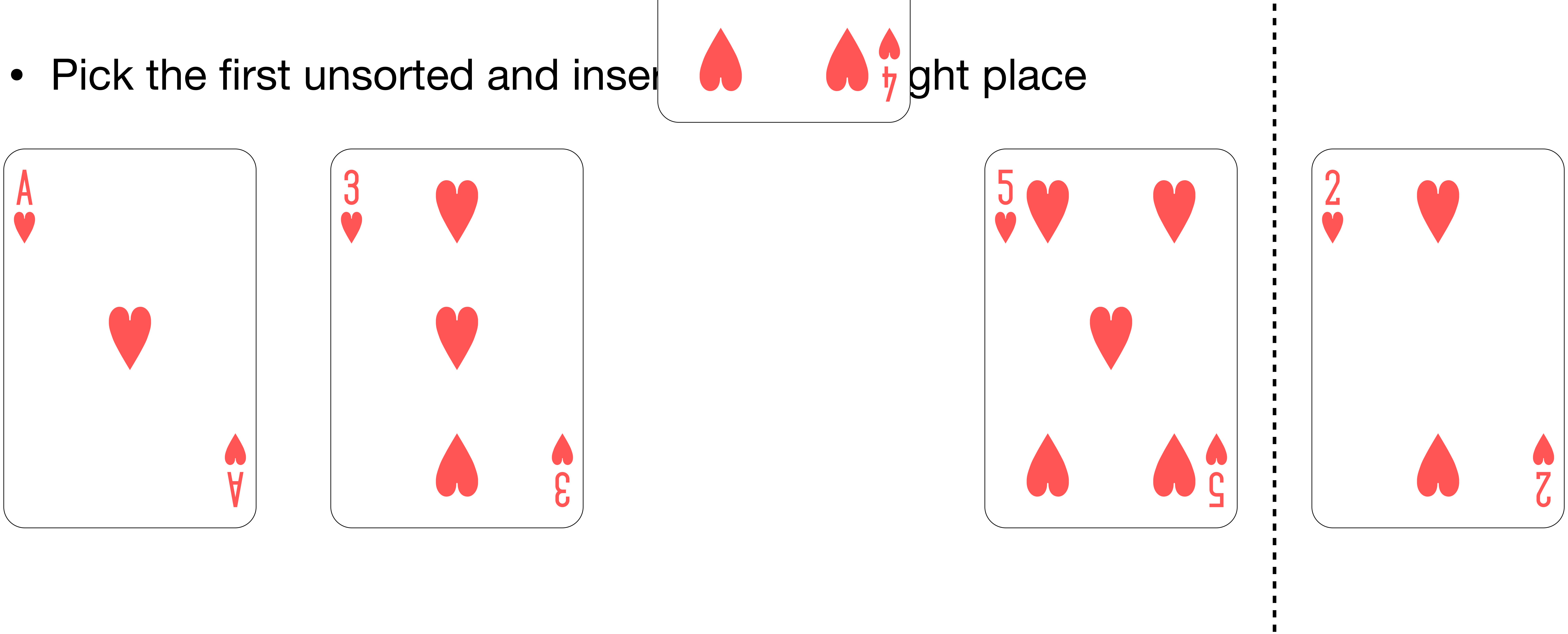


# Sorting

## Example 2



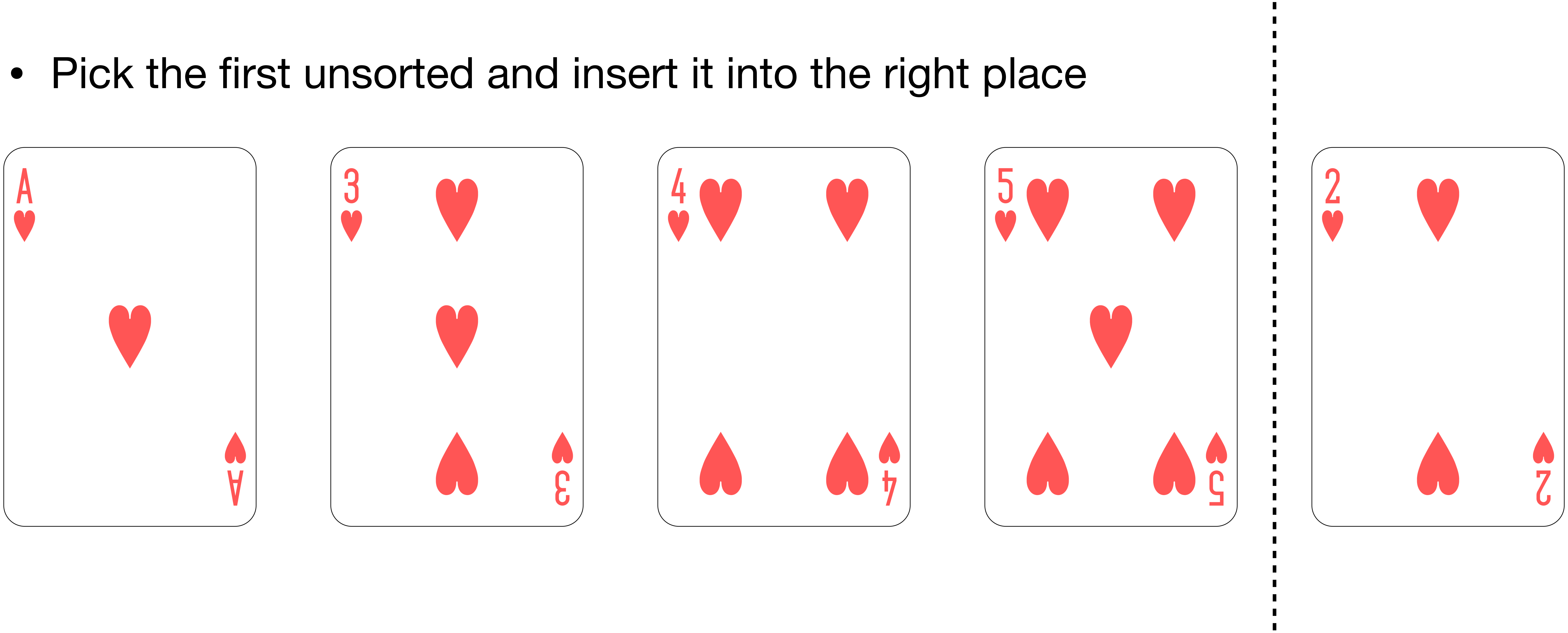

- Pick the first unsorted and inser ght place



# Sorting

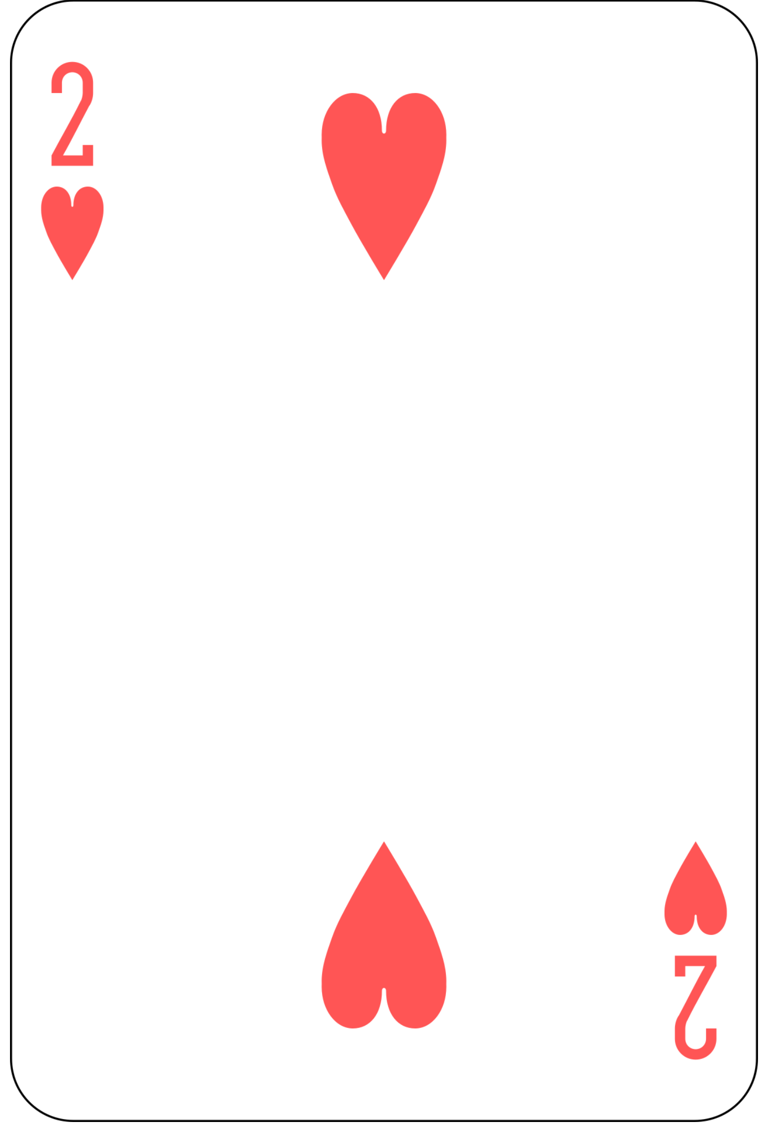
## Example 2

- Pick the first unsorted and insert it into the right place

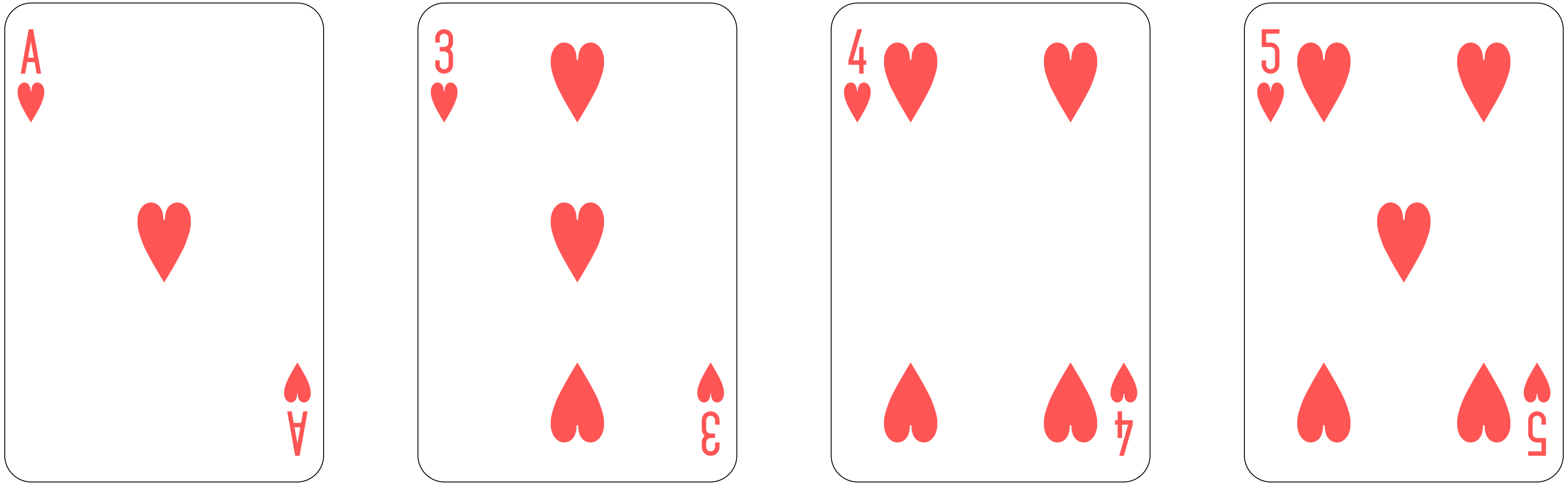


# Sorting

## Example 2

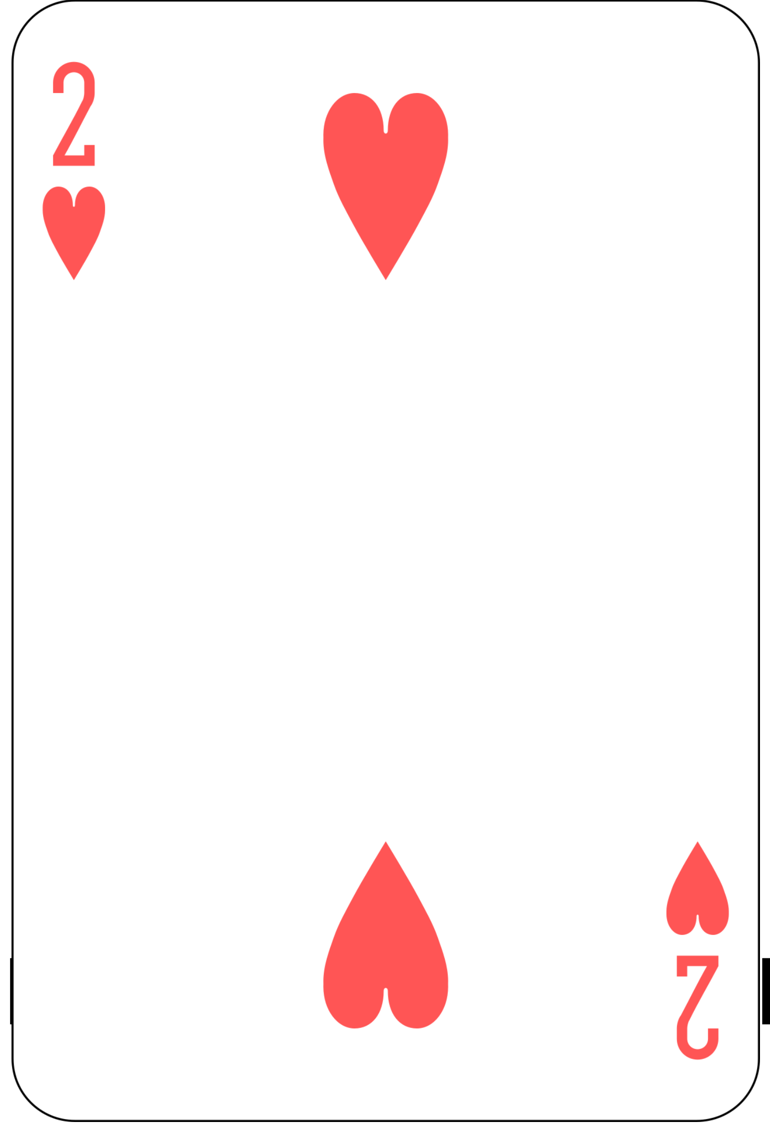



- Pick the first unsorted and insert it into the right place

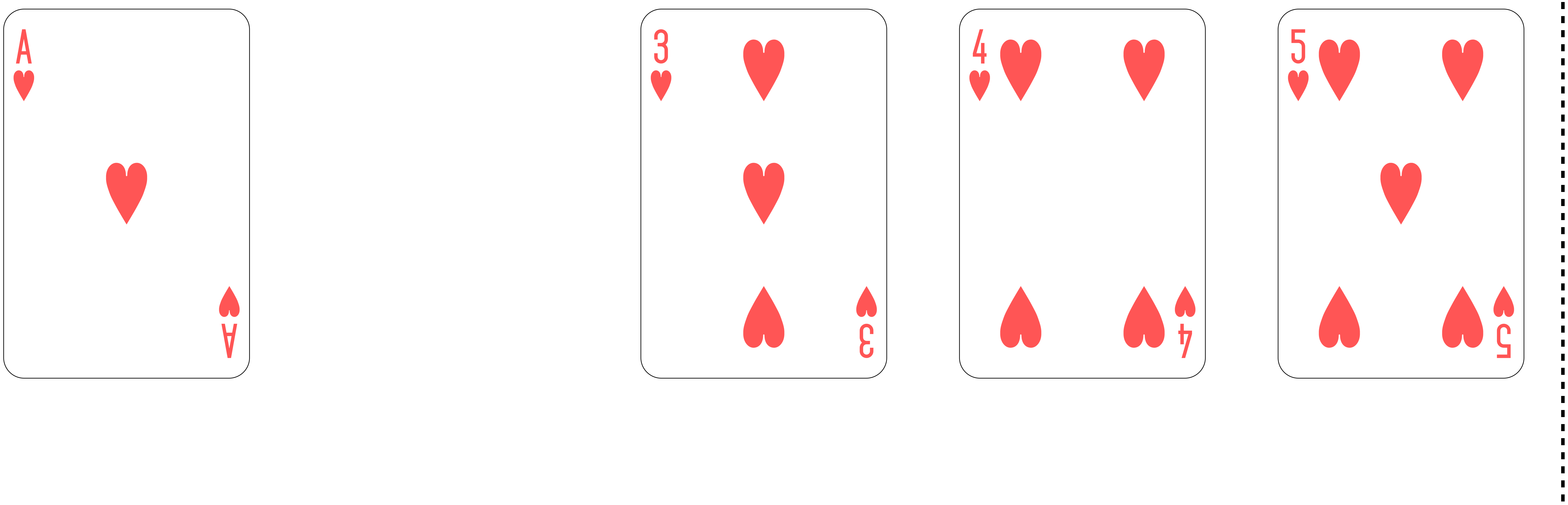


# Sorting

## Example 2



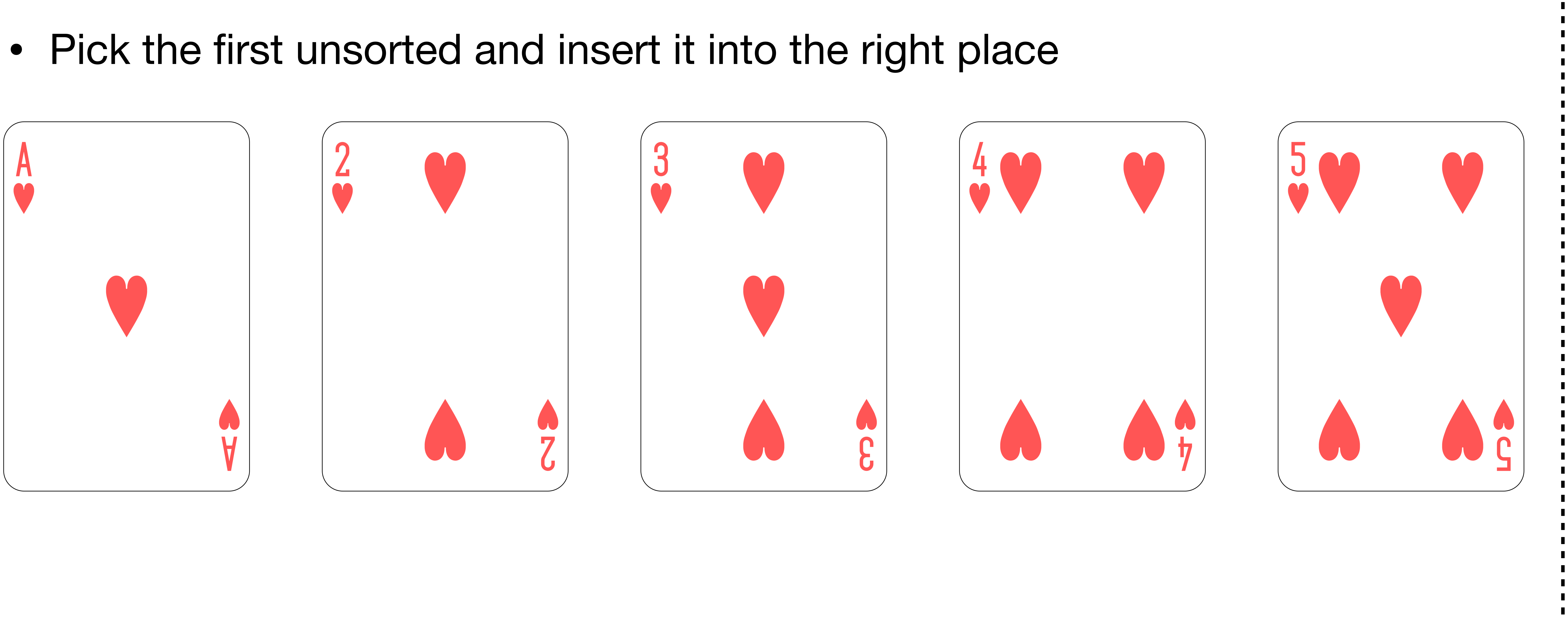
- Pick the first u nsert it into the right place



# Sorting

## Example 2

- Pick the first unsorted and insert it into the right place



# Sorting

## Example 2: Algorithm

```
For i = 2 to n:
    j = i - 1
    while j > 0 and A[j] > A[i]:
