Week 4: Monday

EECS 281

MAXIM ALEKSA

maximal.io

Roadmap

Mon 5/22	Quicksort	Section: Sorting algorithms	Project 2 due
Tue 5/23	Mergesort		
Wed 5/24	Midterm Exam review	Section: Midterm Exam review	
Thu 5/25	Midterm Exam		
Fri 5/26	Lab 4 and Lab 5 due		
Tue 5/30	Intro to Hashing		
Wed 5/31	Hashing and Collision Resolution	Section: Hashing	
Thu 6/1	Tree ADT, Searching in Trees		

Agenda

Disjoint sets

Queue with two stacks

FibonacciIterator

Searching

Sorting

Bubble sort

Insertion sort

Selection sort

Merge sort

Heapsort

Quicksort

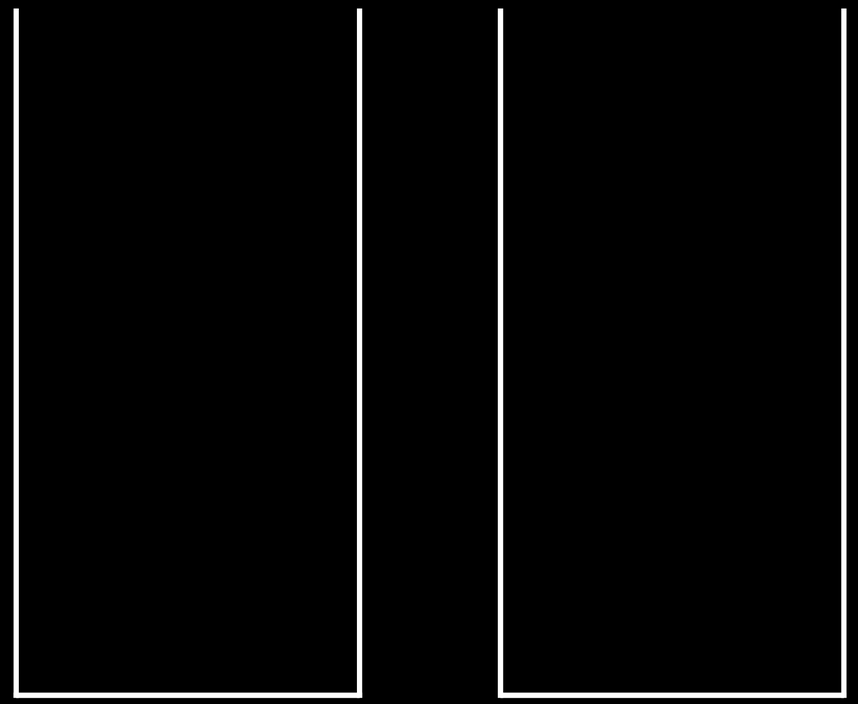
260

slides

Queue with two stacks

Queue with two stacks

Queue with two stacks



Fibonacci Iterator

$$F_n = F_n - 1 + F_{n-1}$$

Disjoint sets

No items are more than in one set

Universe of items: all items that can be a member of a set

Operations:
Union merges two sets into one
Find finds which set an item is in

Disjoint sets

Disjoint sets

Disjoint sets

Microsoft

Skype

Tumblr

Yahoo!

Nokia

Verizon

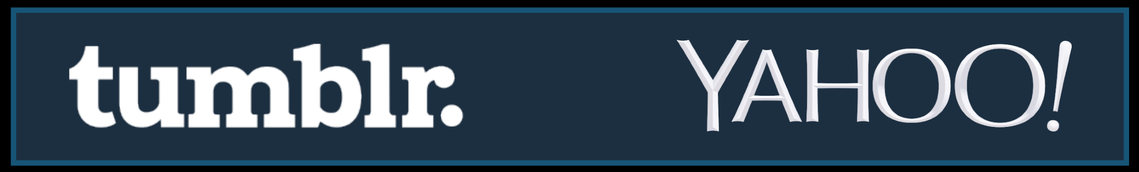
LinkedIn


```
find(Skype) → Skype
find(Tumblr) → Tumblr
find(LinkedIn) → LinkedIn
```

Disjoint sets

Microsoft

Yahoo!



Nokia

Verizon

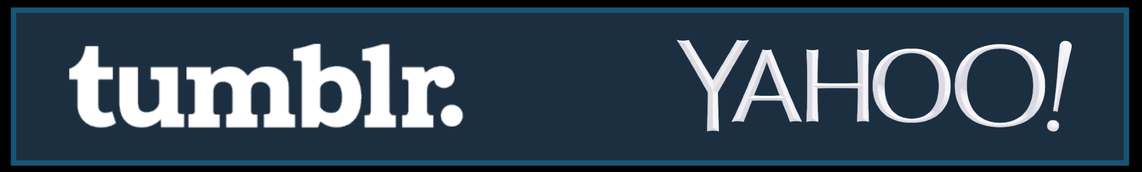
LinkedIn


```
find(Skype) → Microsoft
find(Tumblr) → Yahoo!
find(LinkedIn) → LinkedIn
```

Disjoint sets

Microsoft

Yahoo!



LinkedIn

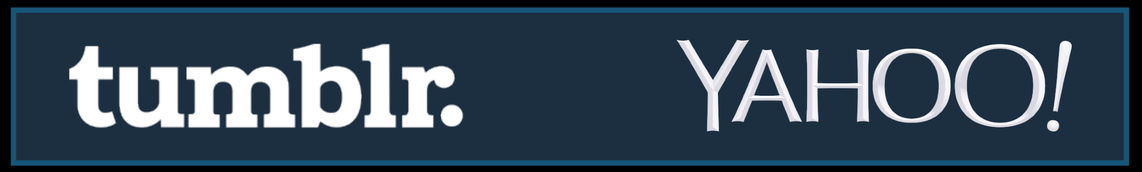
Verizon


```
find(Skype) → Microsoft
find(Nokia) → Microsoft
find(LinkedIn) → LinkedIn
```

Disjoint sets

Microsoft

Yahoo!



Verizon


```
find(Skype) → Microsoft
find(Nokia) → Microsoft
find(LinkedIn) → Microsoft
```

Disjoint sets

Microsoft

Verizon


```
find(Tumblr) → Verizon
find(Yahoo!) → Verizon
find(Verizon) → Verizon
```

Quick Find

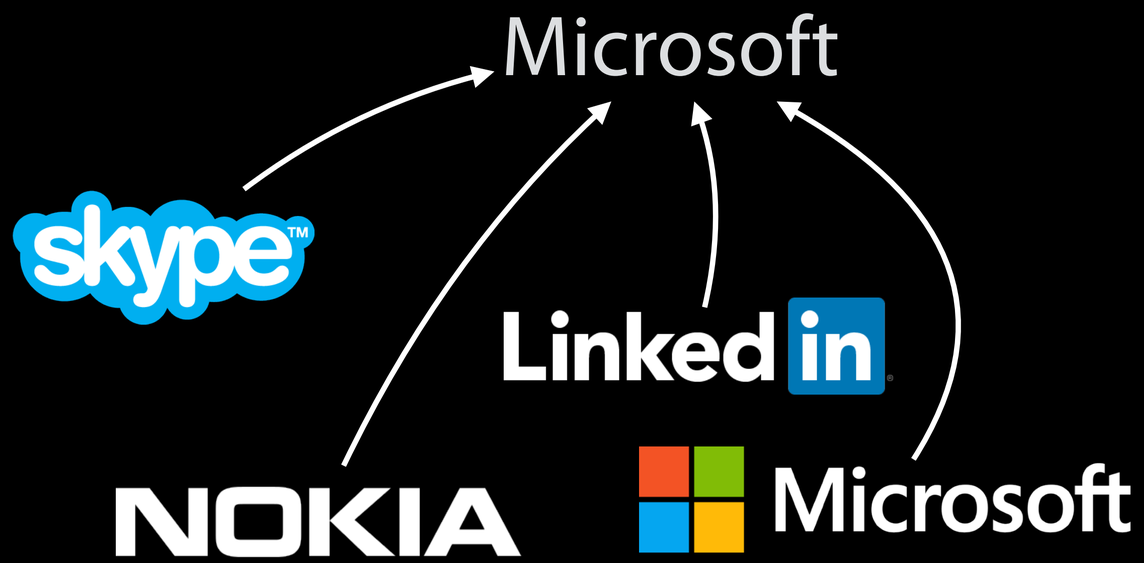
List-based disjoint sets

Each set references list of items in that set

Each item references the set that contains it

Find: Θ(1) time

Union: slow



Quick Find

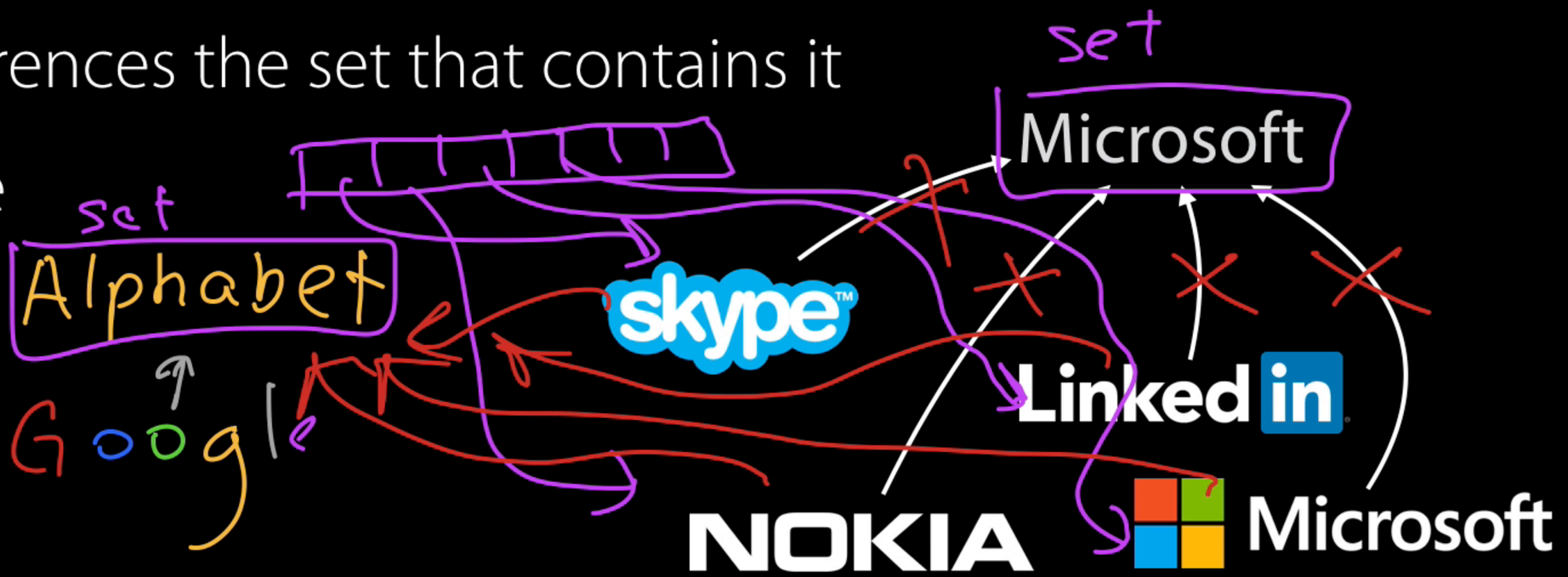
List-based disjoint sets

Each set references list of items in that set

Each item references the set that contains it

✓ Find: Θ(1) time

Union: slow



Quick Union

Union: Θ(1) time

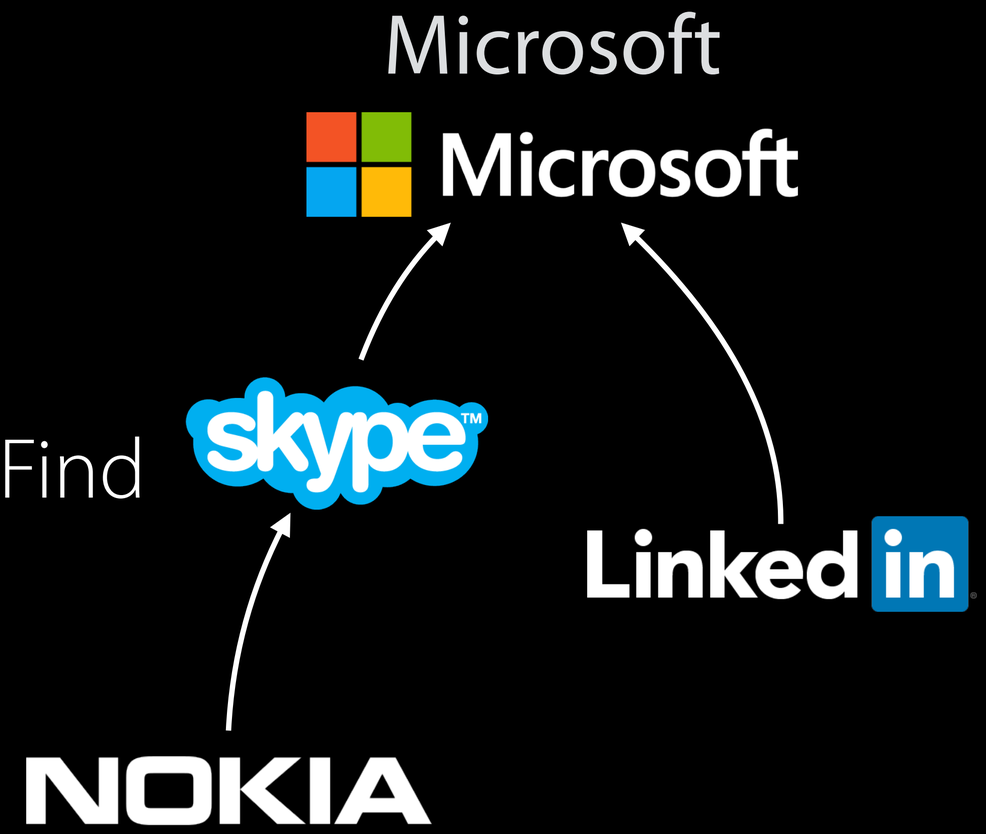
Find: slower

Quick Union is faster overall than Quick Find

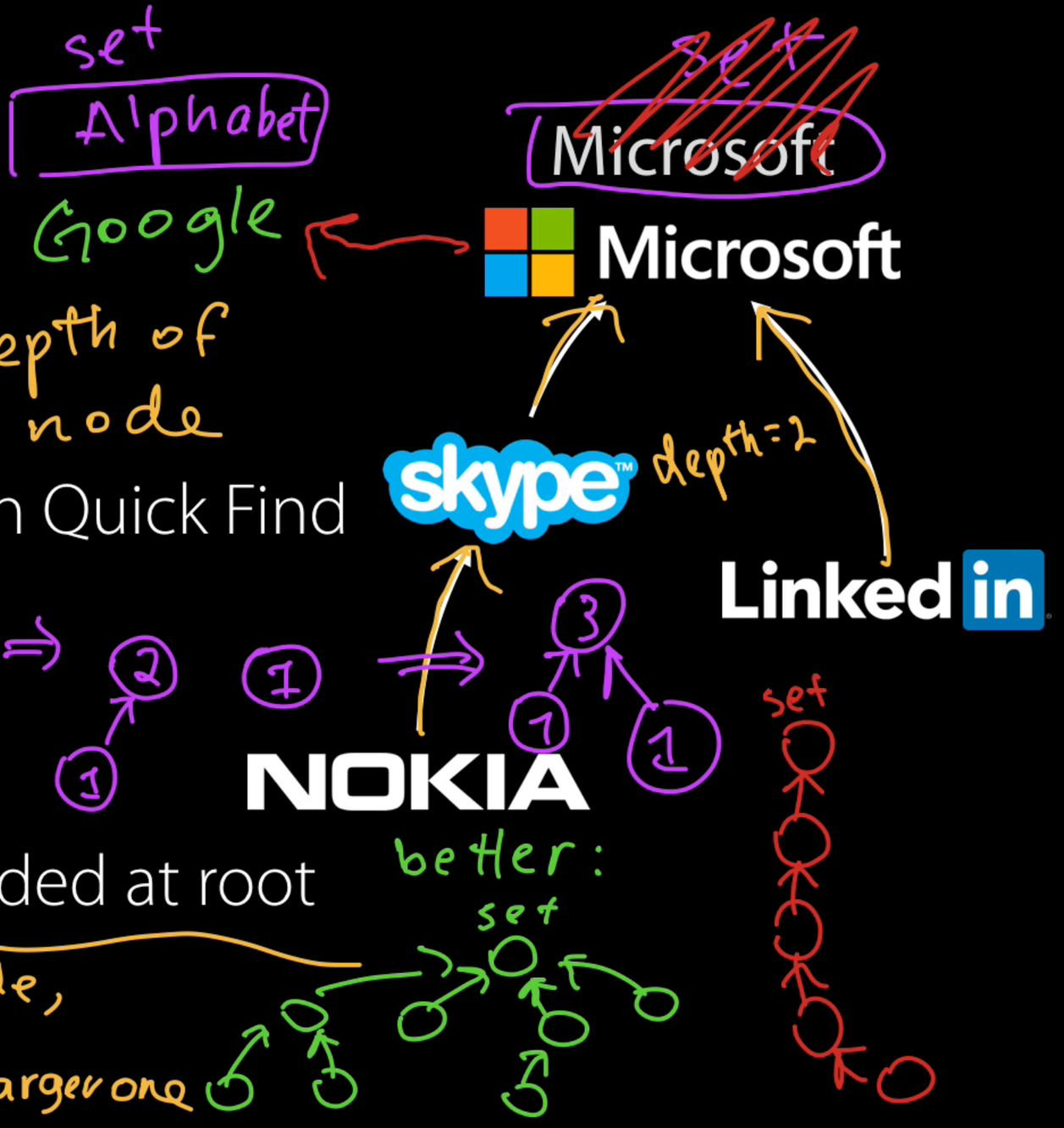
Sets are stored as trees

Only parent references

True identity of each set is recorded at root



Quick Union



✓ Union: Θ(1) time

Find: slower $O(d)$ d = depth of a node

Quick Union is faster overall than Quick Find

Sets are stored as trees

Only parent references

True identity of each set is recorded at root

UNION BY SIZE At each node, record size of tree

Make smaller tree subtree of larger one

Quick Union

Union: Θ(1) time

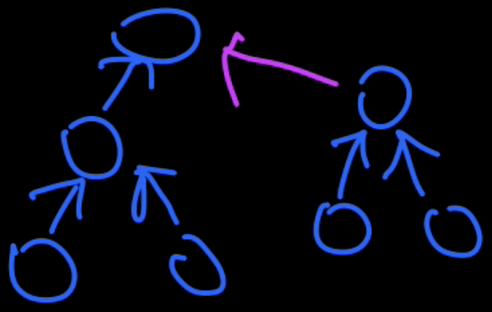
Find: slower

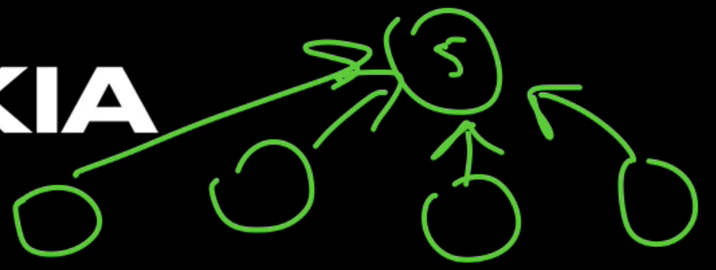
Quick Union is faster overall than Quick Find

Sets are stored as trees

Only parent references

True identity of each set is recorded at root





Microsoft

Microsoft

skype

LinkedIn

NOKIA

Union by size

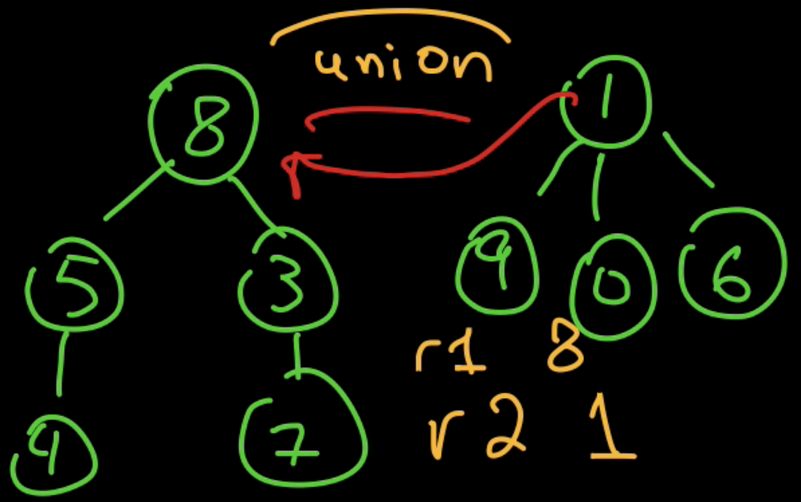
approx. by decr

size != depth

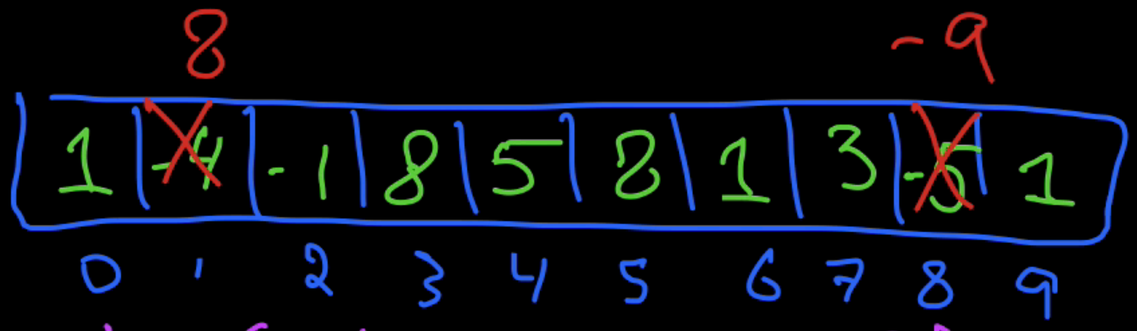
want small

Disjoint Sets as an Array

- Items numbered from zero
- Array to record parent of each item
- If no parent → record size as neg. number



(2)



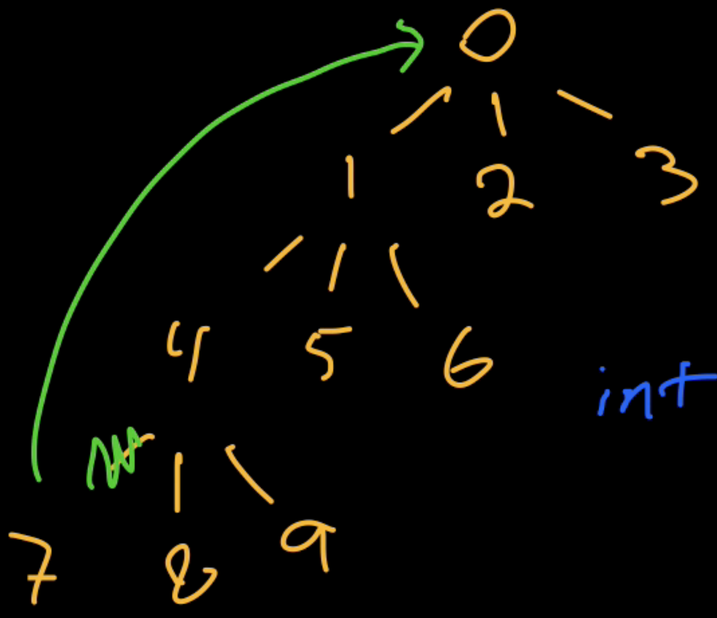

```
void union(int r1, int r2){
    if (a[r2] < a[r1]){
        a[r2] += a[r1];
        a[r1] = r2;
    } else {
        ...
}
```

R2 has larger tree

Optimizing

① Union by size

② Path compression



find(7)
find(4)


```
int find(int x) {
  if (a[x] < 0) {
    return x;
  } else {
    a[x] = find(a[x]);
    return a[x];
  }
}
```

0 ← 4

Sequence of Union & Find

$\Theta(u + f \cdot \alpha(f+u, u))$

extr slow = growing inverse Ackermann fn ← never > 4

avg case find $\Theta(1)$

```
break;
```

Sorting

Sorting

A sort is a permutation of a sequence of elements that brings them into order according to some total order.

A total order $\preccurlyeq$ is…

total	$x \preccurlyeq y$ or $y \preccurlyeq x$ for all x, y.
reflexive	$x \preccurlyeq x$.
antisymmetric	$x \preccurlyeq y$ and $y \preccurlyeq x$ iff $x = y$.
transitive	$x \preccurlyeq y$ and $y \preccurlyeq z$ implies $x \preccurlyeq z$.

Alternative view point

An **inversion** is a pair of elements that are out of order.

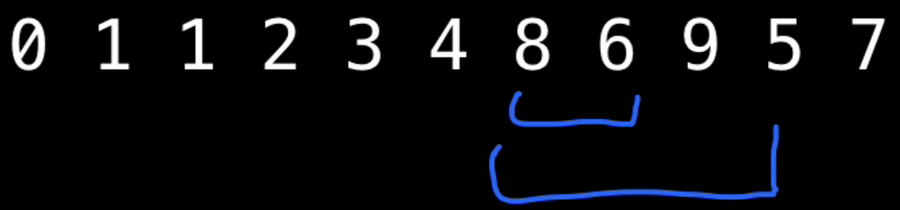
```
0 1 1 2 3 4 8 6 9 5 7
```

6/55 inversions: `(8,6)` `(8,5)` `(8,7)` `(6,5)` `(9,5)` `(9,7)`.

Goal of sorting: reduce the number of inversions to 0.

Alternative view point

An **inversion** is a pair of elements that are out of order.



6/55 inversions: `(8,6)` `(8,5)` `(8,7)` `(6,5)` `(9,5)` `(9,7)`.

Goal of sorting: reduce the number of inversions to 0.

Why sort?

Problems become easier

Searching

Finding median

Data compression

Computer graphics

Sorting Algorithms

Bubble sort

Selection sort

Insertion sort

Heapsort

Merge sort

Quicksort

Sorting Algorithms

Bubble sort

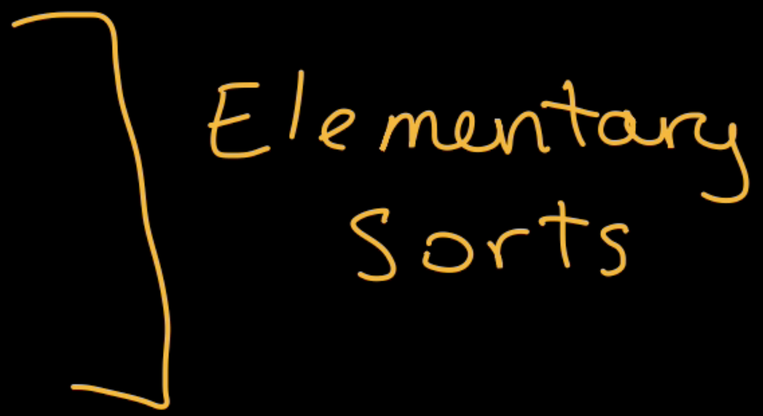
Selection sort

Insertion sort

Heapsort

Merge sort

Quicksort



Bubble sort

Reduce the number of inversions by going through the array and swapping adjacent elements if they are an inversion pair.

Bubble large elements up and bubble small elements down.

```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 … n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```

Bubble sort

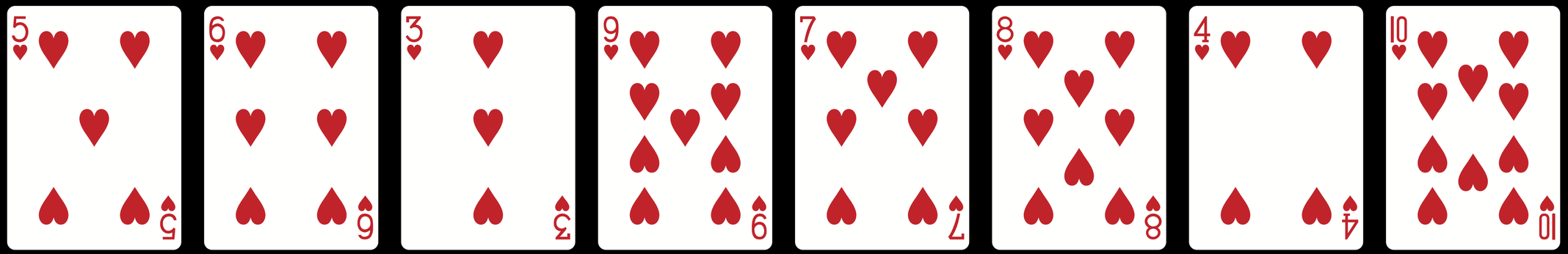
Reduce the number of inversions by going through the array and swapping adjacent elements if they are an inversion pair.

Bubble large elements up and bubble small elements down.

```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```

Bubble sort

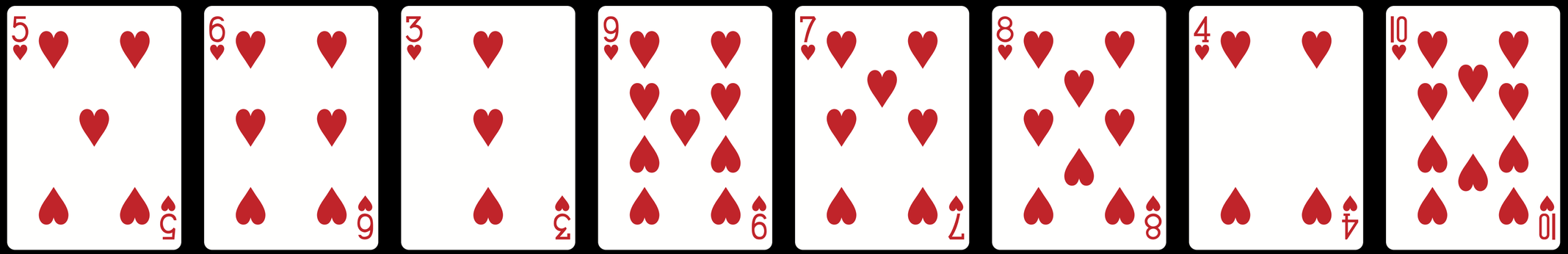
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

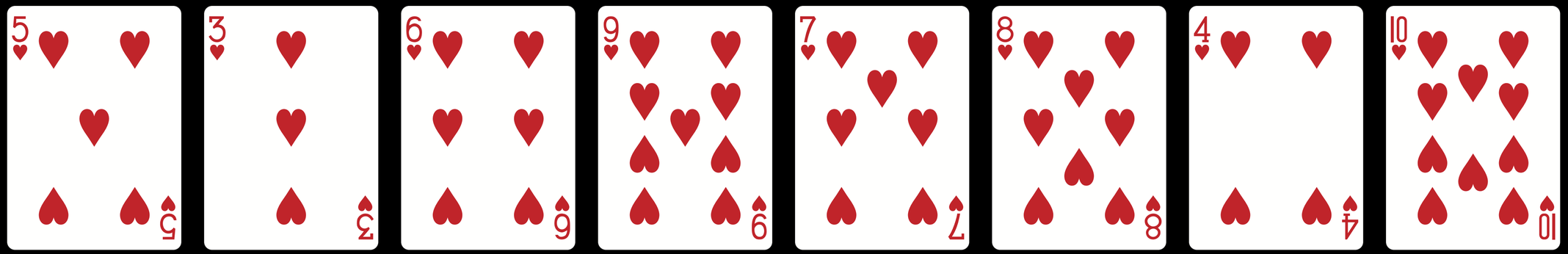
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

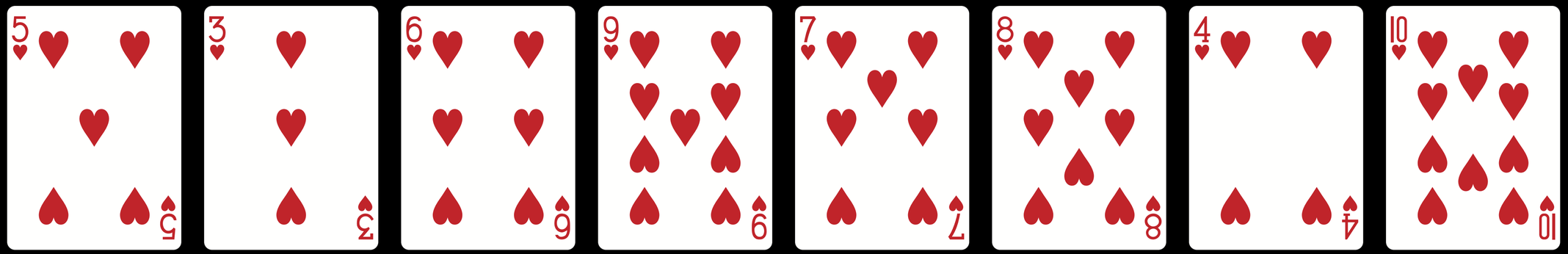
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

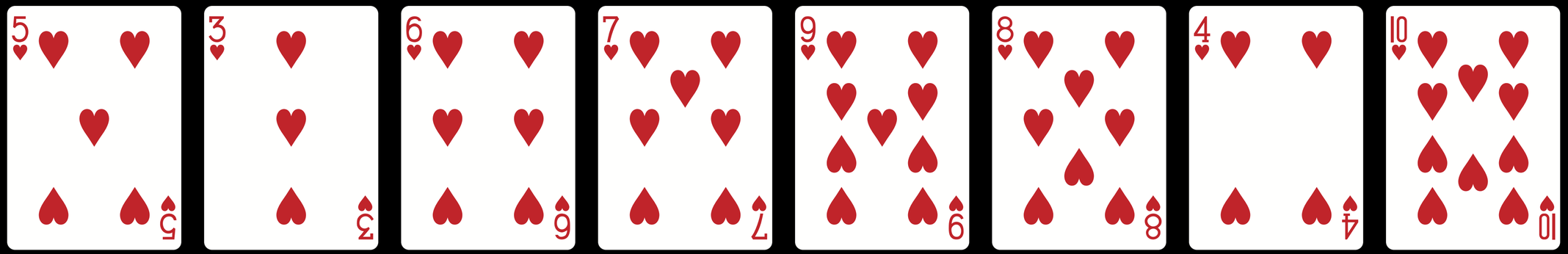
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

