Design and Analysis of Algorithms

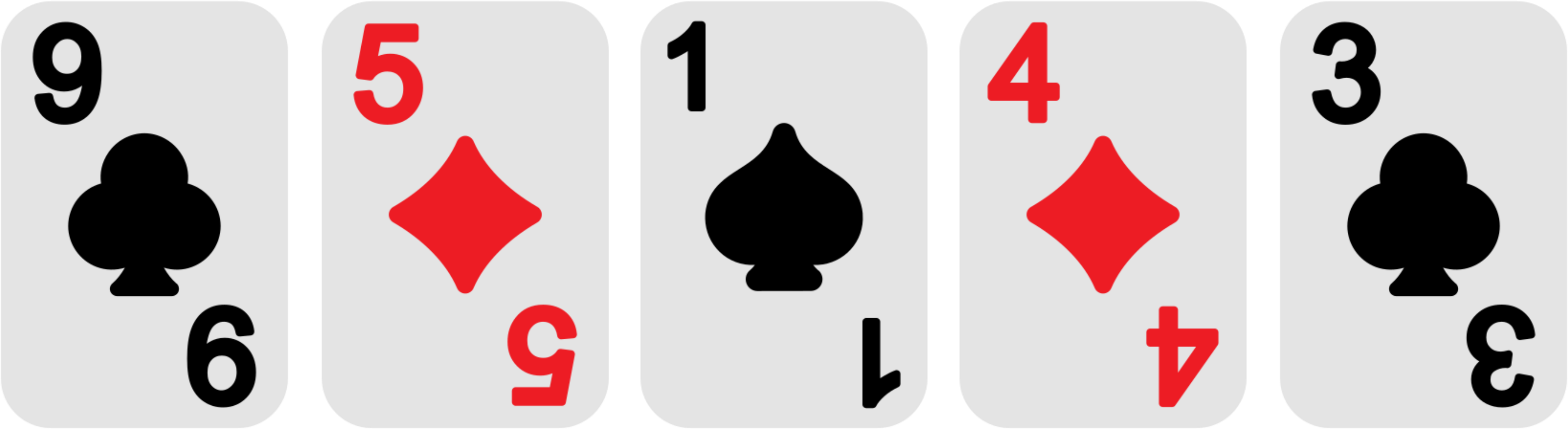
Algorithm Design Techniques

- **Brute Force and Exhaustive Search**
- Divide-and-Conquer
- **Decrease-and-Conquer**
- Transform-and-Conquer
- Space and Time Trade-Offs
- Dynamic Programming
- Greedy Technique
- Iterative Improvement
- Backtracking
- Branch-and-Bound

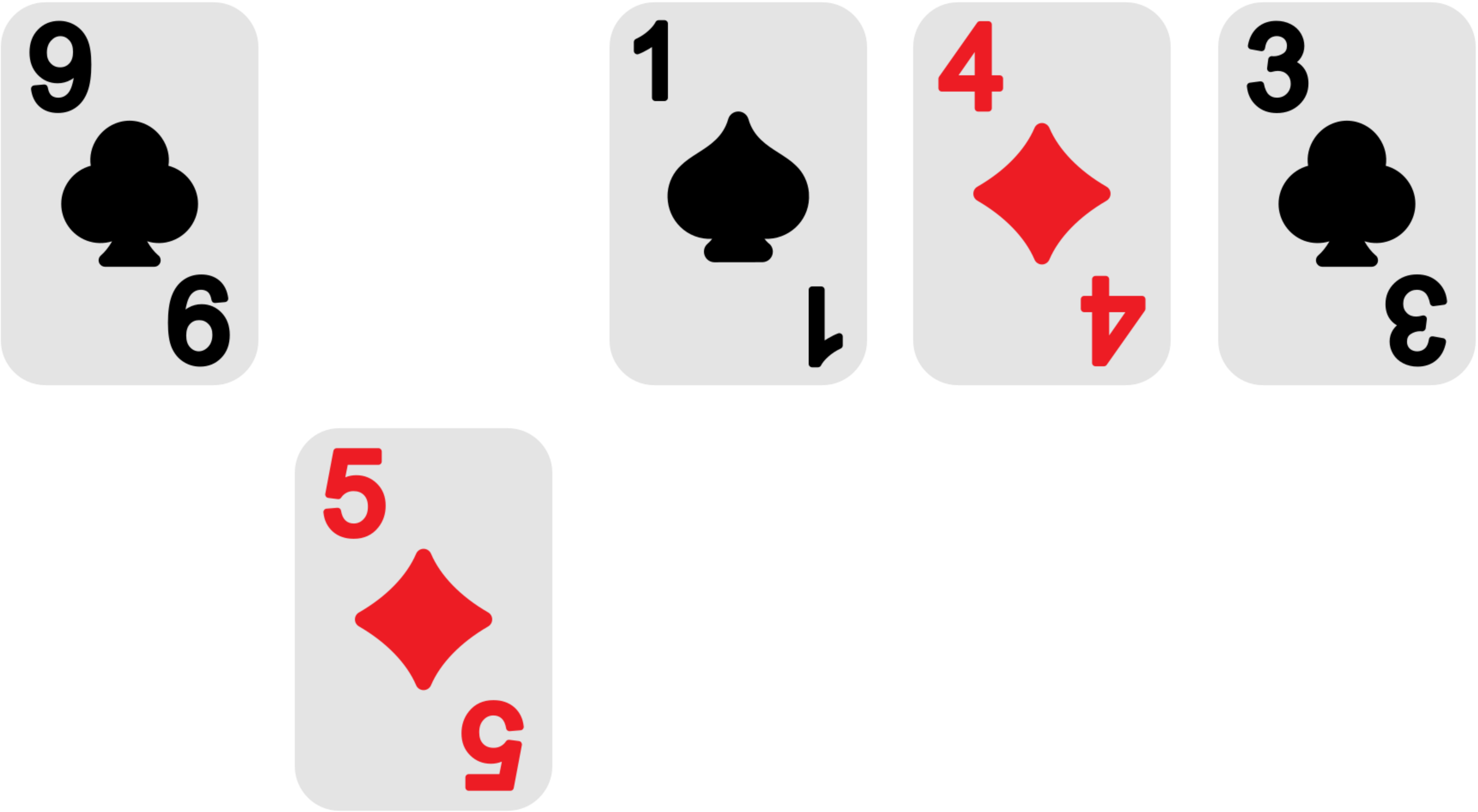
Decrease-and-Conquer

- **Reduce** original problem instance to smaller instance of the same problem.
- **Solve** smaller instance.
- **Extend** solution of smaller instance to obtain solution to original instance.