        A[j + 1] = A[j]
        j = j - 1
    A[j] = A[i]
```

- Take the first unsorted and insert it into the right place in the sorted pile
- Inserting means shifting everything after the card by one place
- This is called *insertion sort*.

# Sorting

## Example 2: Algorithm

```
For i = 2 to n:
    j = i - 1
    while j > 0 and A[j] > A[i]:
        A[j + 1] = A[j]
        j = j - 1
    A[j] = A[i]
```

- Comparison: worst-case $O(n^2)$
- Swap: worst-case $O(n^2)$

# Sorting

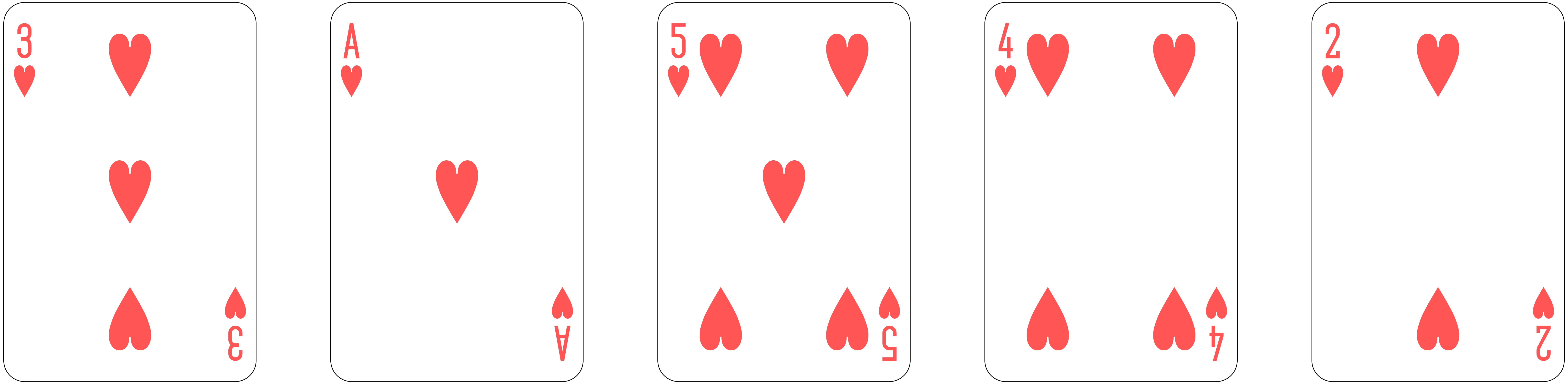
## Example 2: Algorithm

```
For i = 2 to n:
    j = i - 1
    while j > 0 and A[j] > A[i]:
        A[j + 1] = A[j]
        j = j - 1
    A[j] = A[i]
```

- Everything left of the line is sorted
- Take the first one on the right, scanning the left to find a place.