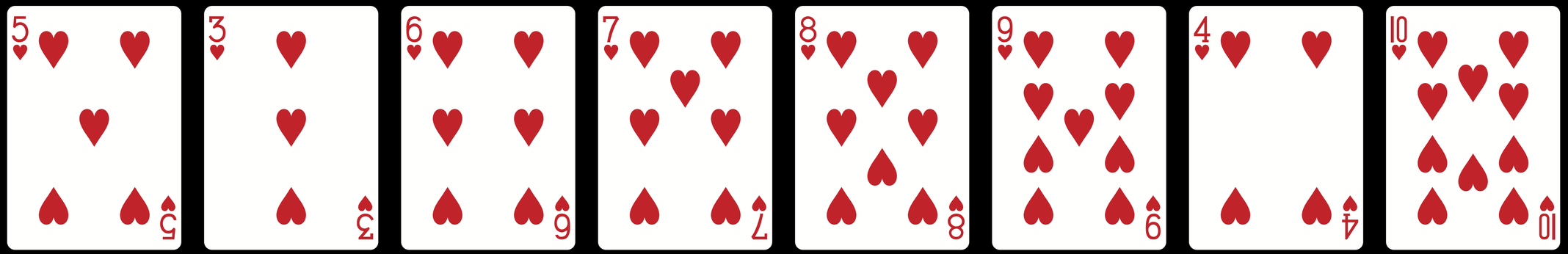
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

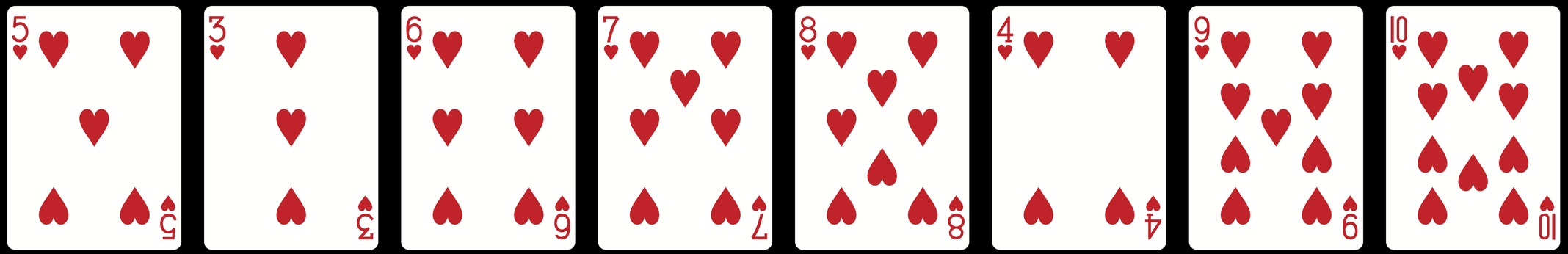
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

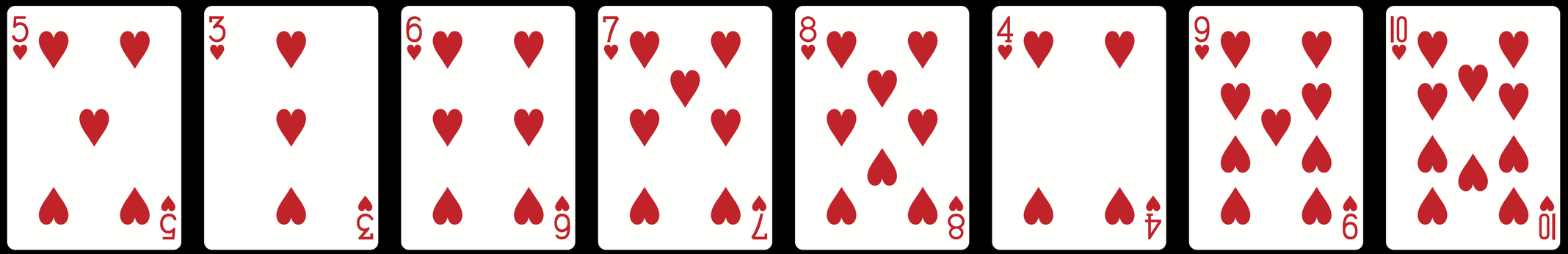
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

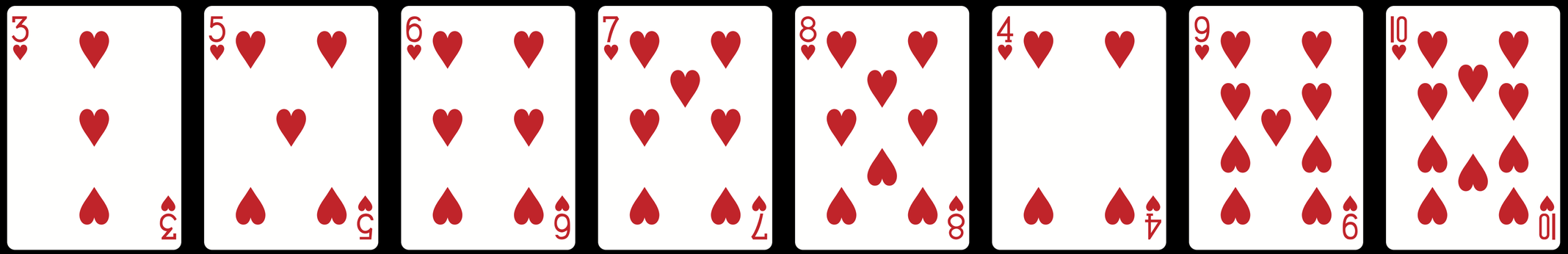
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

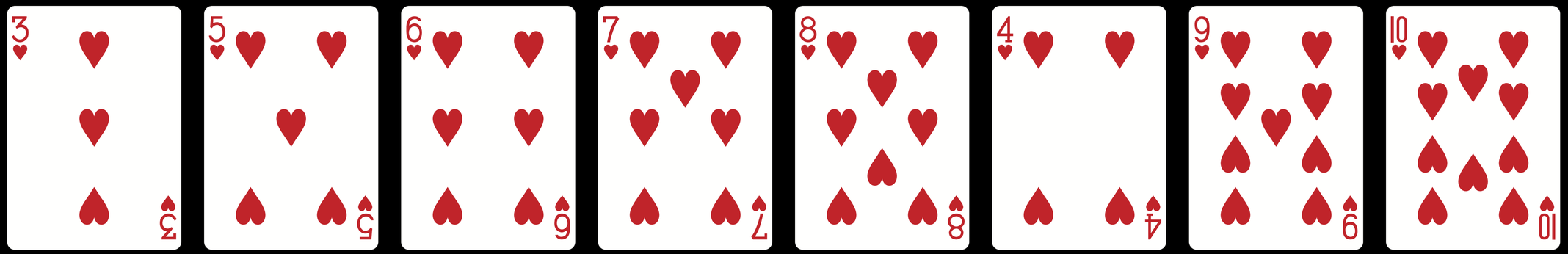
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

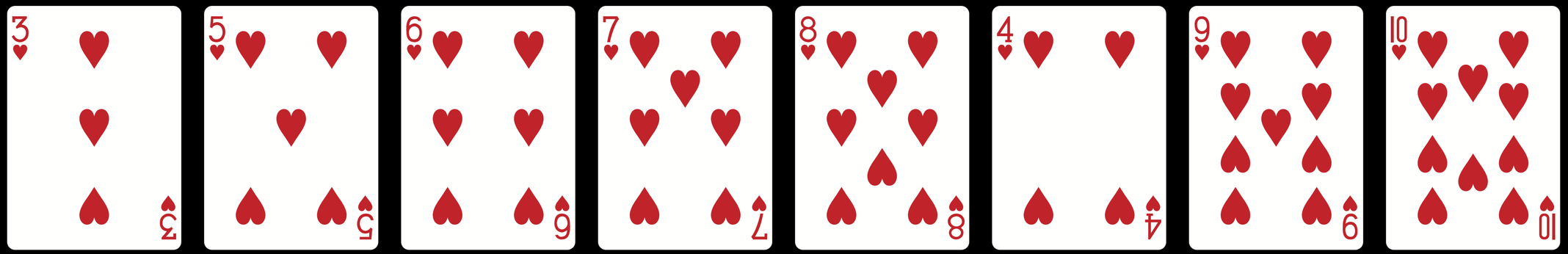
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

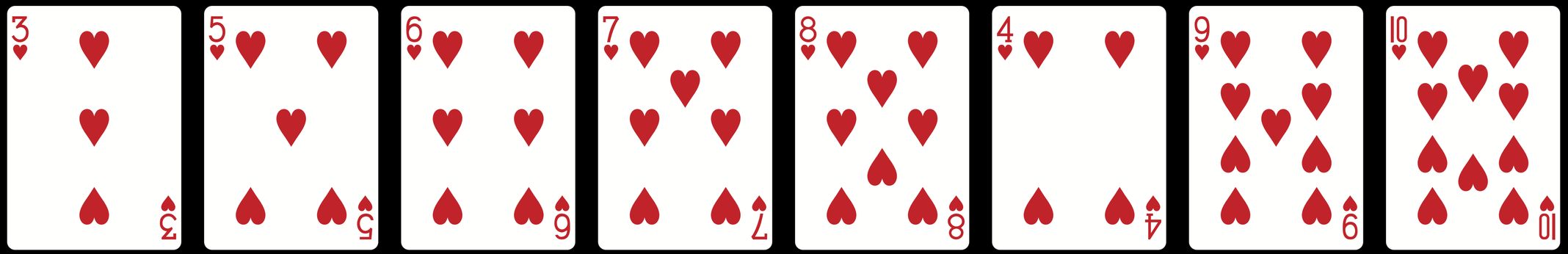
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

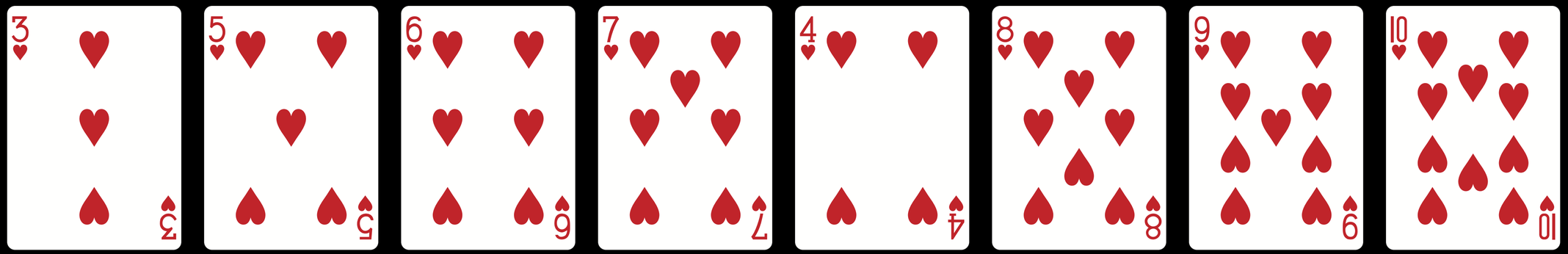
Bubble sort

```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



Bubble sort

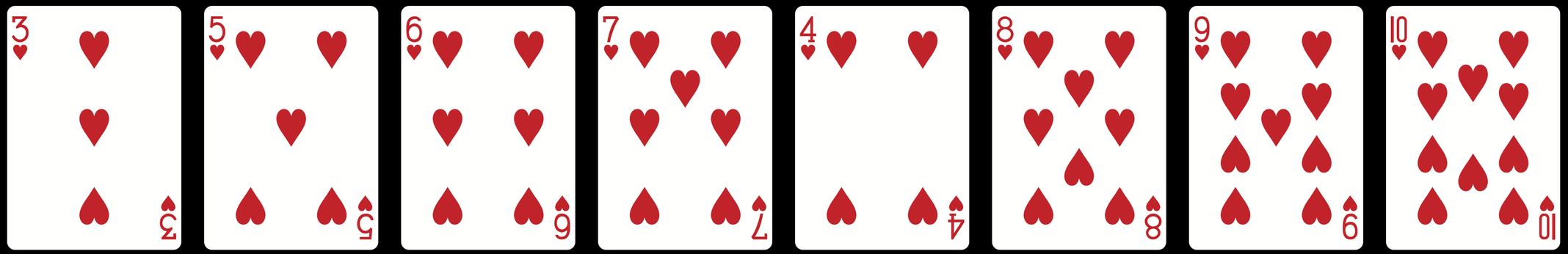
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

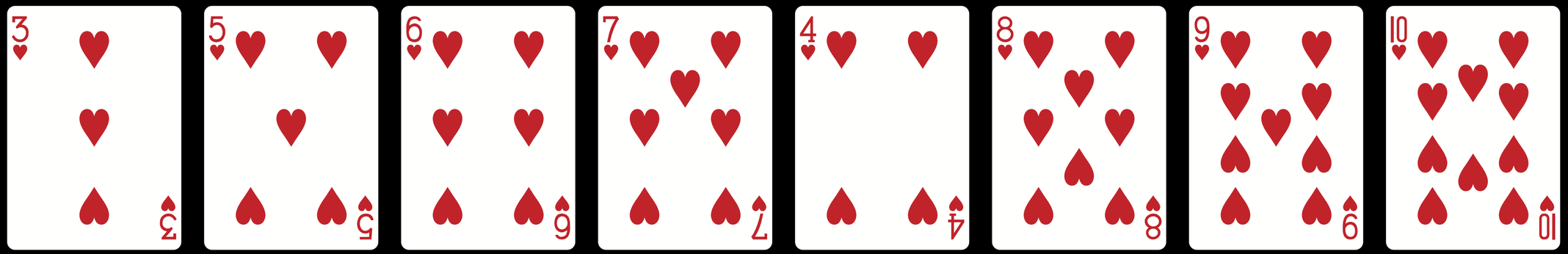
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

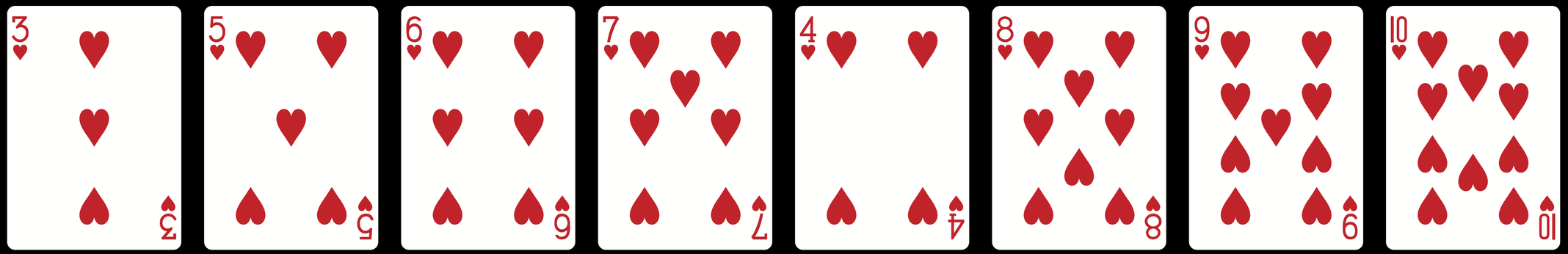
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

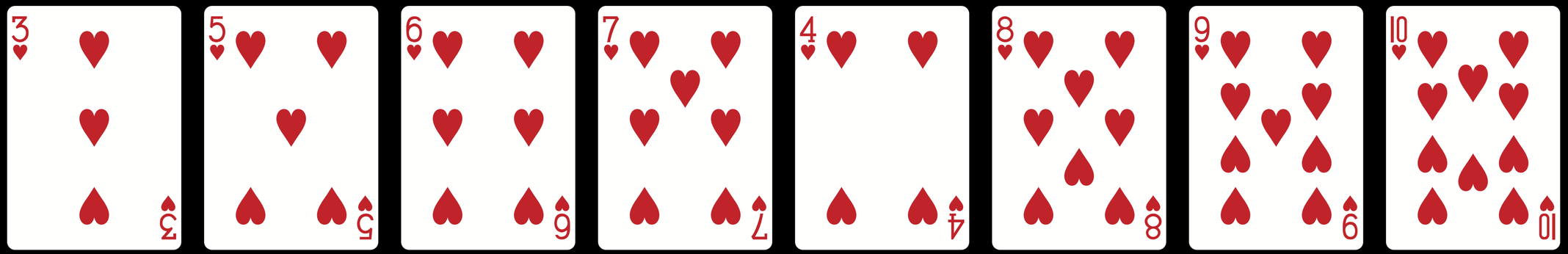
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

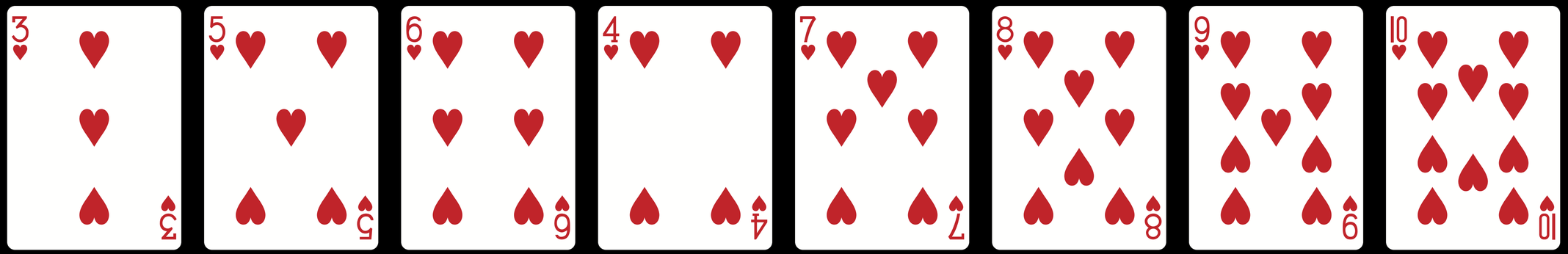
Bubble sort

```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



Bubble sort

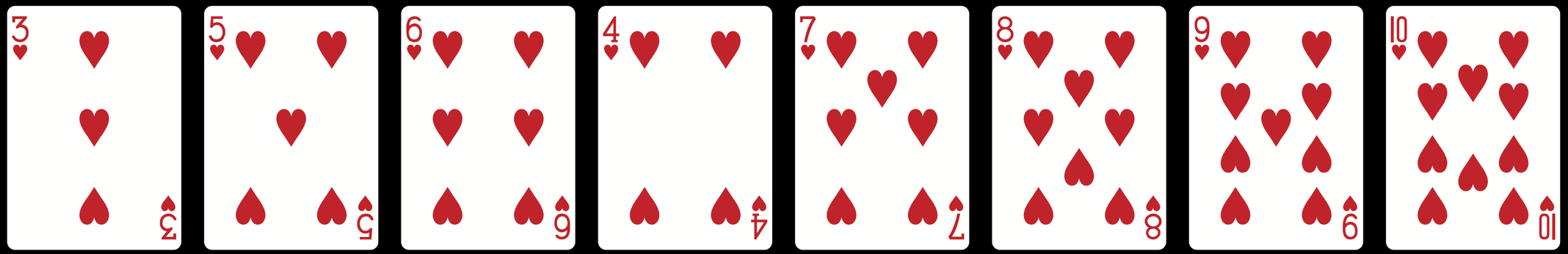
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

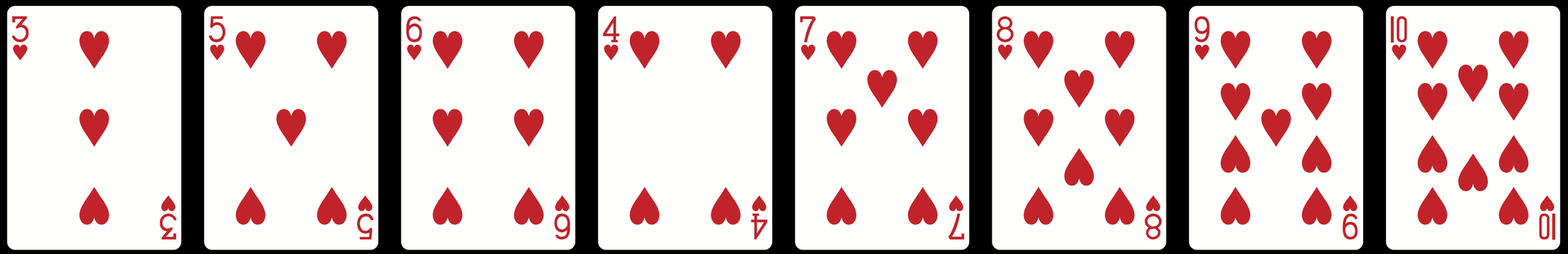
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

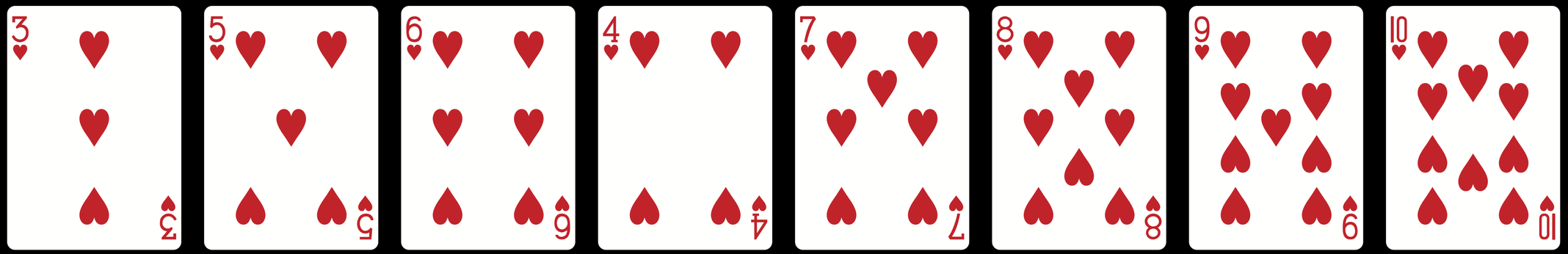
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

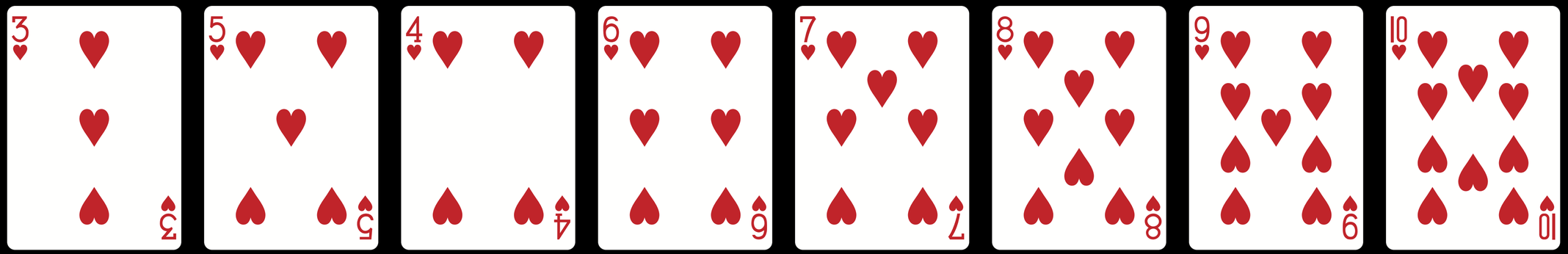
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

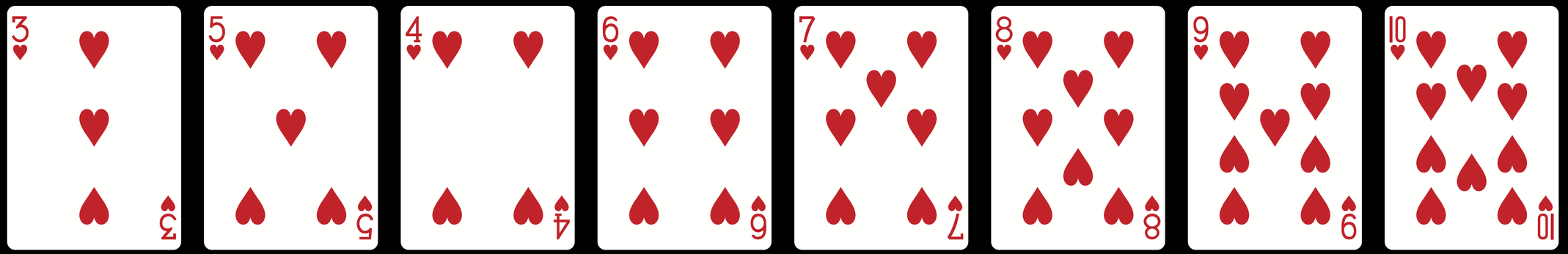
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

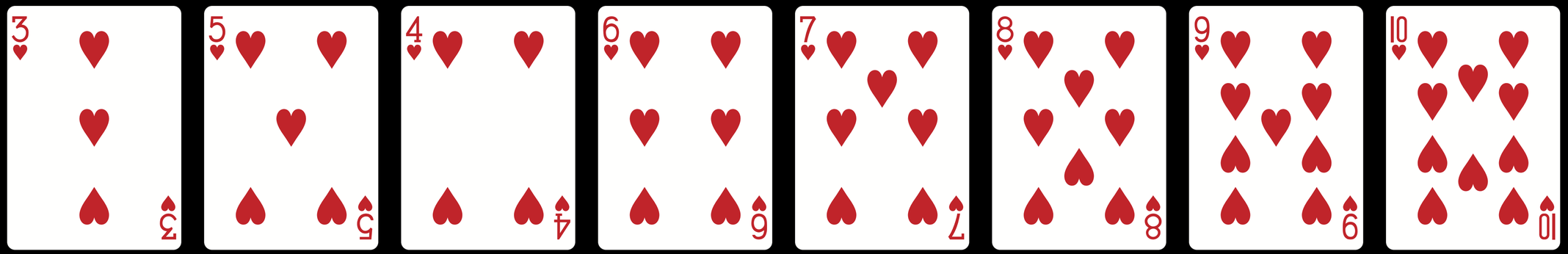
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

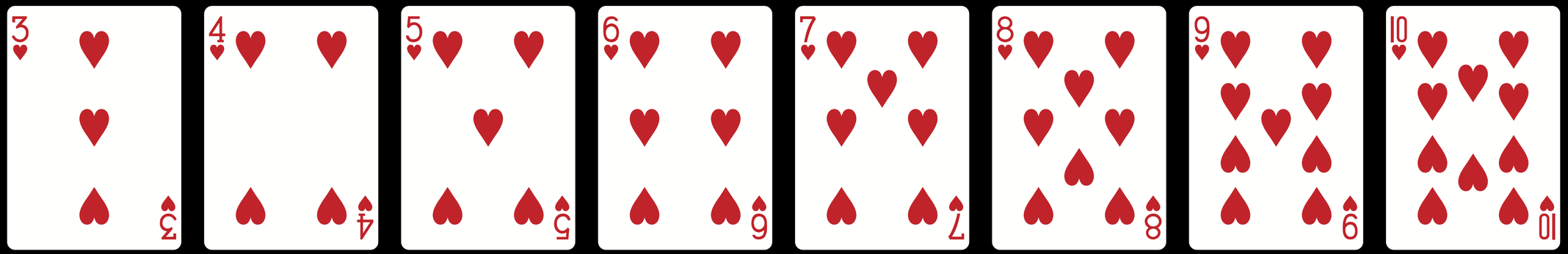
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

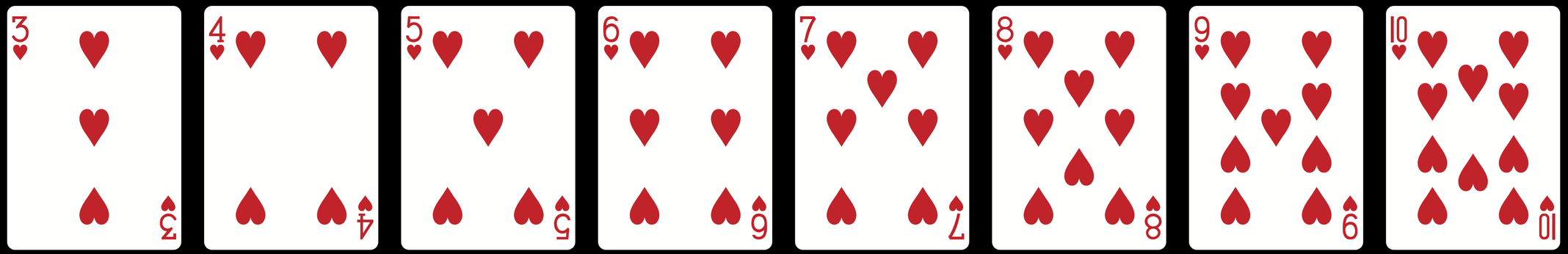
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

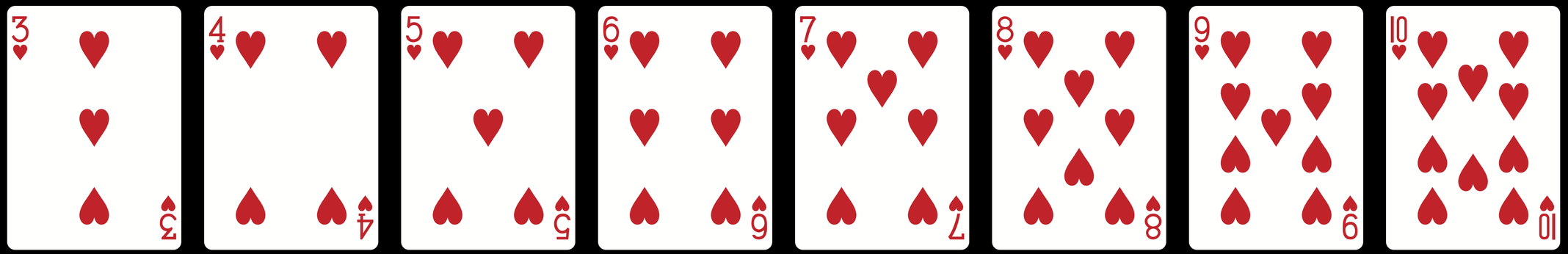
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

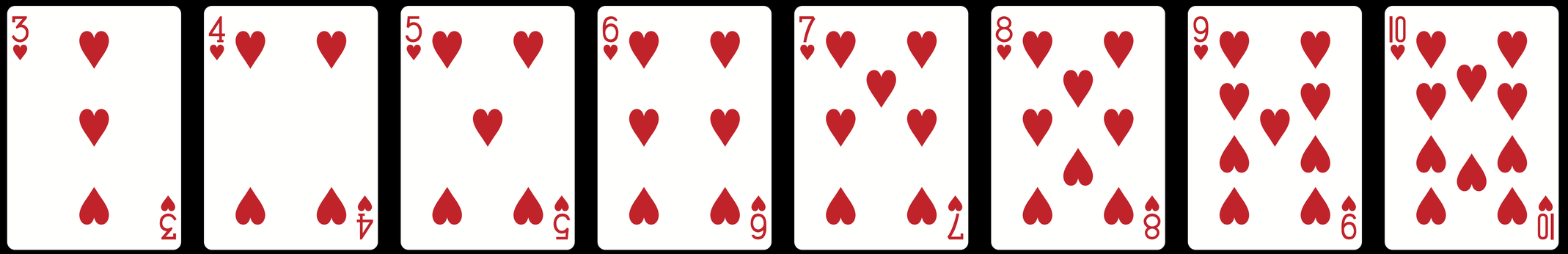
```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



0 1 2 3 4 5 6 7

Bubble sort