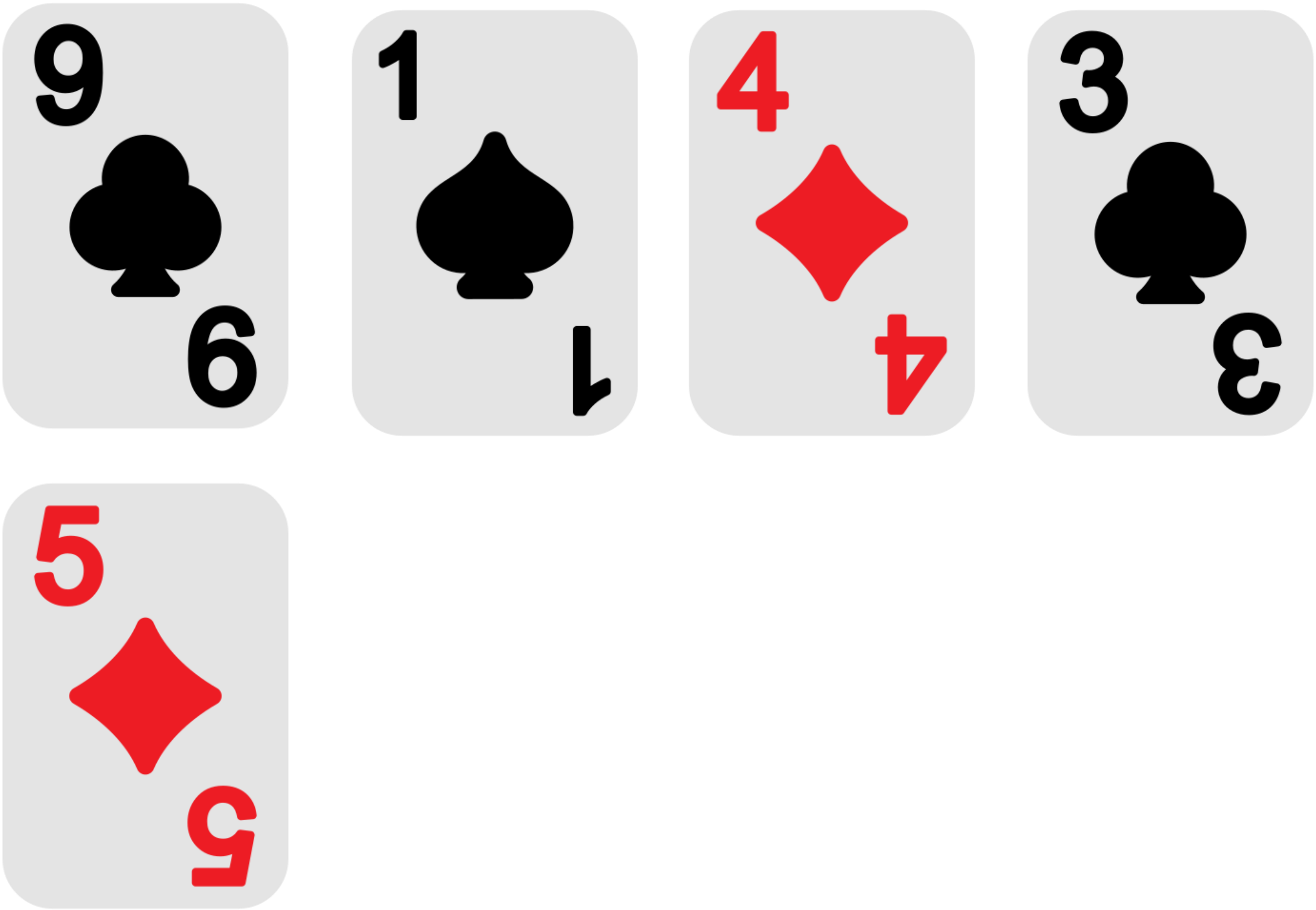
Sorting Playing Cards



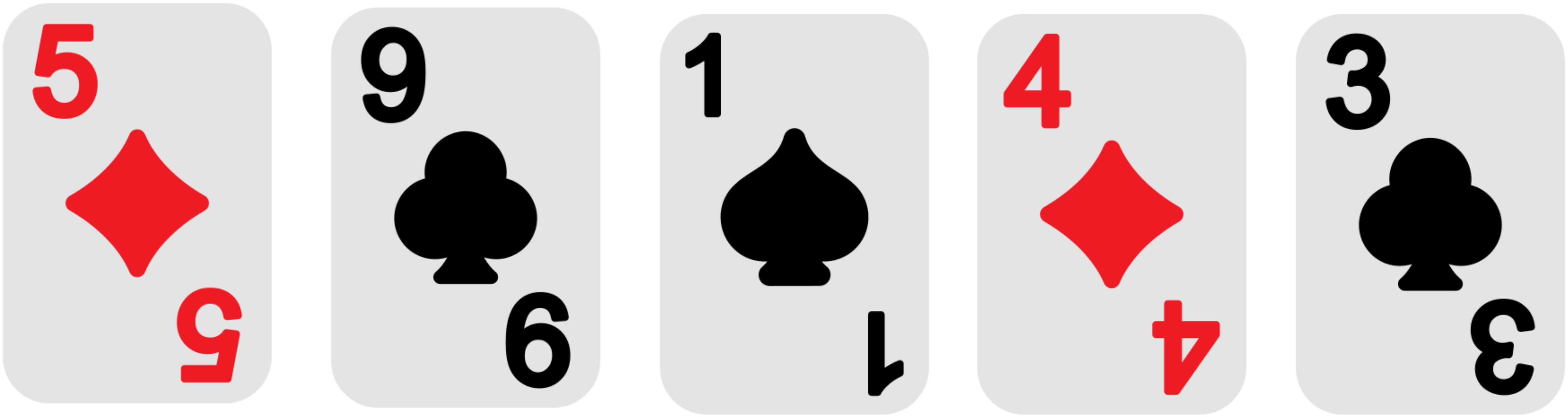
Sorting Playing Cards



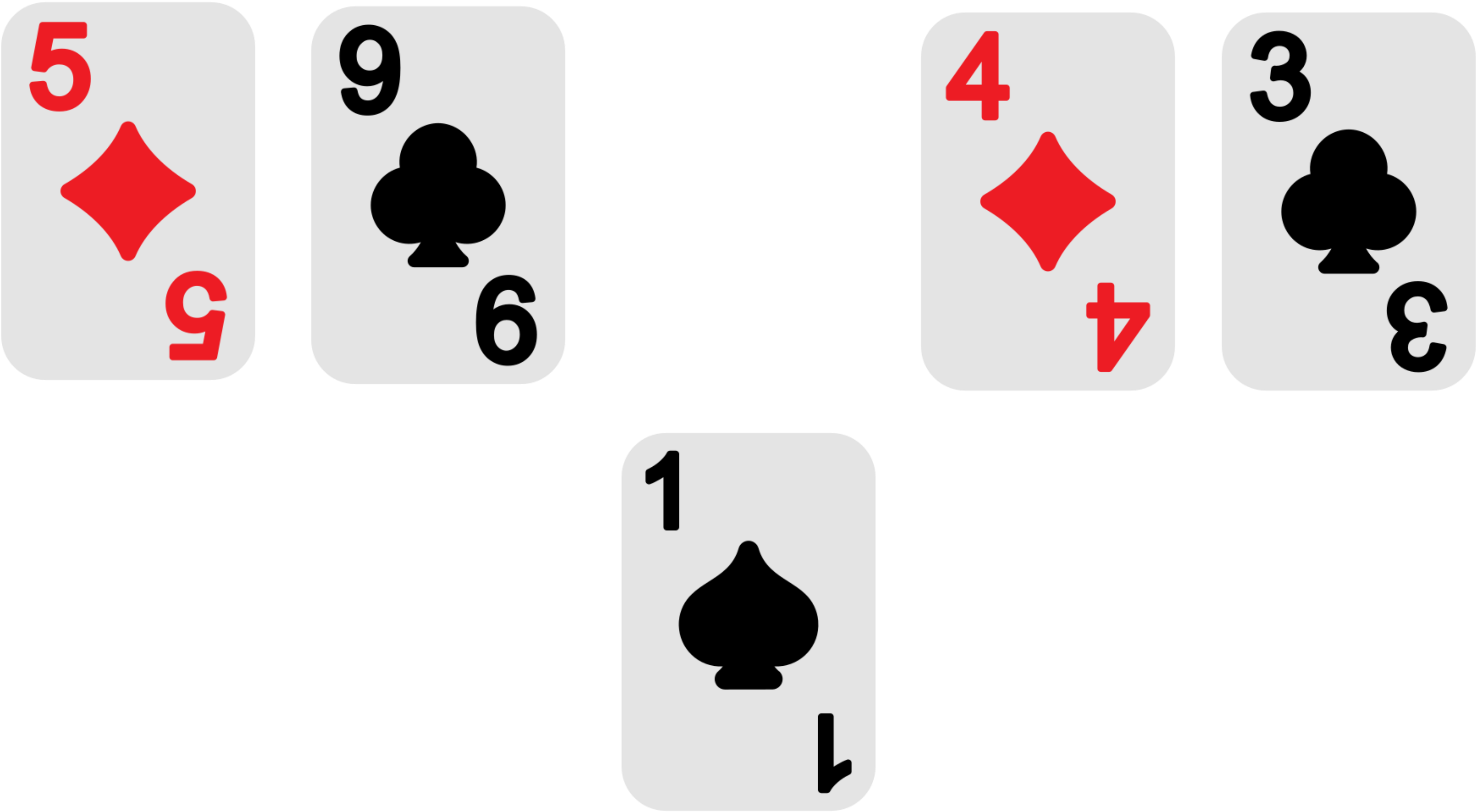
Sorting Playing Cards



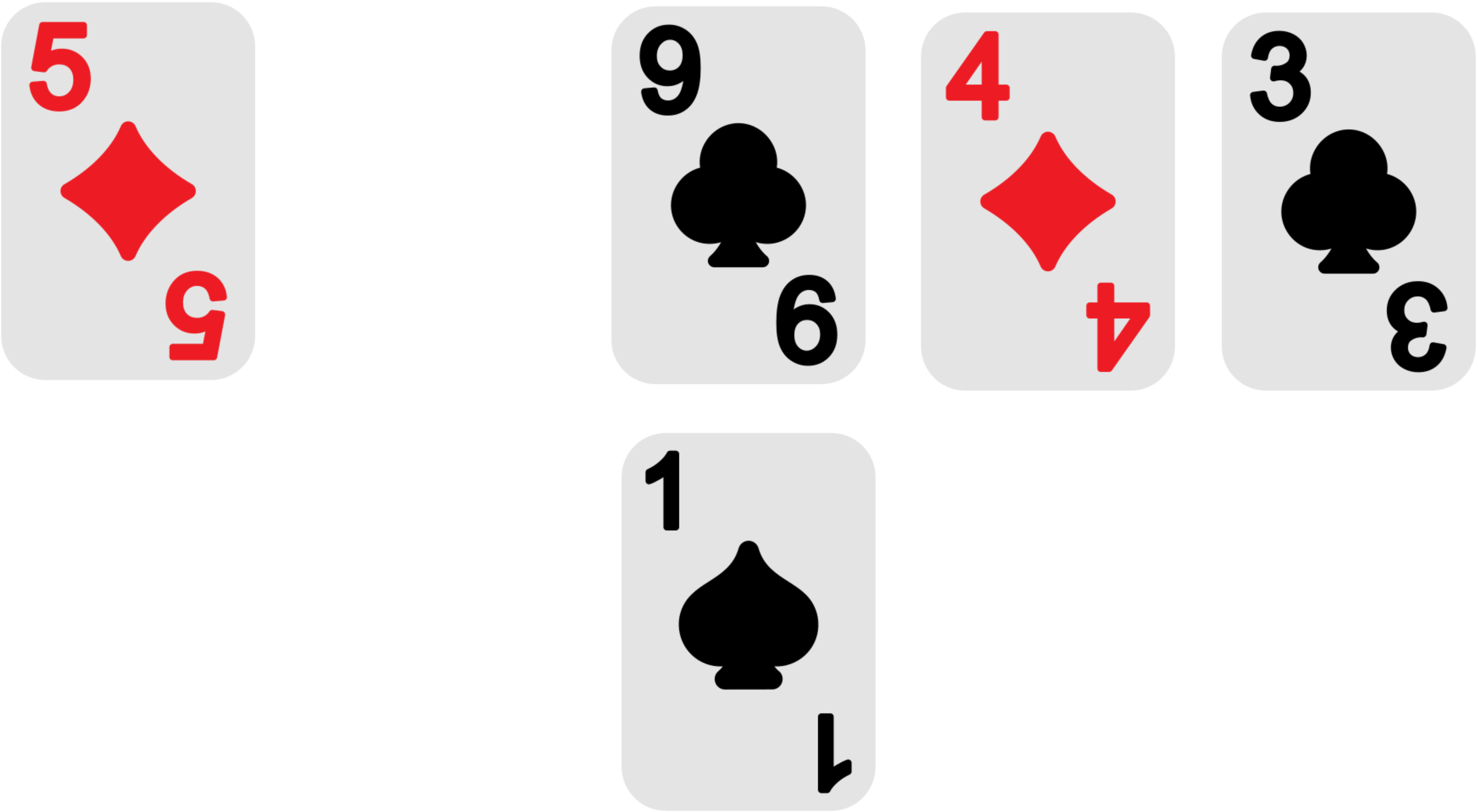
Sorting Playing Cards



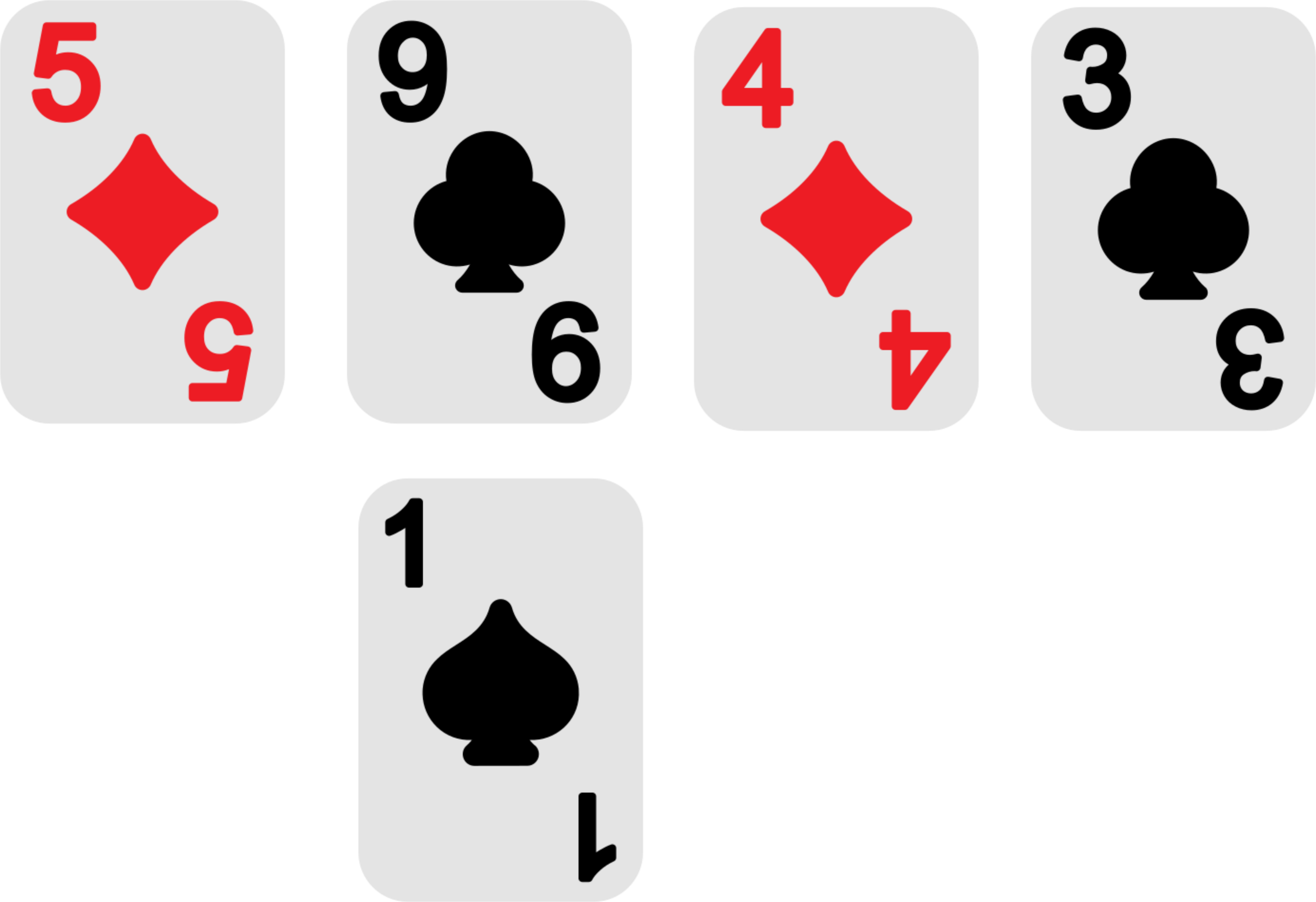