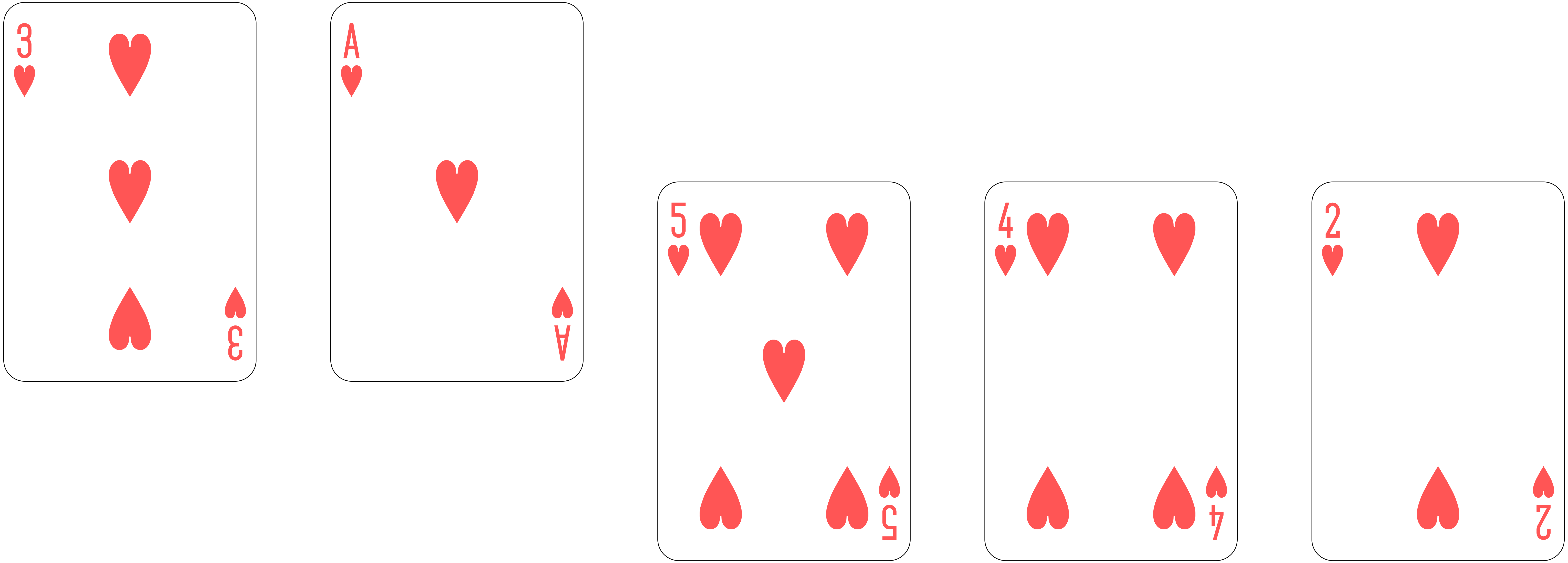
# Sorting
## Example 3

- Take two cards, swap them if out of order.