```
Repeat until A is sorted (no swaps in previous loop):
    for i = 0 ... n – 2:
        if A[i] > A[i + 1]:
            swap A[i] > A[i + 1]
```



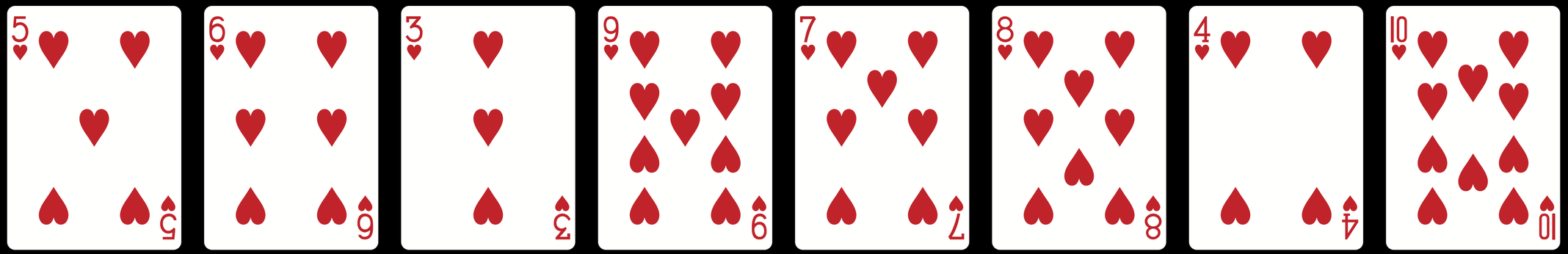
0 1 2 3 4 5 6 7

Selection sort

Find the smallest item in the unsorted part.

Swap this item to the front and “fix” it.

Repeat for unfixed items until all items are fixed.

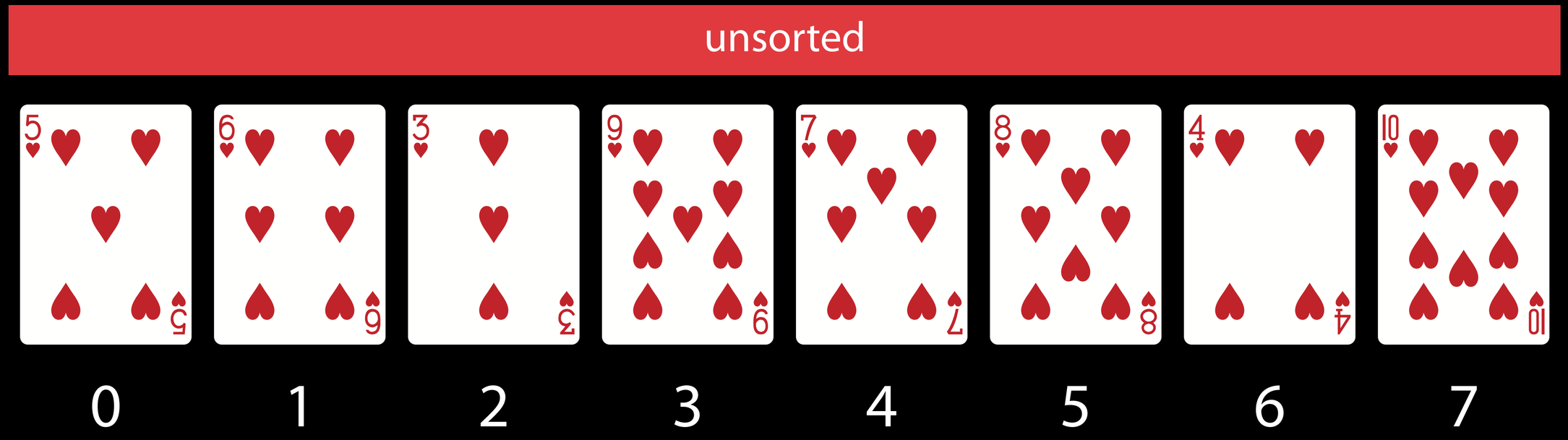


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

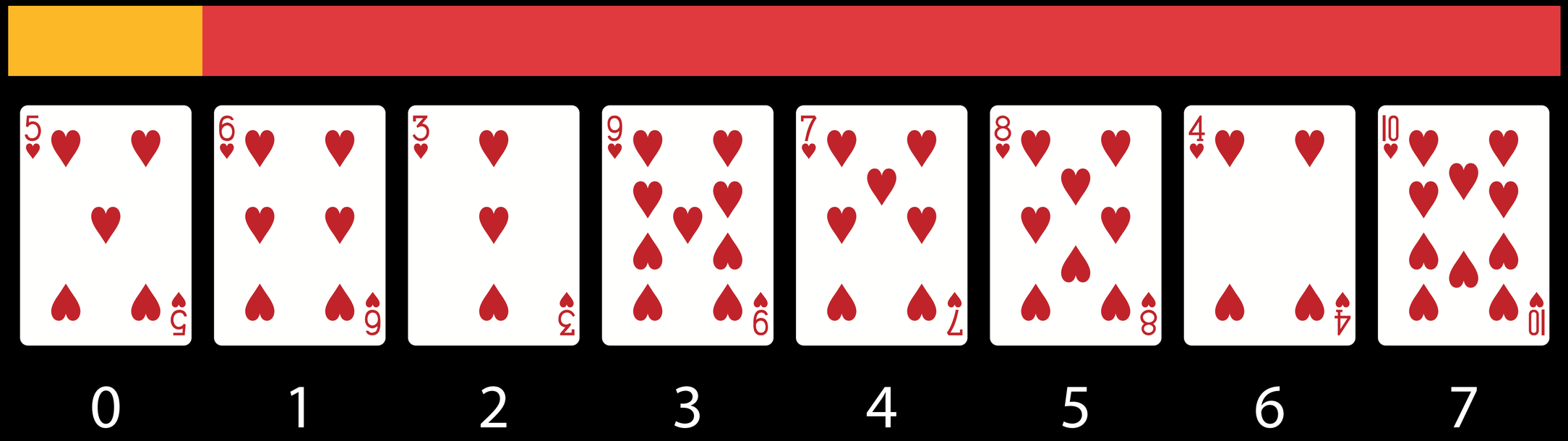


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

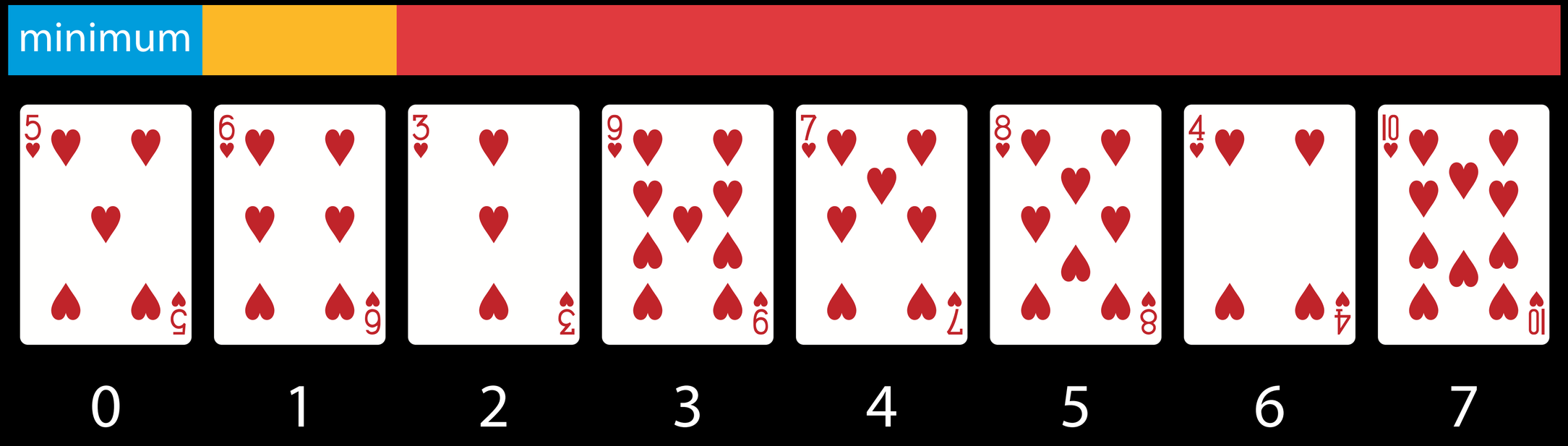


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

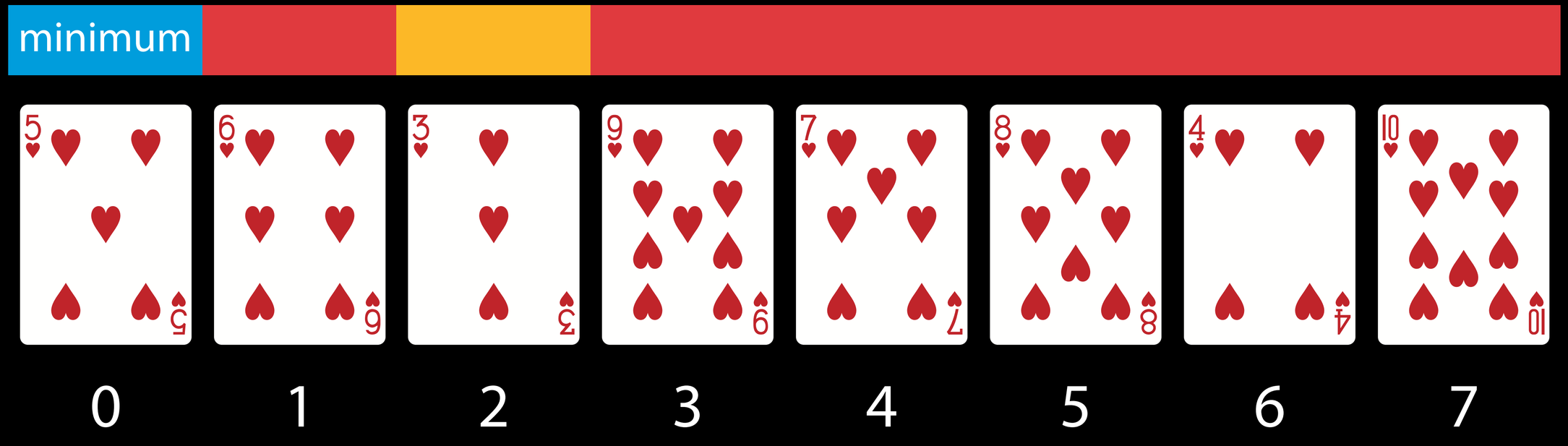


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

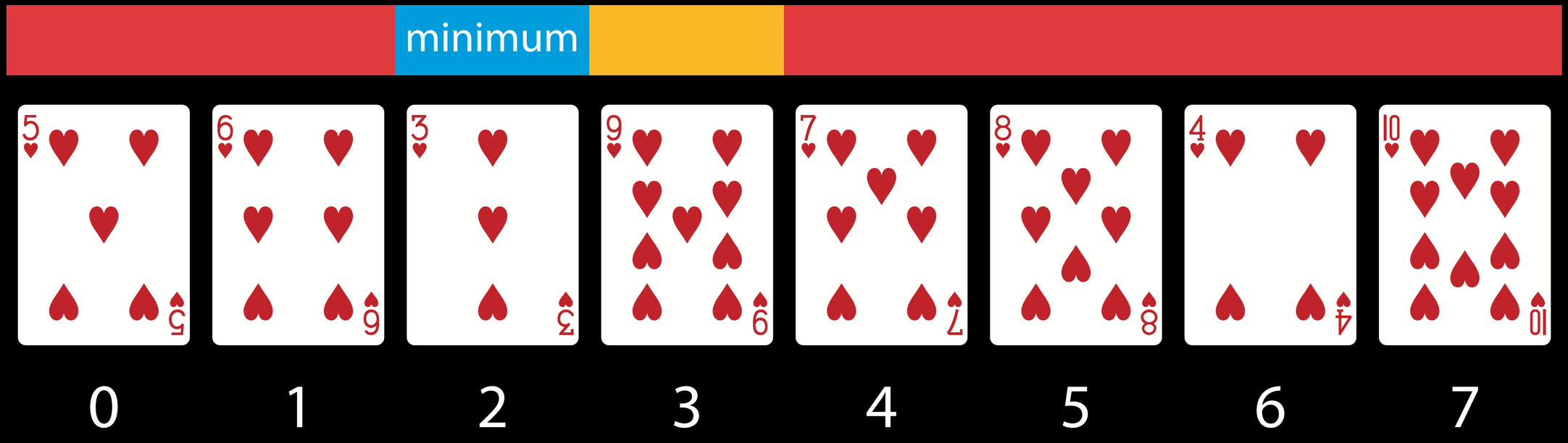


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

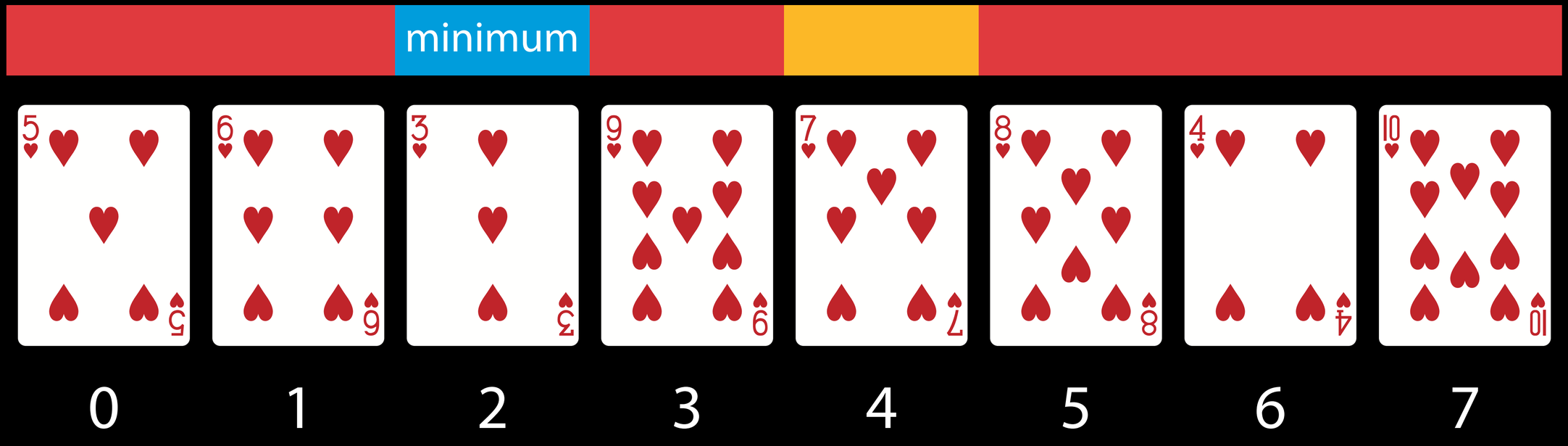


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

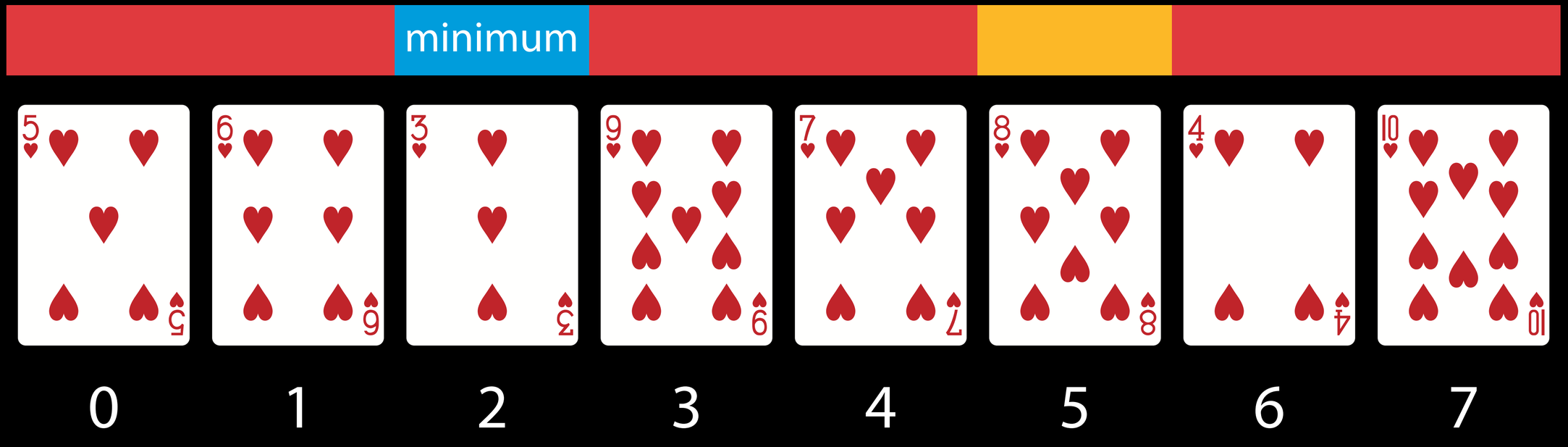


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

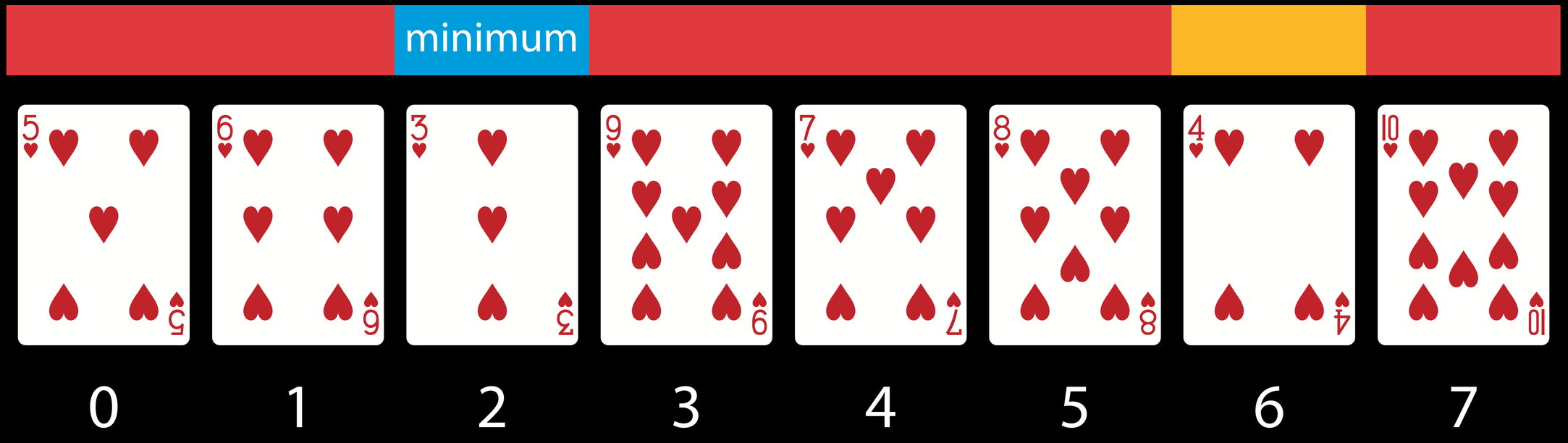


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

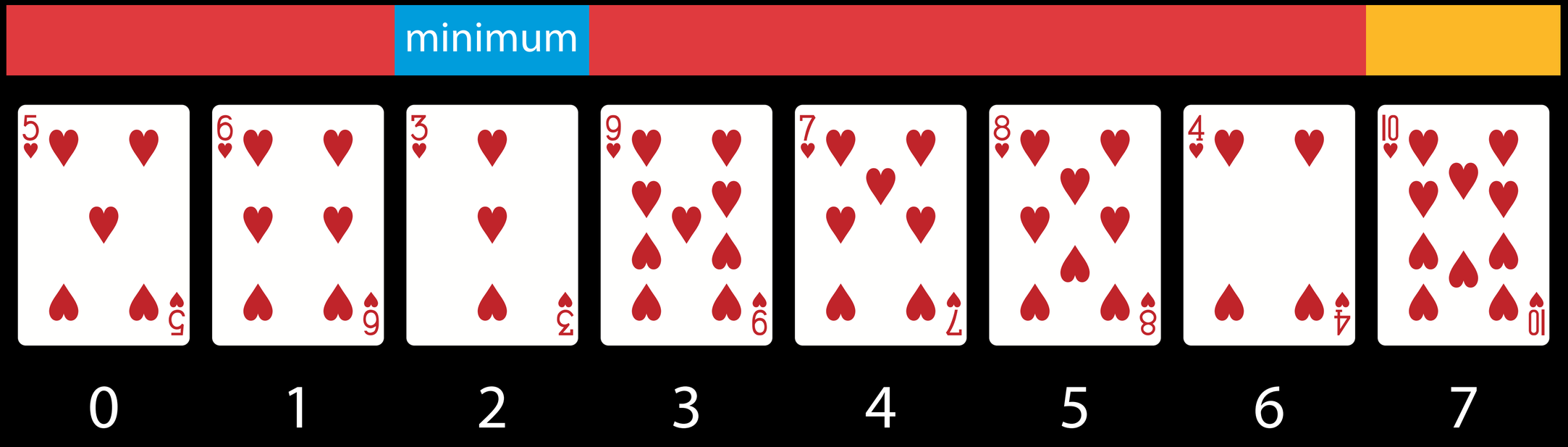


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

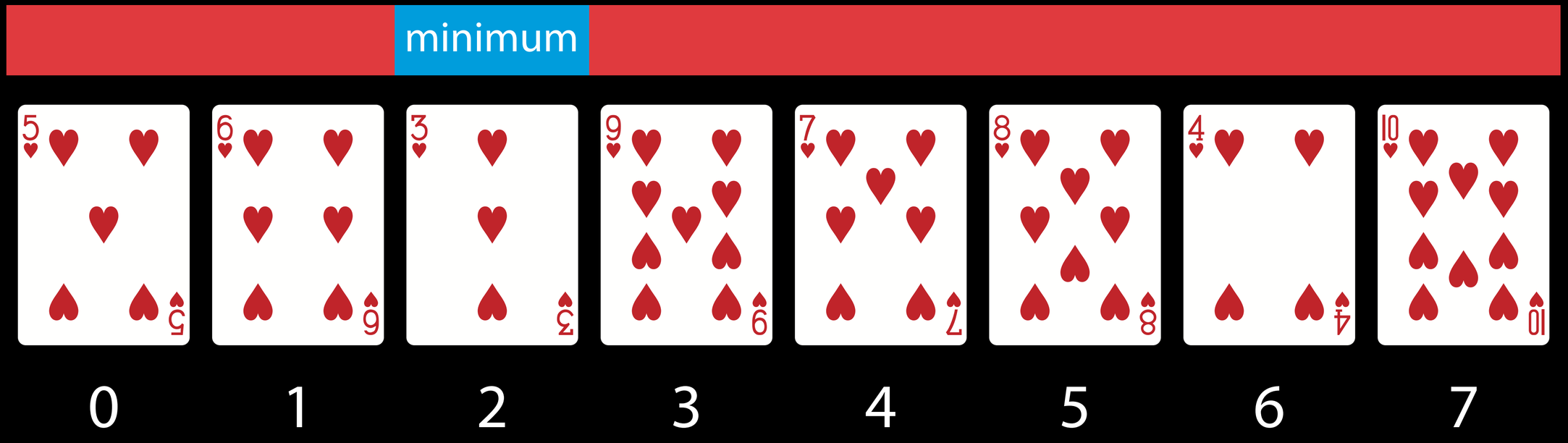


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

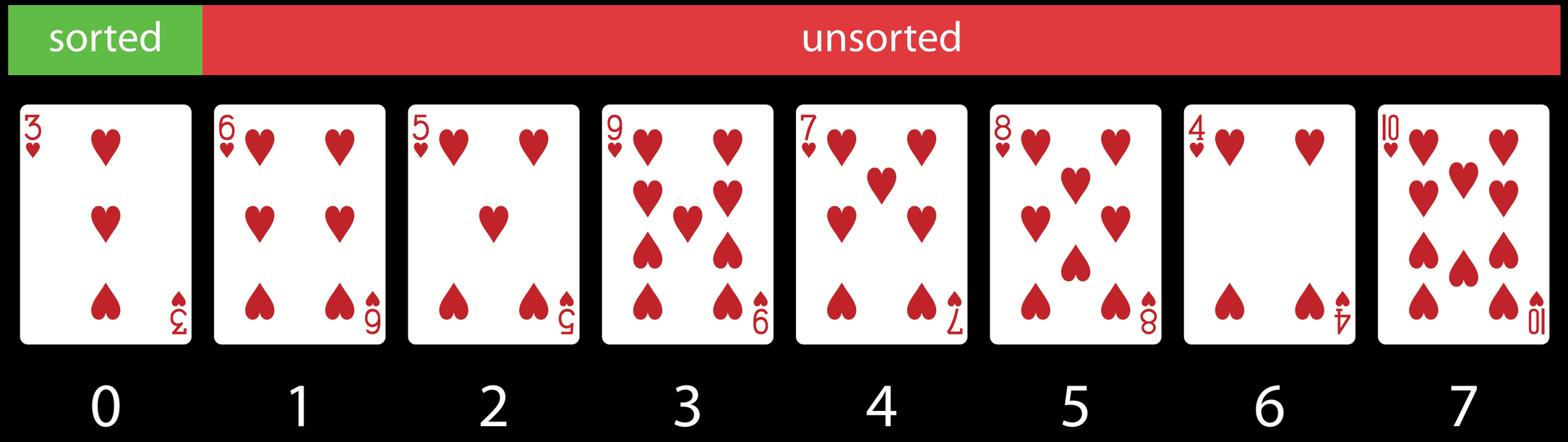


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

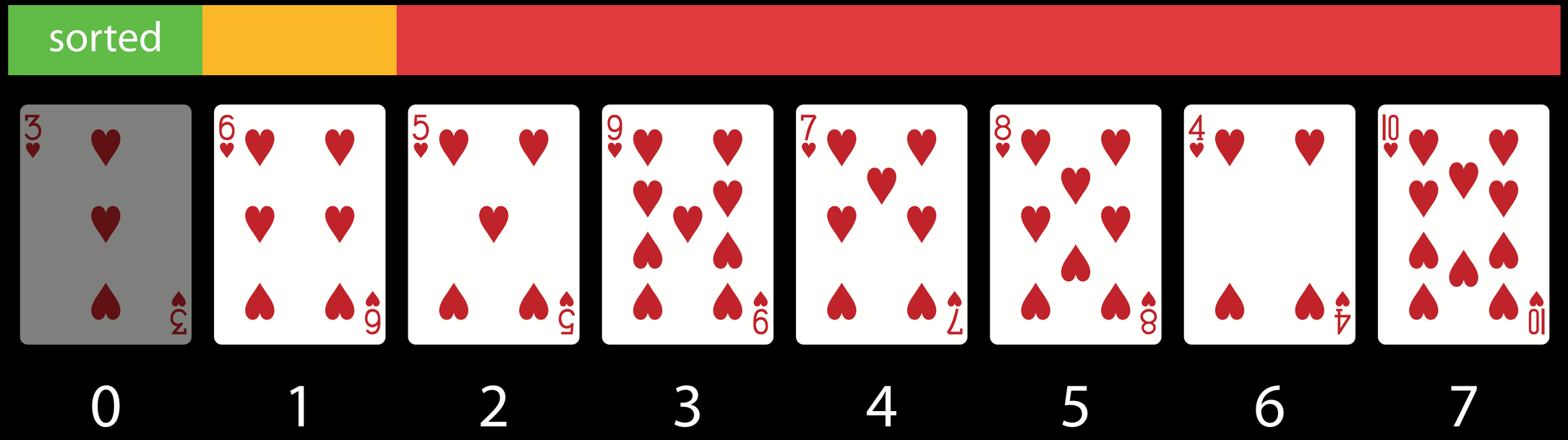


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

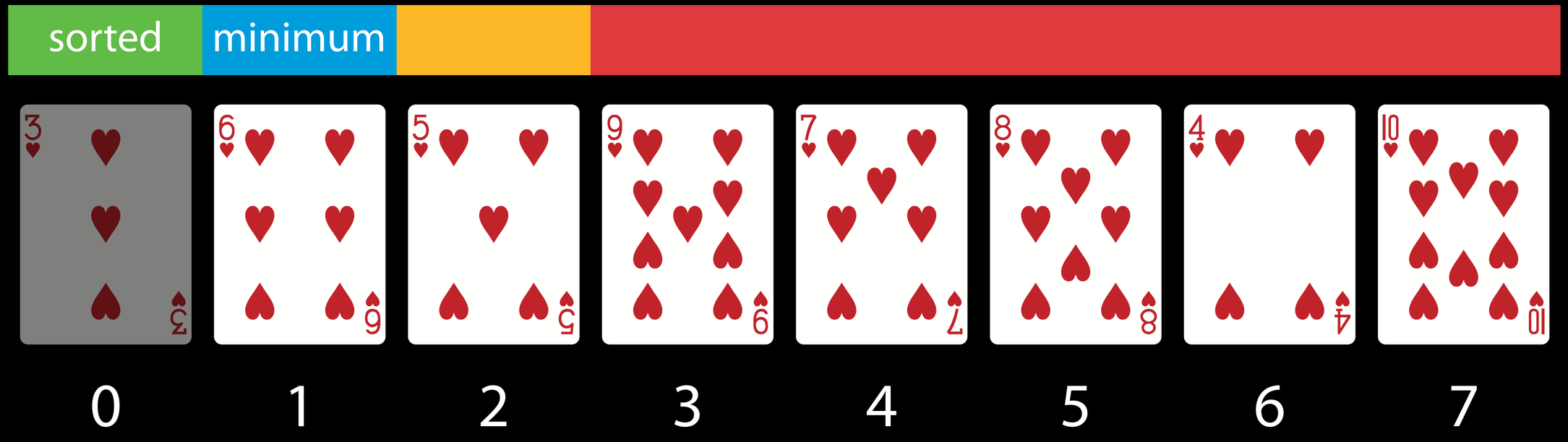