Sorting Playing Cards



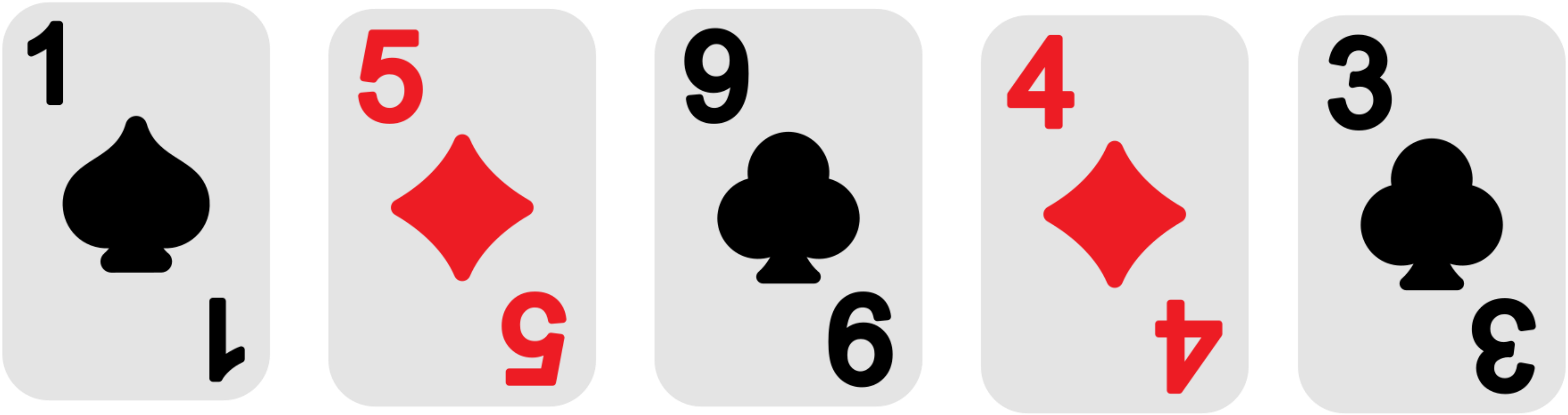
Sorting Playing Cards



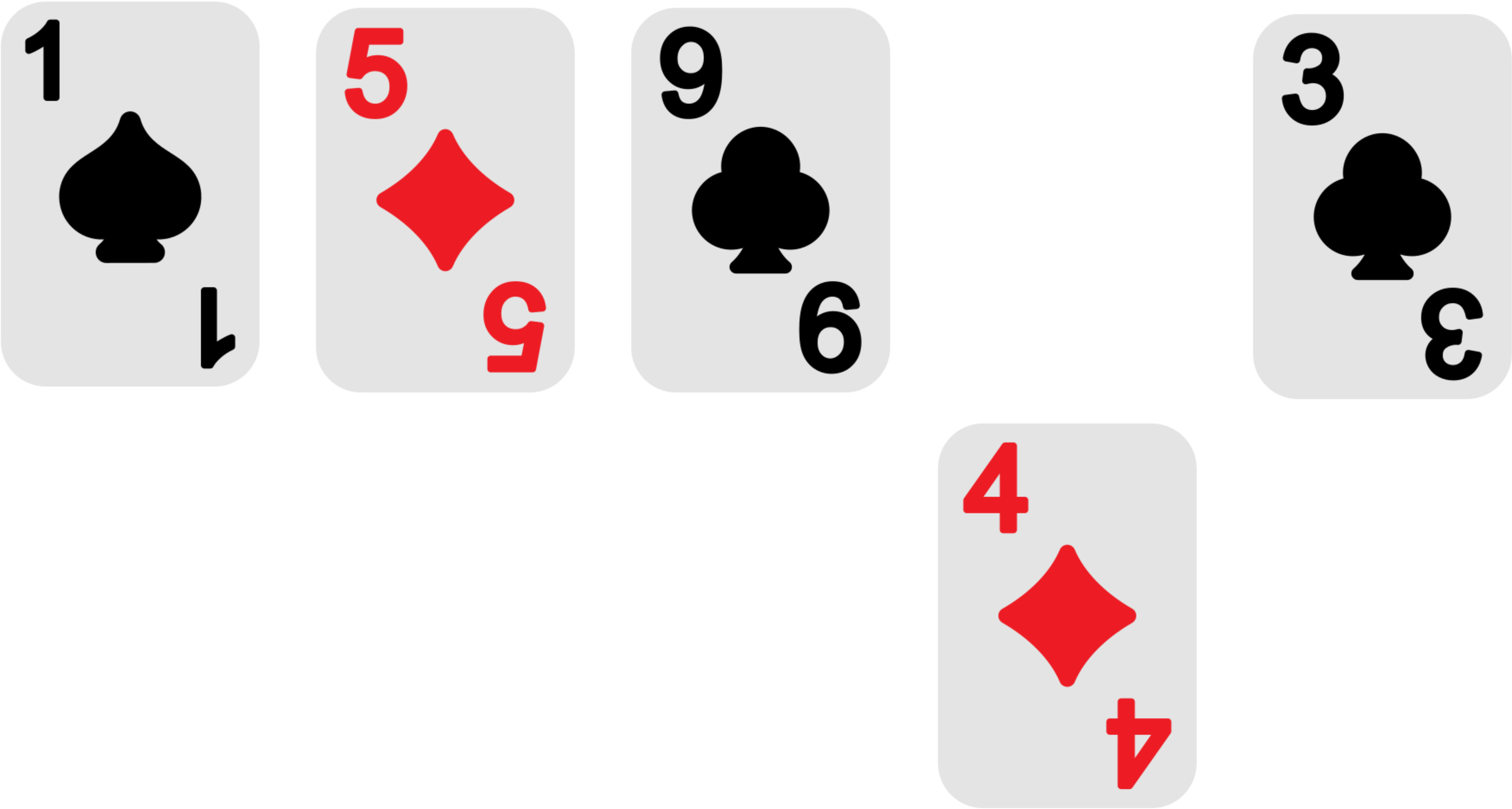
Sorting Playing Cards



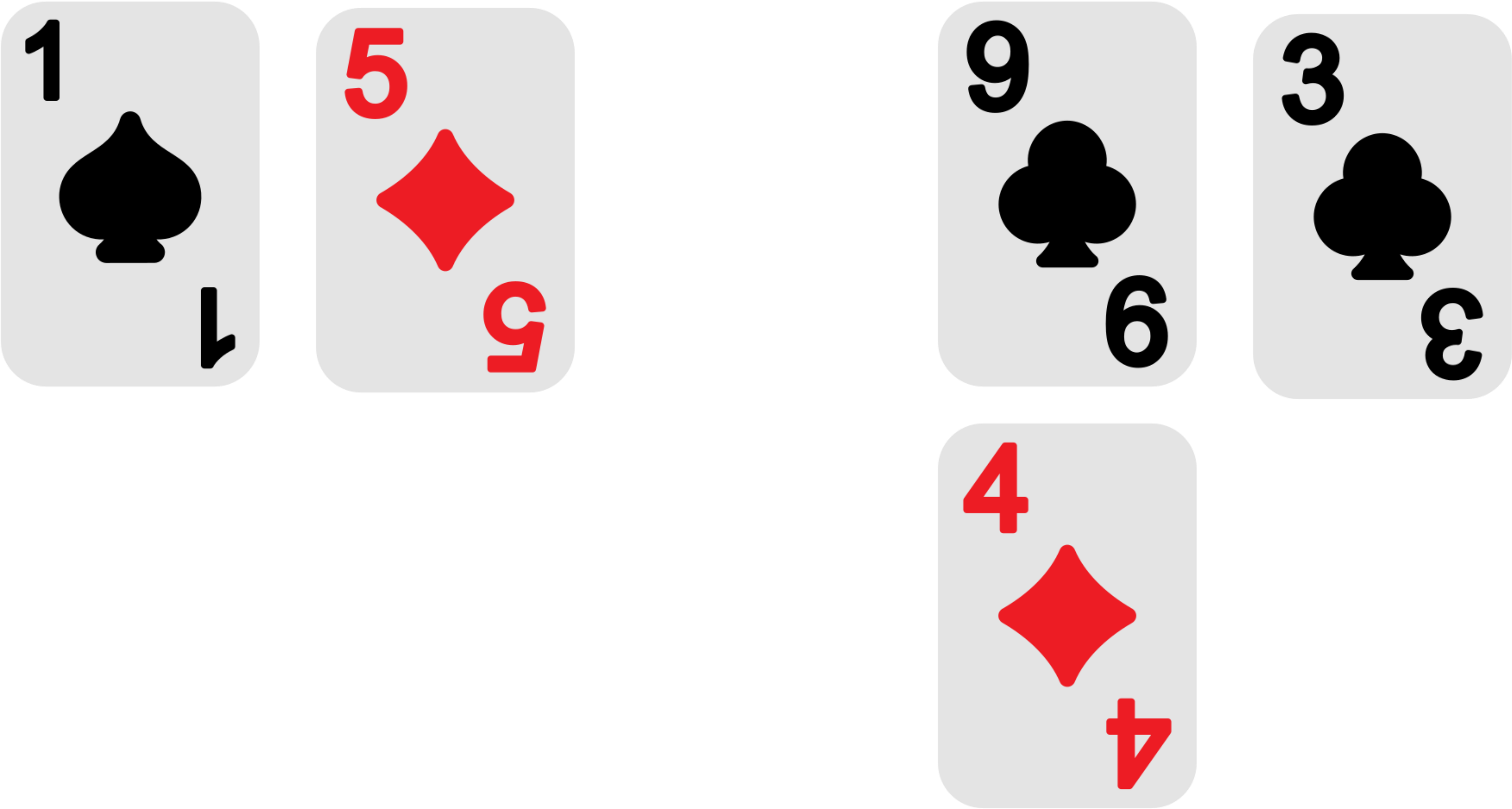
Sorting Playing Cards



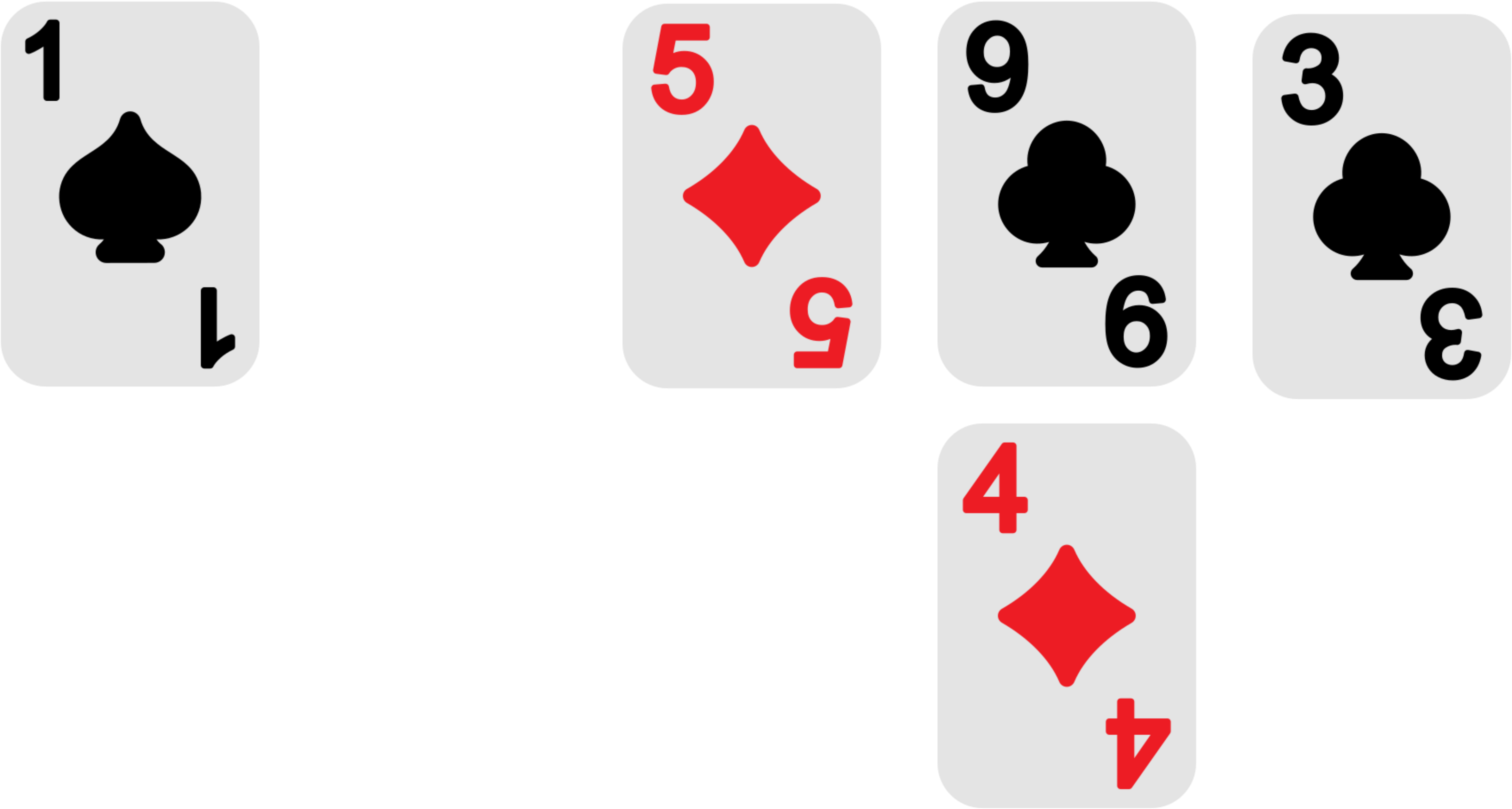
Sorting Playing Cards