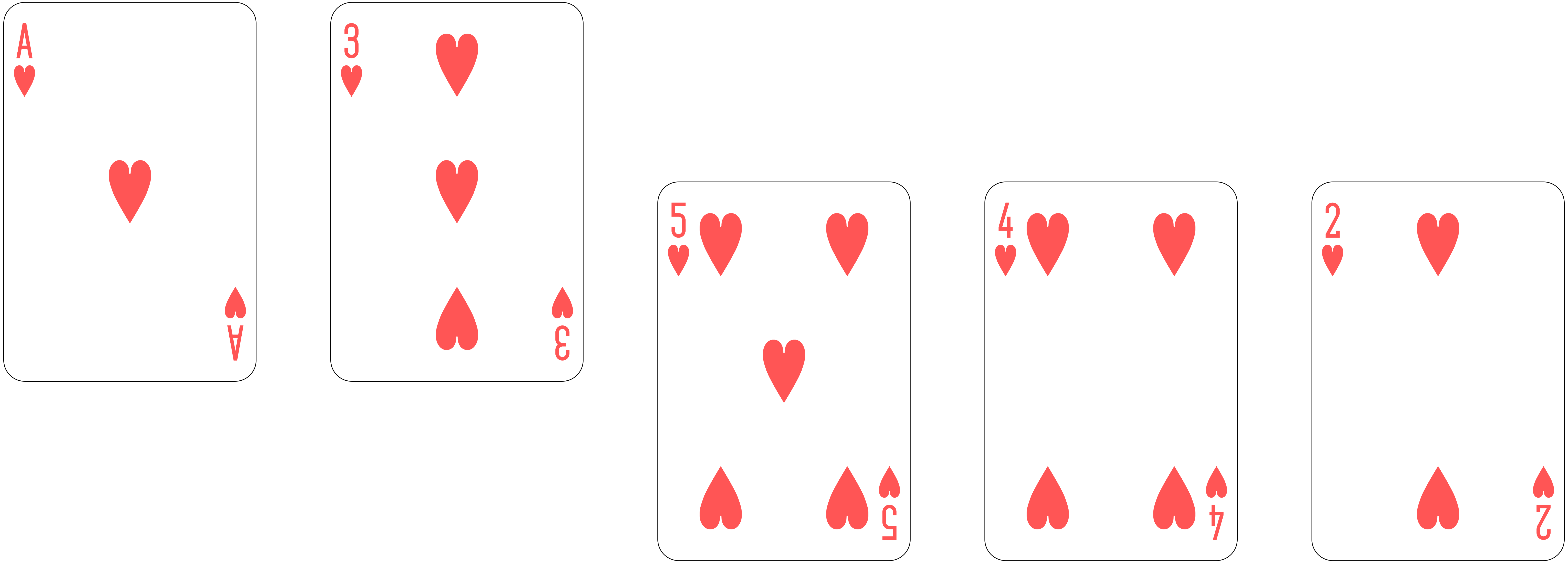
# Sorting

## Example 3



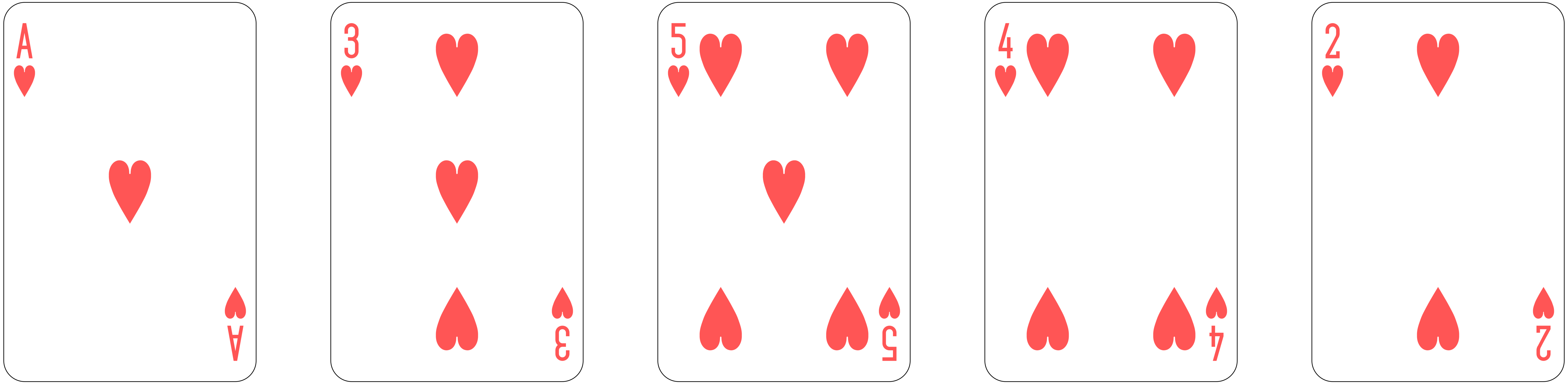
# Sorting

## Example 3



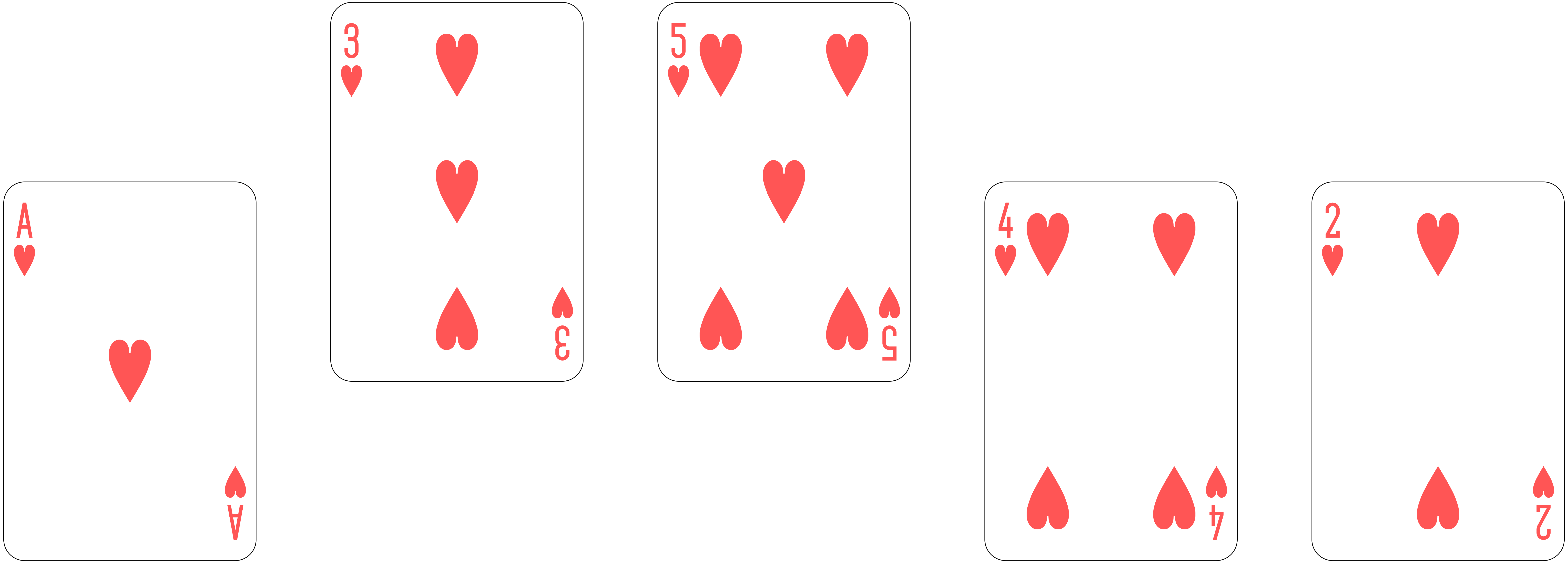
# Sorting

## Example 3



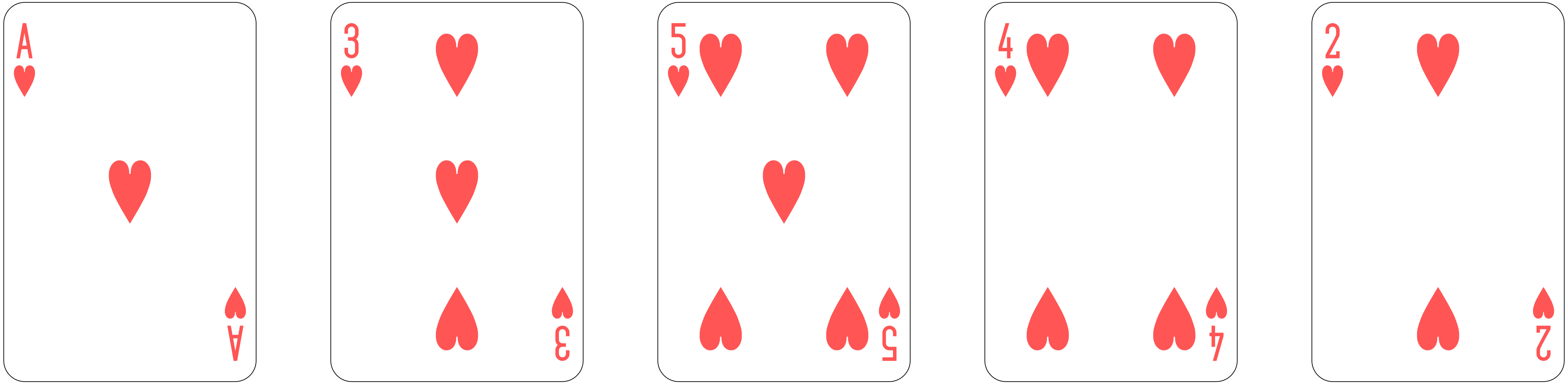
# Sorting

## Example 3



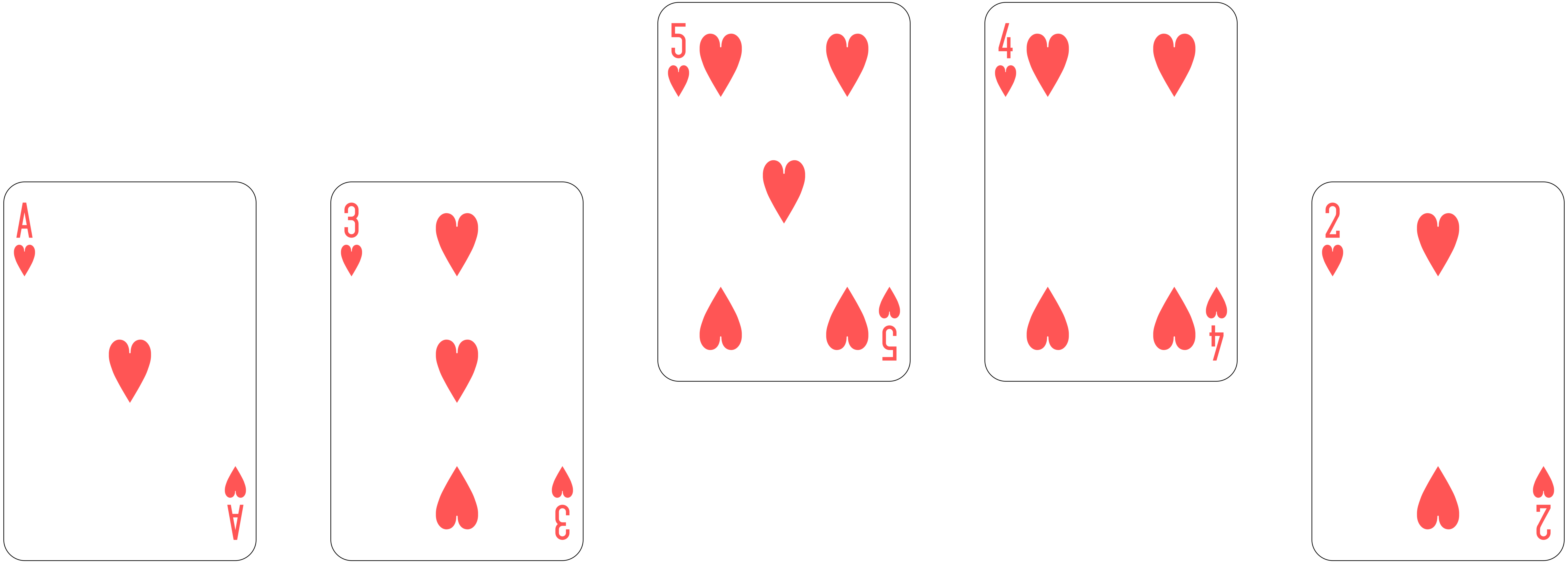
# Sorting

## Example 3



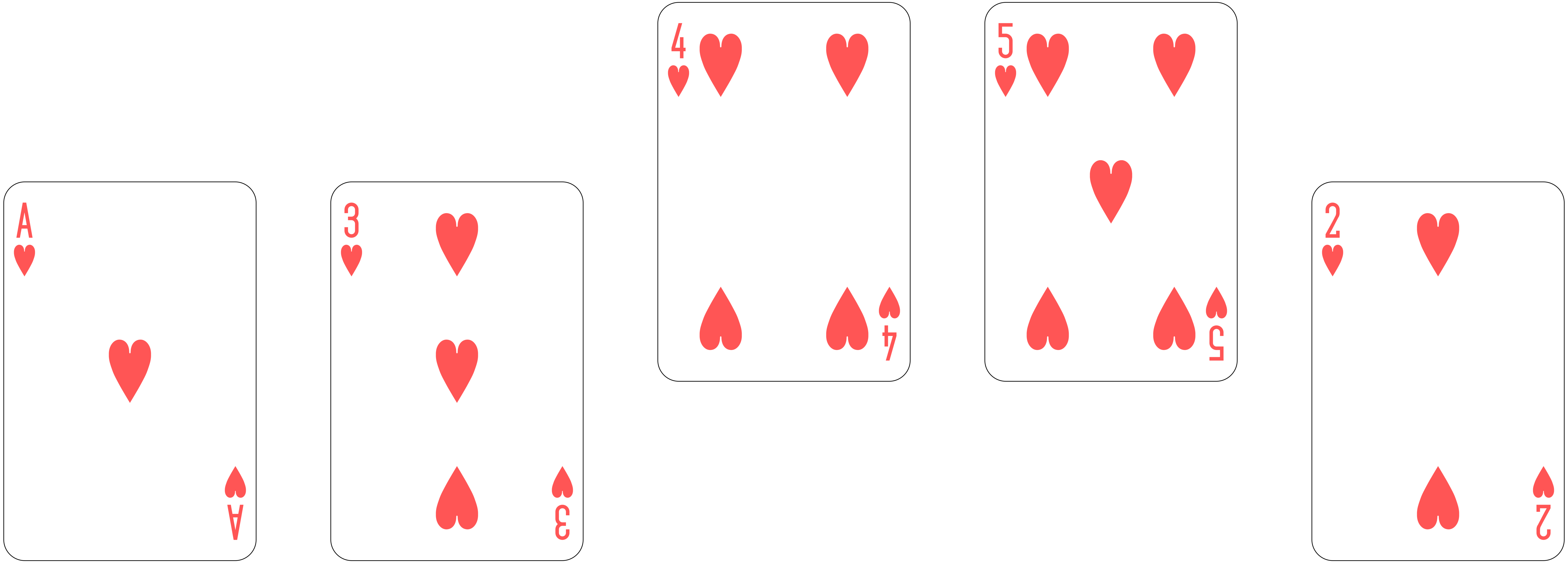
# Sorting

## Example 3



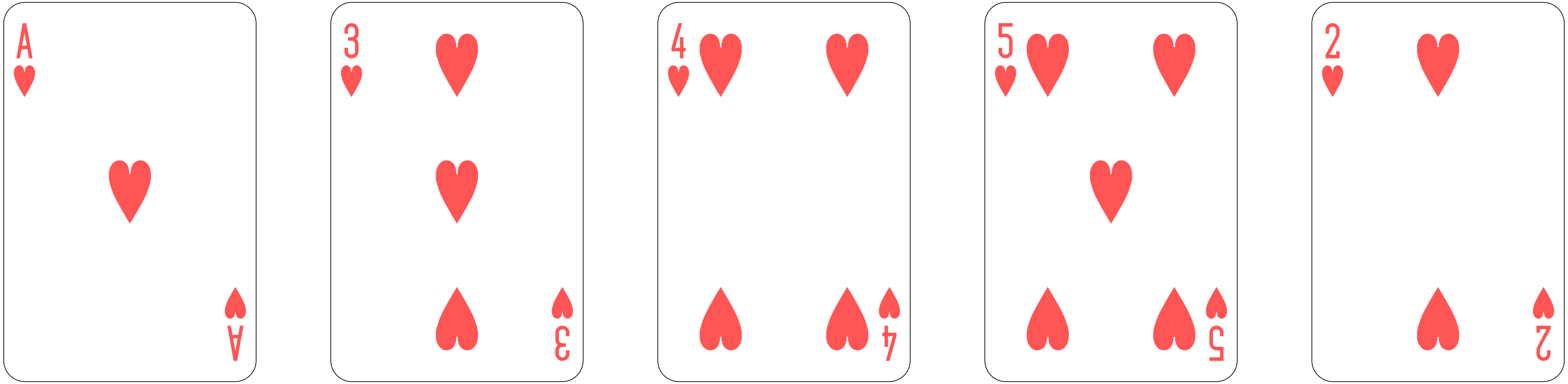
# Sorting

## Example 3



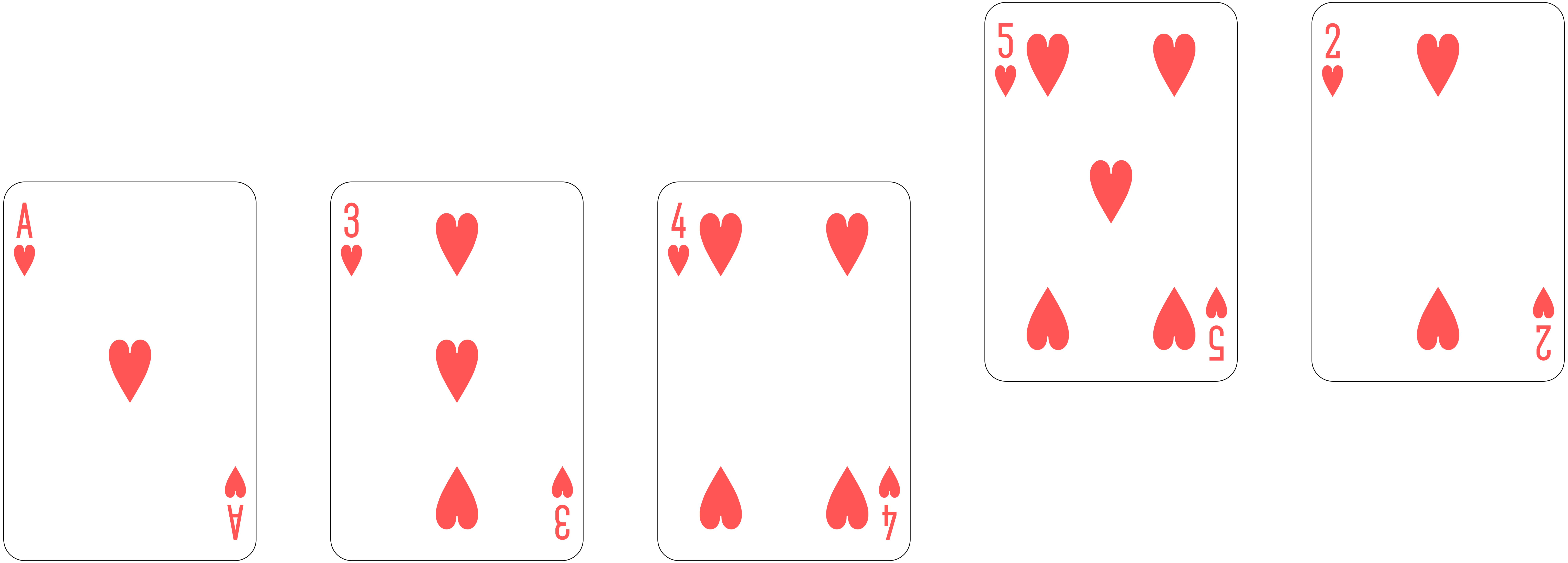
# Sorting

## Example 3



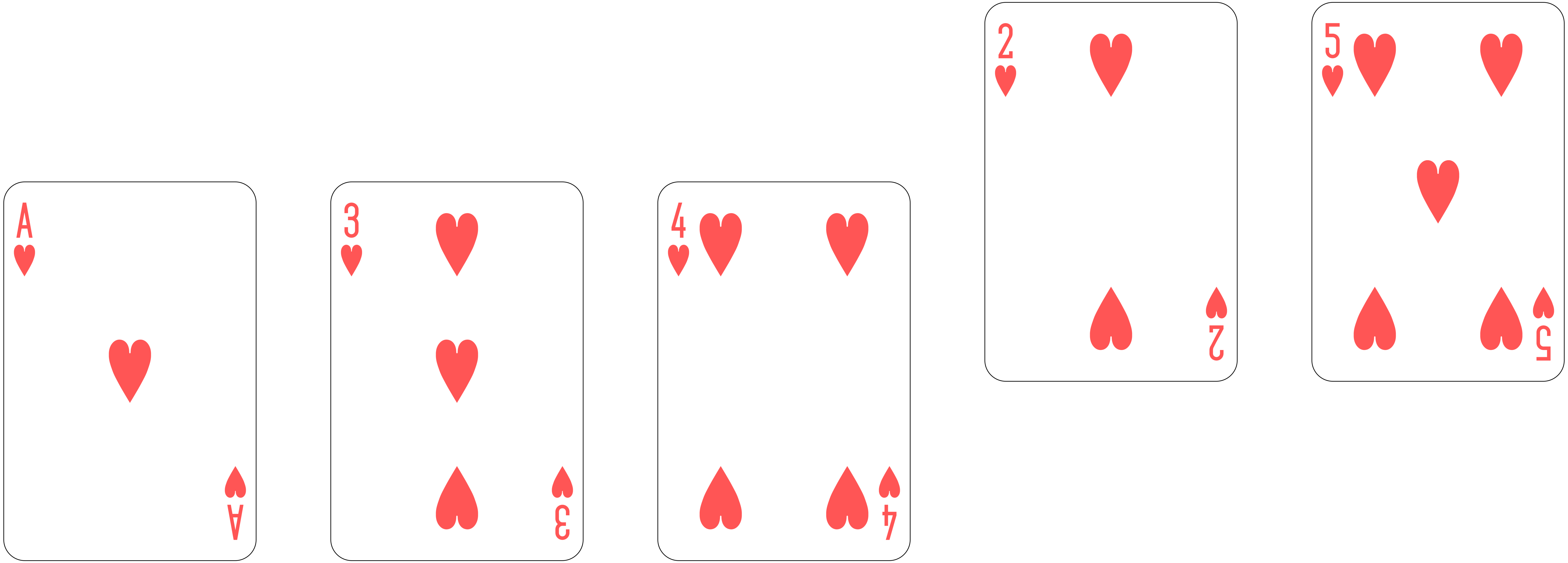
# Sorting

## Example 3



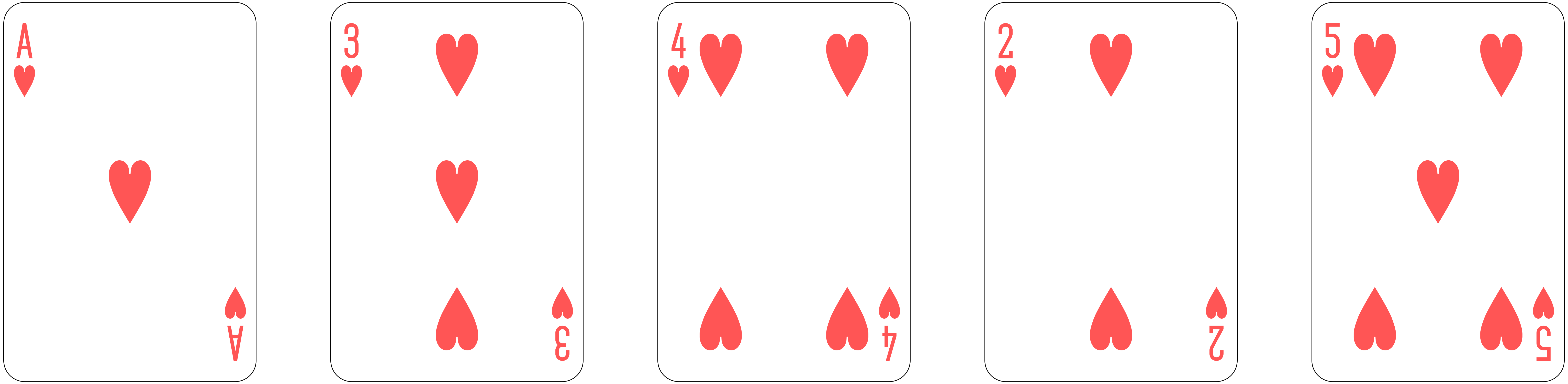
# Sorting

## Example 3



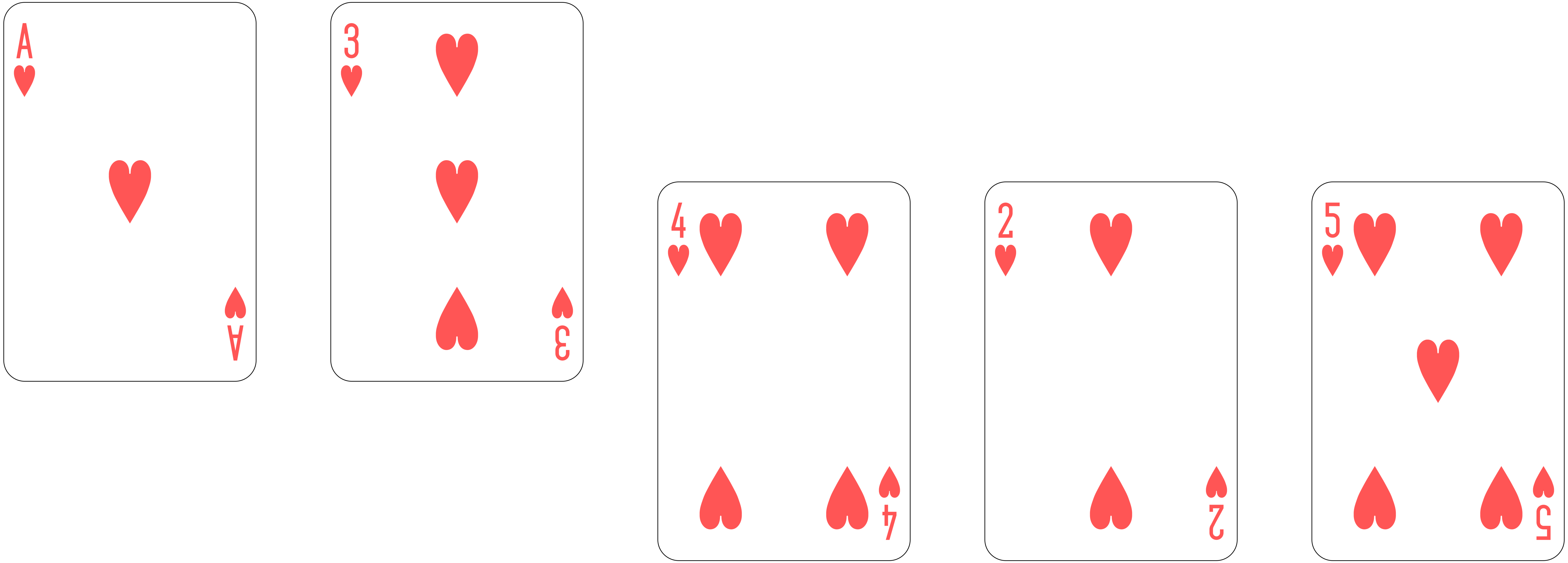
# Sorting

## Example 3



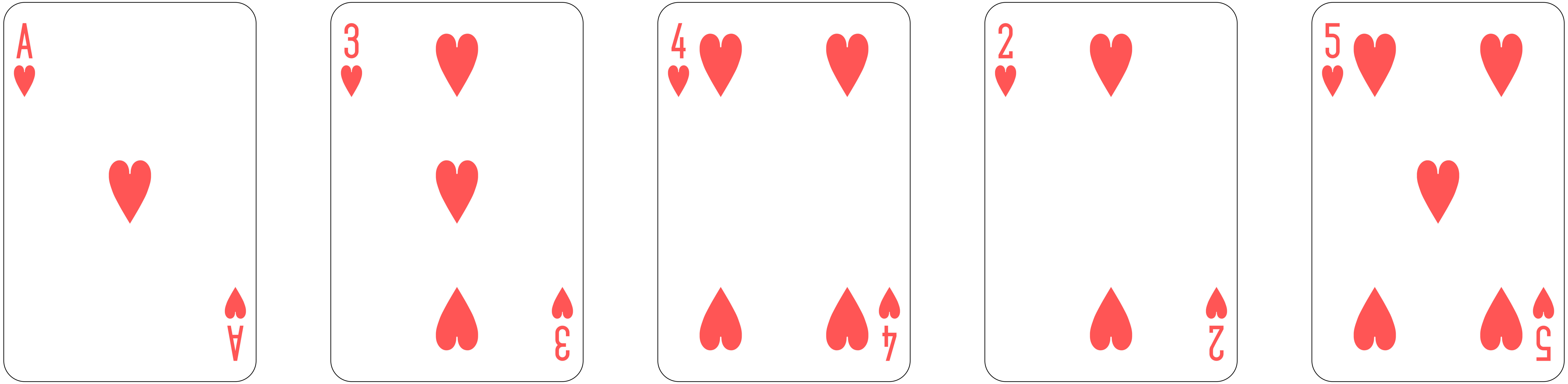
# Sorting

## Example 3



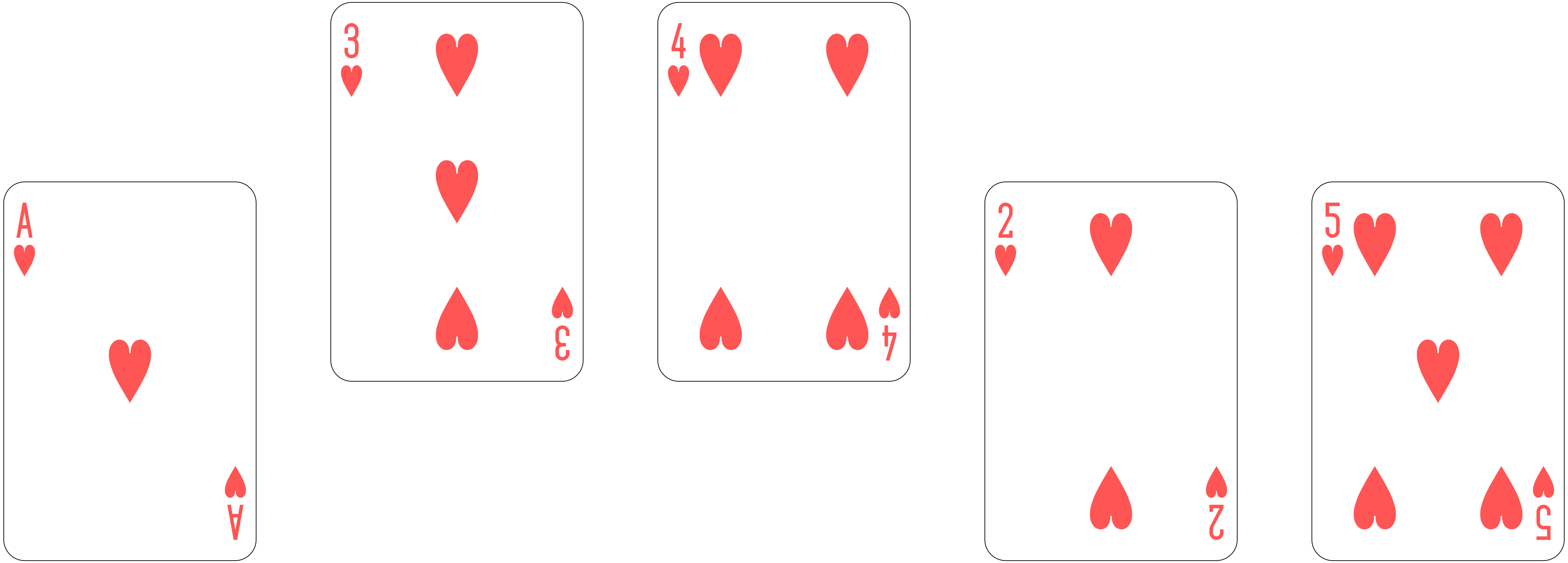
# Sorting

## Example 3



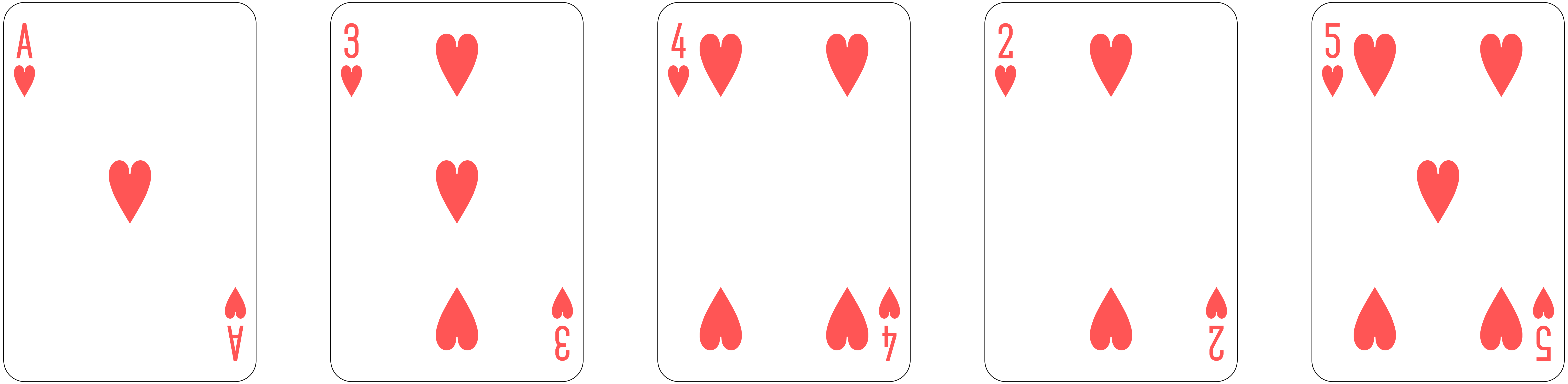
# Sorting

## Example 3



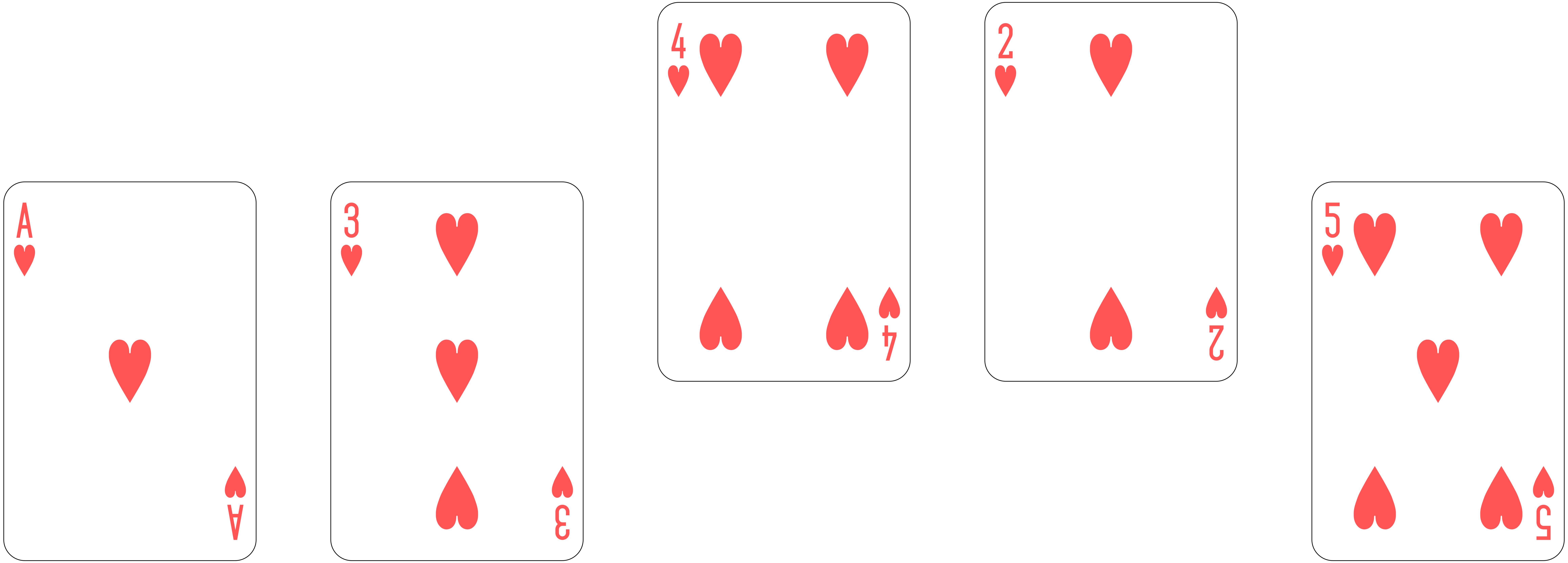