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

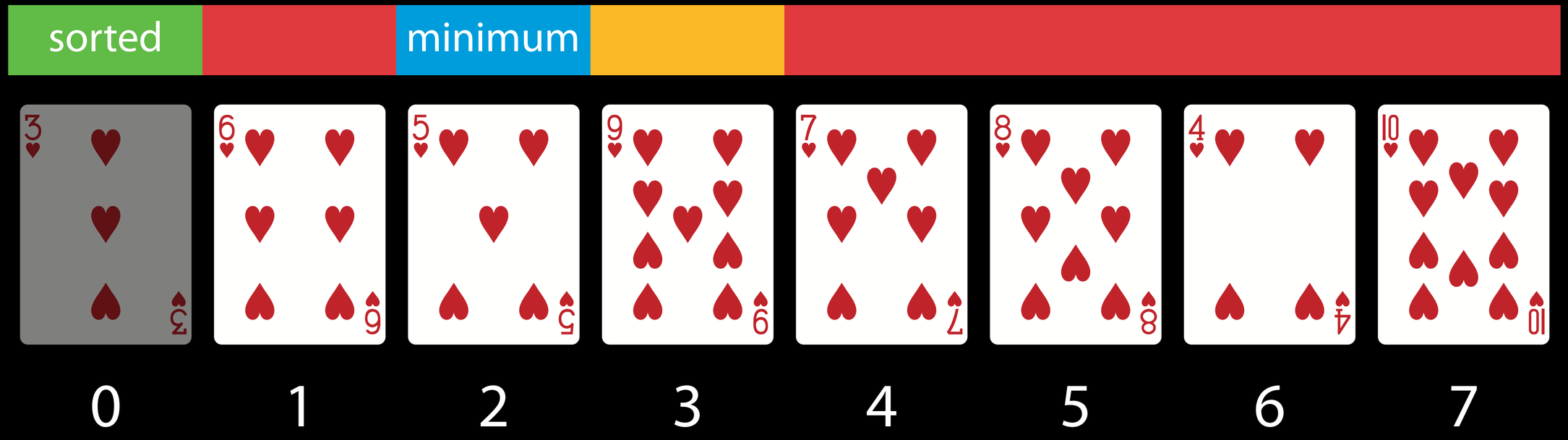


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

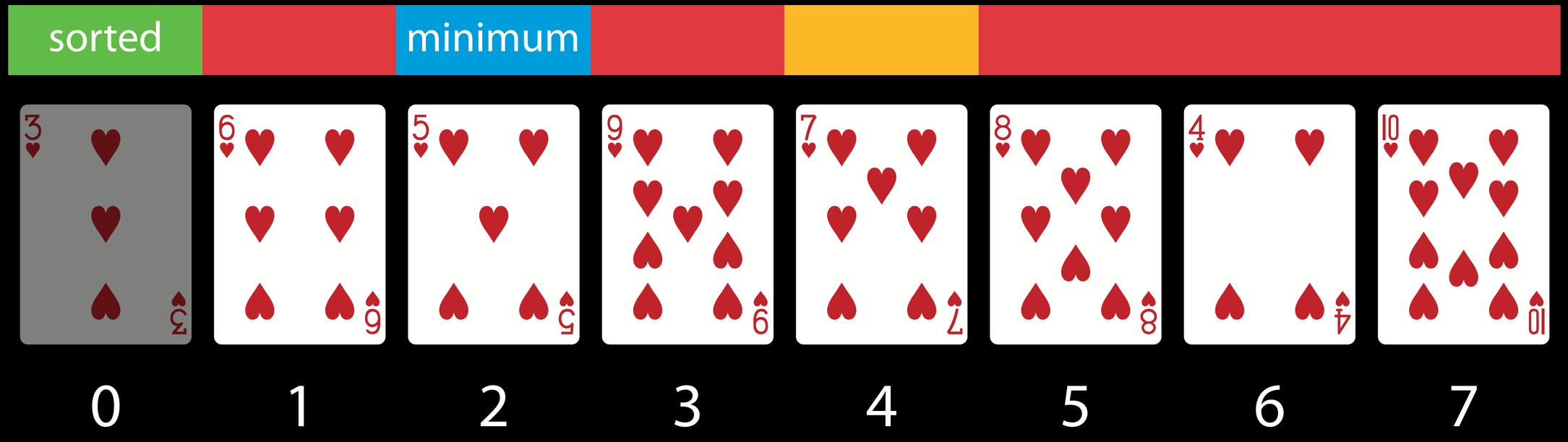


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

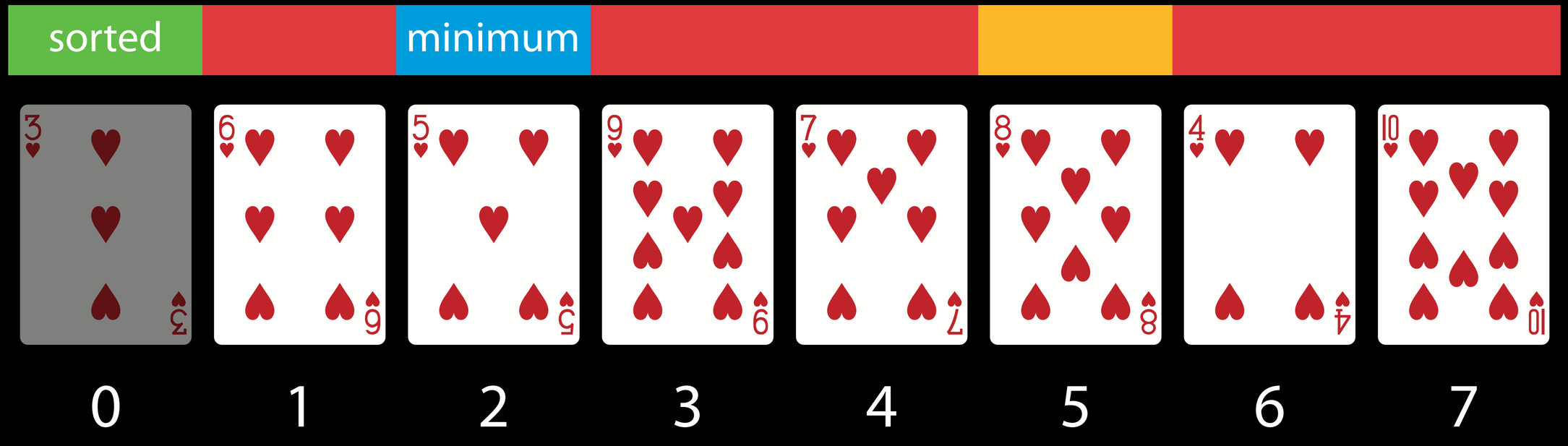


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

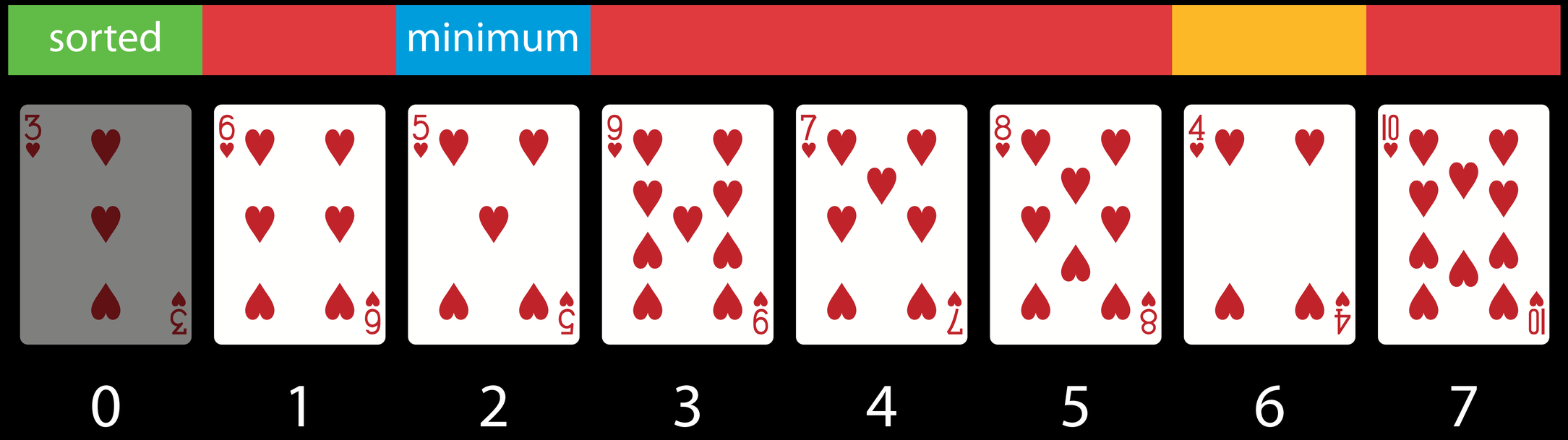


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

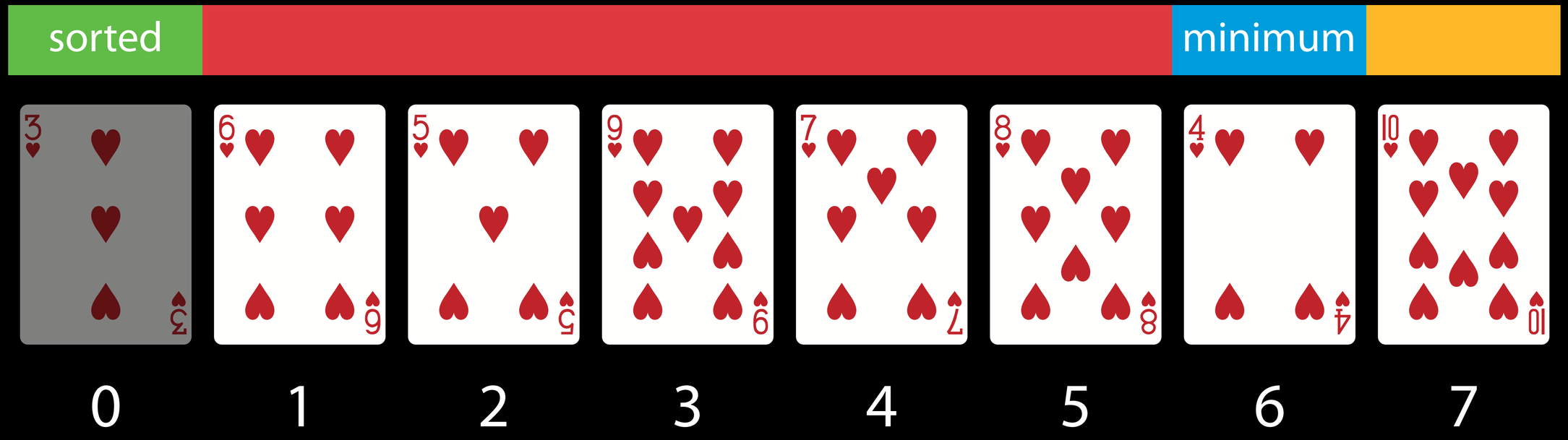


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

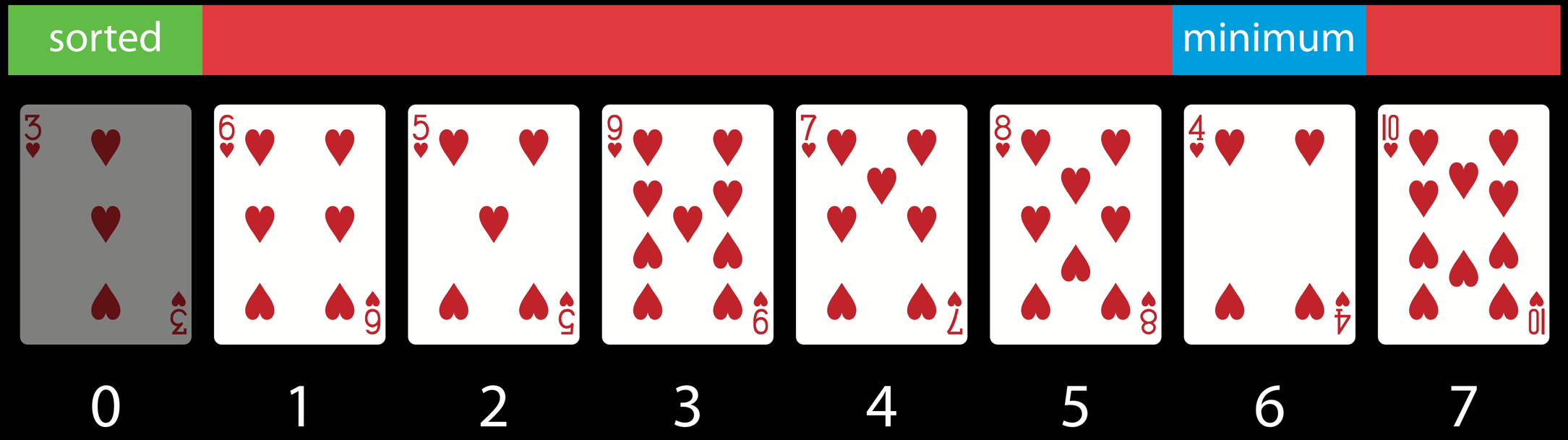


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

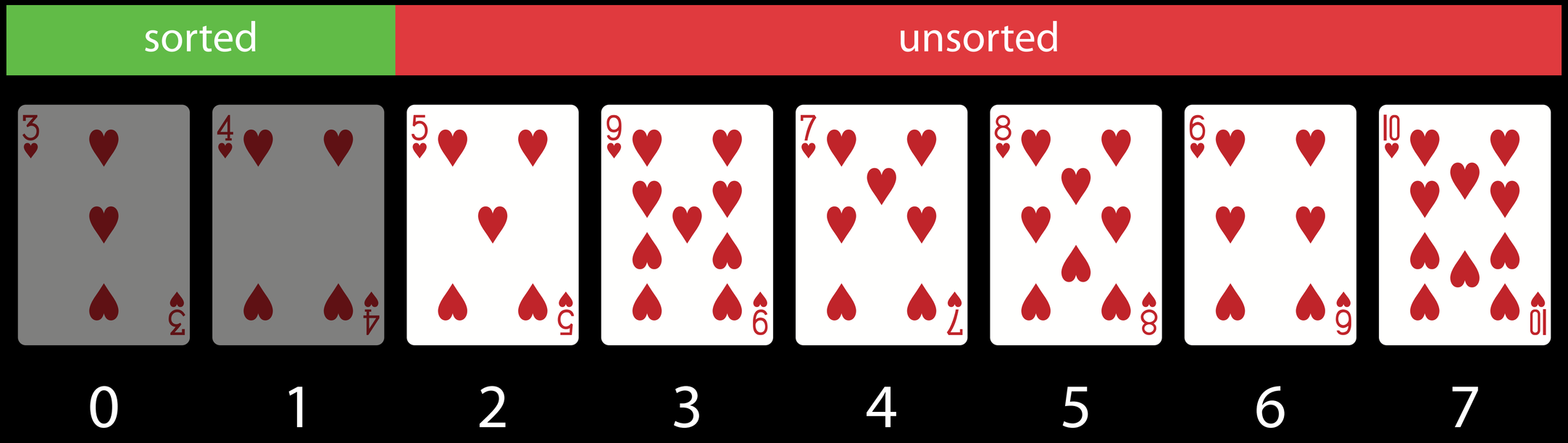


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

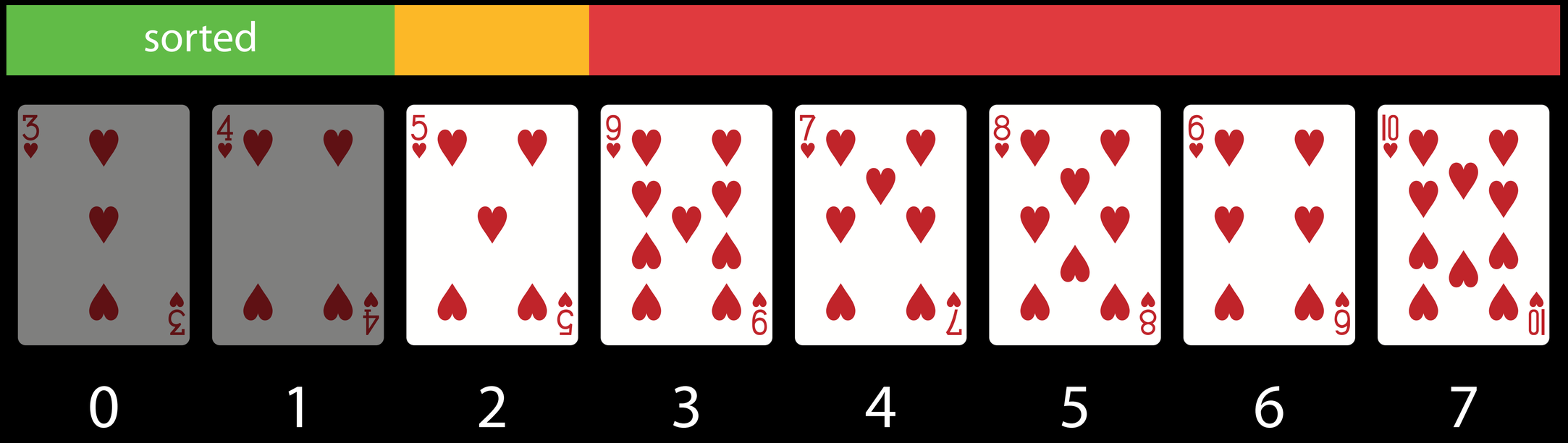


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

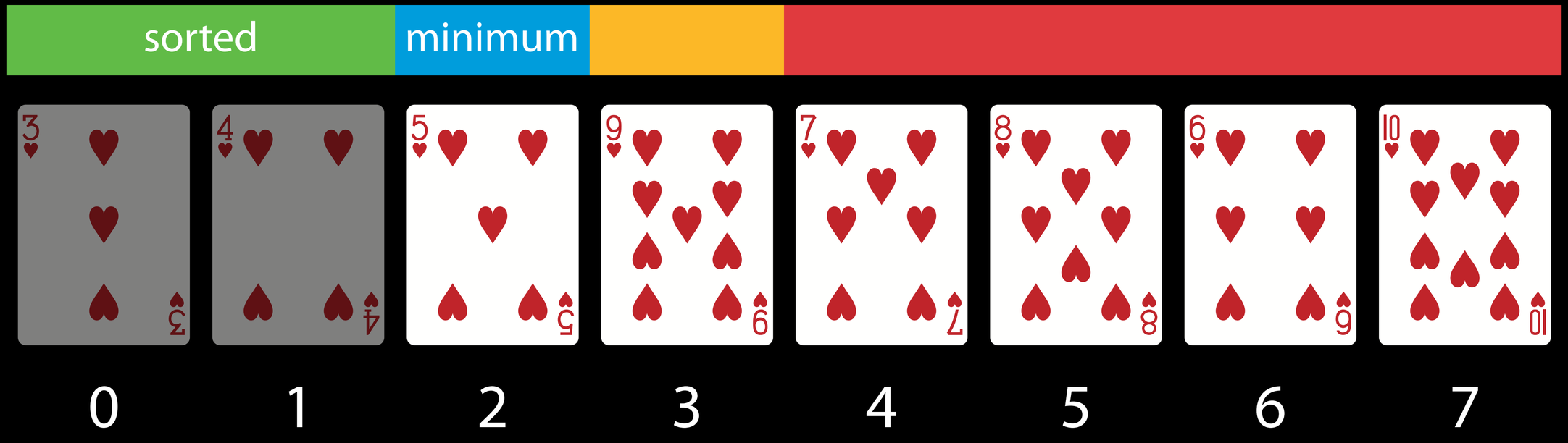


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

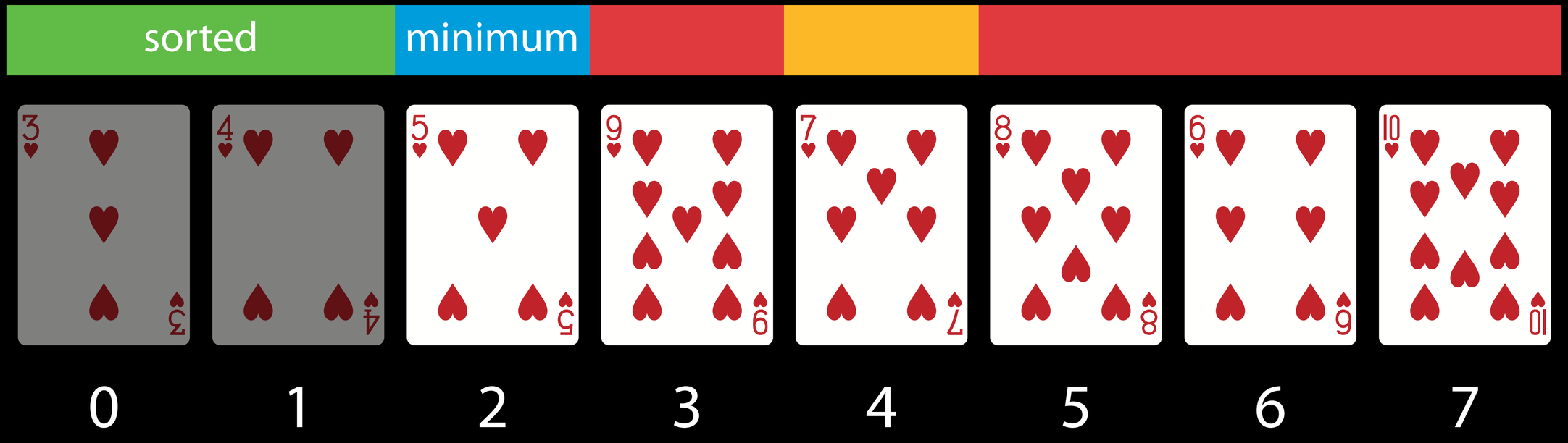


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

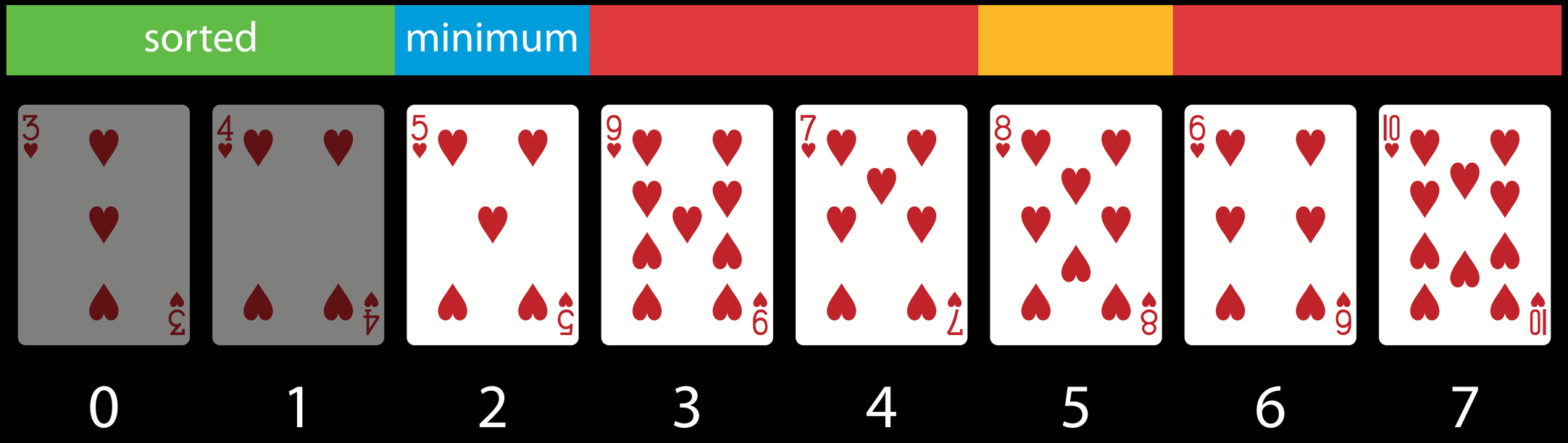


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

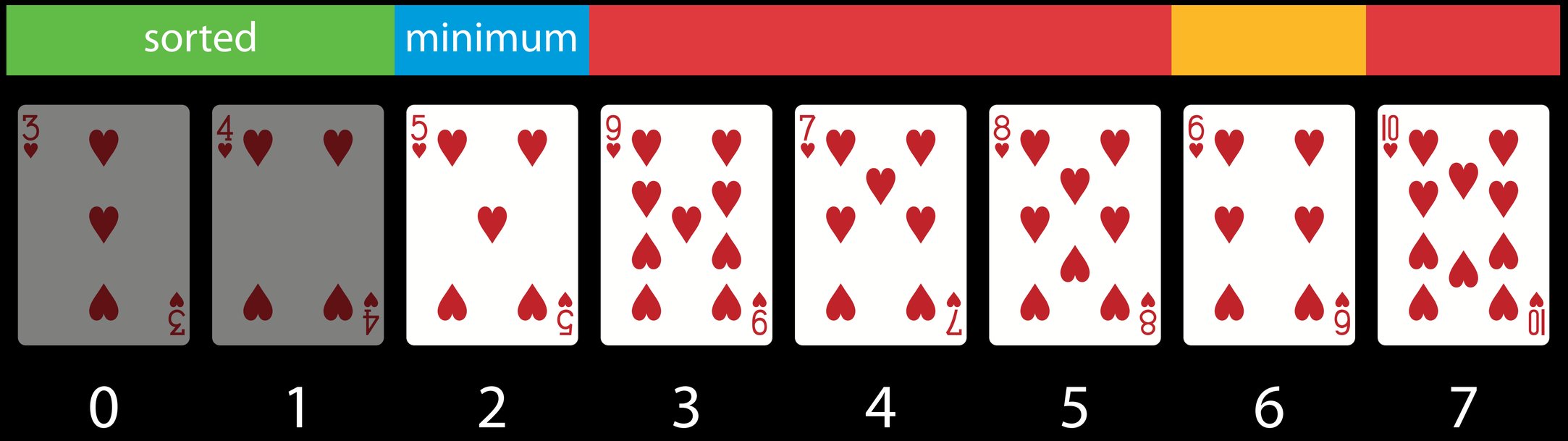


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

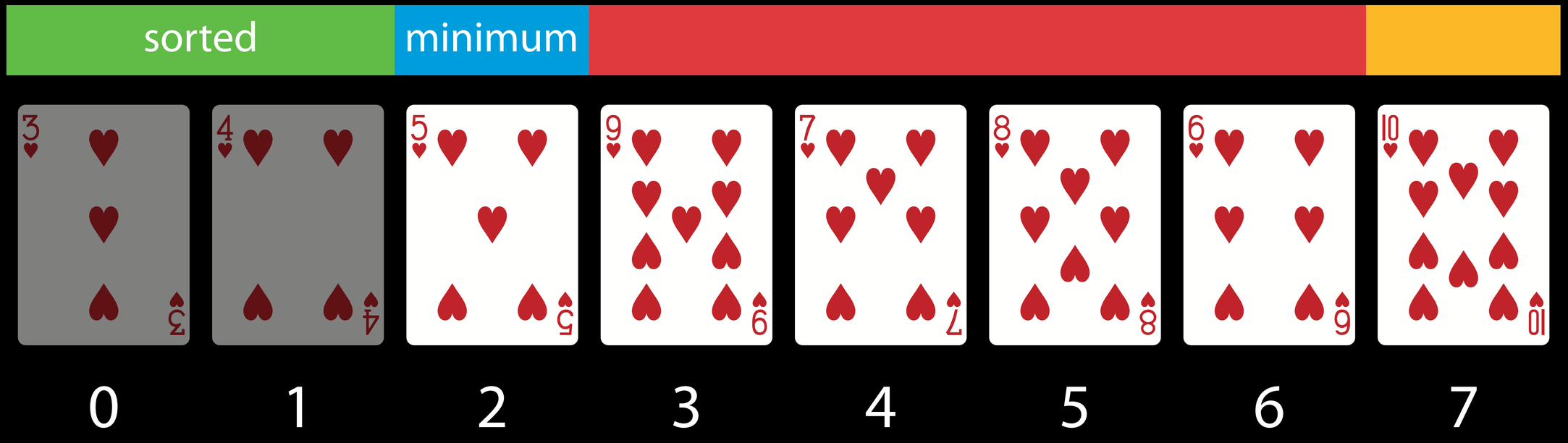


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

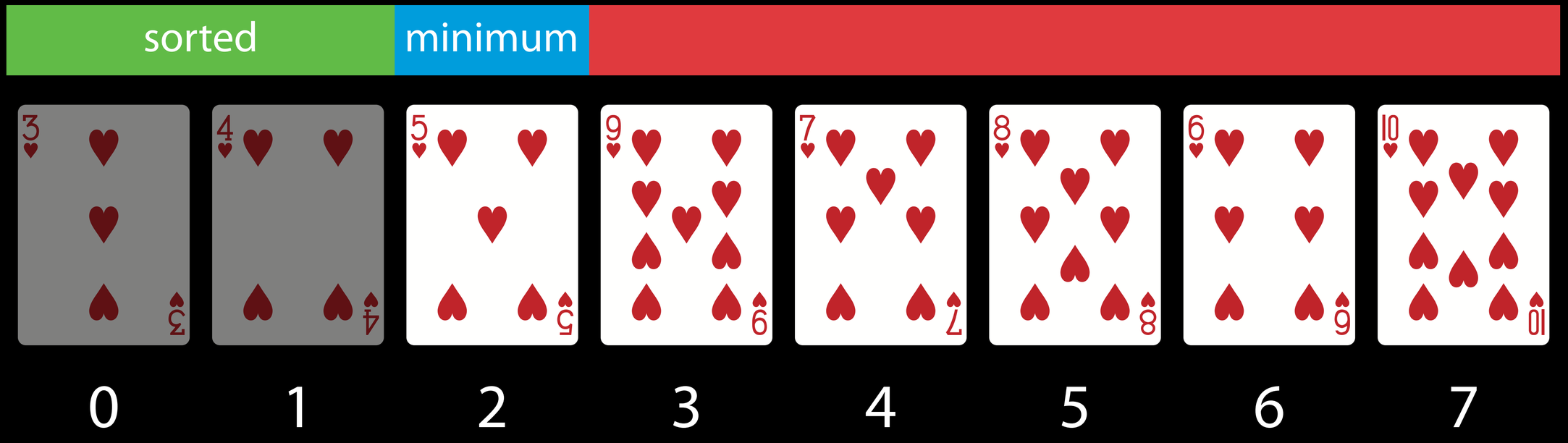


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

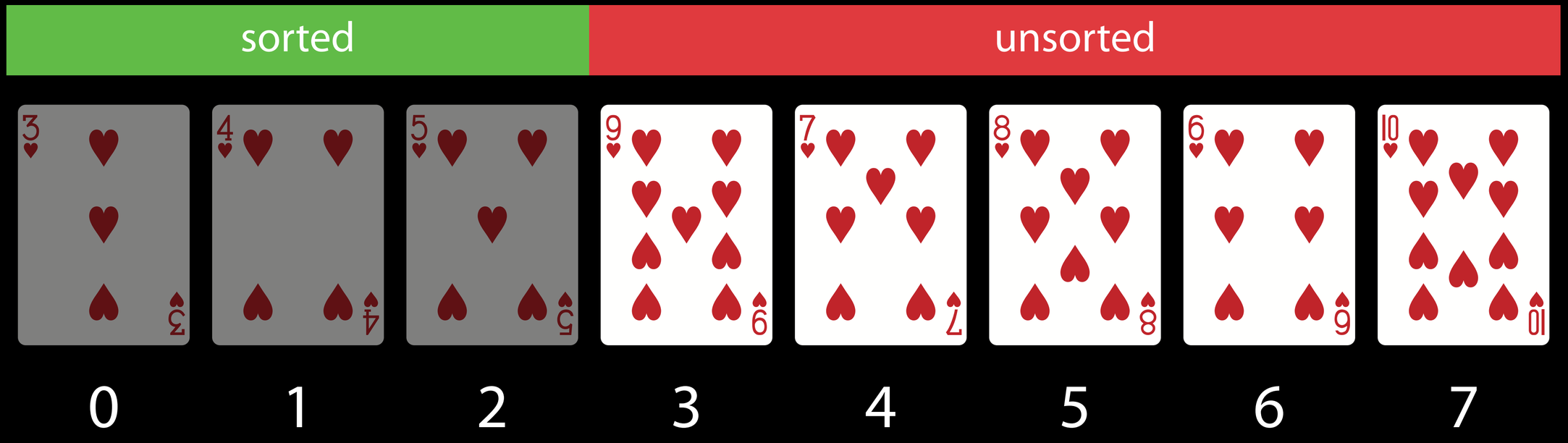


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

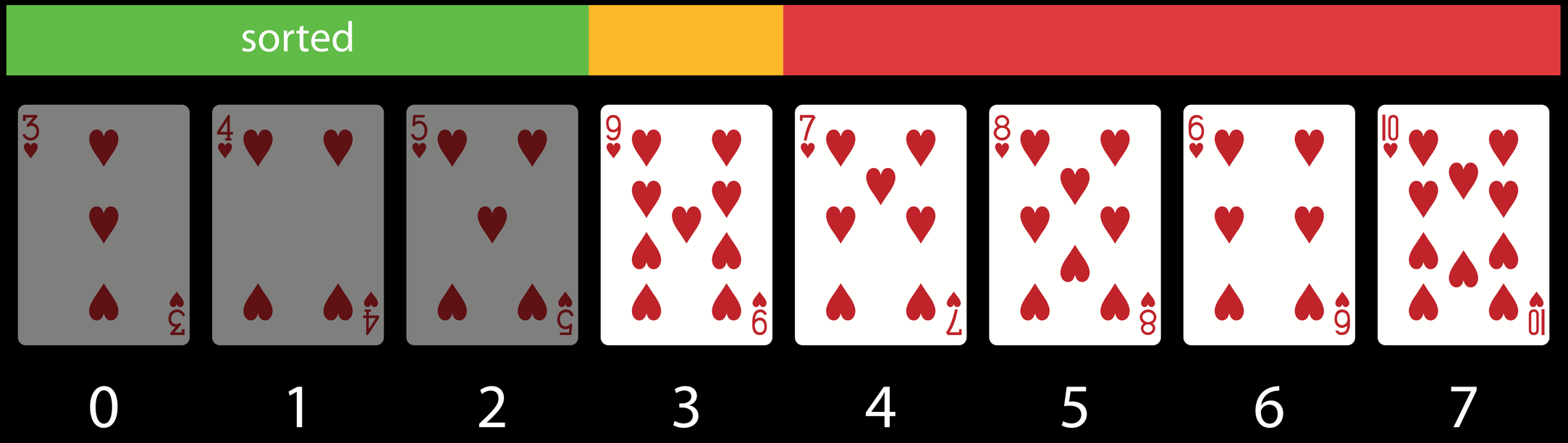