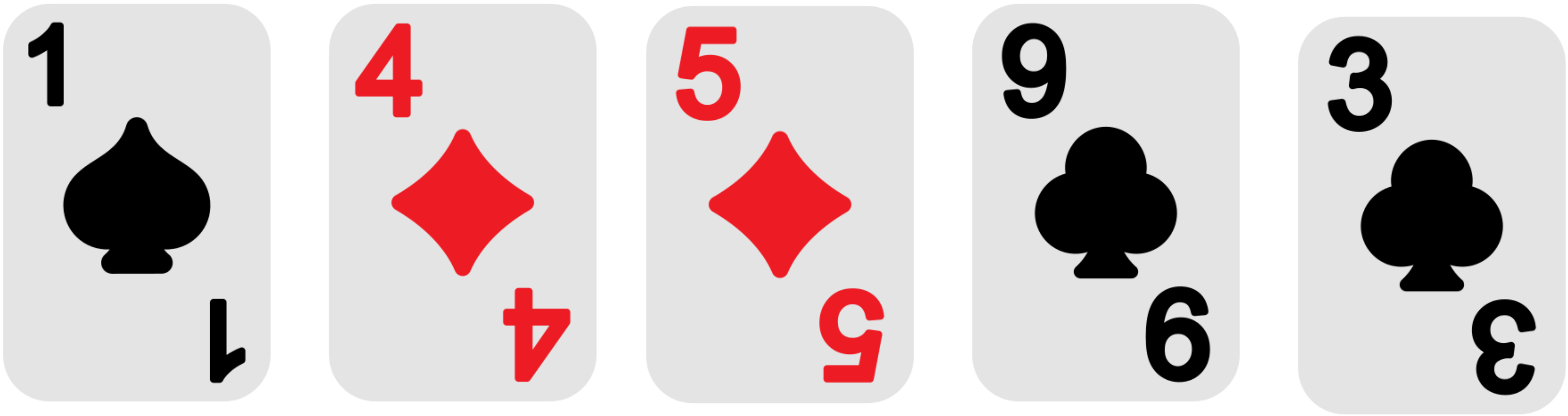
Sorting Playing Cards



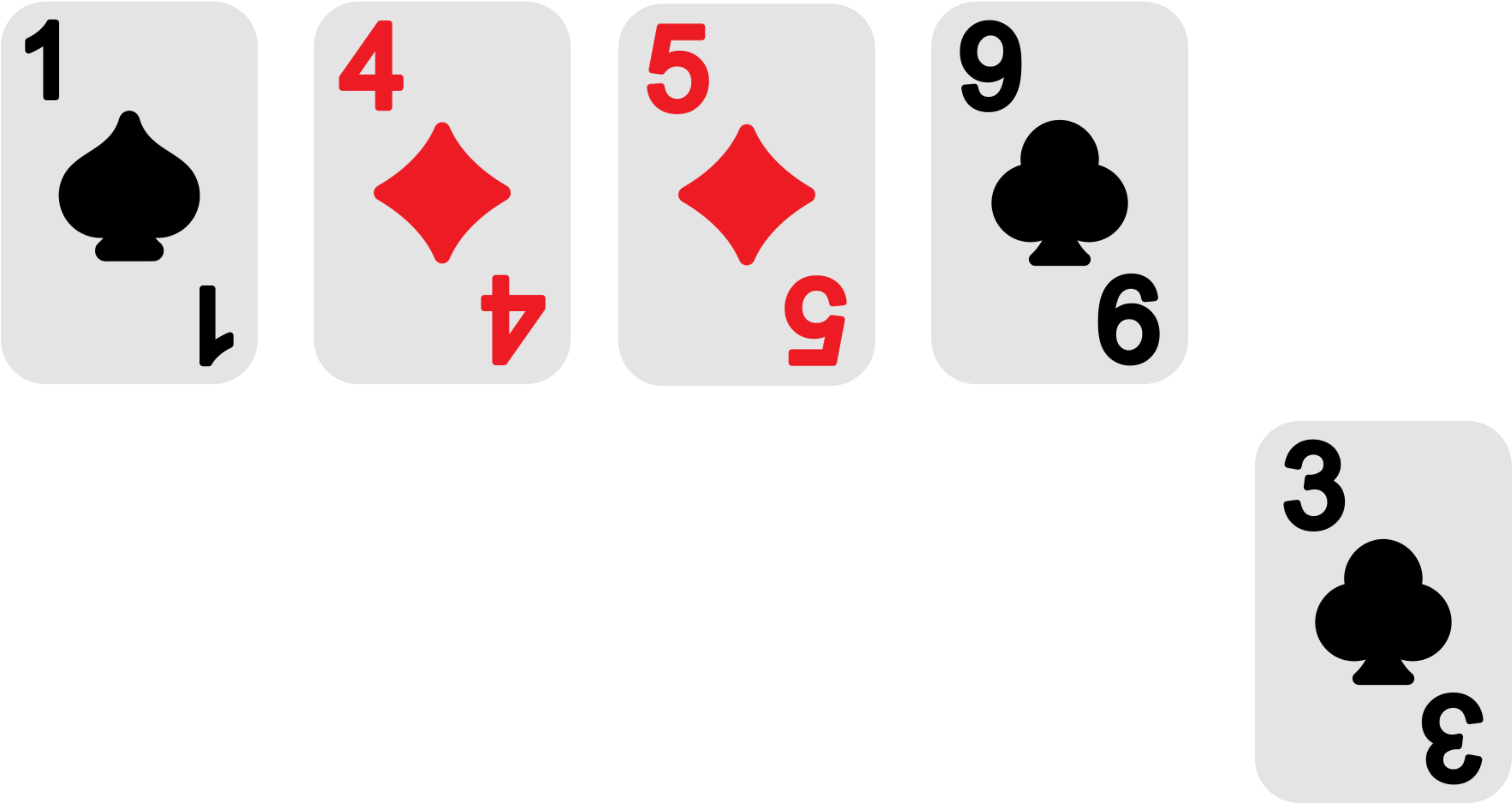
Sorting Playing Cards



Sorting Playing Cards

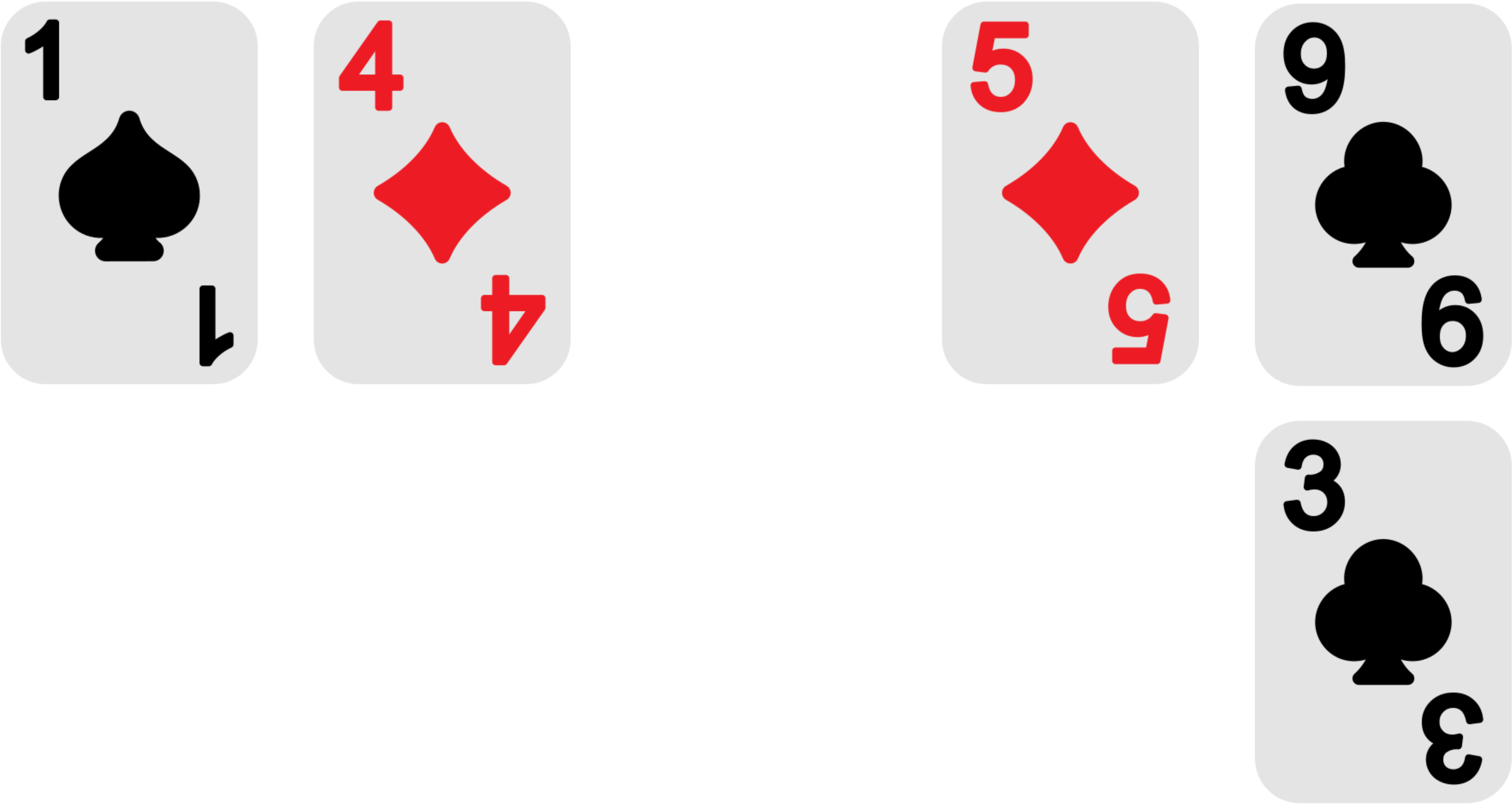


Sorting Playing Cards

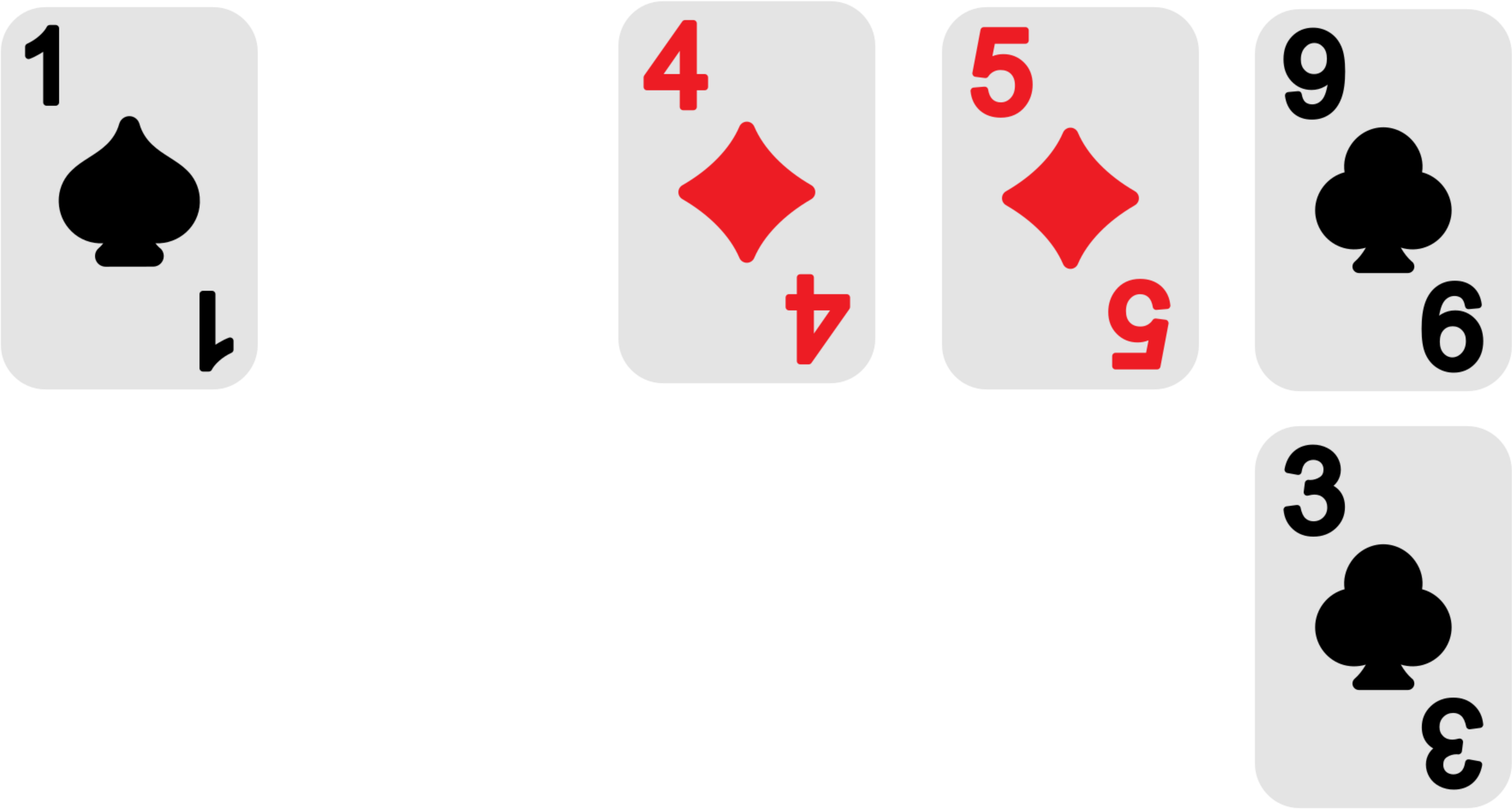


Sorting Playing Cards