# Sorting

## Example 3



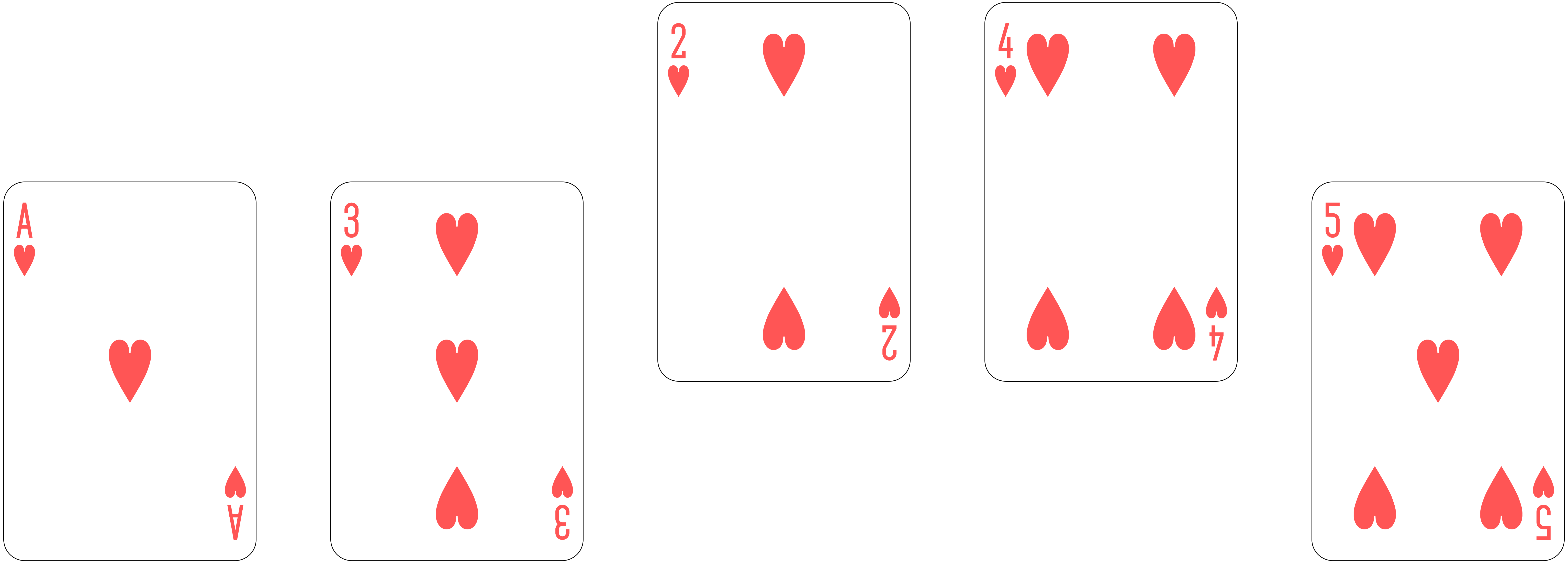
# Sorting

## Example 3



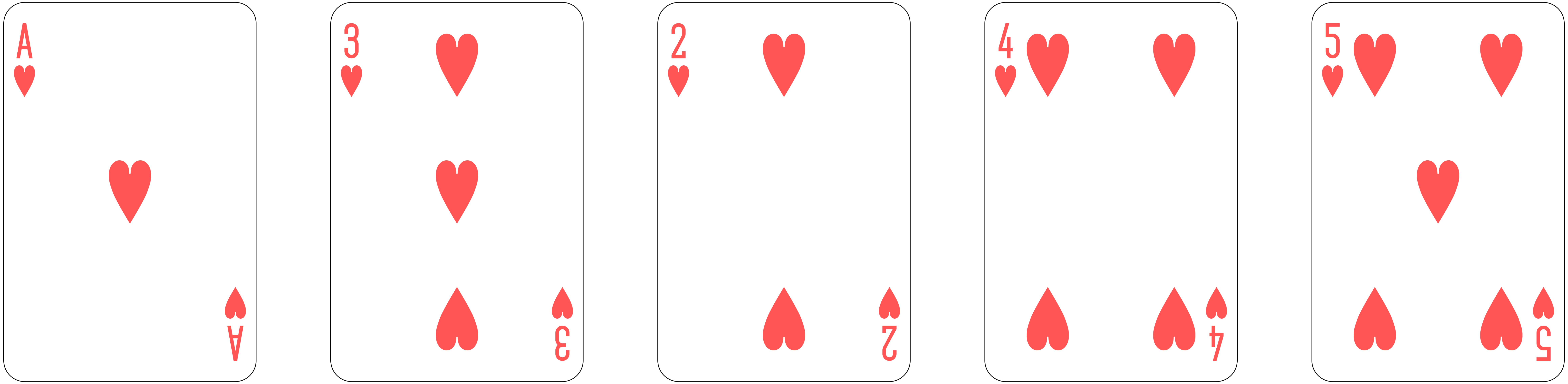
# Sorting

## Example 3



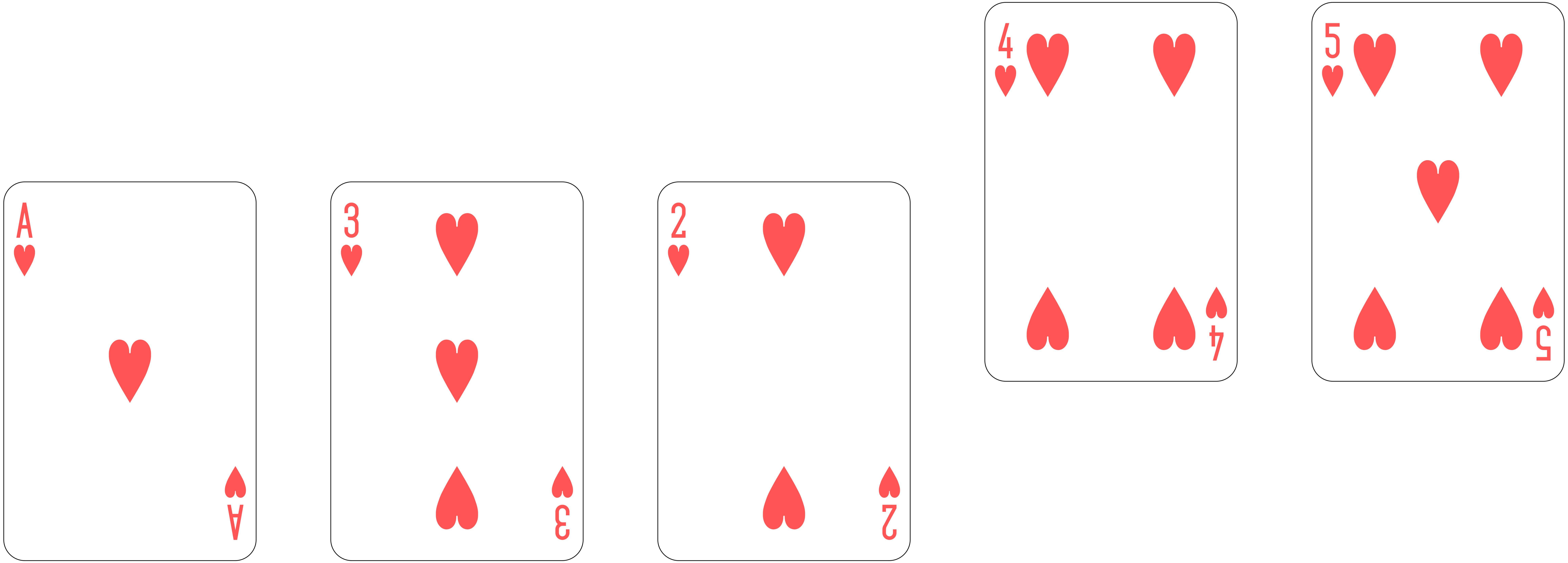
# Sorting

## Example 3



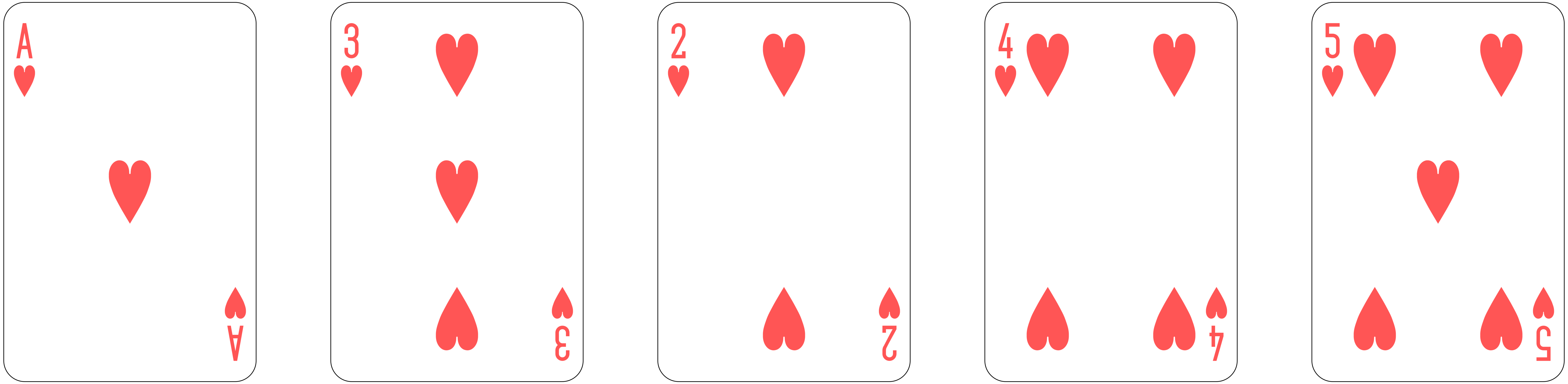
# Sorting

## Example 3



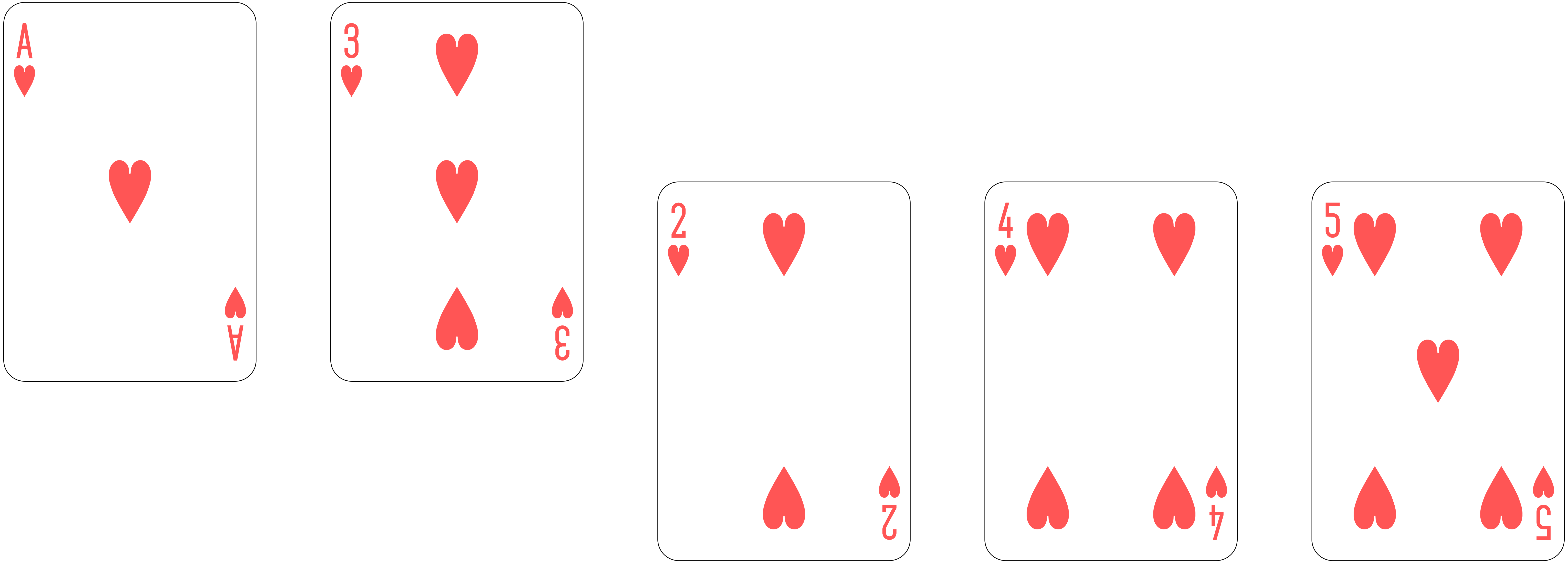
# Sorting

## Example 3



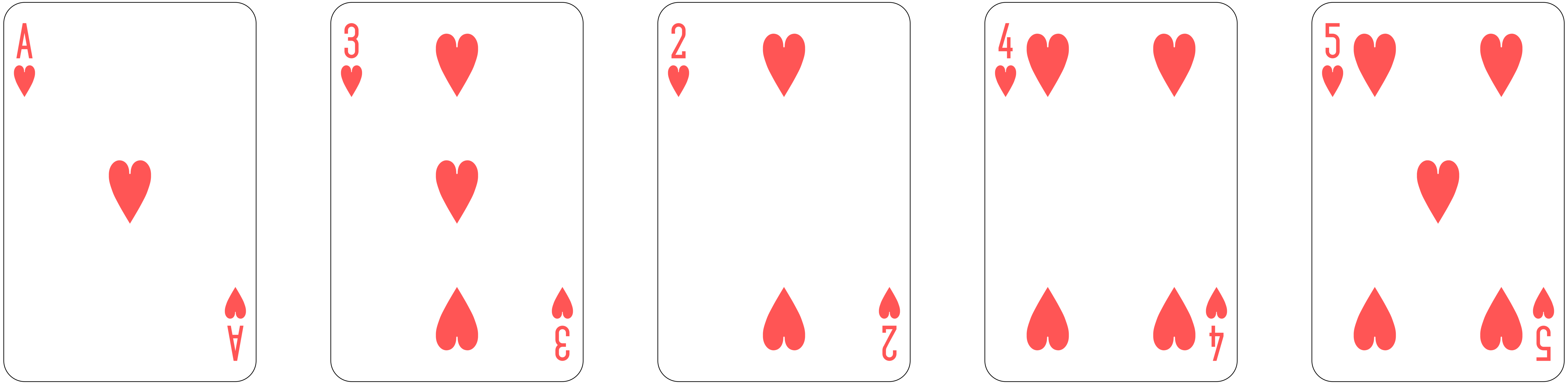
# Sorting

## Example 3



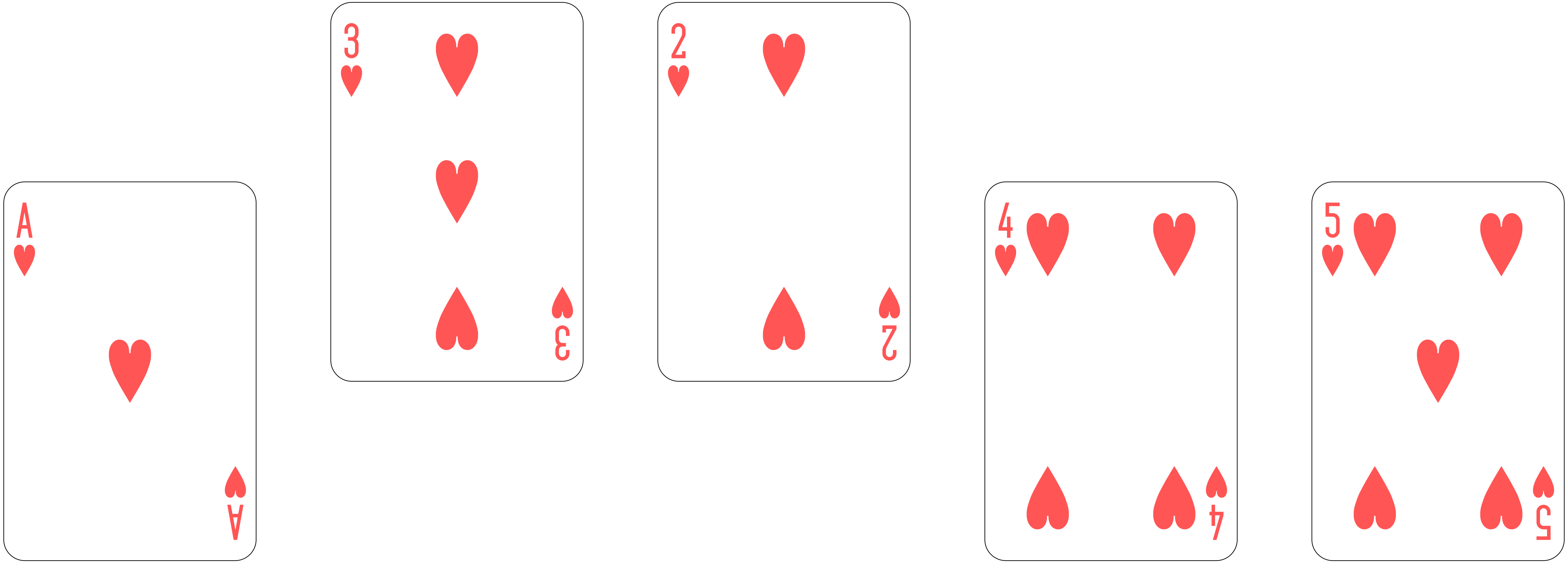
# Sorting

## Example 3



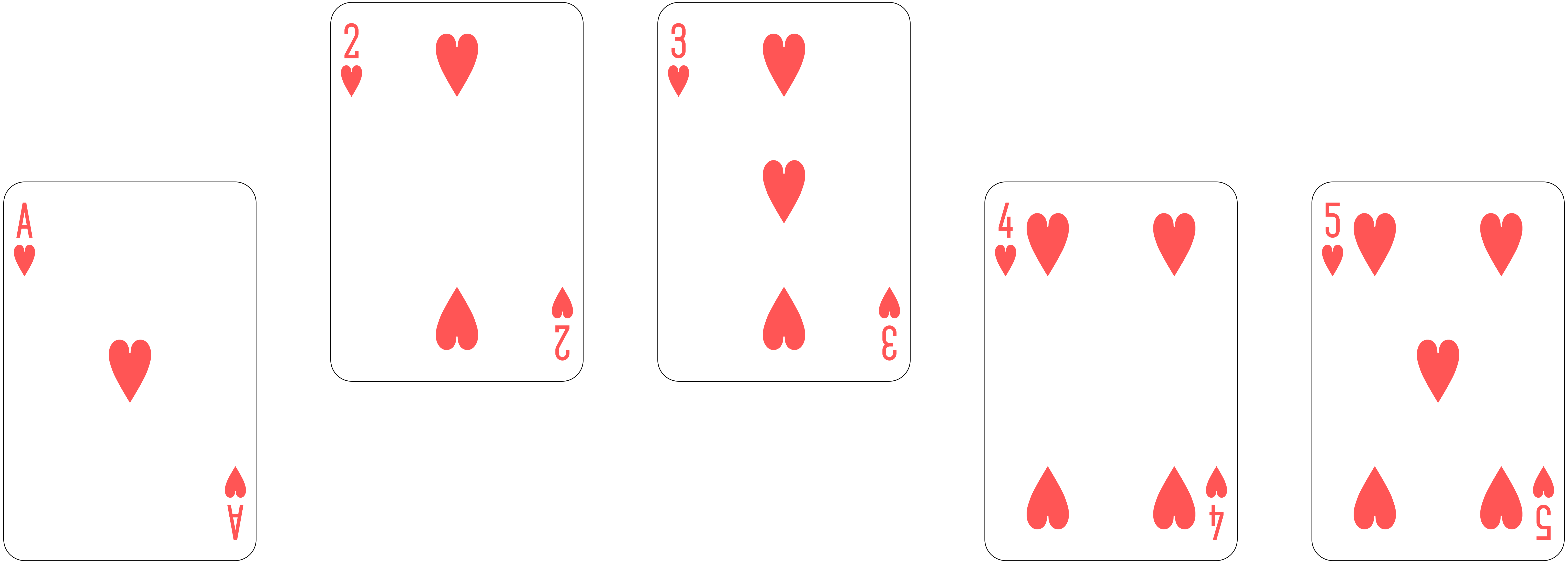
# Sorting

## Example 3

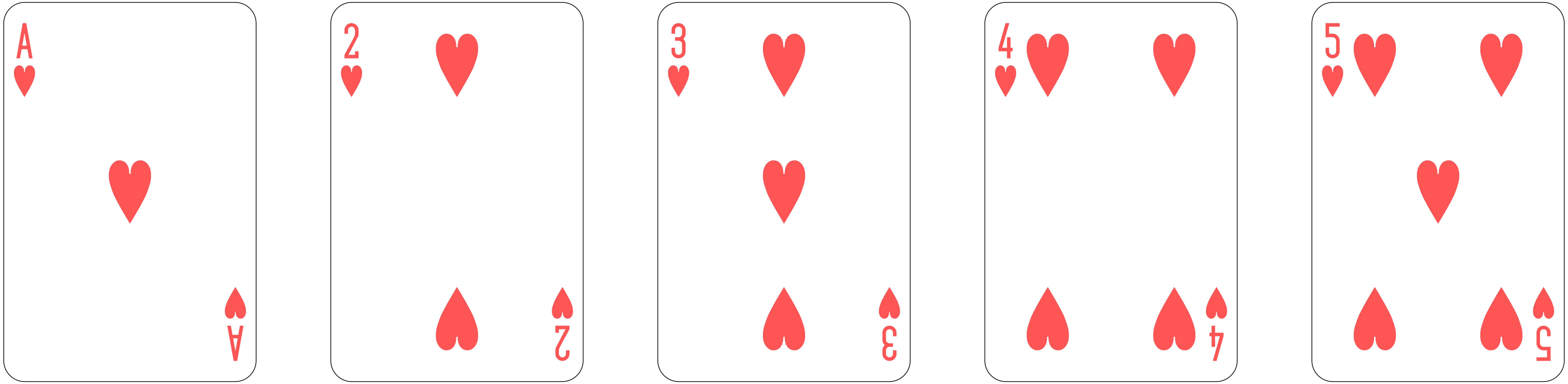


# Sorting

## Example 3

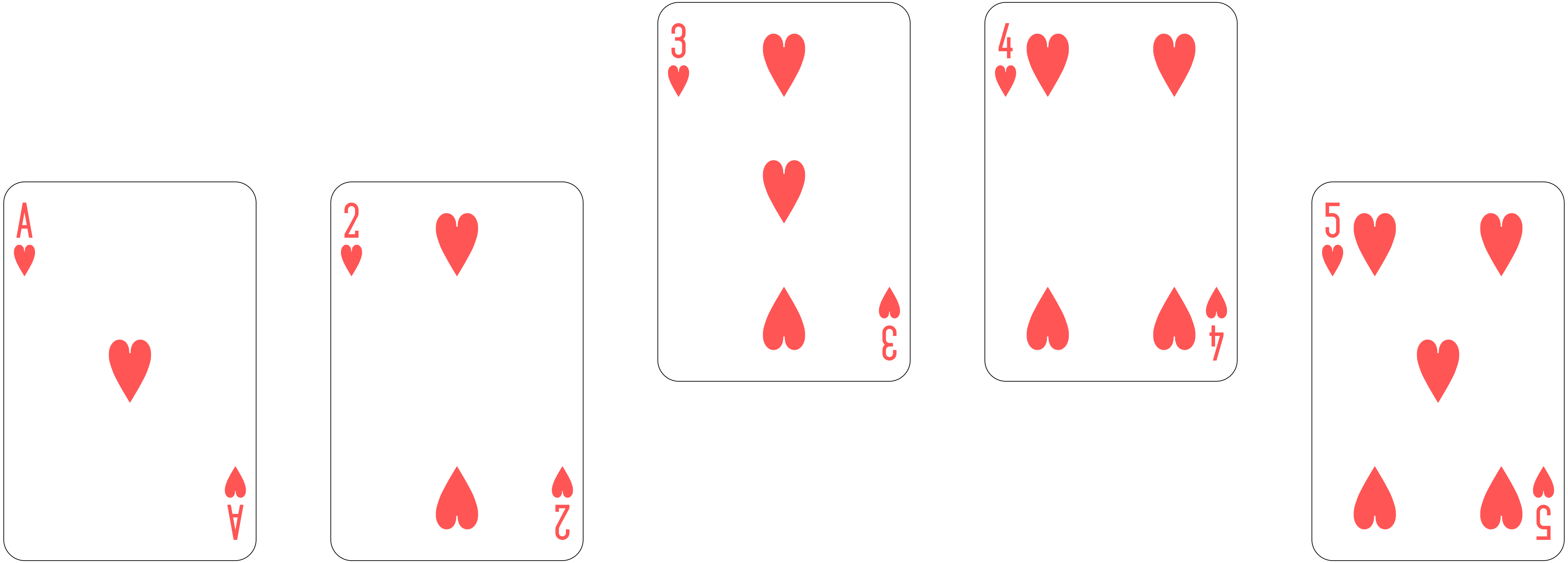


# Sorting
## Example 3



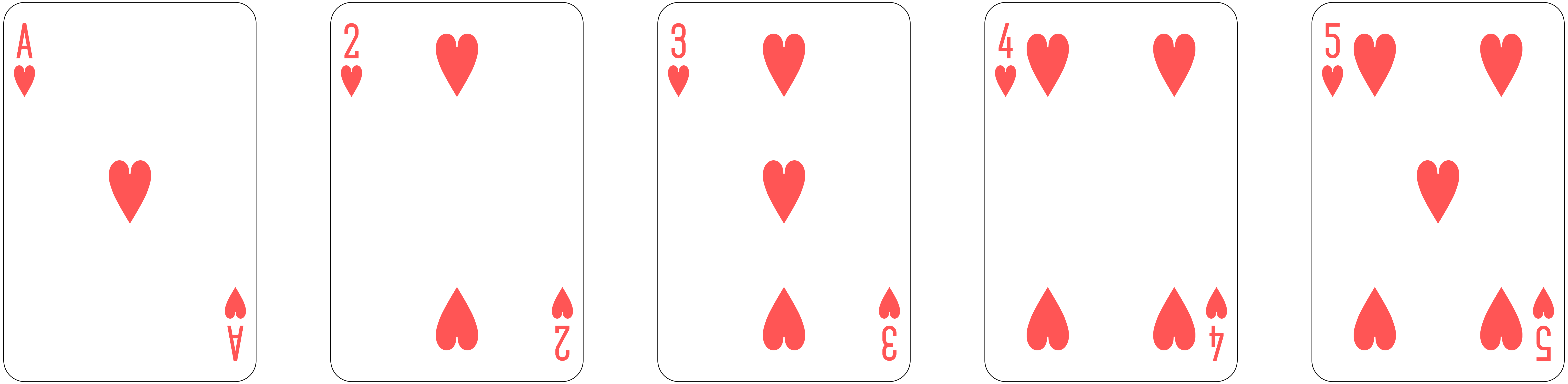
# Sorting

## Example 3



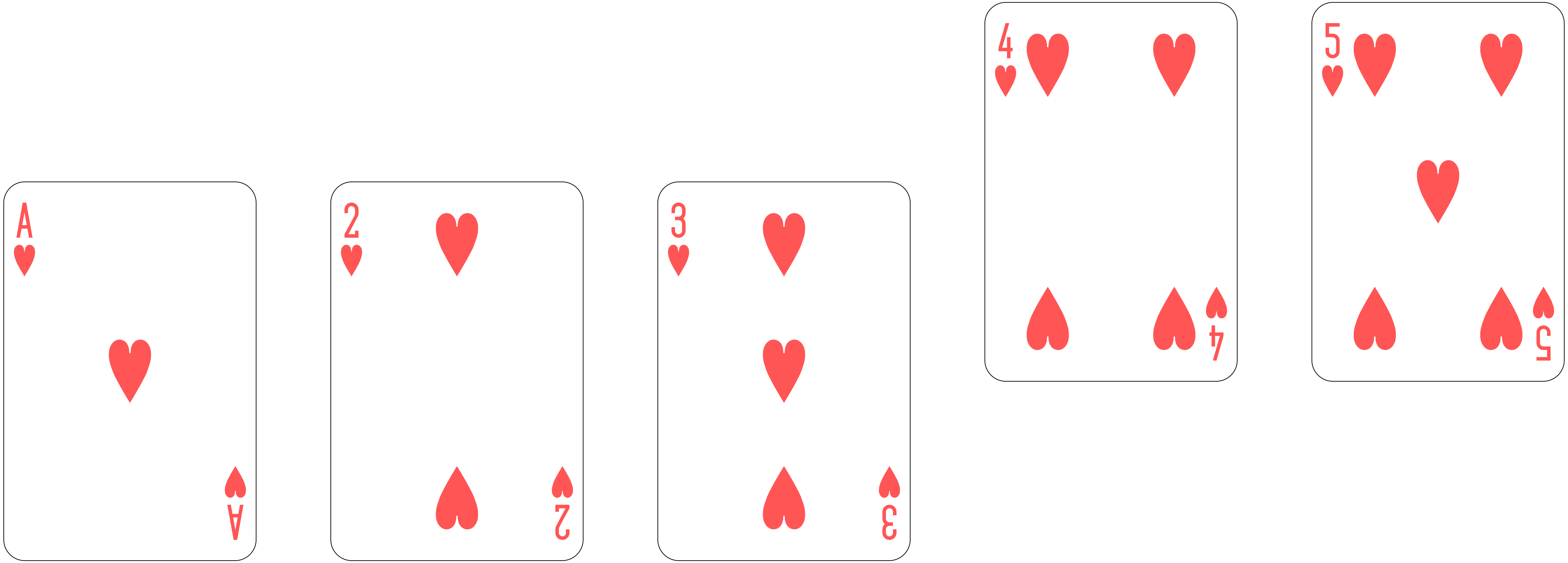
# Sorting

## Example 3



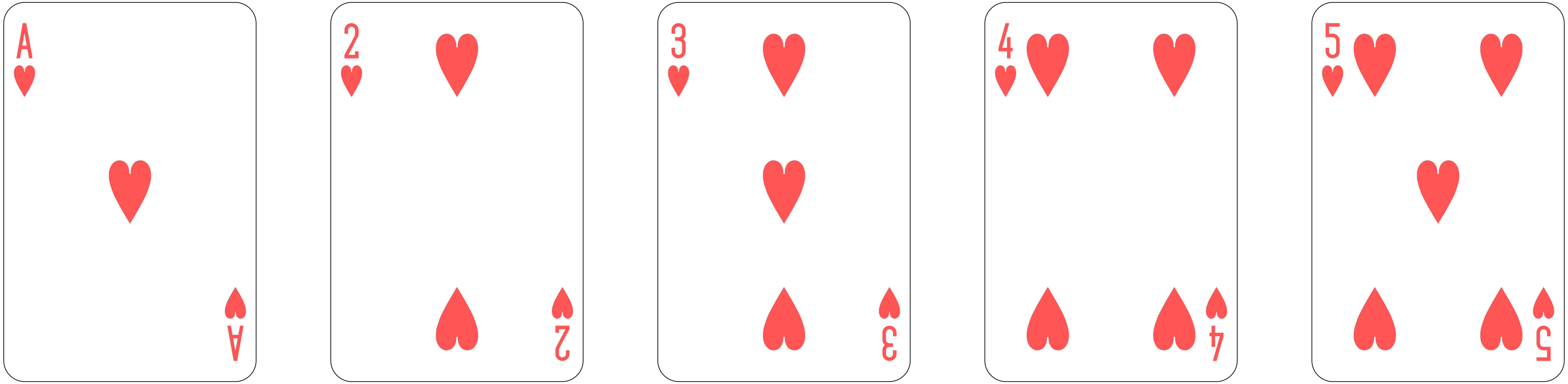
# Sorting

## Example 3



# Sorting

## Example 3



# Sorting

## Example 3: Algorithm

```