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

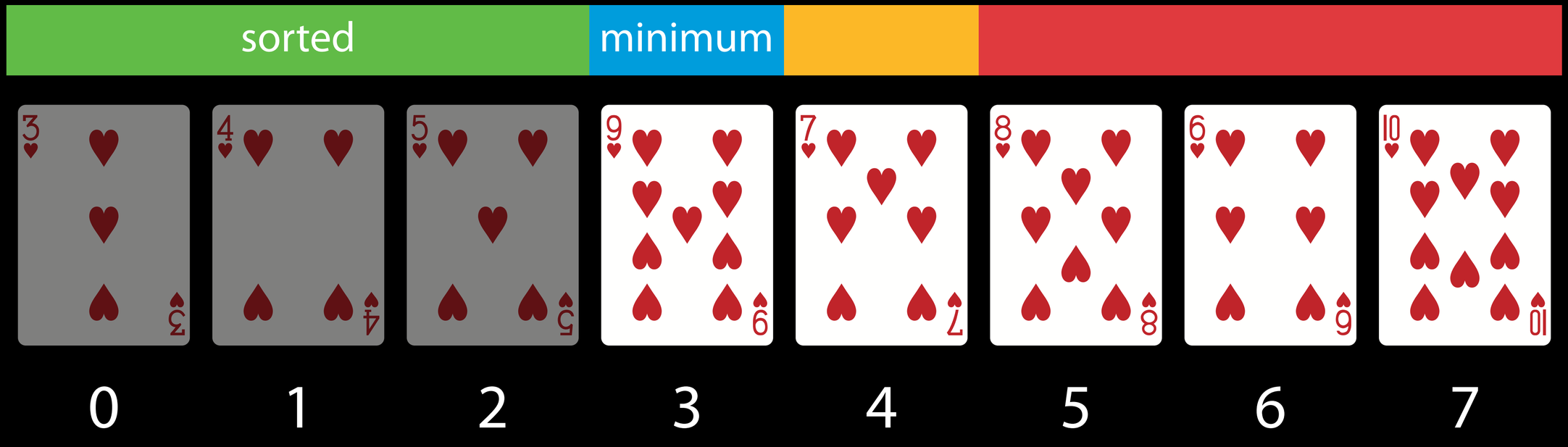


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

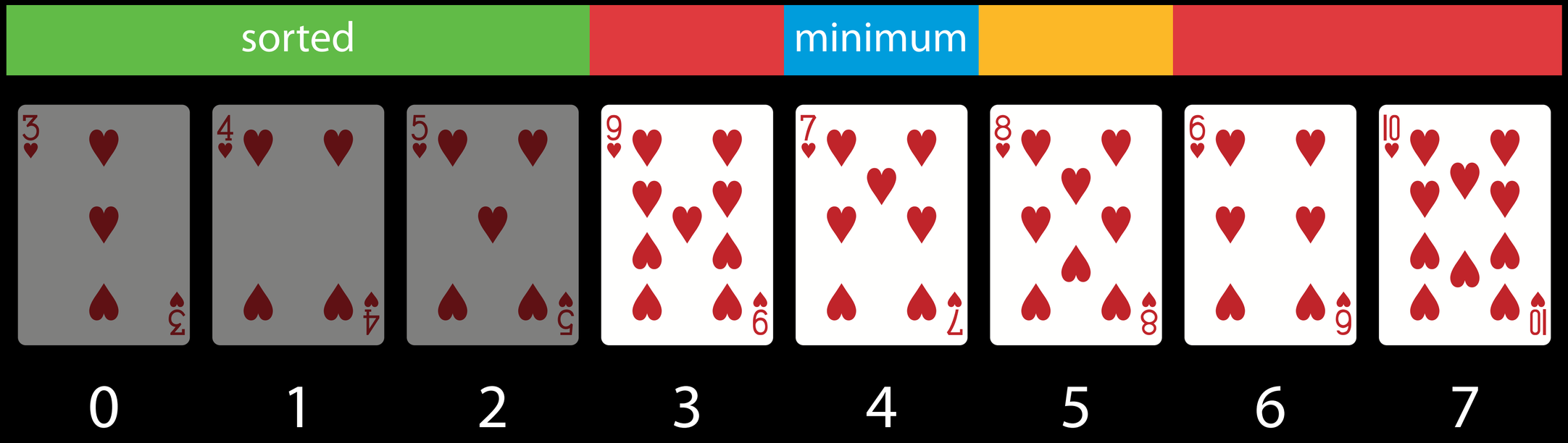


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

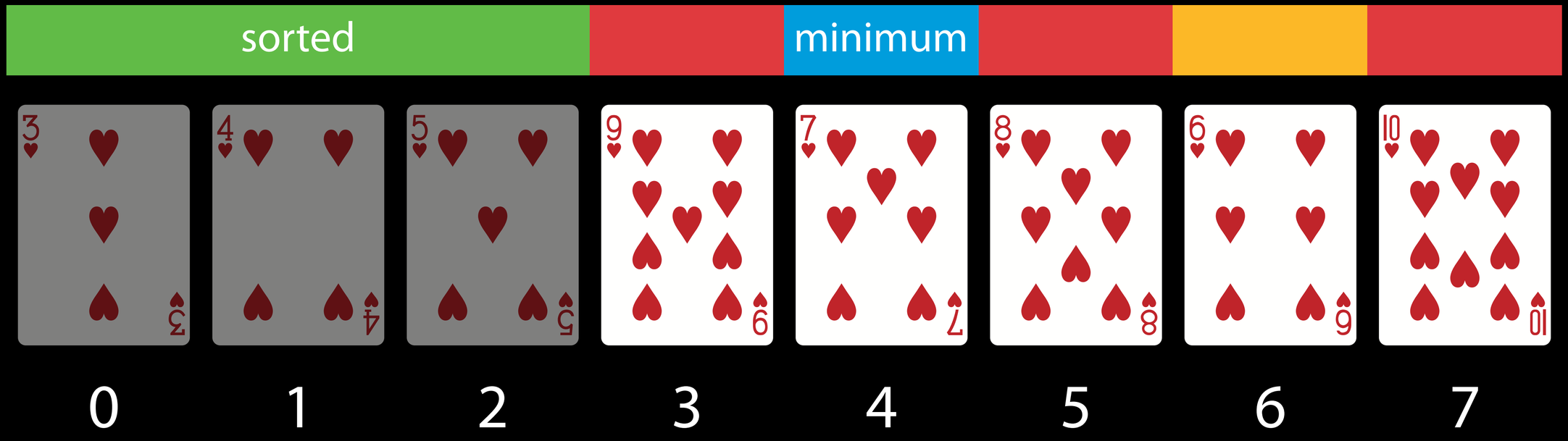


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

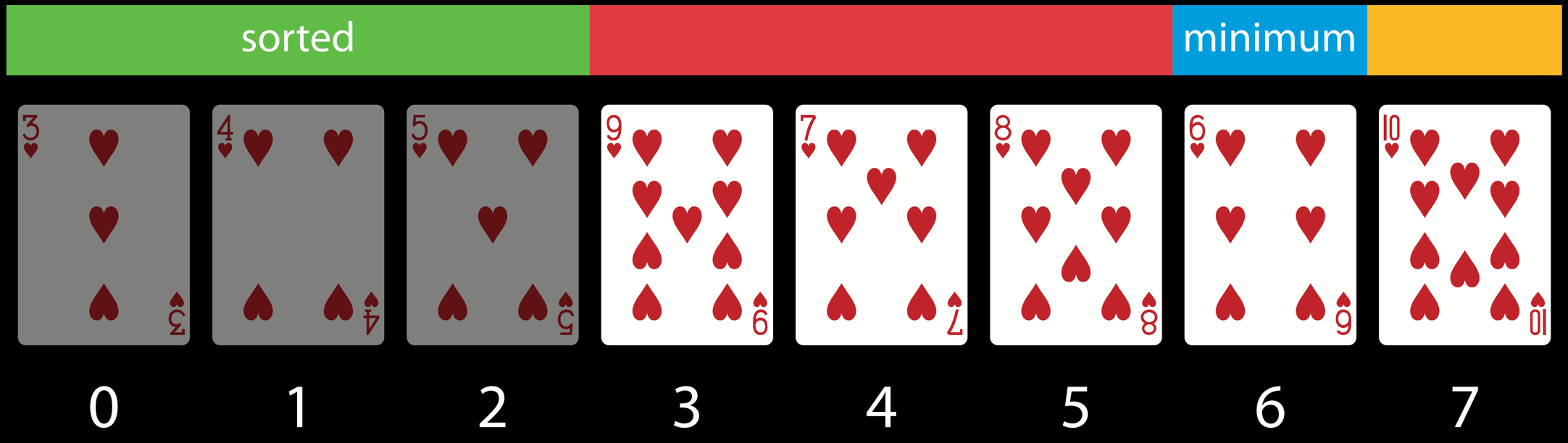


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

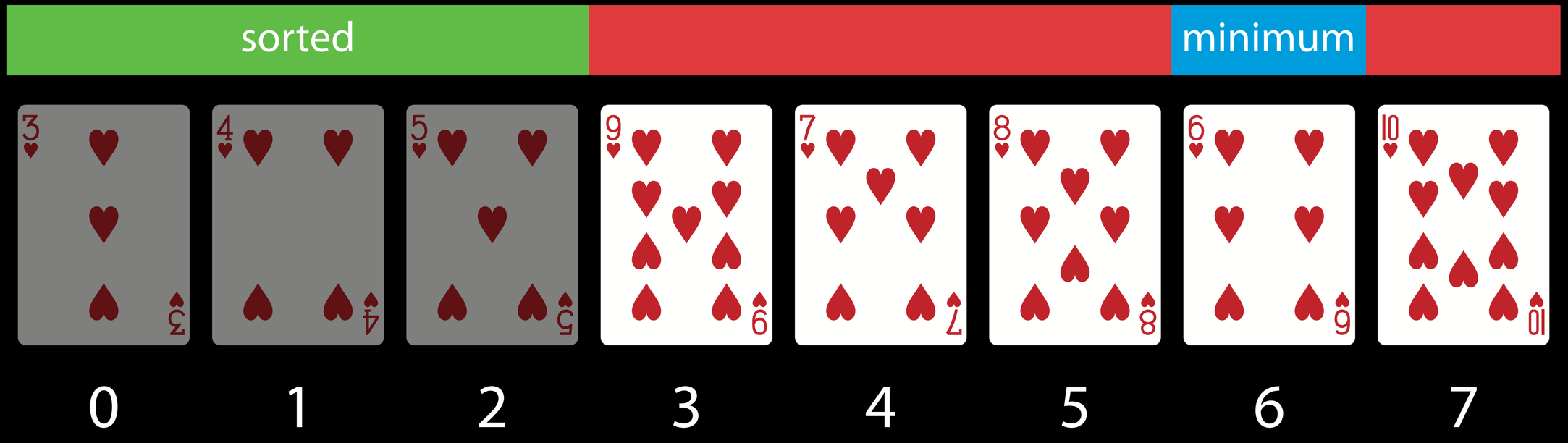


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

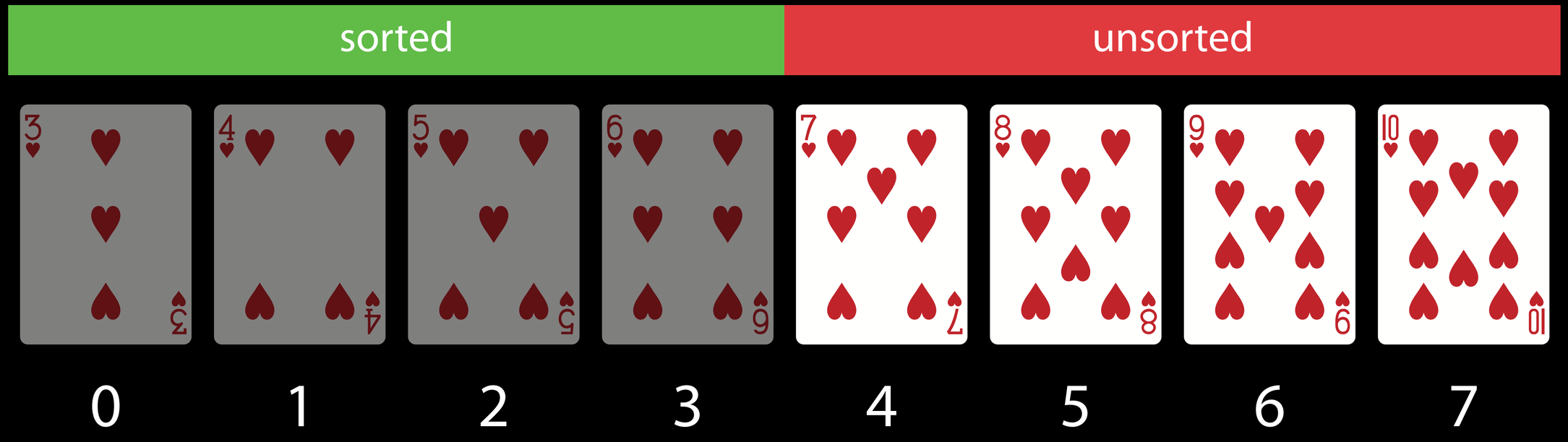


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

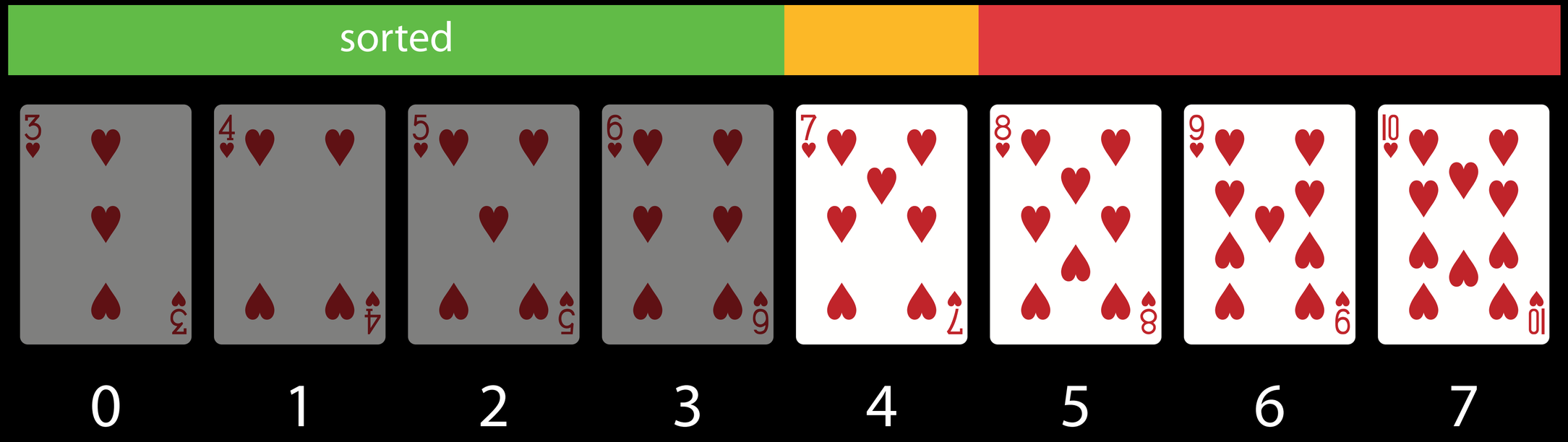


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

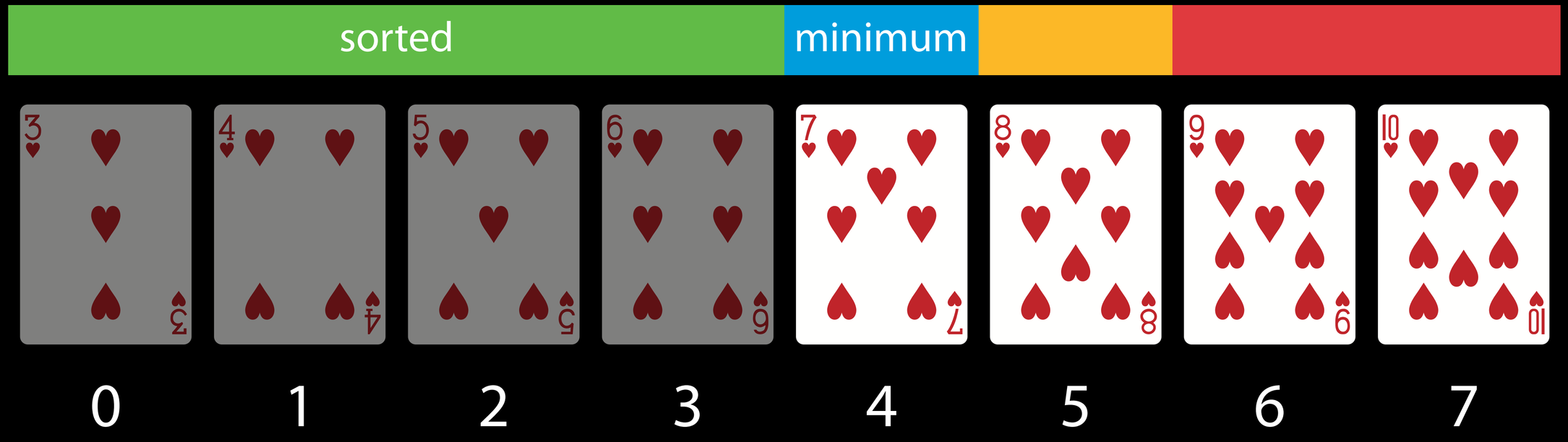


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

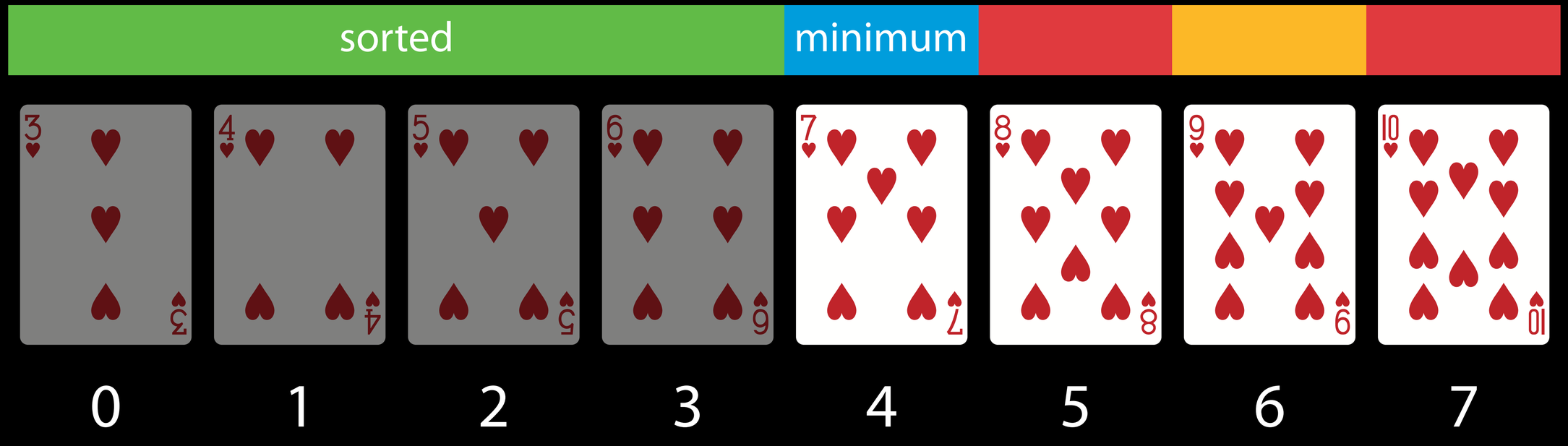


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

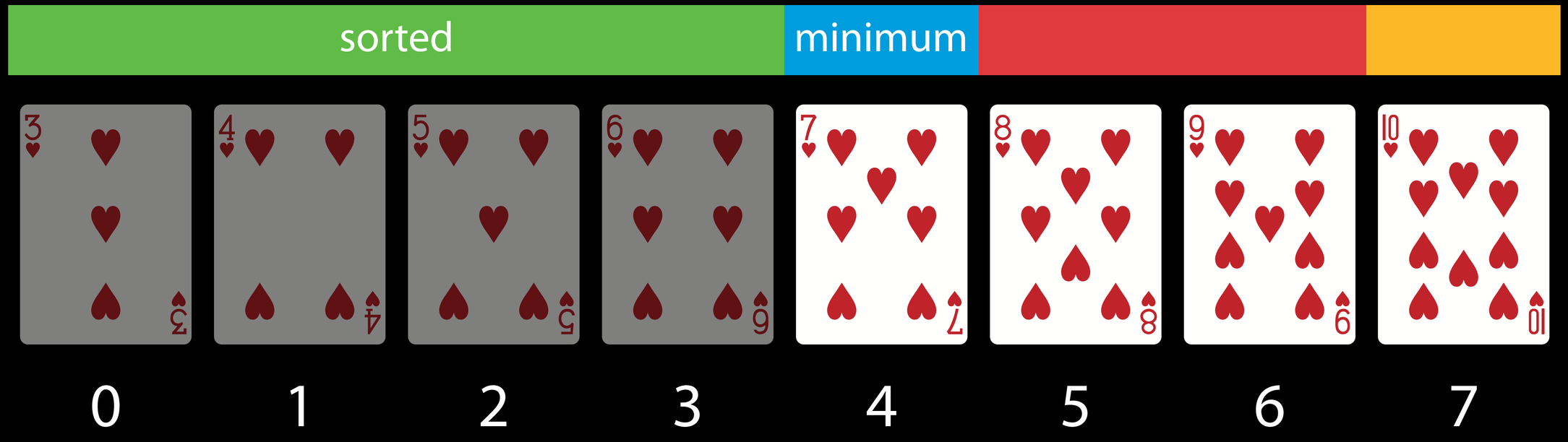


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

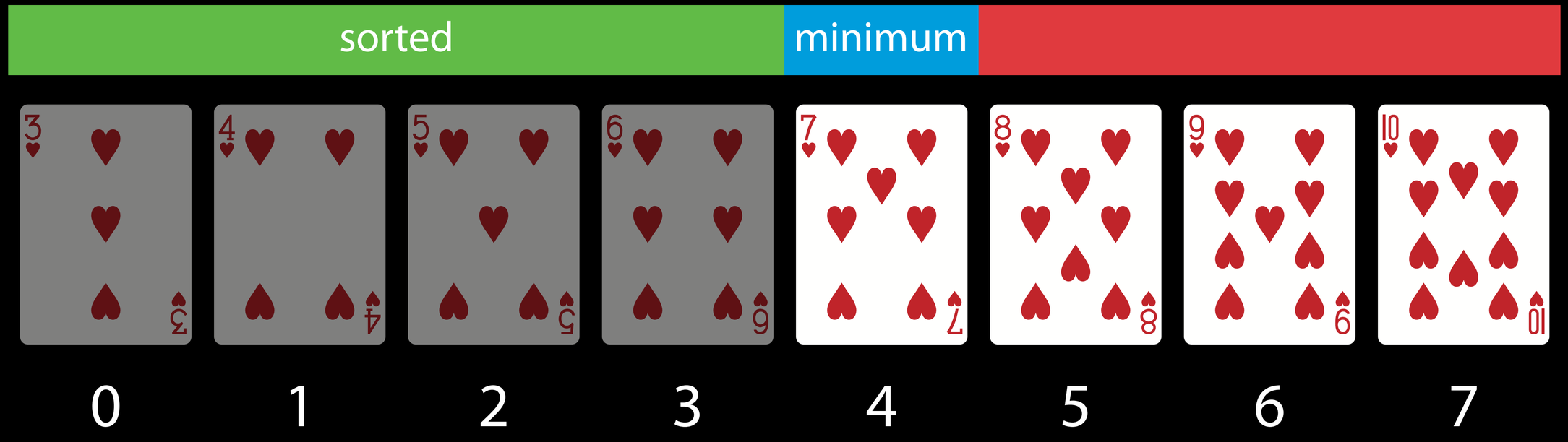


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

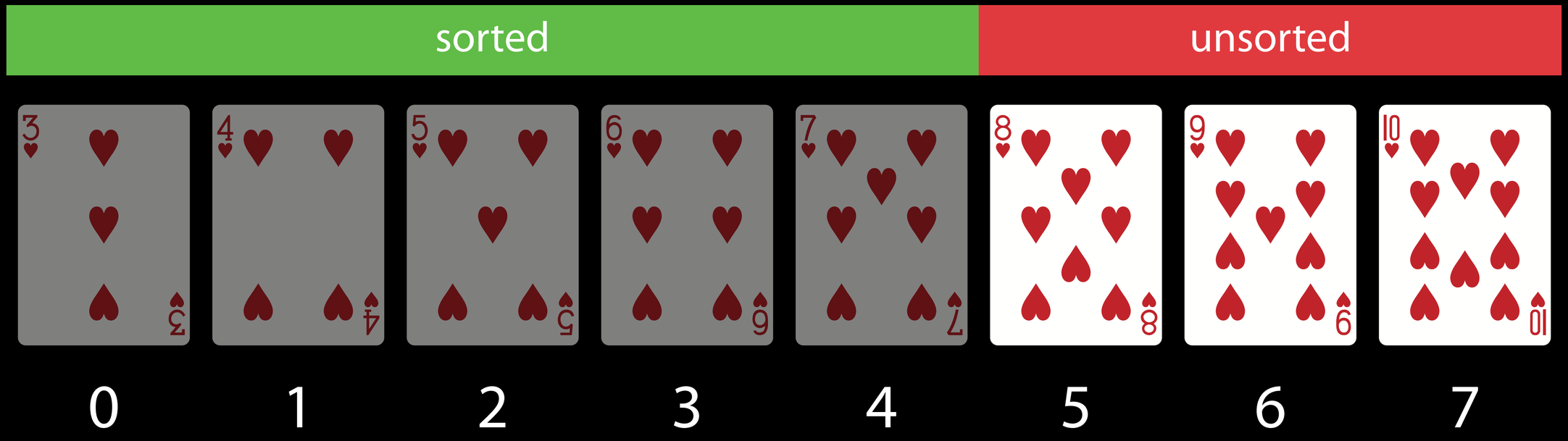


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

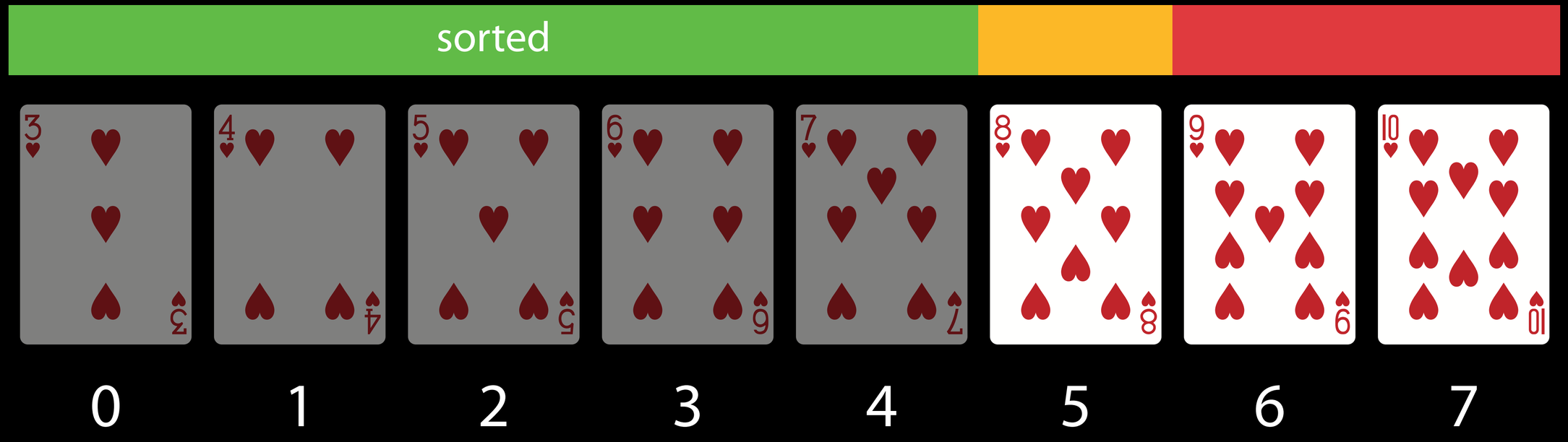


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

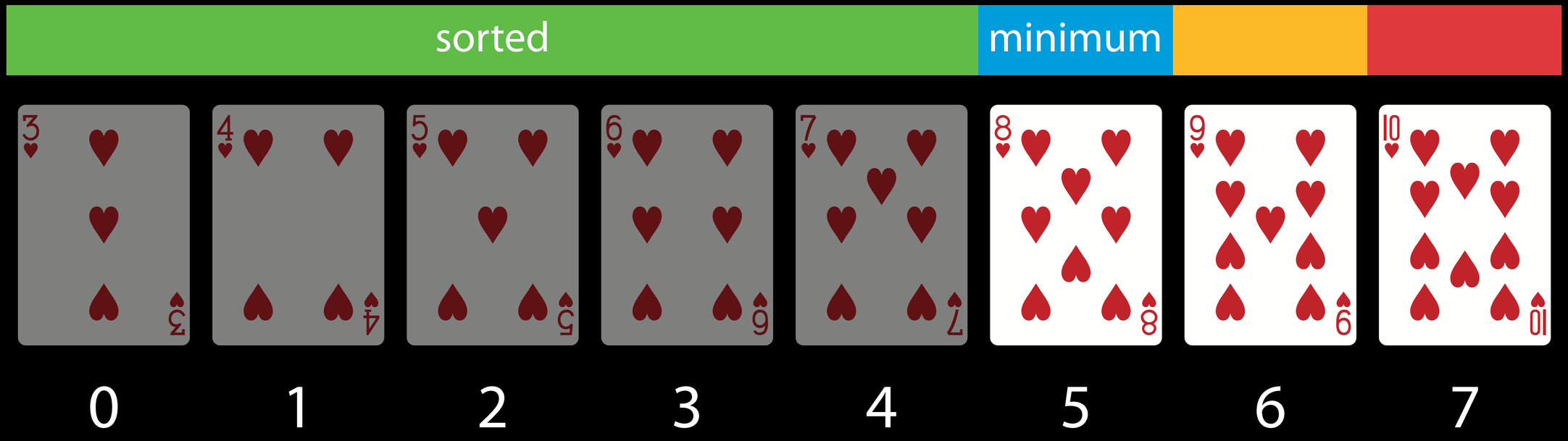


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

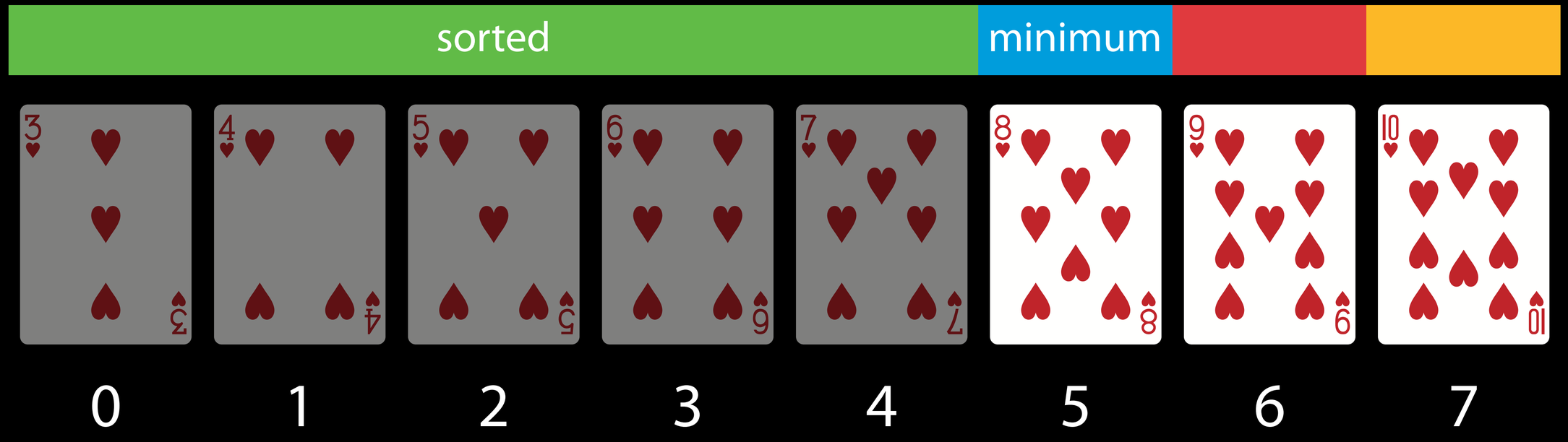


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

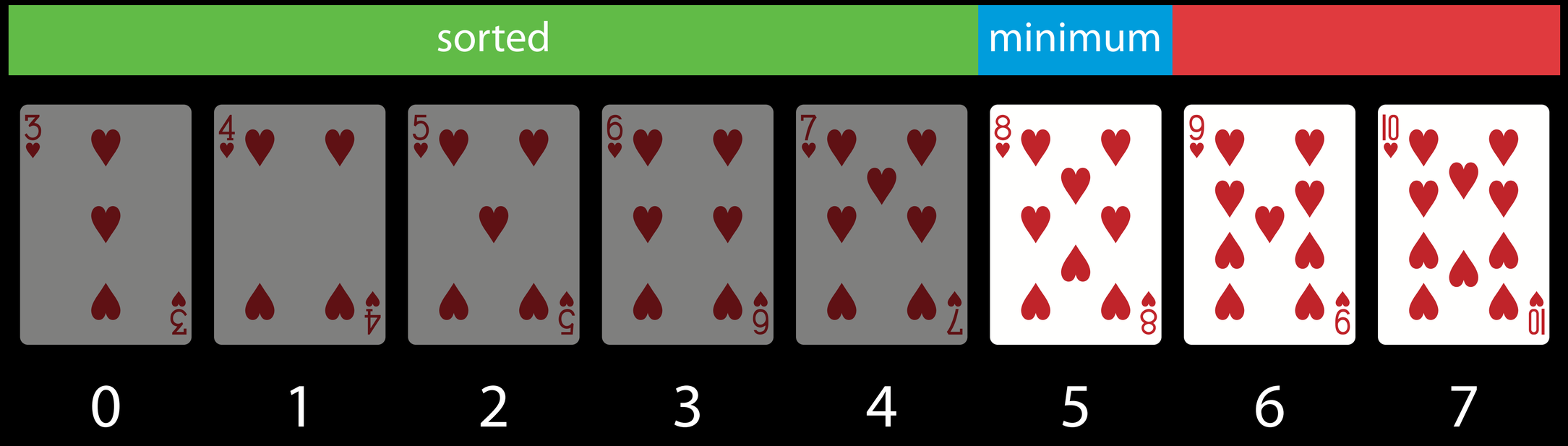