Sorting Playing Cards



Sorting Playing Cards



Sorting Playing Cards

Insertion Sort: Pseudocode

ALGORITHM *InsertionSort*($A[0..n-1]$)
//Sorts a given array by insertion sort
//Input: An array $A[0..n-1]$ of n orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 1$ **to** $n-1$ **do**
    key $\leftarrow A[i]$
    j $\leftarrow i - 1$
    while j ≥ 0 **and** A[j] $>$ key **do**
        A[j + 1] $\leftarrow A$[j]
        j $\leftarrow$ j $-$ 1
    A[j + 1] $\leftarrow$ key

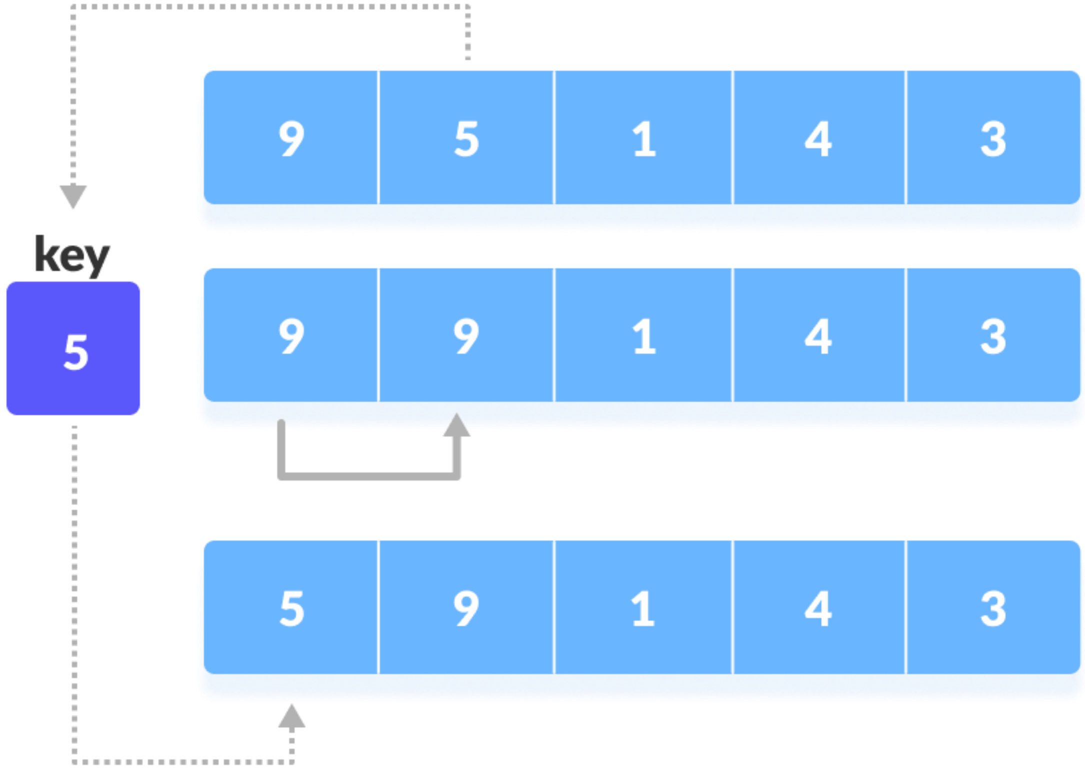
Working of Insertion Sort

- Suppose we need to sort the following array.