while True:
    swapped = False
    for i = 0 to n - 1:
        if A[i] > A[i + 1]:
            swap A[i], A[i + 1]
            swapped = True
    if not swapped:
        break
```

- Swap two adjacent elements if they are out of order
- How many rounds do we need to sort the entire list?
    - $n$. Why?
    - Every round, the largest element is pushed to the right.
- $O(n^2)$ comparisons, $O(n^2)$ swaps

# Sorting

## Example 3: Algorithm

```
while True:
    swapped = False
    for i = 0 to n - 1:
        if A[i] > A[i + 1]:
            swap A[i], A[i + 1]
            swapped = True
    if not swapped:
        break
```

- The largest element "bubbles" up.
- This is called *bubble sort*.

# Sorting

- Three $O(n^2)$ algorithms: Insertion sort, selection sort, bubble sort
- There are better algorithms
  - We will revisit after learning about trees!
- It's a whole can of worms

# Sorting

- Three $O(n^2)$ ... rt
- There are be
  - We will re
- It's a whole

## Sort Benchmark Home Page

**New:** We are happy to announce the 2022 winners listed below. The new, 2022 records are listed in green. Congratulations to the winners!

### Background

Until 2007, the sort benchmarks were primarily defined, sponsored and administered by Jim Gray. Following Jim's disappearance at sea in January 2007, the colleagues and sort benchmark winners. The Sort Benchmark committee members include:

- Chris Nyberg of Ordinal Technology Corp
- Mehul Shah of Aryn.ai
- George Porter of UC San Diego Computer Science & Engineering Dept

## Top Results

| | Daytona | Indy |
|---|---|---|
| Gray | 2016, 44.8 TB/min<br>**Tencent Sort**<br>100 TB in 134 Seconds<br>512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN)<br>Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao<br>Tencent Corporation<br>Mark R. Nutter, Jeremy D. Schaub | 2016, 60.7 TB/min<br>**Tencent Sort**<br>100 TB in 98.8 