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

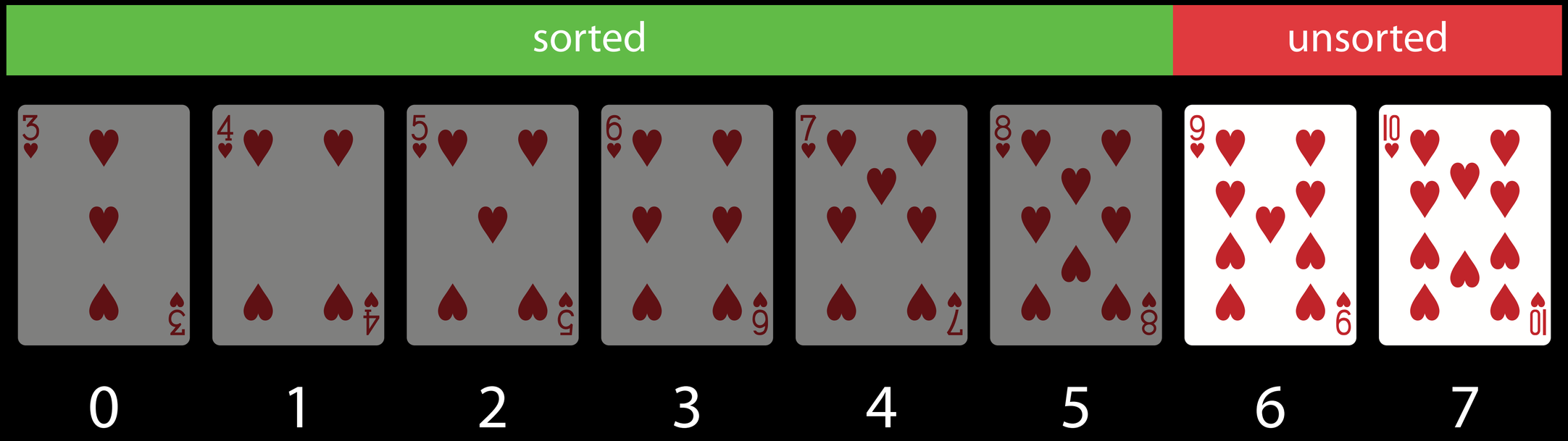


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

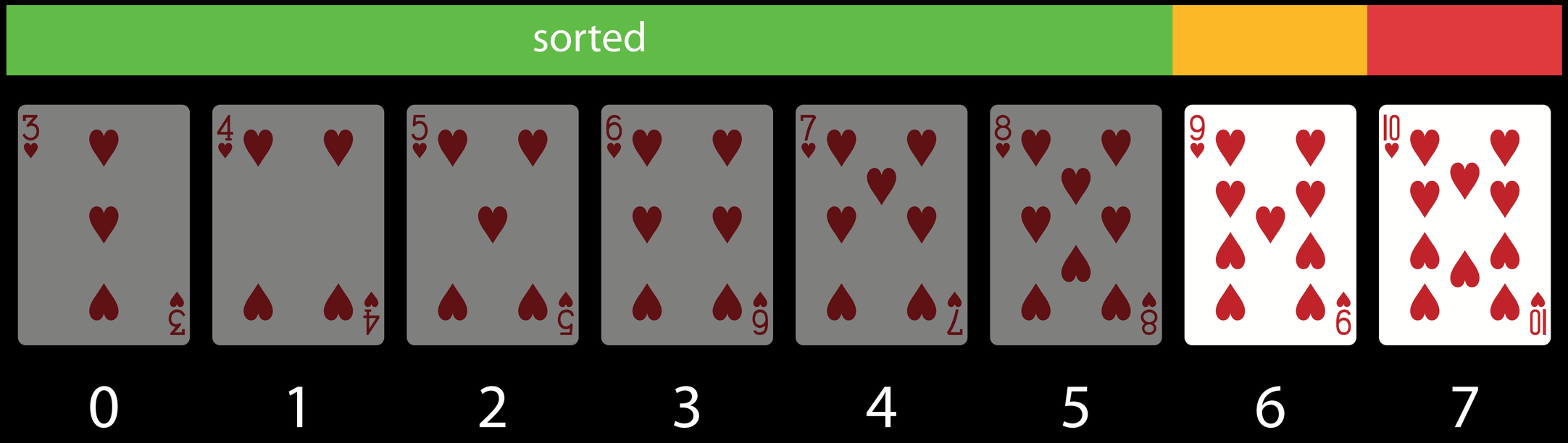


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

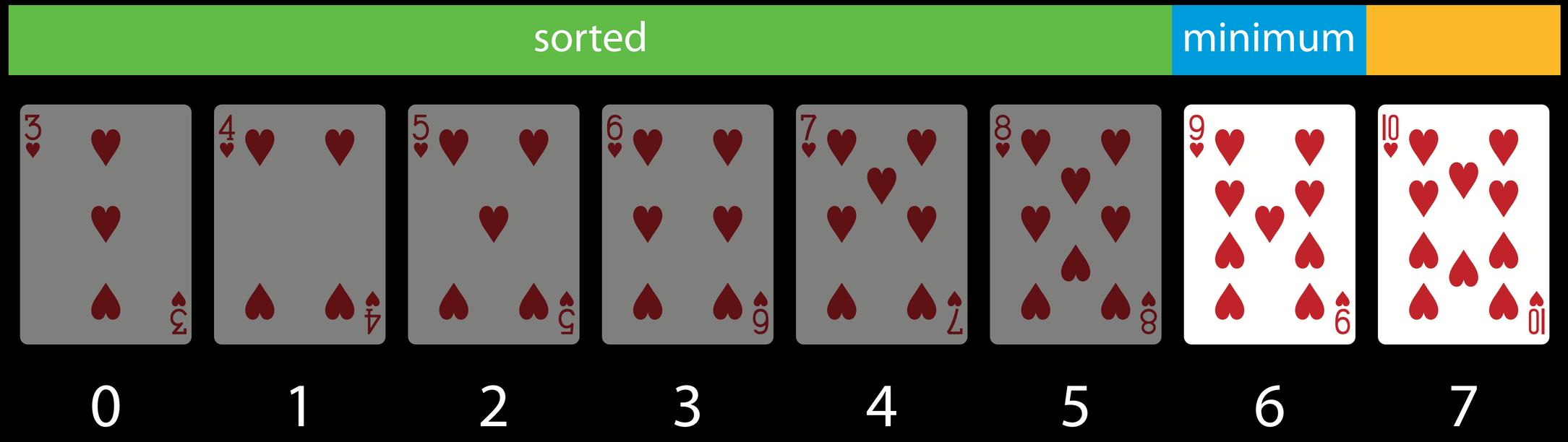


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

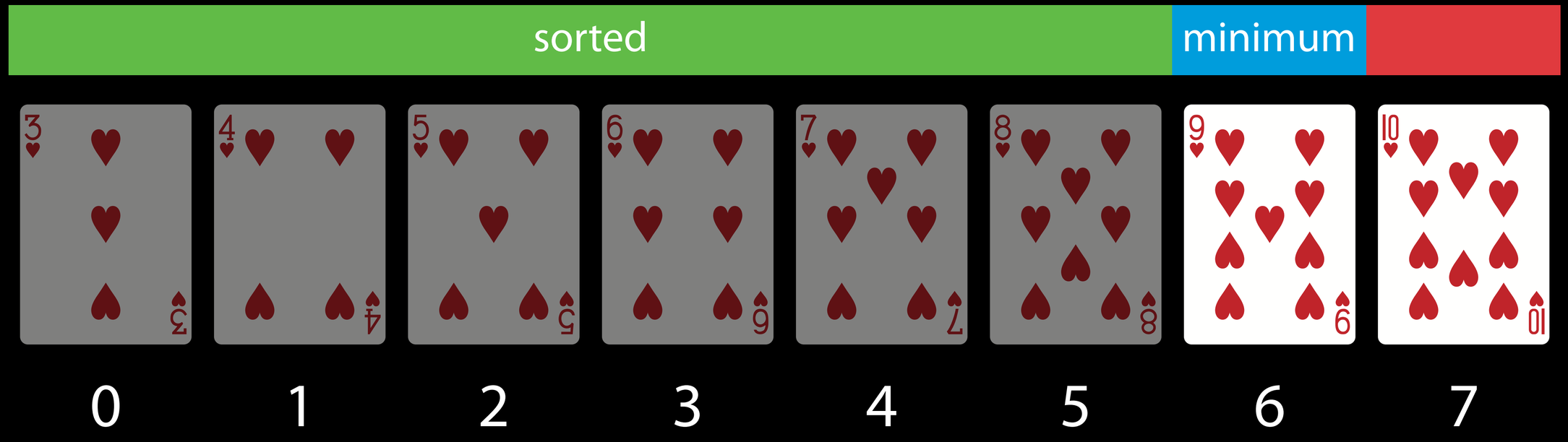


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

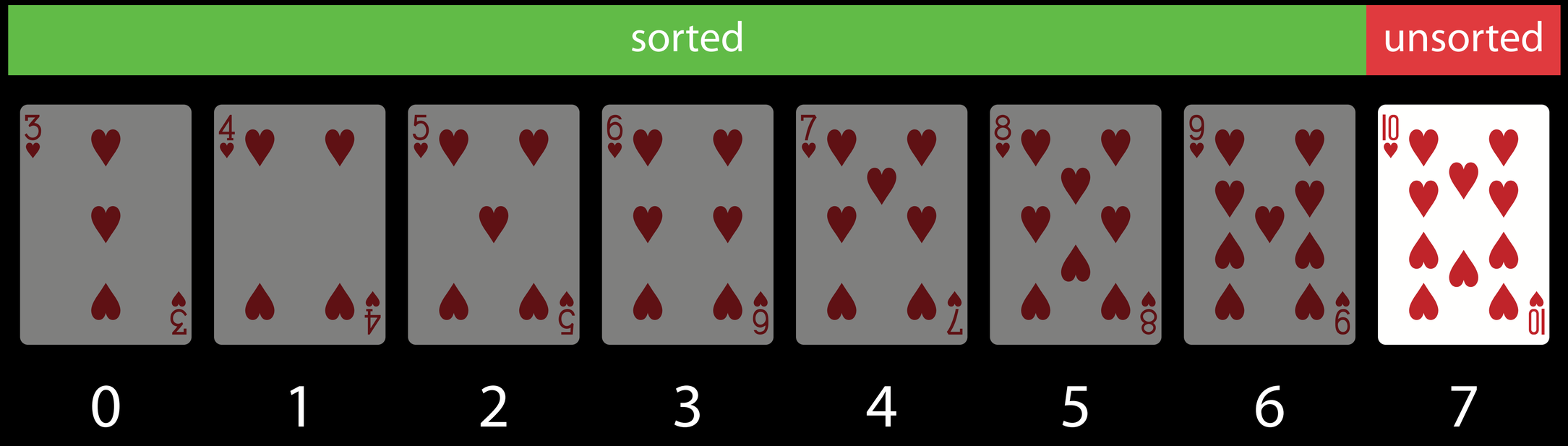


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

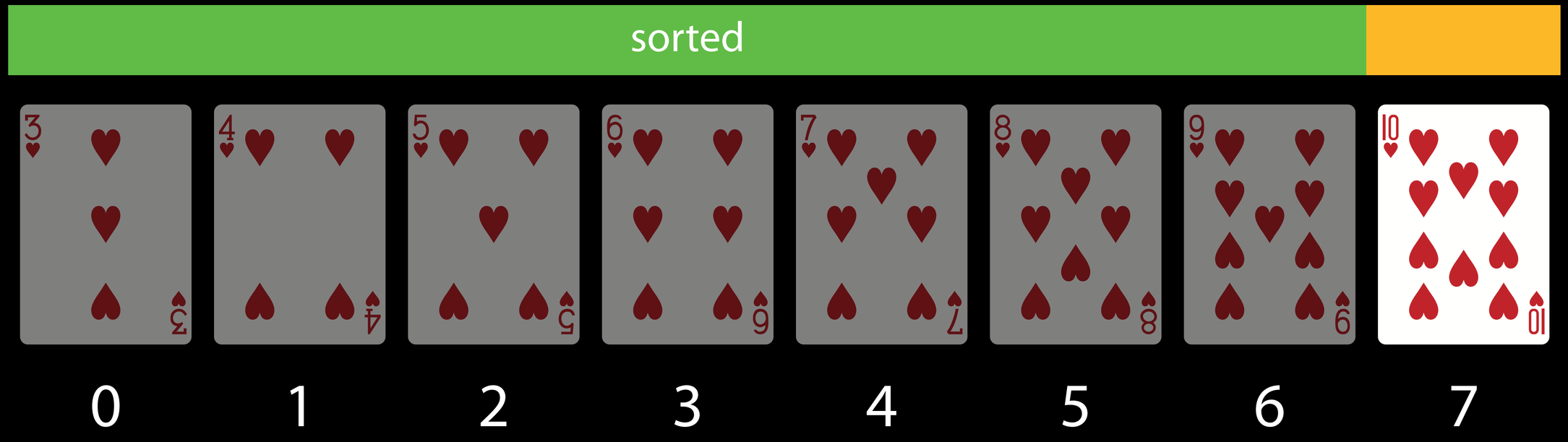


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

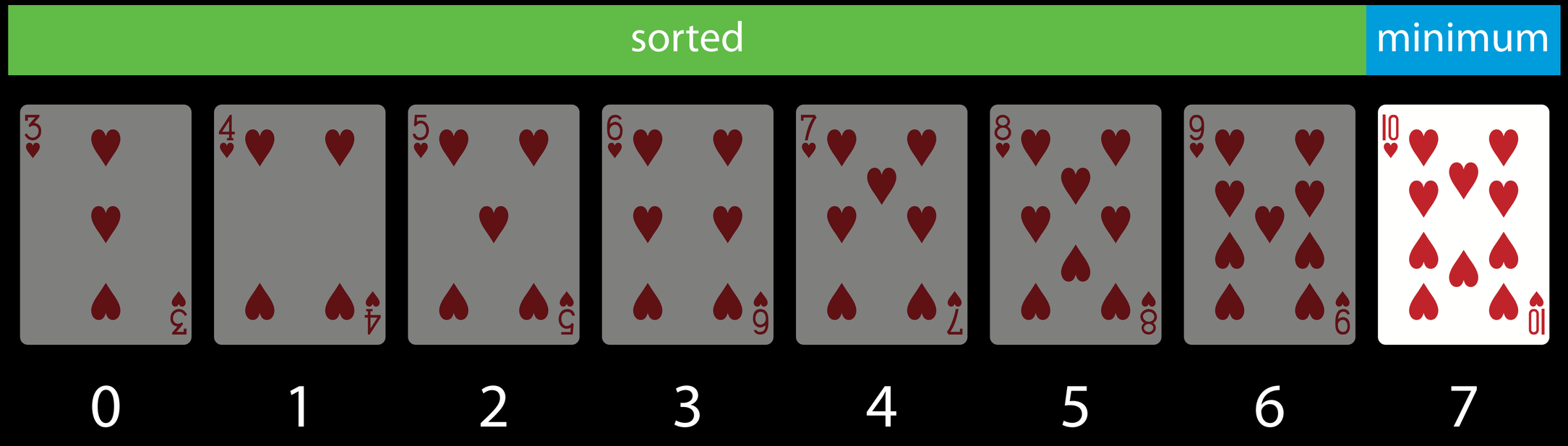


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

Repeat for all unsorted items until all items are in the sorted part.

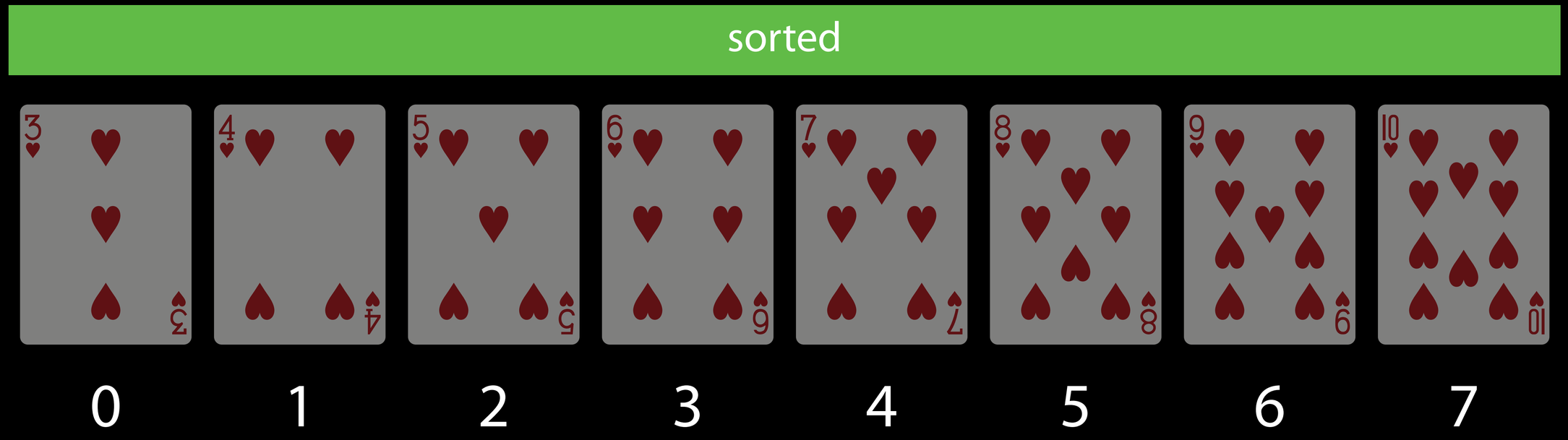


Selection sort

Select the smallest item in the unsorted part.

Swap it to the front of the unsorted part to put it in correct position.

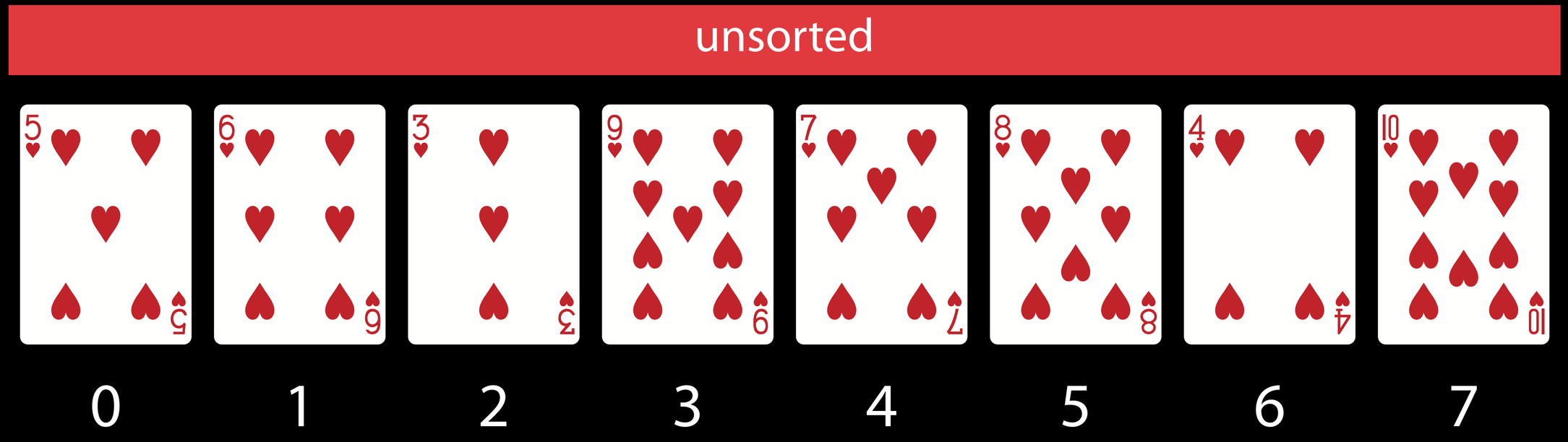
Repeat for all unsorted items until all items are in the sorted part.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

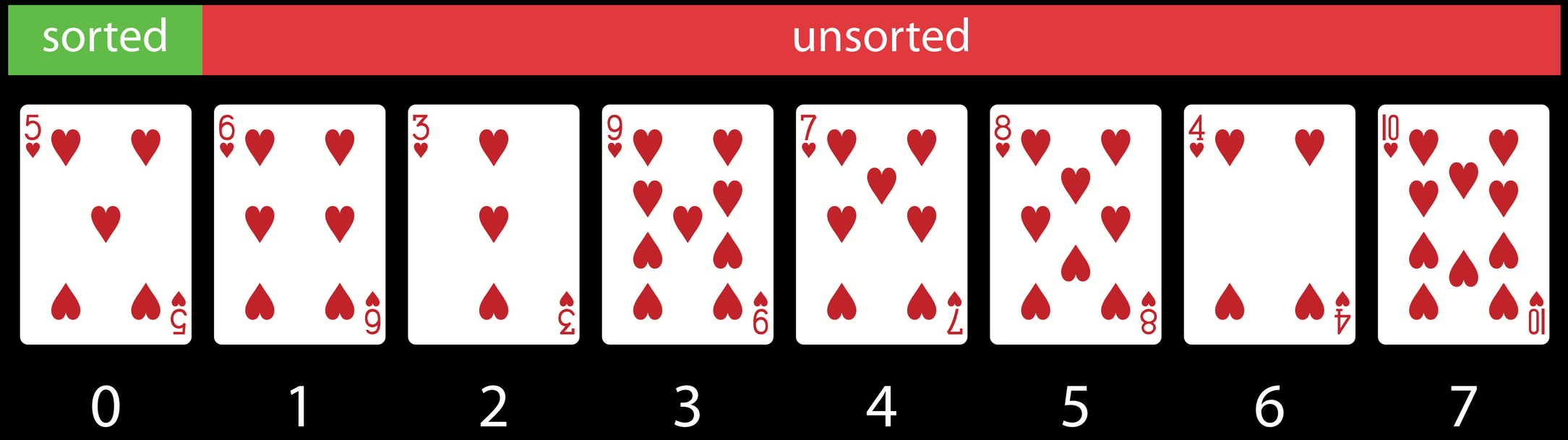
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

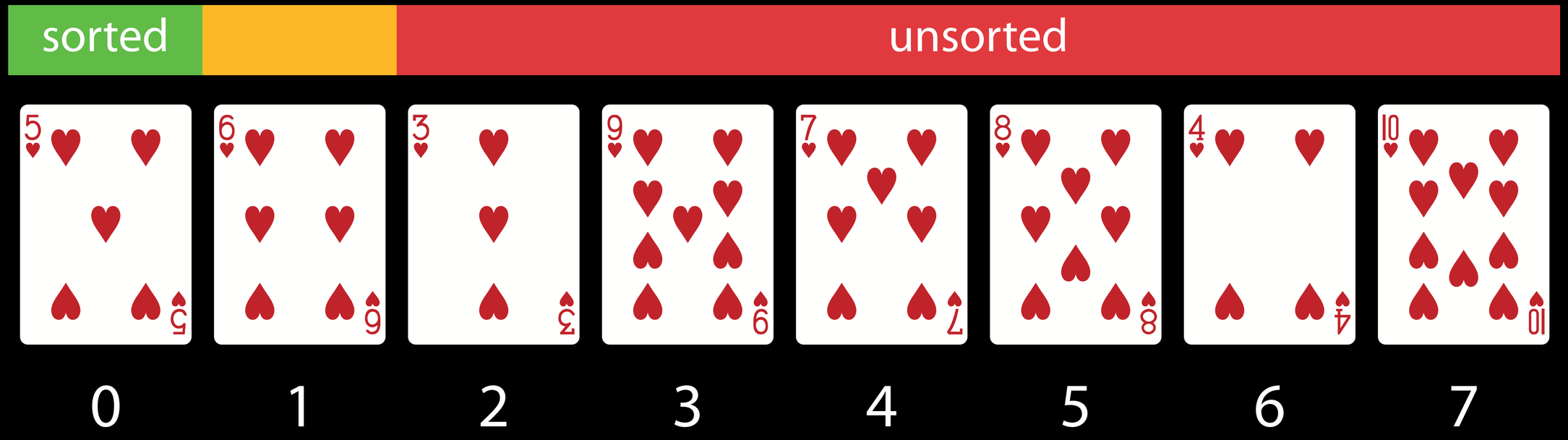
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

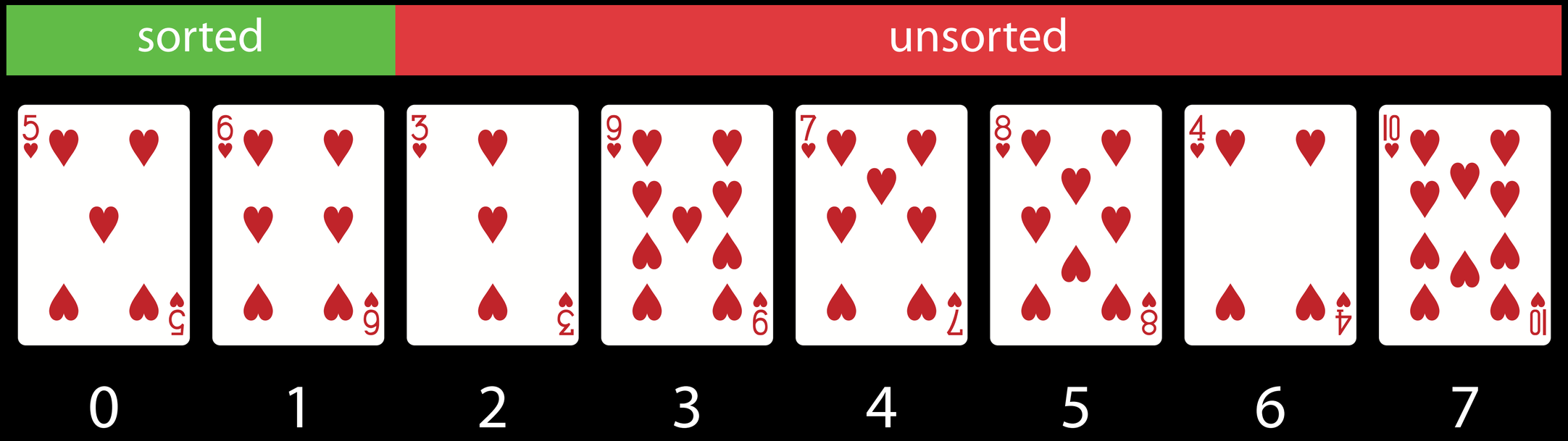
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

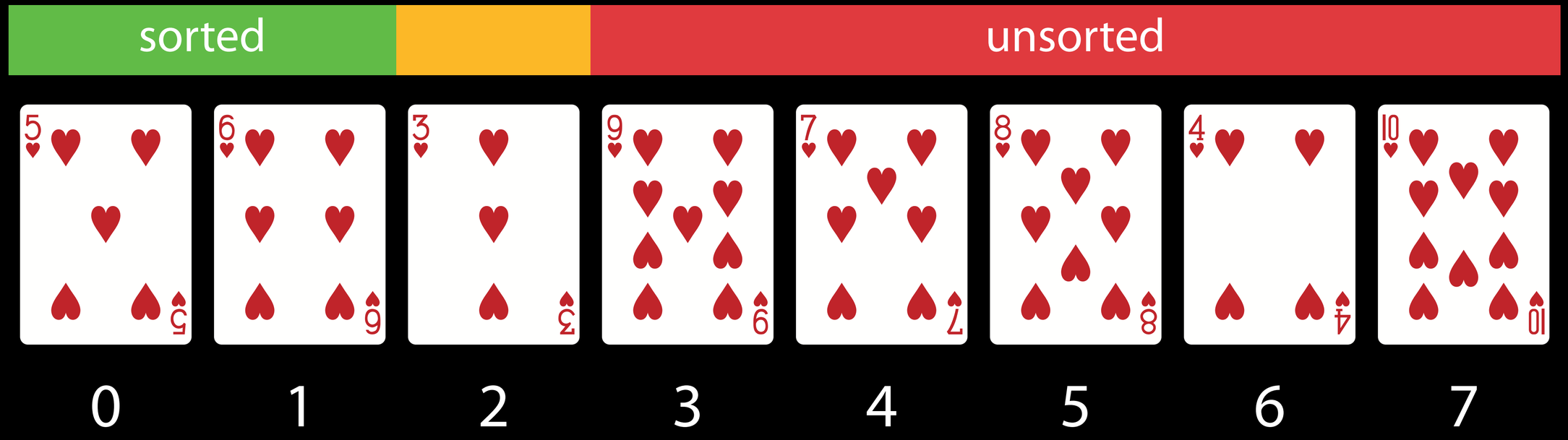
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

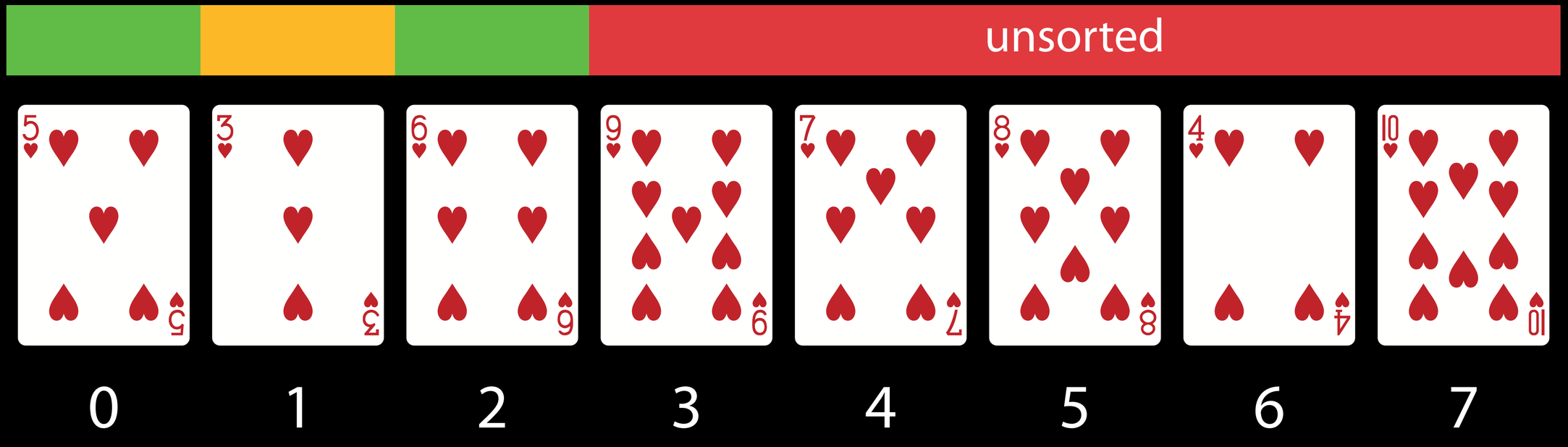