9	5	1	4	3

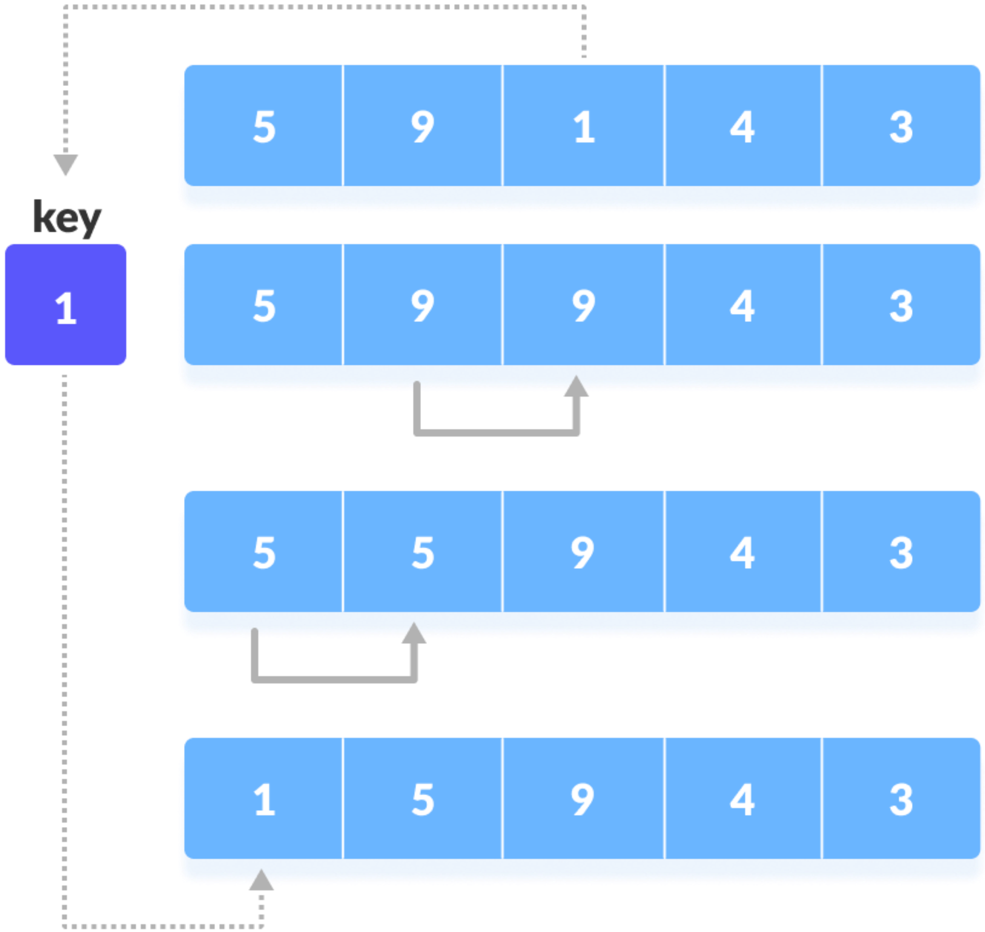
Working of Insertion Sort

- The first element in the array is assumed to be sorted.
- Take the second element and store it separately in key.
- Compare key with the first element.



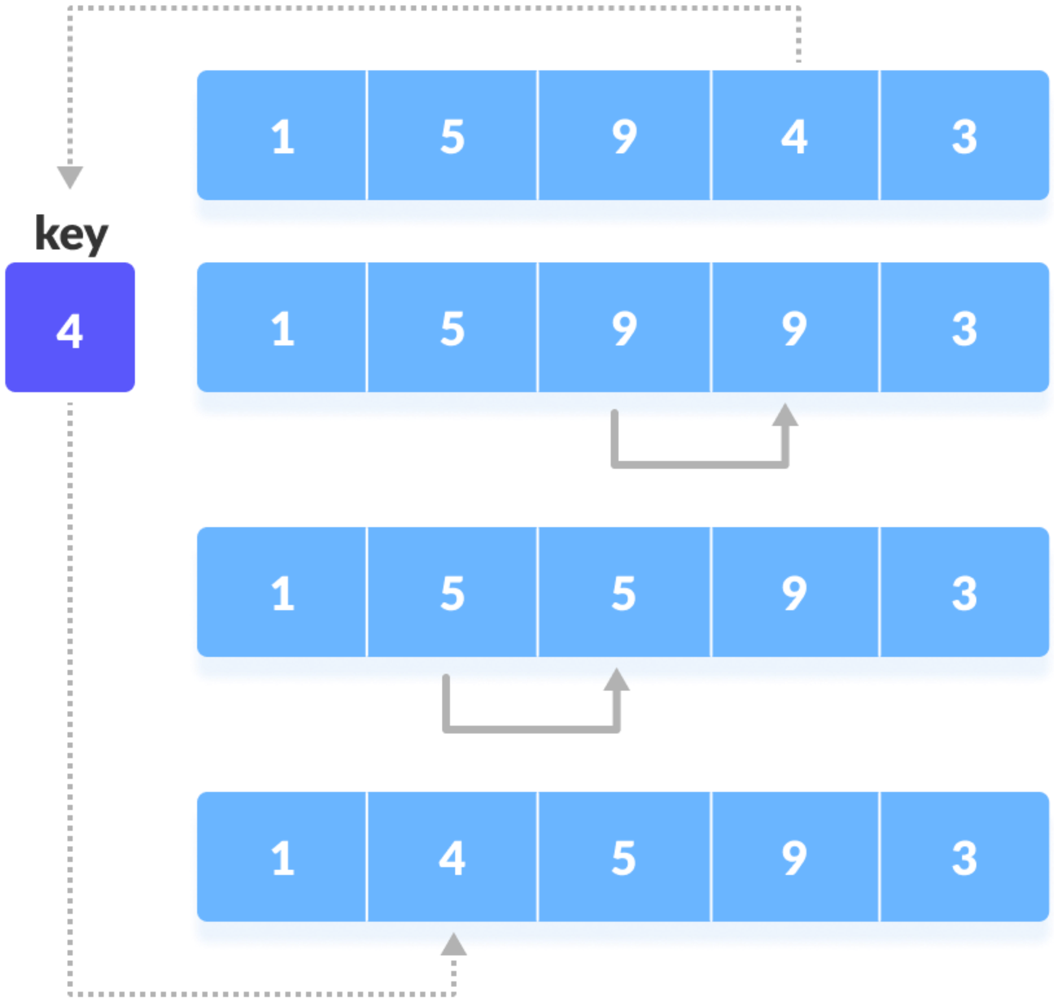
Working of Insertion Sort

- Now, the first two elements are sorted.
- Take the third element and compare it with the elements on the left.



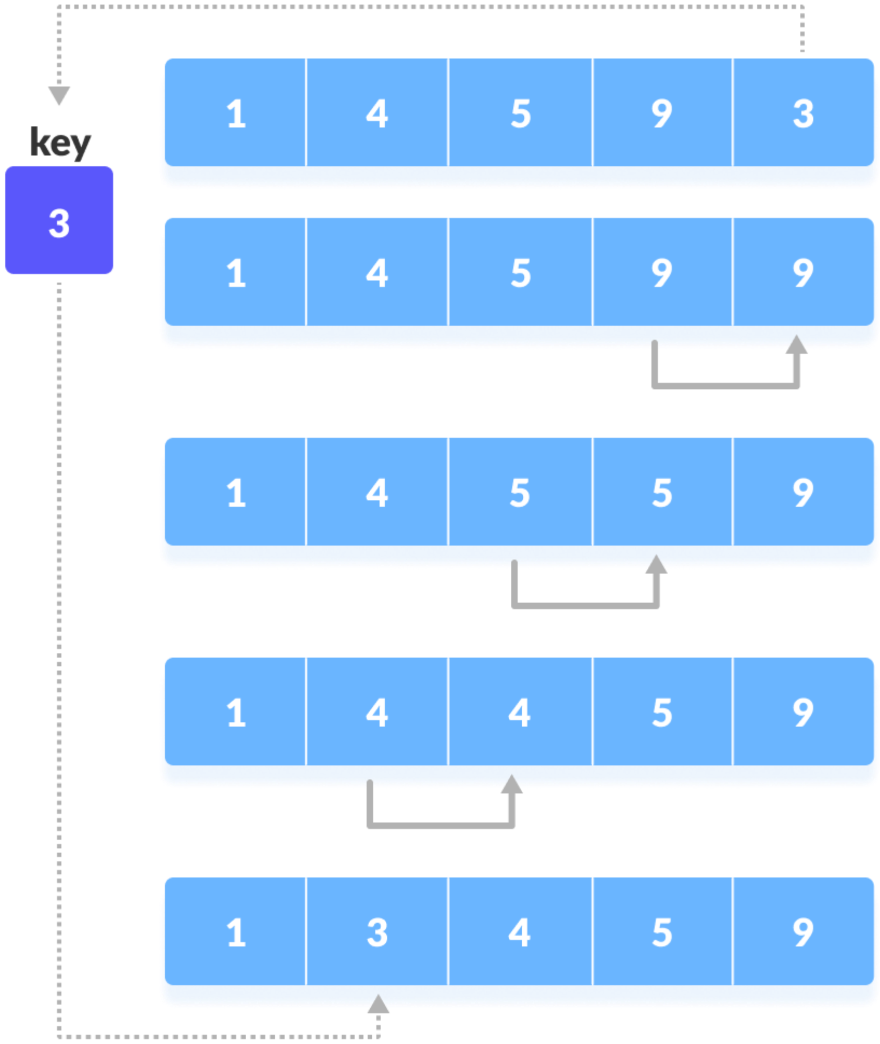
Working of Insertion Sort

- Similarly, place every unsorted element at its correct position.



Working of Insertion Sort

- Similarly, place every unsorted element at its correct position.