Seconds<br>512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN)<br>Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao<br>Tencent Corporation<br>Mark R. Nutter, Jeremy D. Schaub |
| Cloud | 2016, $1.44 / TB<br>**NADSort**<br>100 TB for $144<br>394 Alibaba Cloud ECS ecs.n1.large nodes x (Haswell E5-2680 v3, 8 GB memory, 40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk)<br>Qian Wang, Rong Gu, Yihua Huang<br>Nanjing University<br>Reynold Xin<br>Databricks Inc.<br>Wei Wu, Jun Song, Junluan Xia<br>Alibaba Group Inc. | 2022, $0.97 / TB<br>**Exoshuffle-CloudSort**<br>100 TB for $97<br>40 Amazon EC2 i4i.4xlarge nodes<br>1 Amazon EC2 r6i.2xlarge node<br>Amazon S3 storage<br>Frank Sifei Luan<br>UC Berkeley<br>Stephanie Wang<br>UC Berkeley and Anyscale<br>Samyukta Yagati, Sean Kim, Kenneth Lien, Isaac Ong, Tony Hong<br>UC Berkeley<br>SangBin Cho, Eric Liang<br>Anyscale<br>Ion Stoica<br>UC Berkeley and Anyscale |
| | 2016, 37 TB | 2016, 55 TB |