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

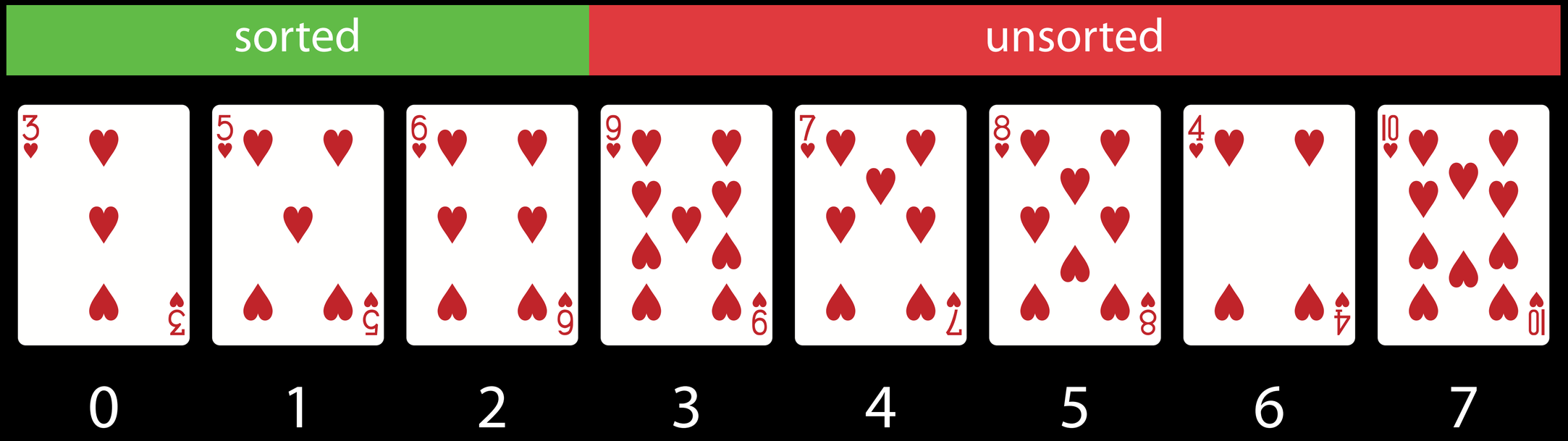
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

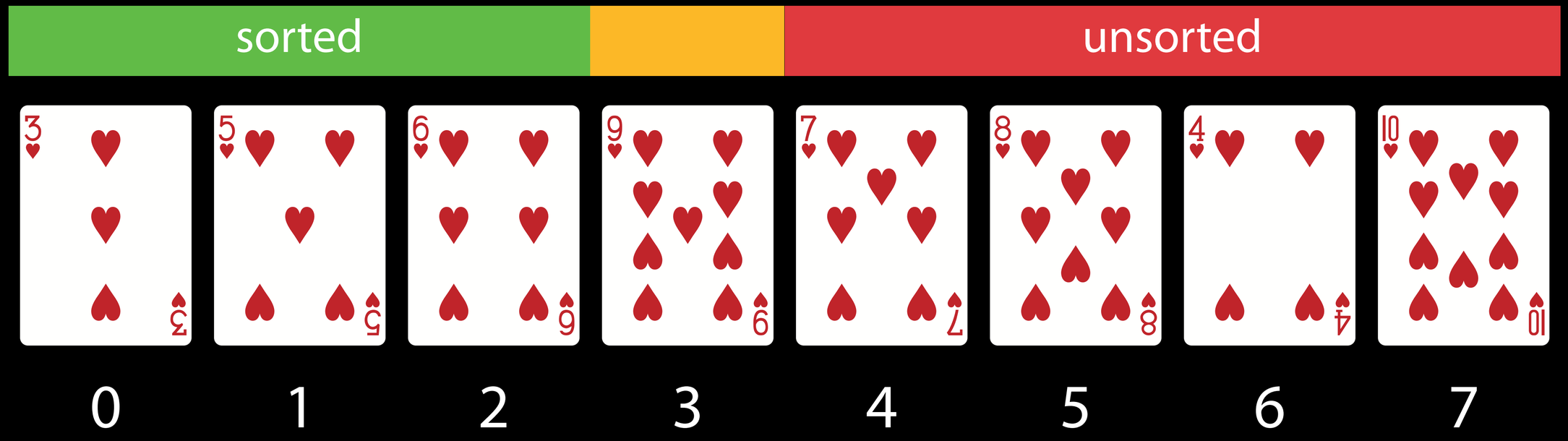
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

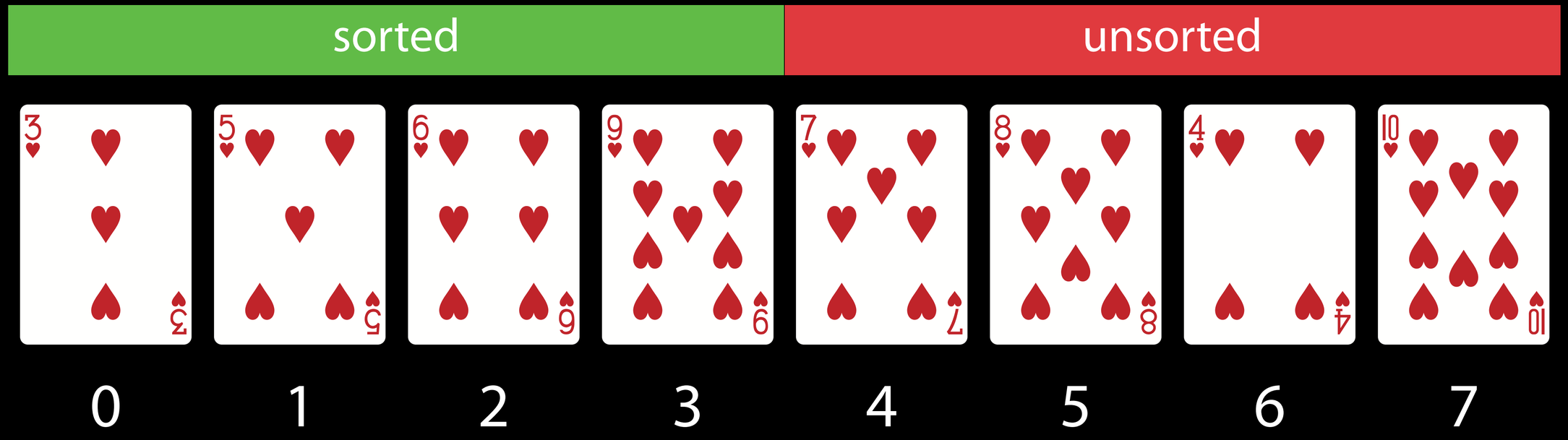
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

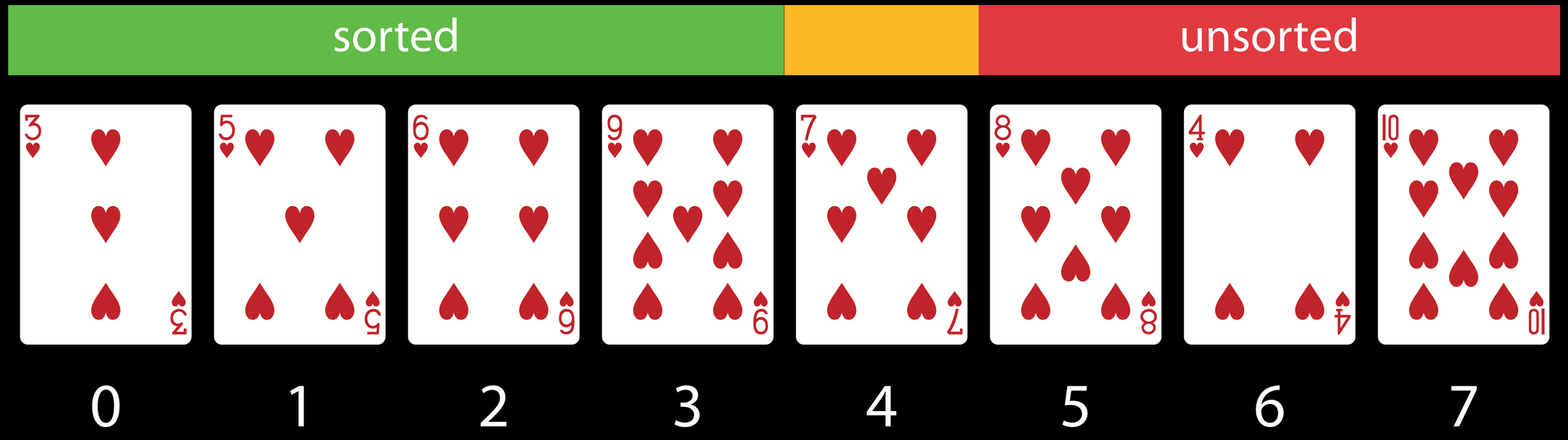
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

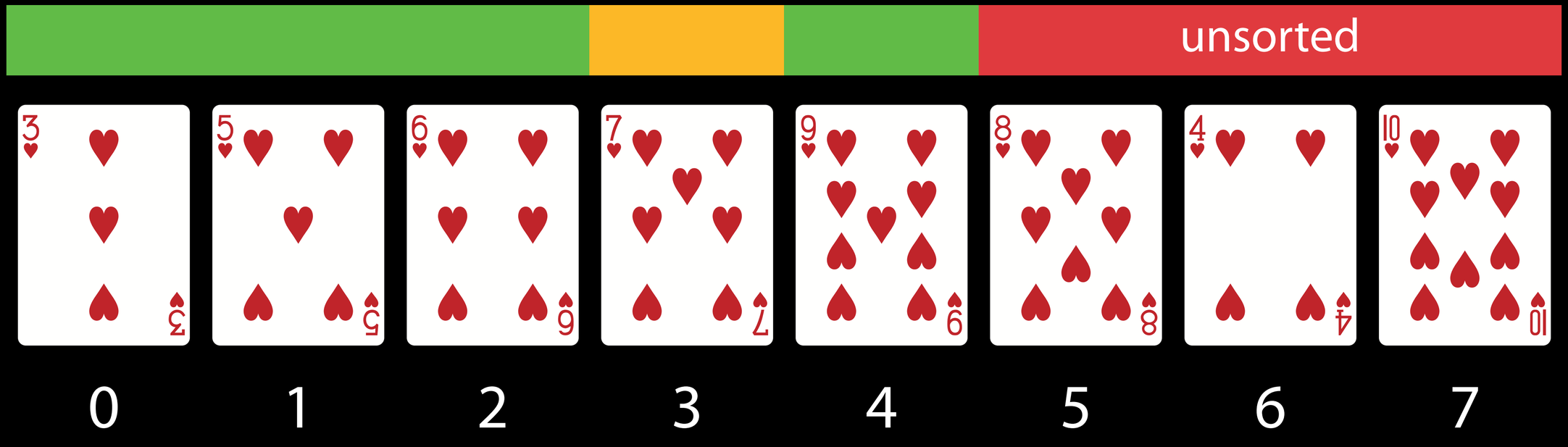
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

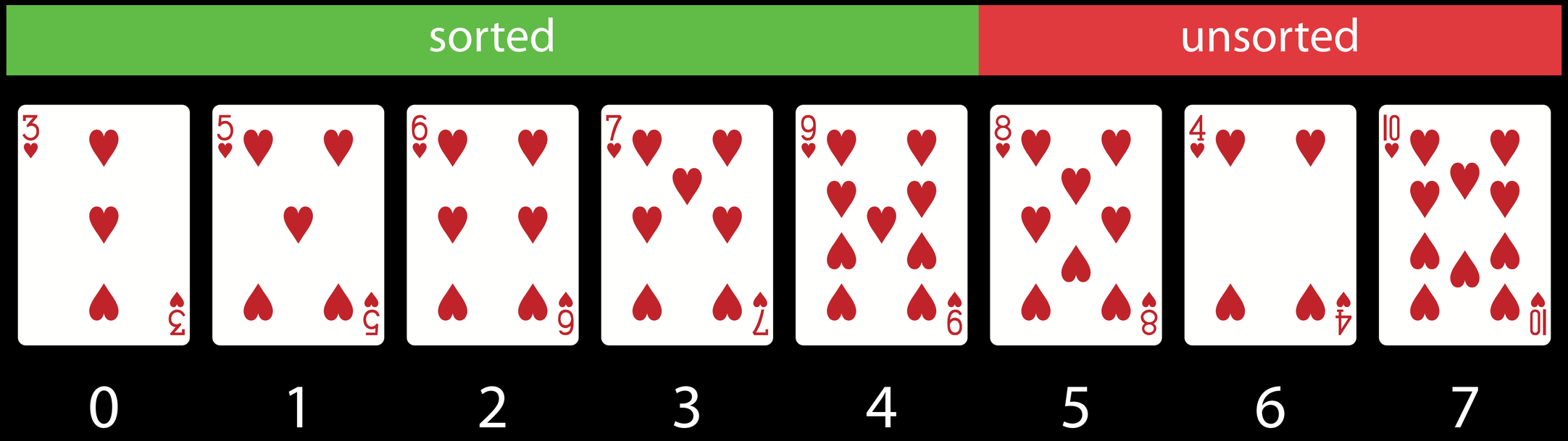
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

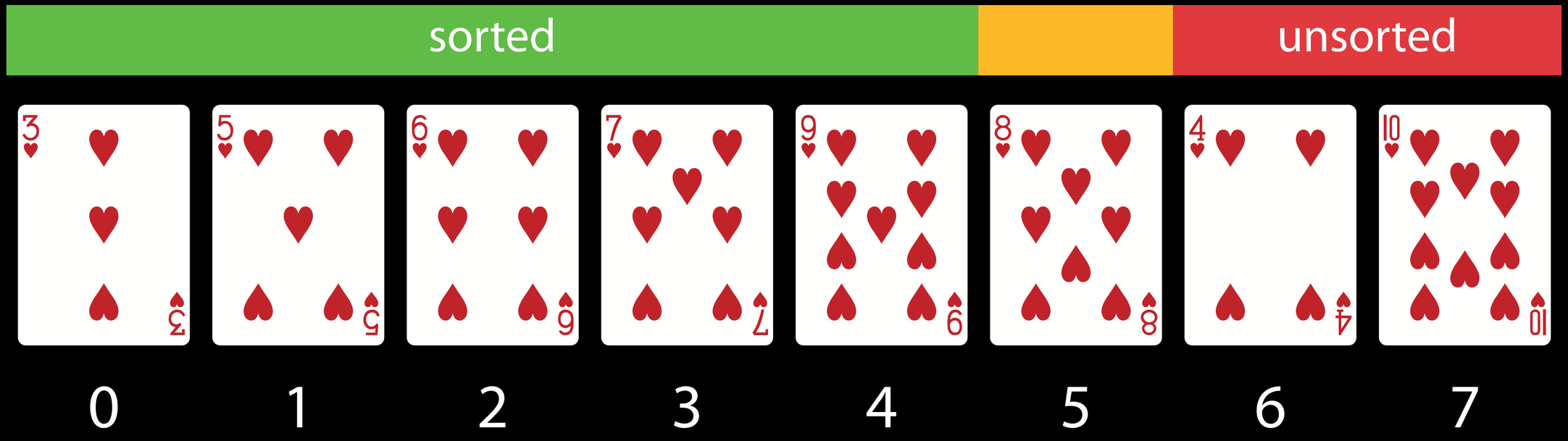
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

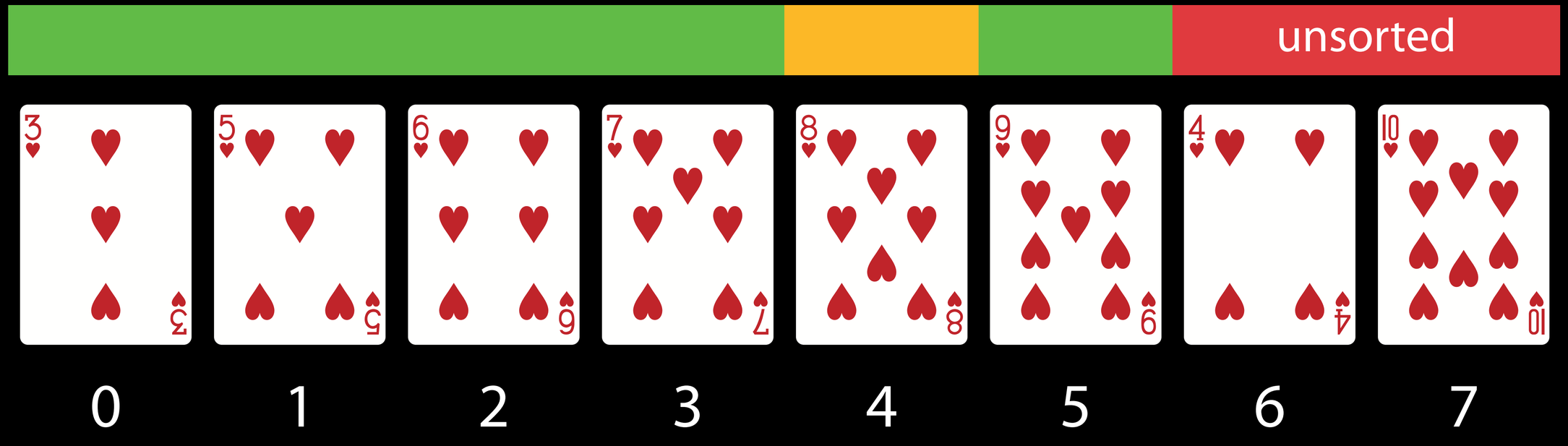
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

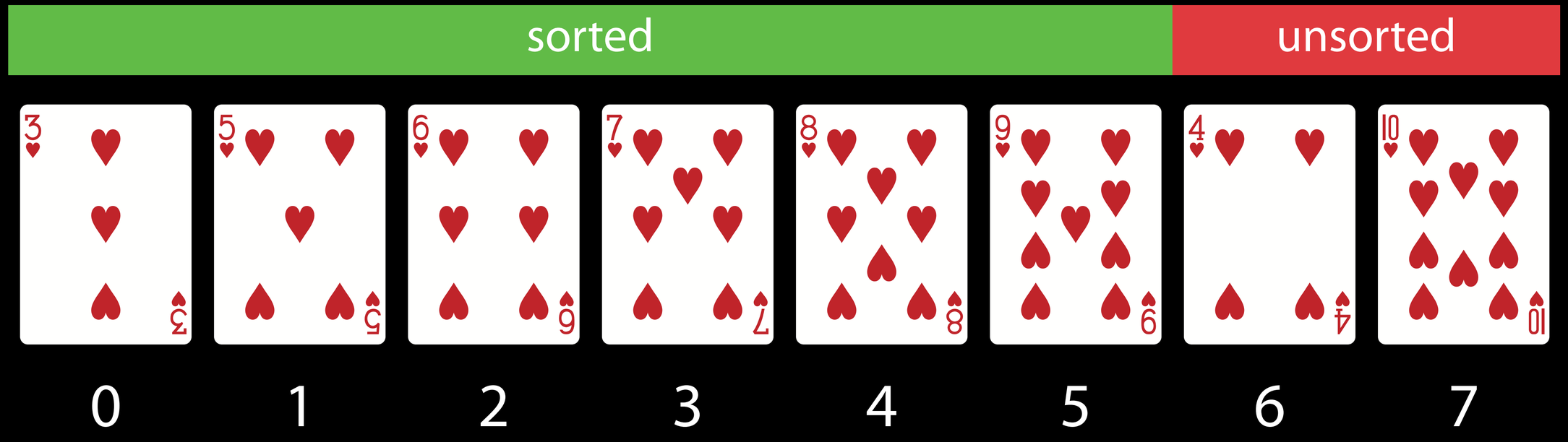
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

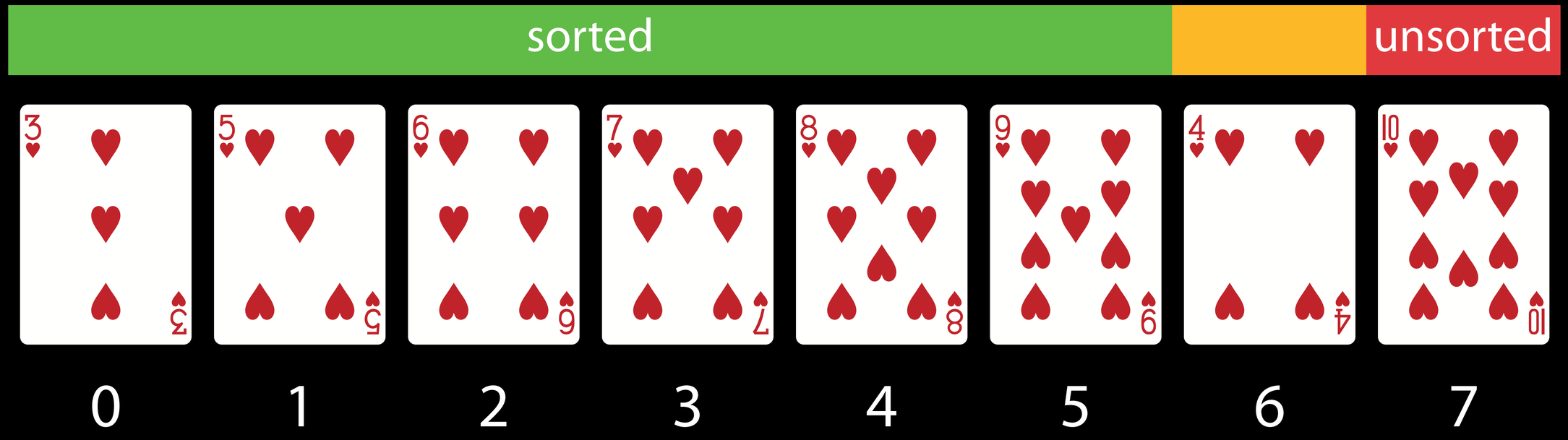
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

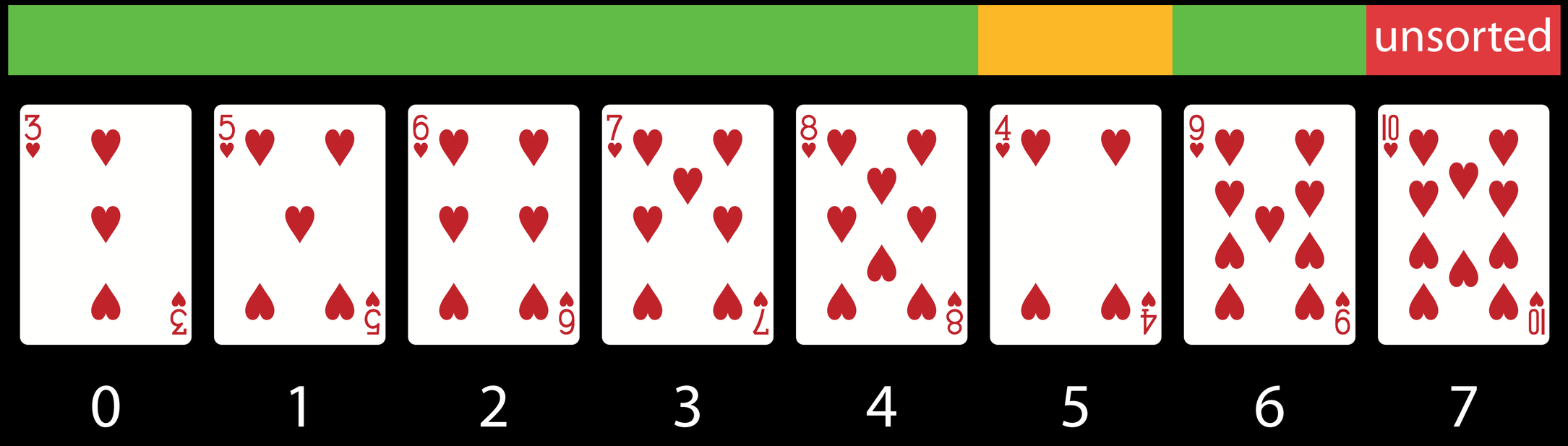
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

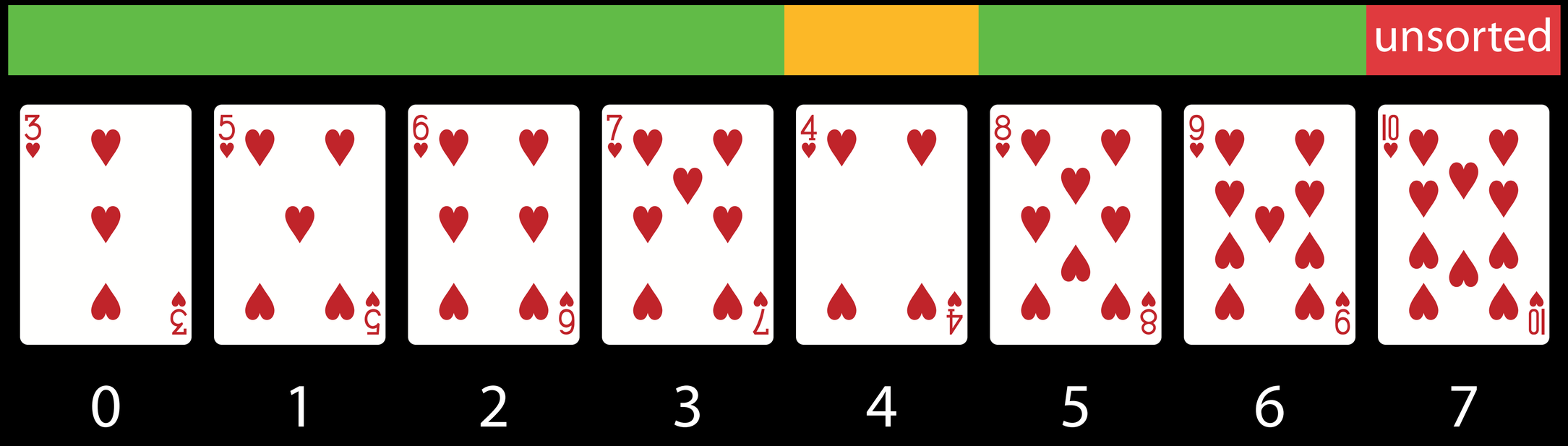
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

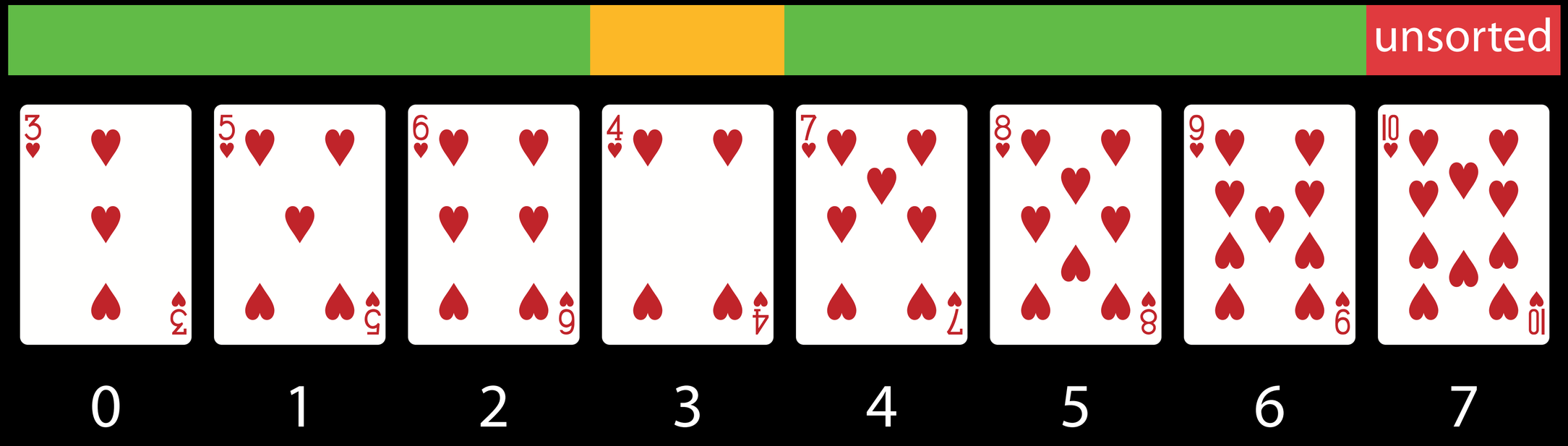
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

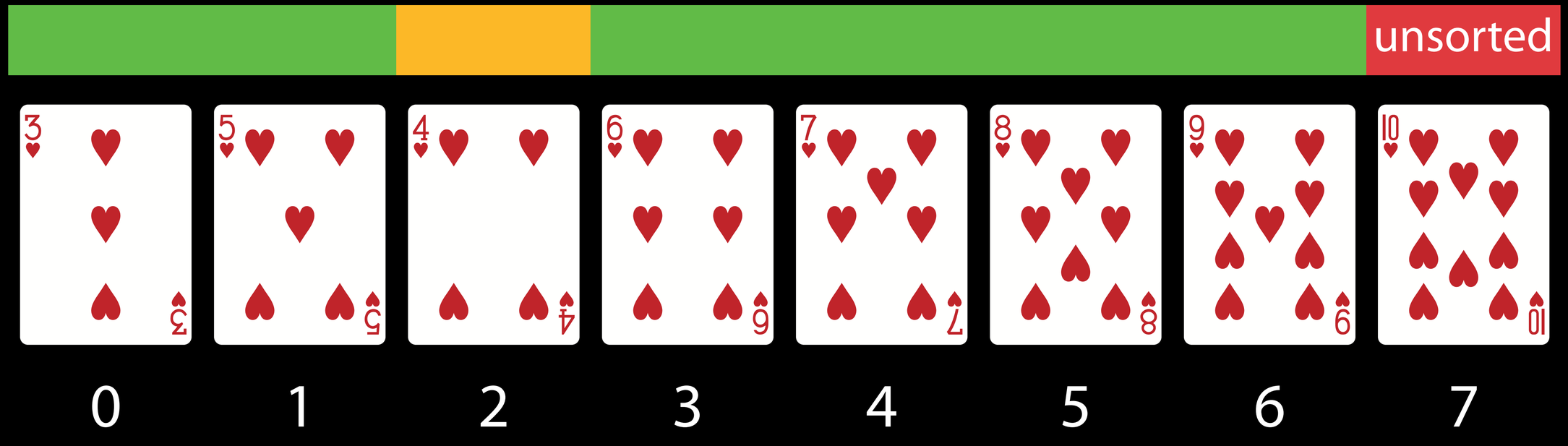
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

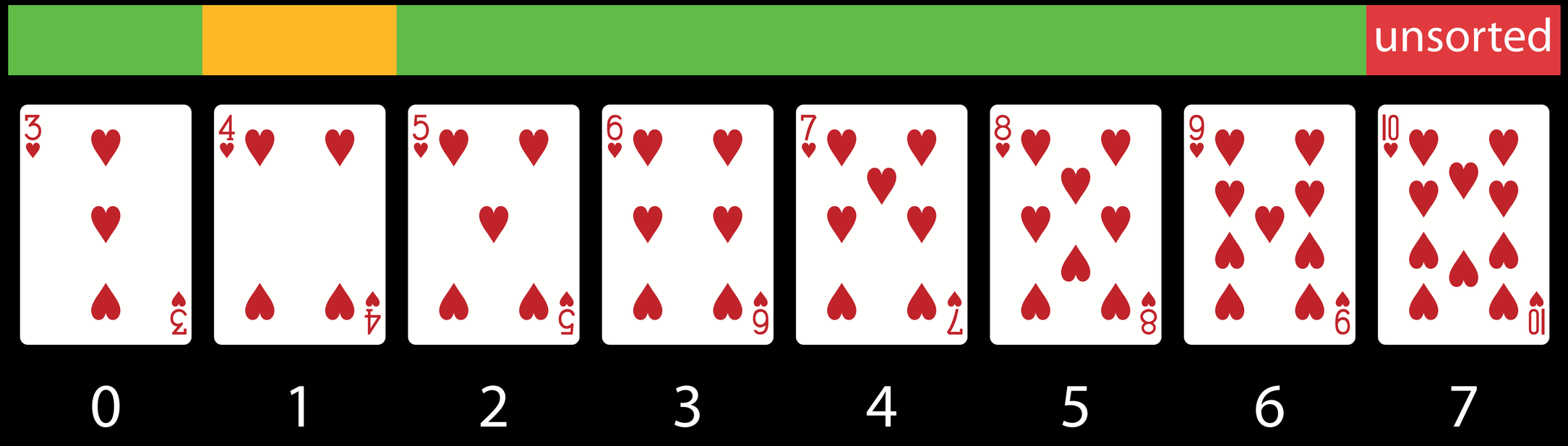
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

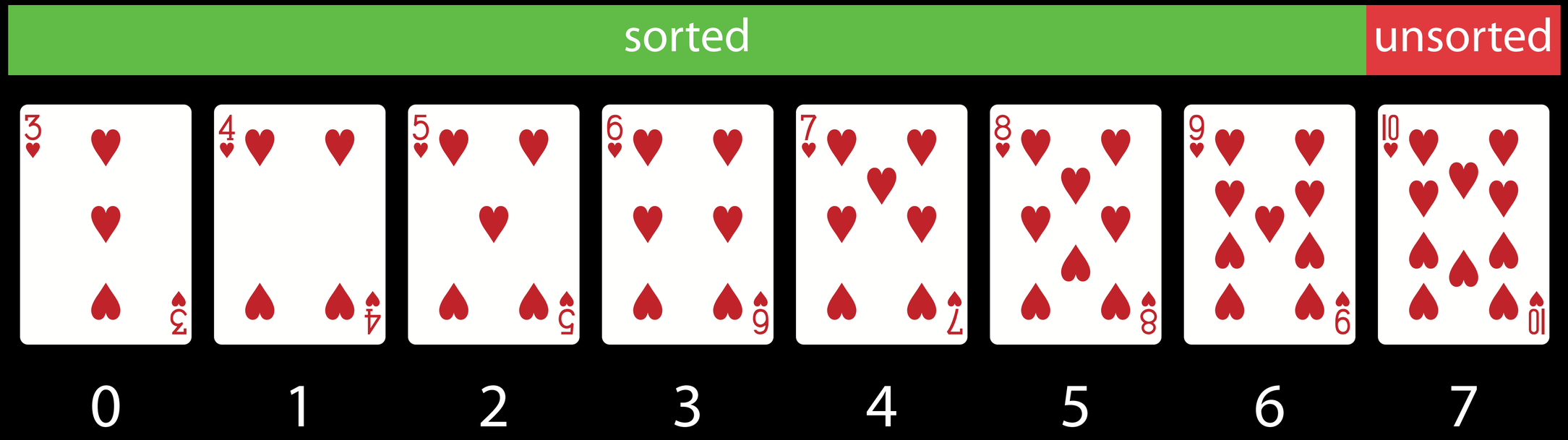
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