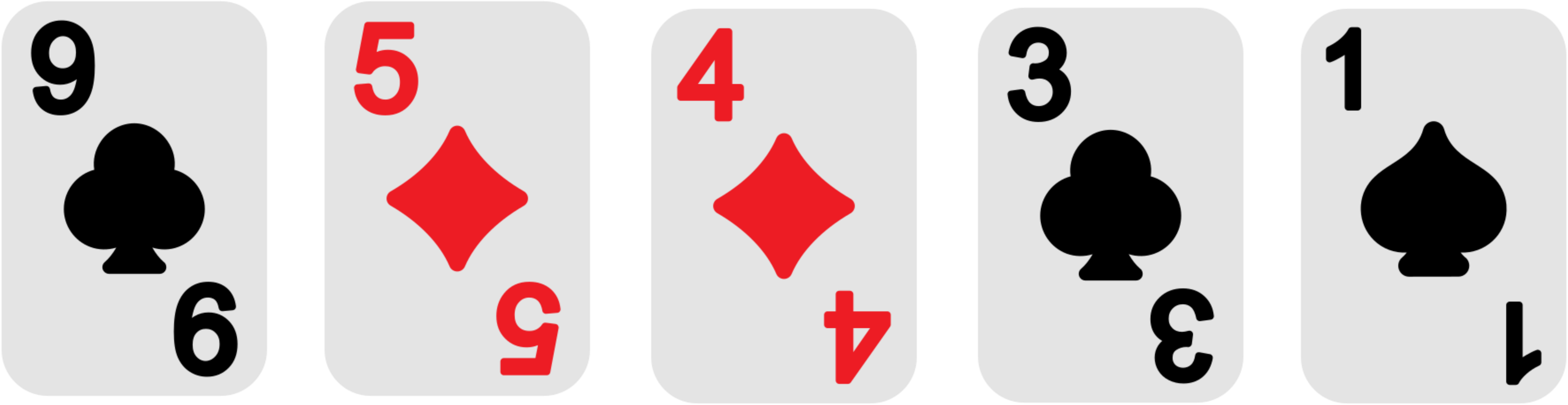
Example Of Sorting With Insertion Sort

Sort 6, 4, 1, 8, 5

6 |	**4**	1	8	5
4	6 |	**1**	8	5
1	4	6 |	**8**	5
1	4	6	8 |	**5**
1	4	5	6	8

Insertion Sort: Best Case

Insertion Sort: Worst Case



Insertion Sort Complexity

Time Complexity	
Best	$O(n)$
Worst	$O(n^2)$
Average	$O(n^2)$
Space Complexity	$O(1)$
Stability	Yes

Sorting Algorithms

- Bubble Sort
- Selection Sort
- **Insertion Sort**
- Merge Sort
- Quicksort
- Counting Sort
- Radix Sort
- Bucket Sort
- Heap Sort
- Shell Sort

Summations

$$\sum_{i=m}^{n} 1 = n - m + 1$$

$$\sum_{i=1}^{n} 1 = n$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{n} i^2 = \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$$

Mystery 1

a. What does this algorithm compute?
b. What is its basic operation?
c. How many times is the basic operation executed?

ALGORITHM *Mystery*(x, y)

//Input: Two numbers x and y

$S \leftarrow x + y$

return S

Mystery 1 – Solution

a. What does this algorithm compute? The sum of x and y
b. What is its basic operation? Addition
c. How many times is the basic operation executed? $T(n) = 1$

ALGORITHM *Add*(x, y)

//Input: Two numbers x and y

//Output: The sum of x and y

$S \leftarrow x + y$

return S

Mystery 2

a. What does this algorithm compute?
b. What is its basic operation?
c. How many times is the basic operation executed?

ALGORITHM *Mystery*(n)

//Input: A nonnegative integer n

$S \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**

$\quad S \leftarrow S + i$

return S

Mystery 2 – Solution

a. What does this algorithm compute? The sum of integers from 1 to n
b. What is its basic operation? Addition
c. How many times is the basic operation executed? $T(n) = n$

ALGORITHM $Sum(n)$

$S \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**

$S \leftarrow S + i$

return S

Mystery 2 – Analysis

- The algorithm makes one addition on each execution of the loop.
- Let us denote $T(n)$ the number of times this addition is executed.
- We try to find a formula expressing it as a function of size n.
- Which is repeated for each value of the loop's variable i within the bounds 1 and n.
- Therefore, we get the following sum for $T(n)$:

$$T(n) = \sum_{i=1}^{n} 1 = n$$

Mystery 3

a. What does this algorithm compute?
b. What is its basic operation?
c. How many times is the basic operation executed?

ALGORITHM *Mystery*(A[0..$n-1$], K)
    //Input: An array A[0..n − 1] and a key K
    $i \leftarrow 0$
    for $i \leftarrow 0$ **to** $n-1$ **do**
        if A[i] = K
        return i
    return −1

Mystery 3 – Solution

a. What does this algorithm compute?

Searches for a given value in a given array by sequential search, and returns the index of the matching element, or −1 if there are no matching elements

b. What is its basic operation? Comparison

c. How many times is the basic operation executed? $T(n) = n$

ALGORITHM *SequentialSearch*(A[0..n − 1], K)

$i \leftarrow 0$

for $i \leftarrow 0$ **to** $n - 1$ **do**

  if A[i] = K

  return i

return −1

Mystery 4

a. What does this algorithm compute?

b. What is its basic operation?

c. How many times is the basic operation executed?

ALGORITHM *Mystery*(A[0..$n-1$])

//Input: An array A[0..$n-1$] of real numbers

$maxval \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n-1$ **do**

  if $A[i] > maxval$

    $maxval \leftarrow A[i]$

return *maxval*

Mystery 4 – Solution

a. What does this algorithm compute?

   Determines the value of the largest element in a given array

b. What is its basic operation? Comparison

c. How many times is the basic operation executed? $T(n) = n - 1$

ALGORITHM *MaxElement*(A[0..$n - 1$])

    maxval ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval

Mystery 4 – Analysis

- Let us denote $C(n)$ the number of times this comparison is executed.
- We try to find a formula expressing it as a function of size n.
- The algorithm makes one comparison on each execution of the loop.
- Which is repeated for each value of the loop's variable i within the bounds 1 and $n-1$.
- Therefore, we get the following sum for $C(n)$:

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n)$$

Mystery 5

a. What does this algorithm compute?

b. What is its basic operation?

c. How many times is the basic operation executed?

ALGORITHM *Mystery*(A[0..$n-1$, 0..$n-1$], B[0..$n-1$, 0..$n-1$])

//Input: Two n × n matrices A and B

for $i \leftarrow 0$ **to** $n-1$ **do**

  for $j \leftarrow 0$ **to** $n-1$ **do**

    C[i, j] $\leftarrow$ A[i, j] + B[i, j]

return C

Mystery 5 – Solution

a. What does this algorithm compute? Adds two square matrices of order n

b. What is its basic operation? Addition

c. How many times is the basic operation executed? $T(n) = n^2$

ALGORITHM *MatrixAddition*(A[0..$n-1$, 0..$n-1$], B[0..$n-1$, 0..$n-1$])

//Input: Two n × n matrices A and B

//Output: Matrix C = A + B

for $i \leftarrow 0$ **to** $n - 1$ **do**

  for $j \leftarrow 0$ **to** $n - 1$ **do**

    $C[i, j] \leftarrow A[i, j] + B[i, j]$

return C

Mystery 6

a. What does this algorithm compute?
b. What is its basic operation?
c. How many times is the basic operation executed?

ALGORITHM *Mystery*(A[0..$n-1$])
//Input: An array A[0..n − 1]
for $i \leftarrow 0$ **to** $n - 2$ **do**
  for $j \leftarrow i + 1$ **to** $n - 1$ **do**
    if A[i] = A[j]
      return false
return true

Mystery 6 – Solution

a. What does this algorithm compute?

   Determines whether all the elements in a given array are distinct

b. What is its basic operation? Comparison

c. How many times is the basic operation executed? $T(n) = n(n - 1)/2$

ALGORITHM *UniqueElements*(A[0..$n - 1$])

    for $i \leftarrow 0$ **to** $n - 2$ **do**

        for $j \leftarrow i + 1$ **to** $n - 1$ **do**

            if $A[i] = A[j]$

                return false

    return true

Mystery 6 – Analysis

- Only one comparison is made for each repetition of the innermost loop.
- For each value of the loop variable j between its limits $i + 1$ and $n - 1$.
- This is repeated for each value of the outer loop.
- For each value of the loop variable i between its limits 0 and $n - 2$.

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2).$$

Mystery 7

a. What does this algorithm compute?

b. What is its basic operation?

c. How many times is the basic operation executed?

```
ALGORITHM Mystery(A[0..n − 1, 0..n − 1], B[0..n − 1, 0..n − 1])
    //Input: Two n × n matrices A and B
    for i ← 0 to n − 1 do
        for j ← 0 to n − 1 do
            C[i, j] ← 0
            for k ← 0 to n − 1 do
                C[i, j] ← C[i, j] + A[i, k] ∗ B[k, j]
    return C
```

Mystery 7 – Solution

a. What does this algorithm compute? Multiplies two square matrices of order n

b. What is its basic operation? Multiplication

c. How many times is the basic operation executed? $T(n) = n^3$

ALGORITHM *MatrixMultiplication*(A[0..$n-1$, 0..$n-1$], B[0..$n-1$, 0..$n-1$])

```
    //Input: Two n × n matrices A and B
    for i ← 0 to n − 1 do
        for j ← 0 to n − 1 do
            C[i, j] ← 0
            for k ← 0 to n − 1 do
                C[i, j] ← C[i, j] + A[i, k] ∗ B[k, j]
    return C
```

Mystery 7 – Analysis

- Obviously, there is just one multiplication executed on each repetition of the algorithm's innermost loop.
- The total number of multiplications $M(n)$ is expressed by the following triple sum:

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1$$

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$