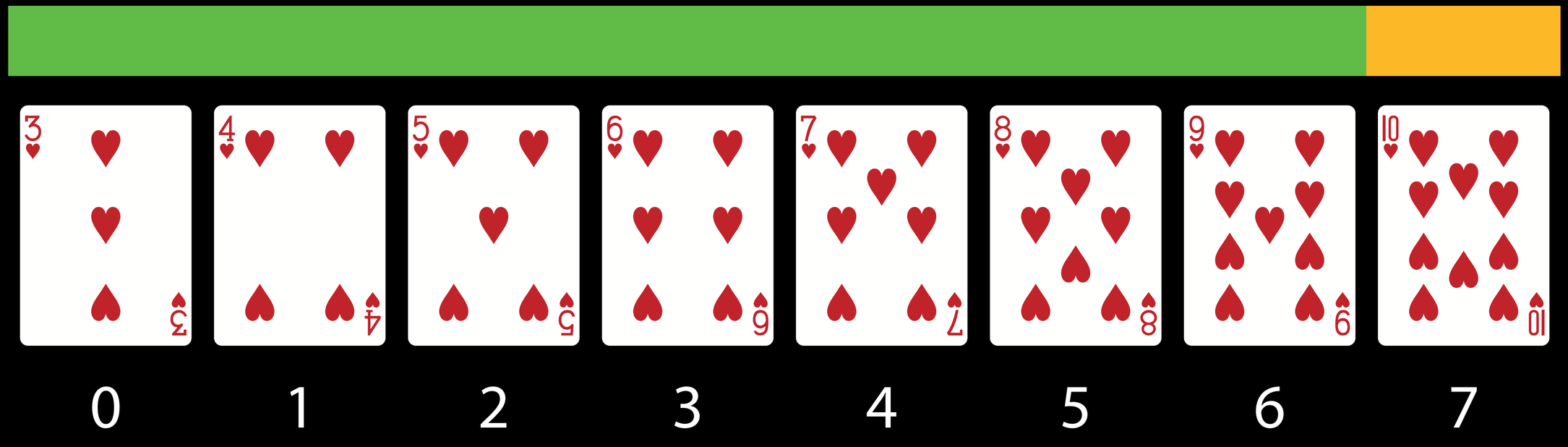
Do everything in place using swapping.



Insertion sort

Add each item from unsorted input, inserting into sorted output at the correct position.

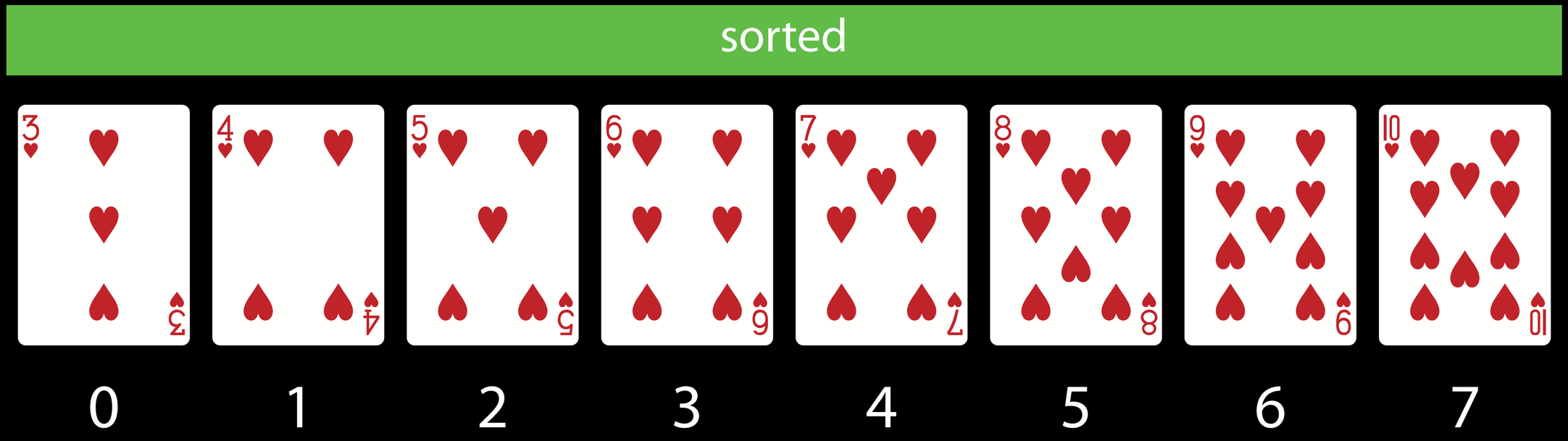
Do everything in place using swapping.



Insertion sort

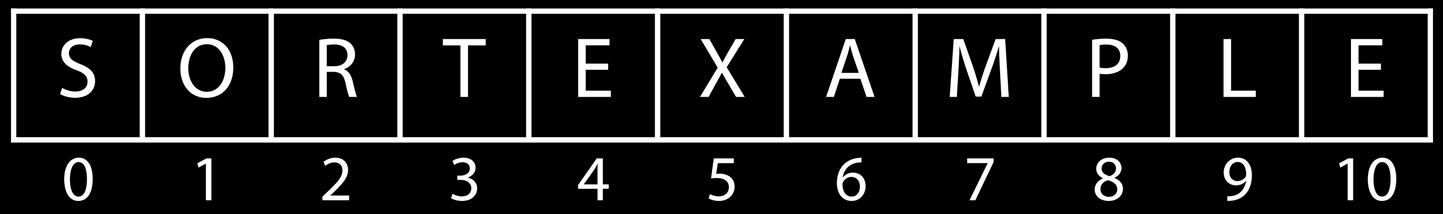
Add each item from unsorted input, inserting into sorted output at the correct position.

Do everything in place using swapping.



Heapsort

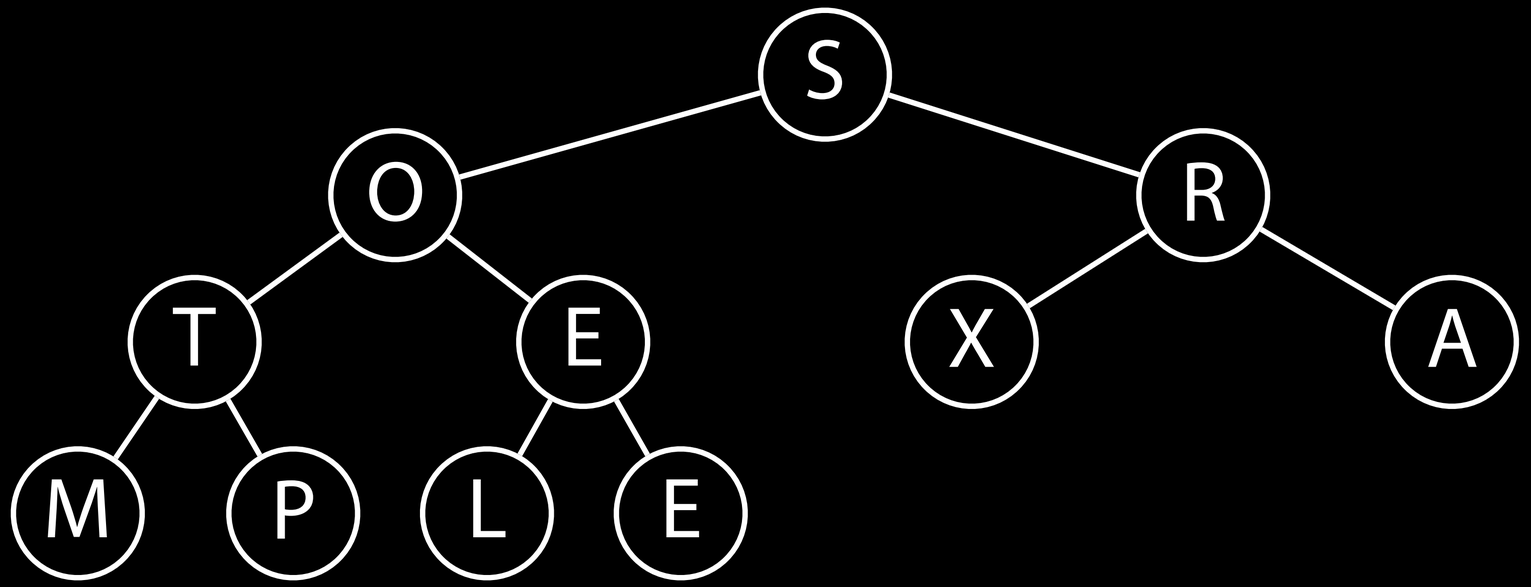
Build a max-heap using bottom-up method.



Heapsort

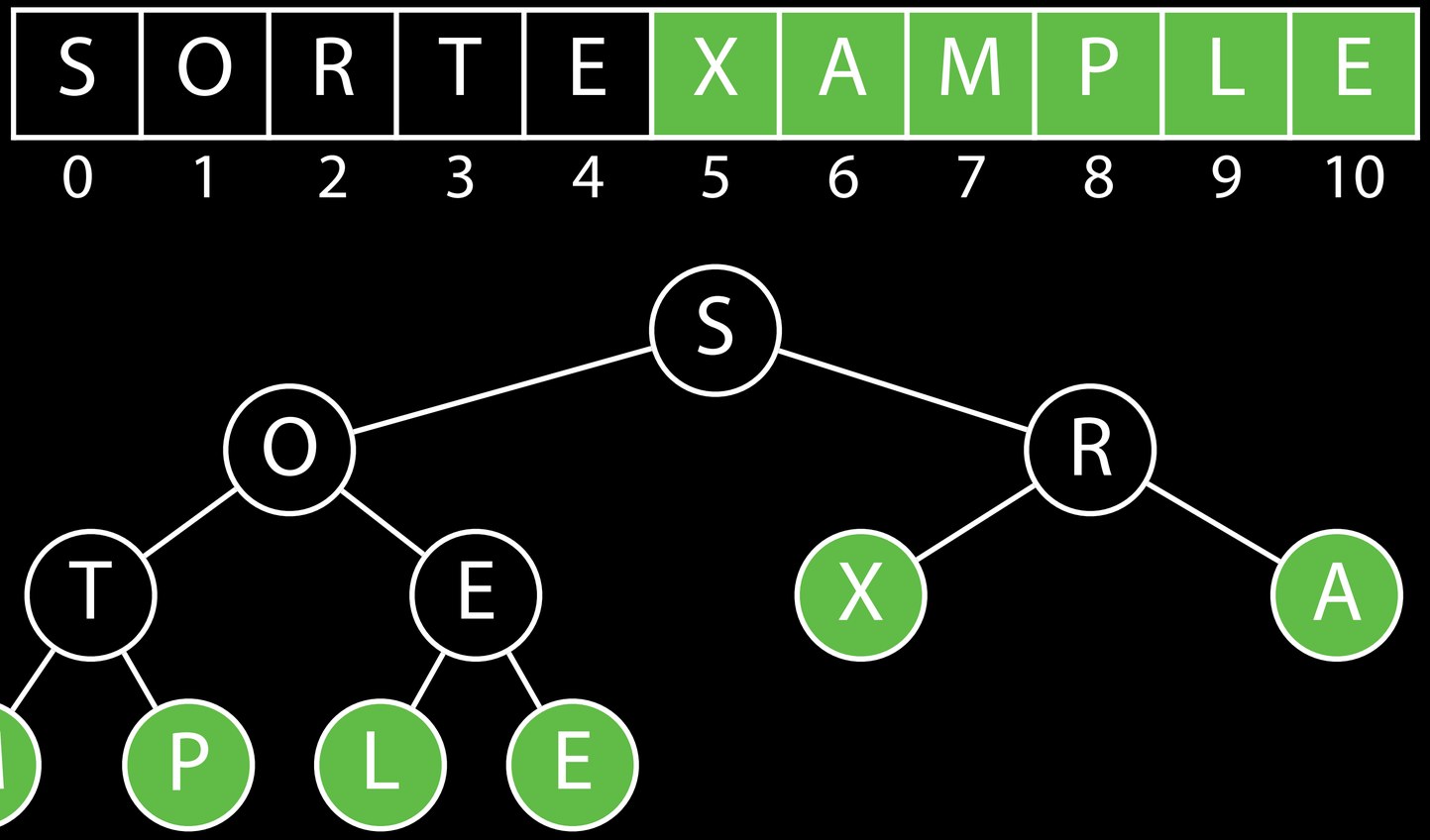
Build a max-heap using bottom-up method.

S	O	R	T	E	X	A	M	P	L	E
0	1	2	3	4	5	6	7	8	9	10



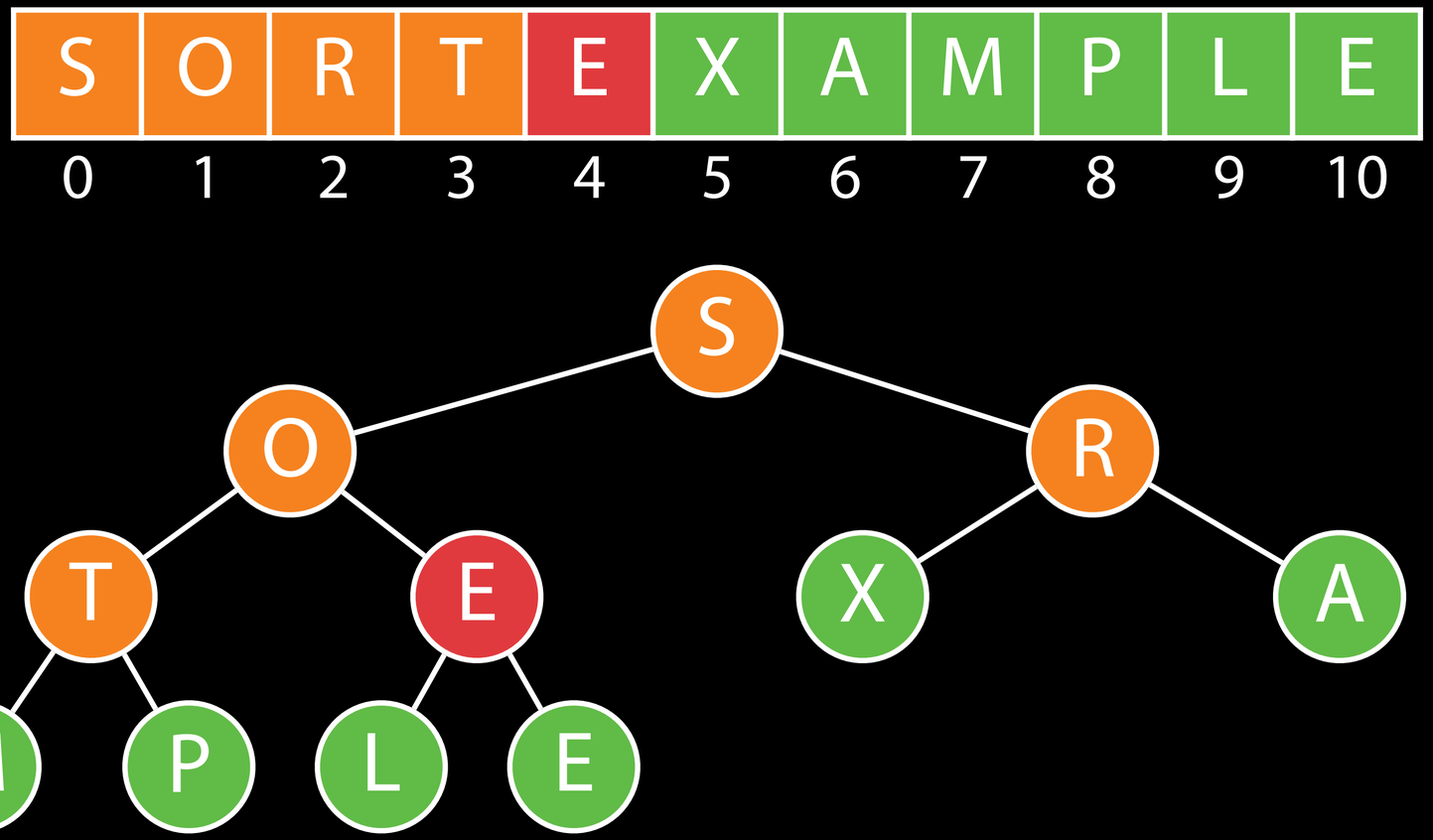
Heapsort

Build a max-heap using bottom-up method.



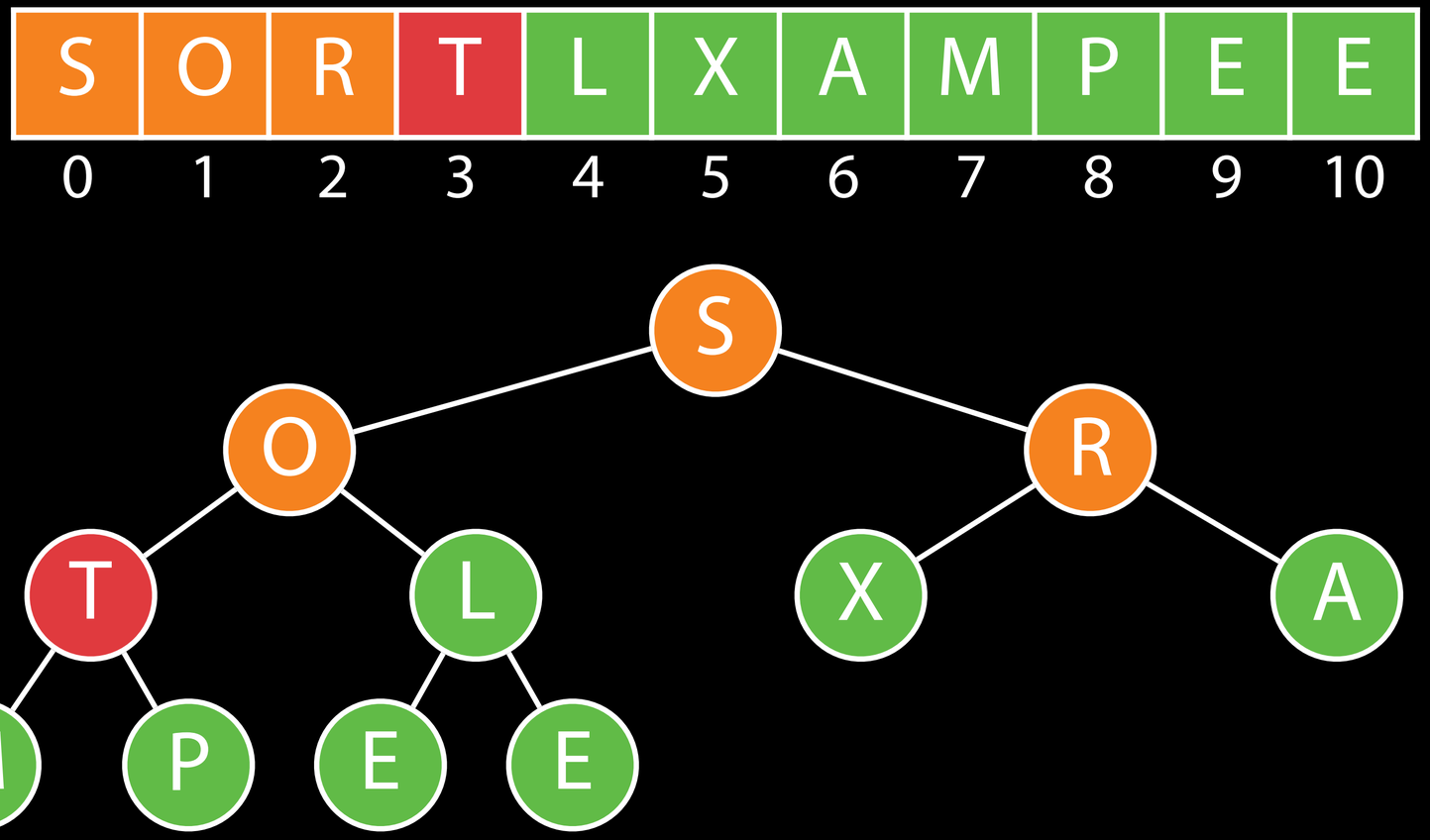
Heapsort

Build a max-heap using bottom-up method.



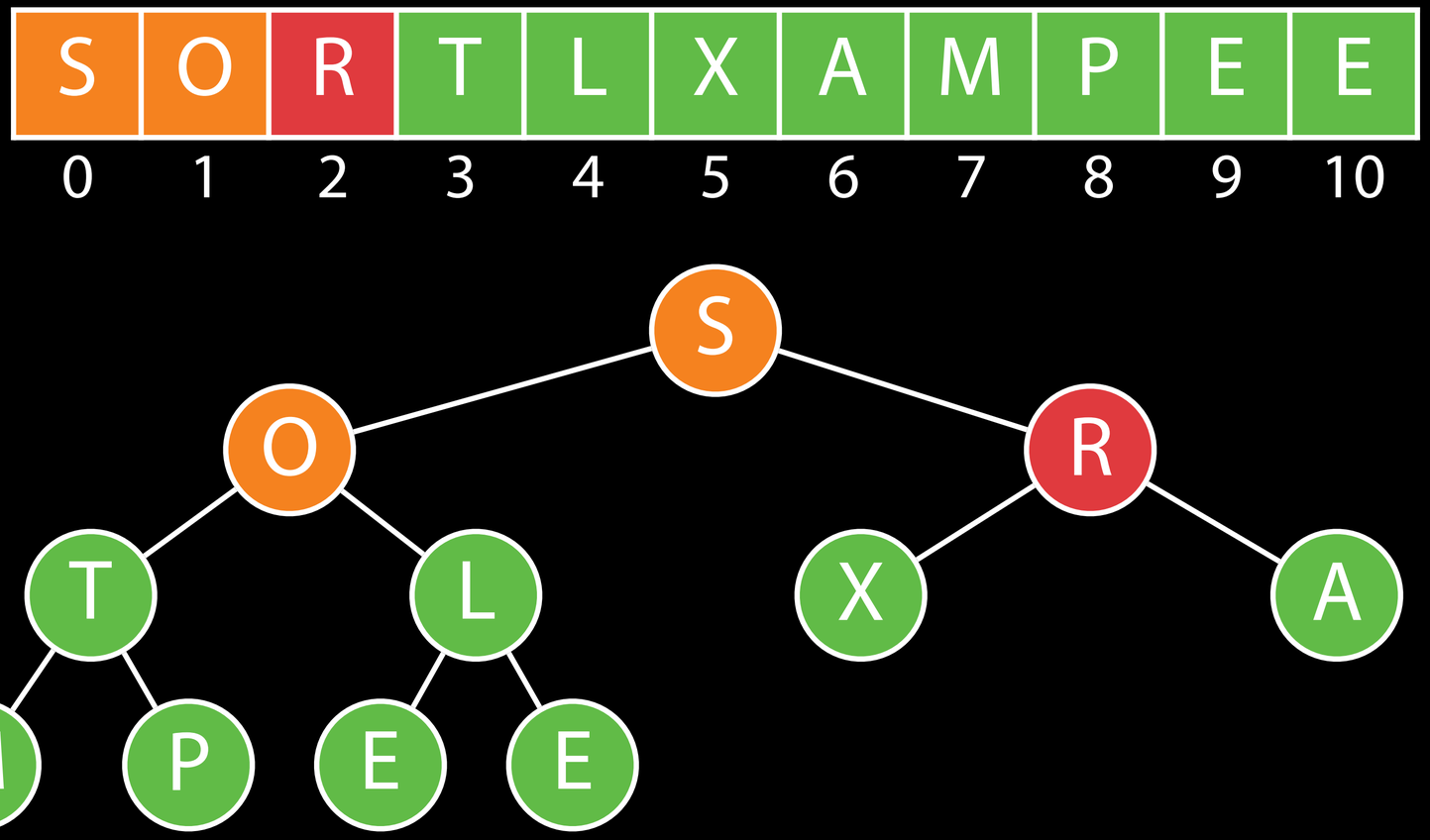
Heapsort

Build a max-heap using bottom-up method.



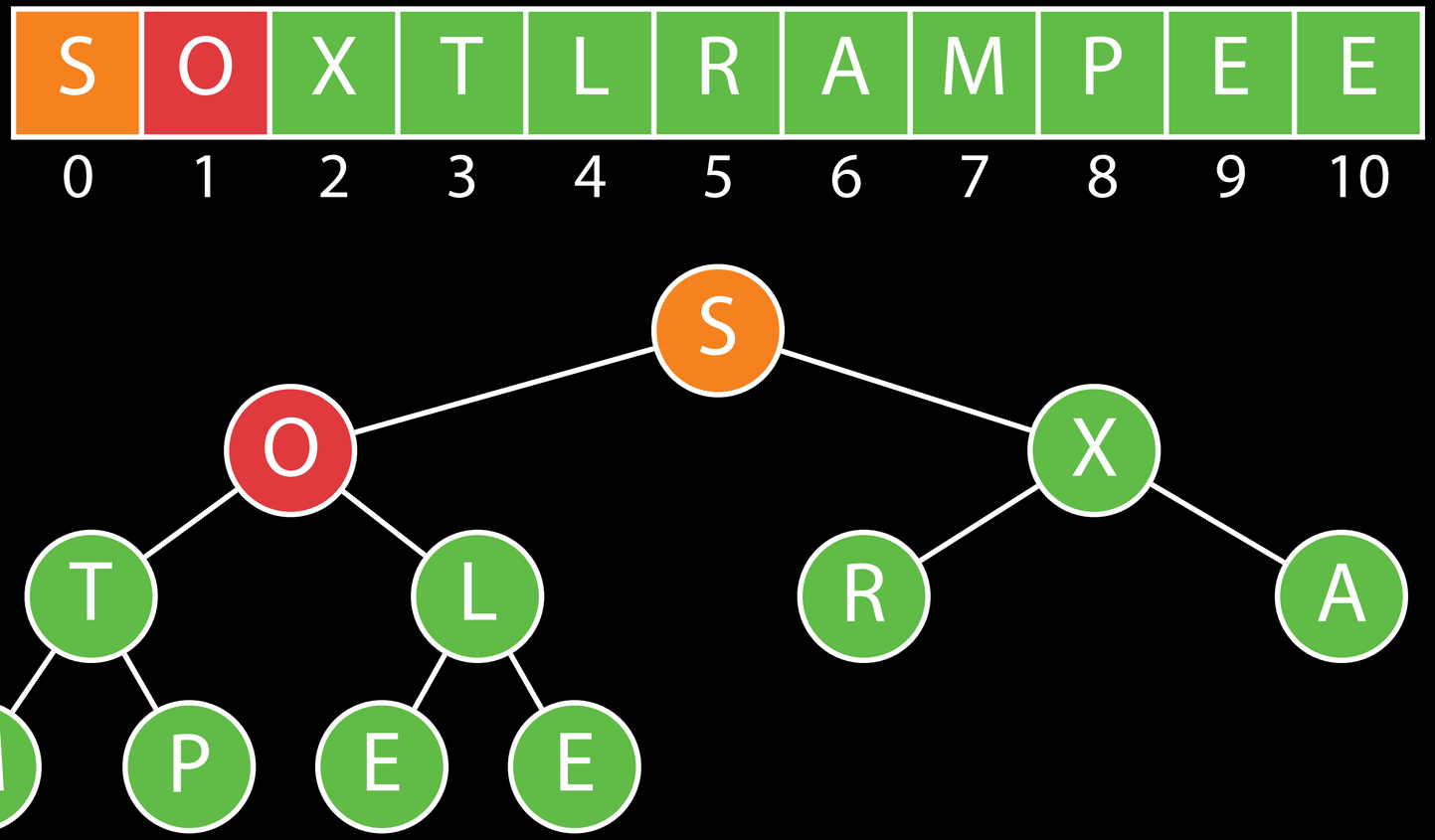
Heapsort

Build a max-heap using bottom-up method.



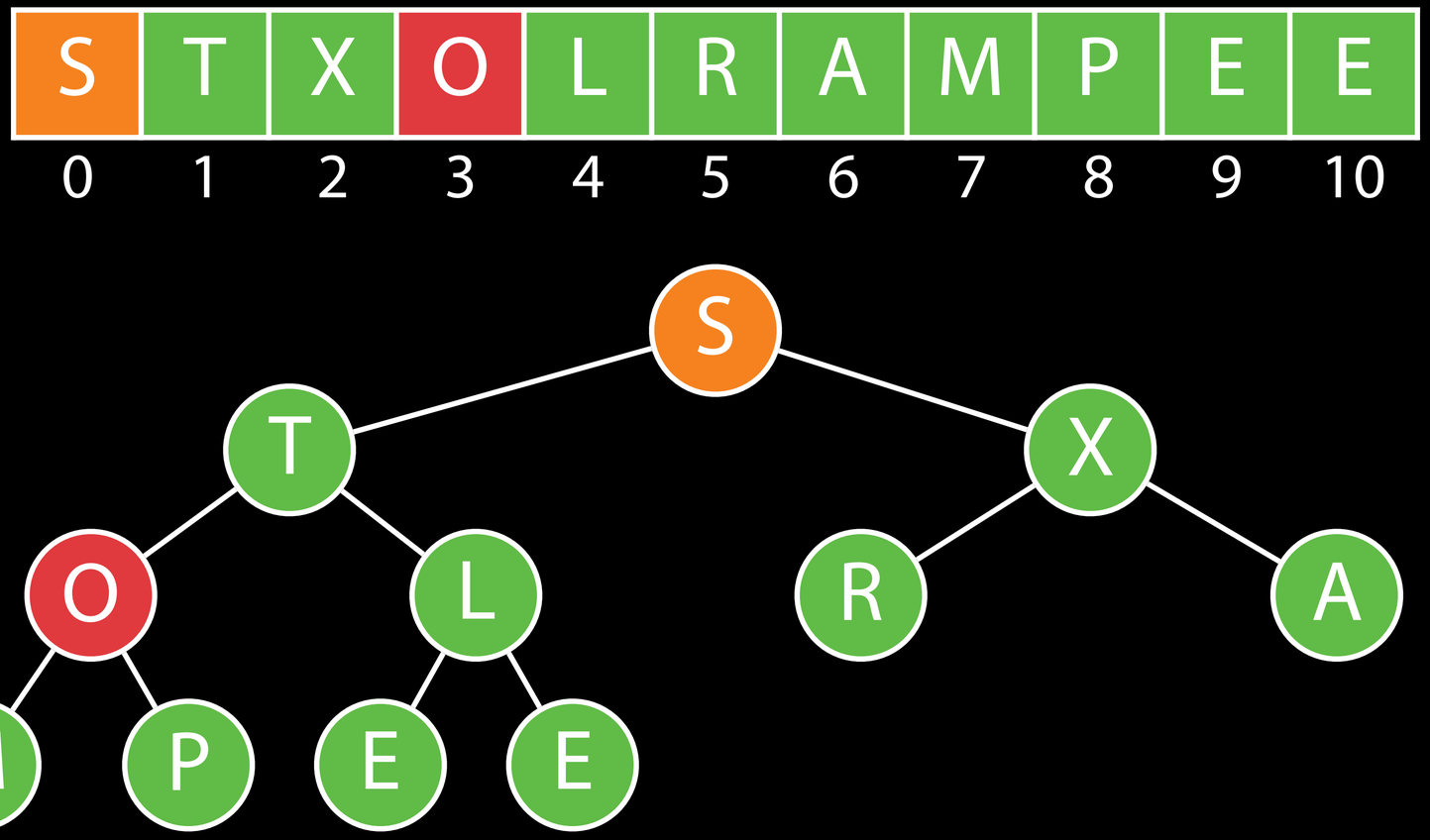
Heapsort

Build a max-heap using bottom-up method.



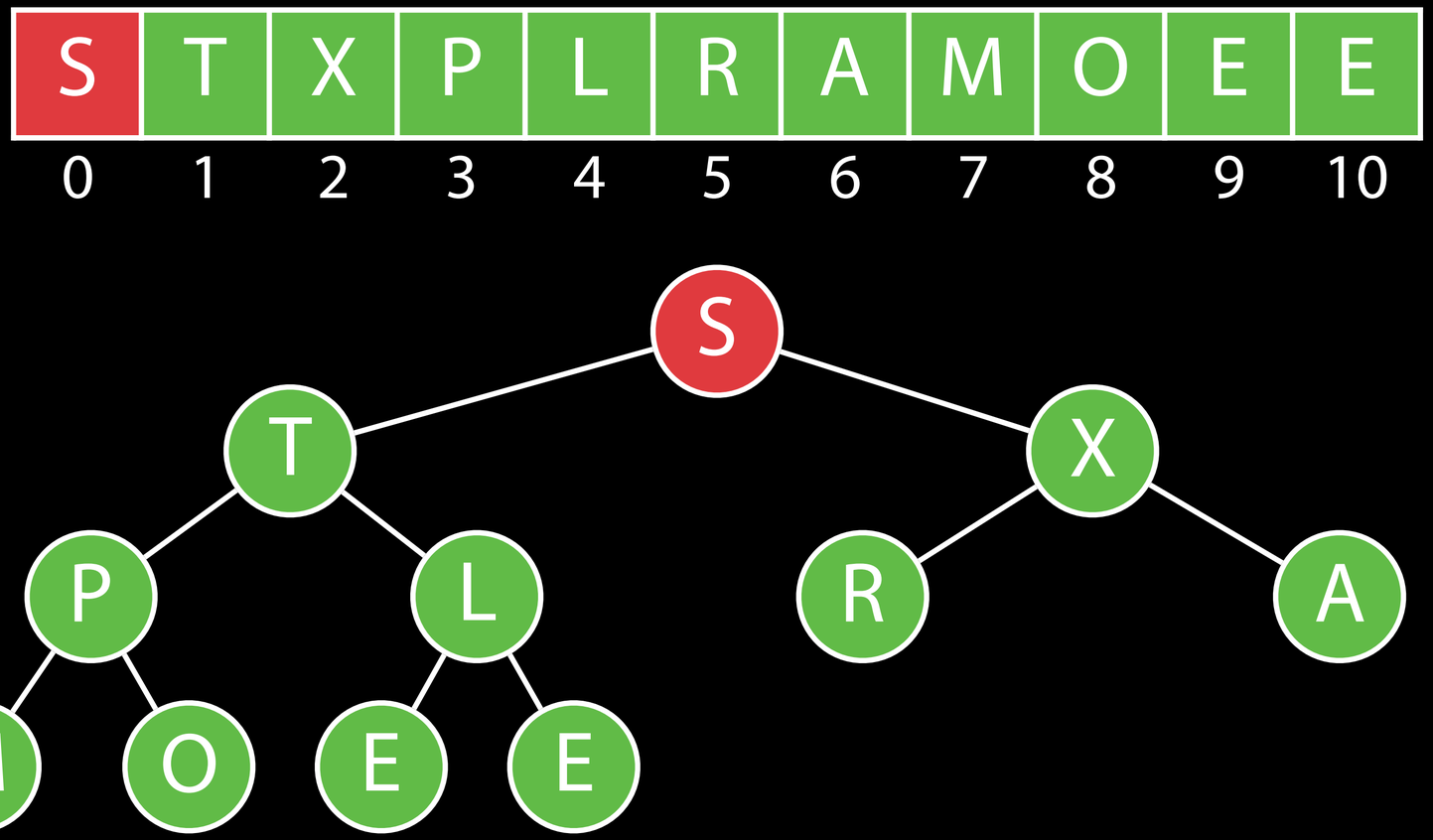
Heapsort

Build a max-heap using bottom-up method.



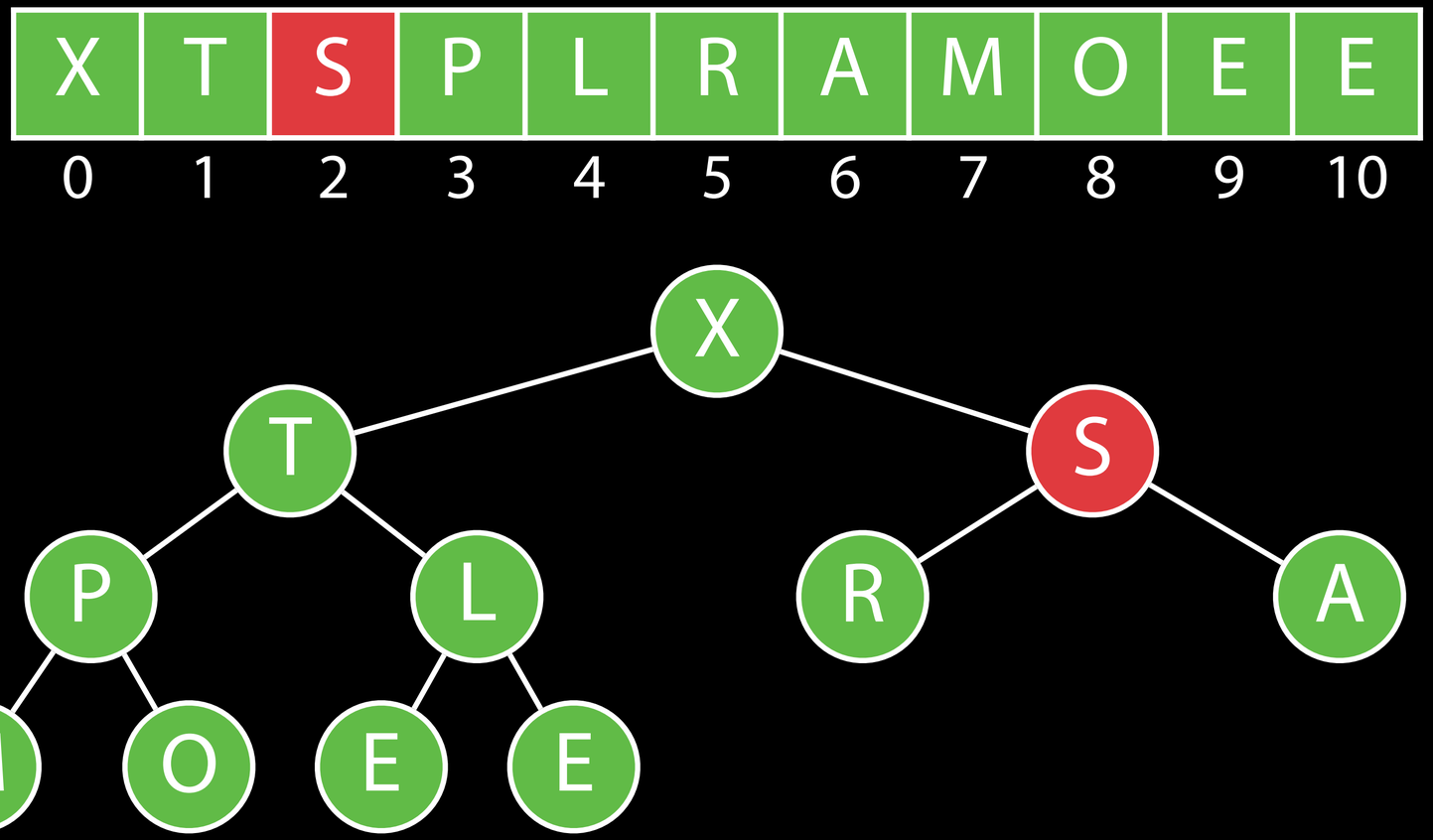
Heapsort

Build a max-heap using bottom-up method.



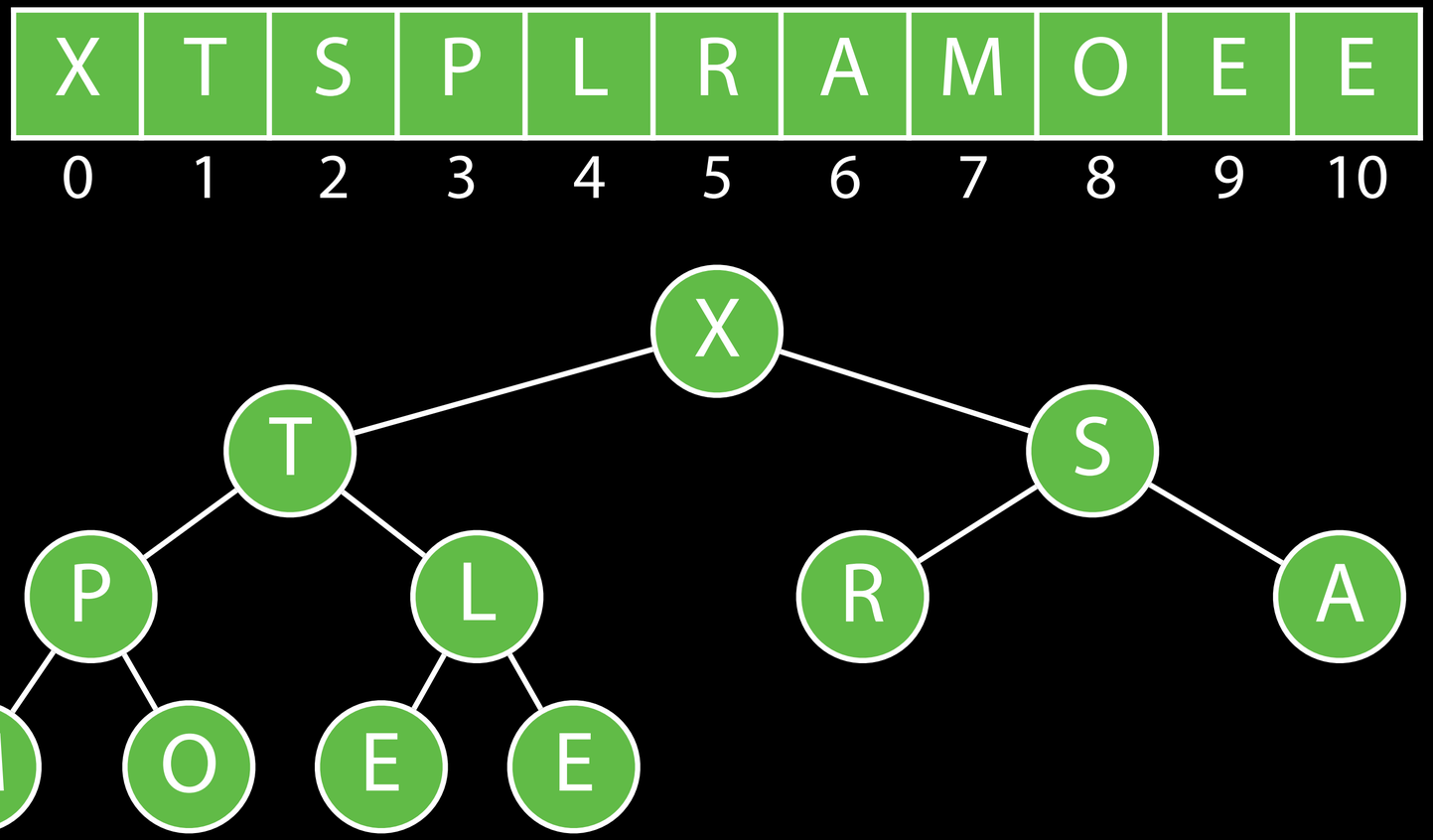
Heapsort

Build a max-heap using bottom-up method.



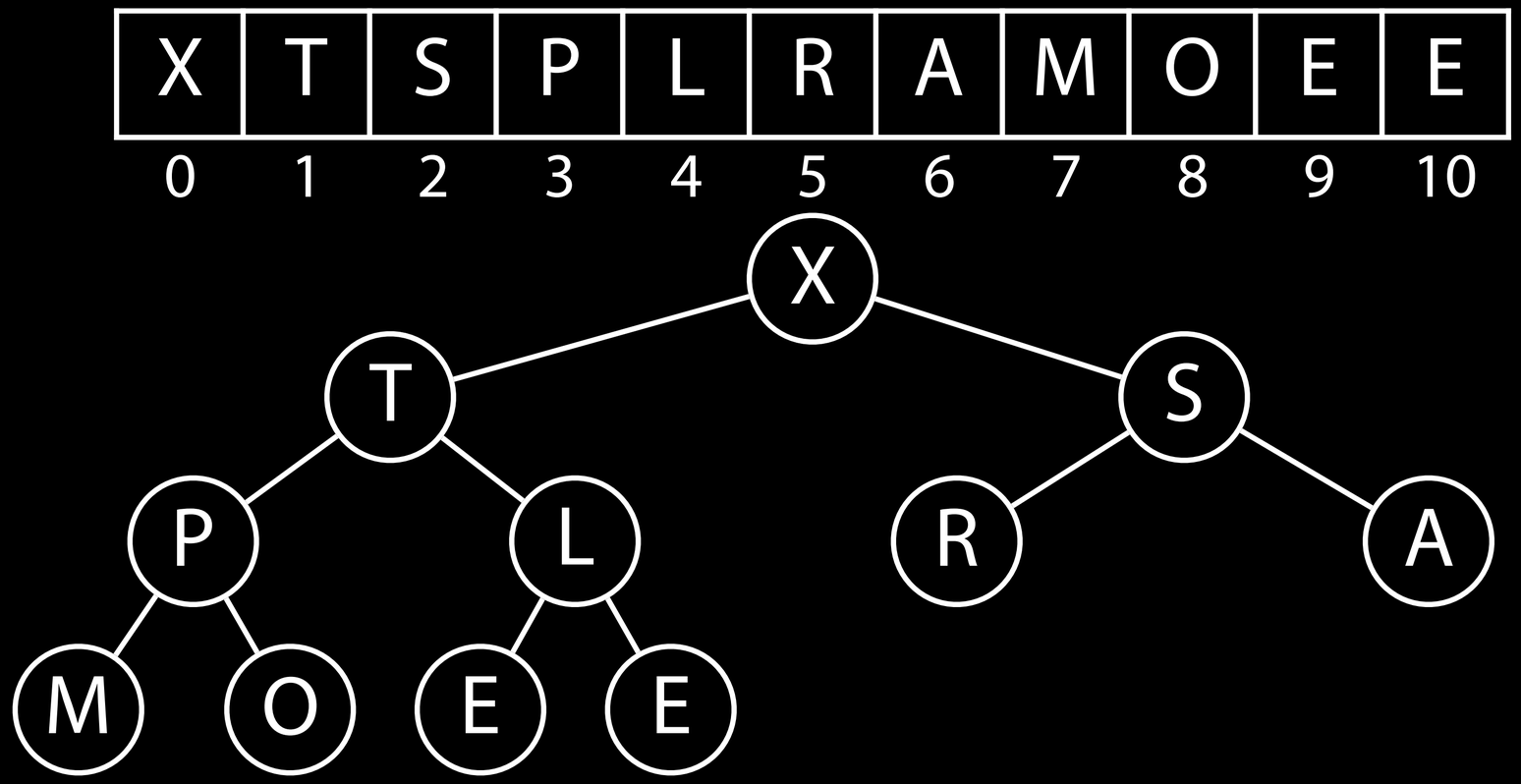
Heapsort

Build a max-heap using bottom-up method.



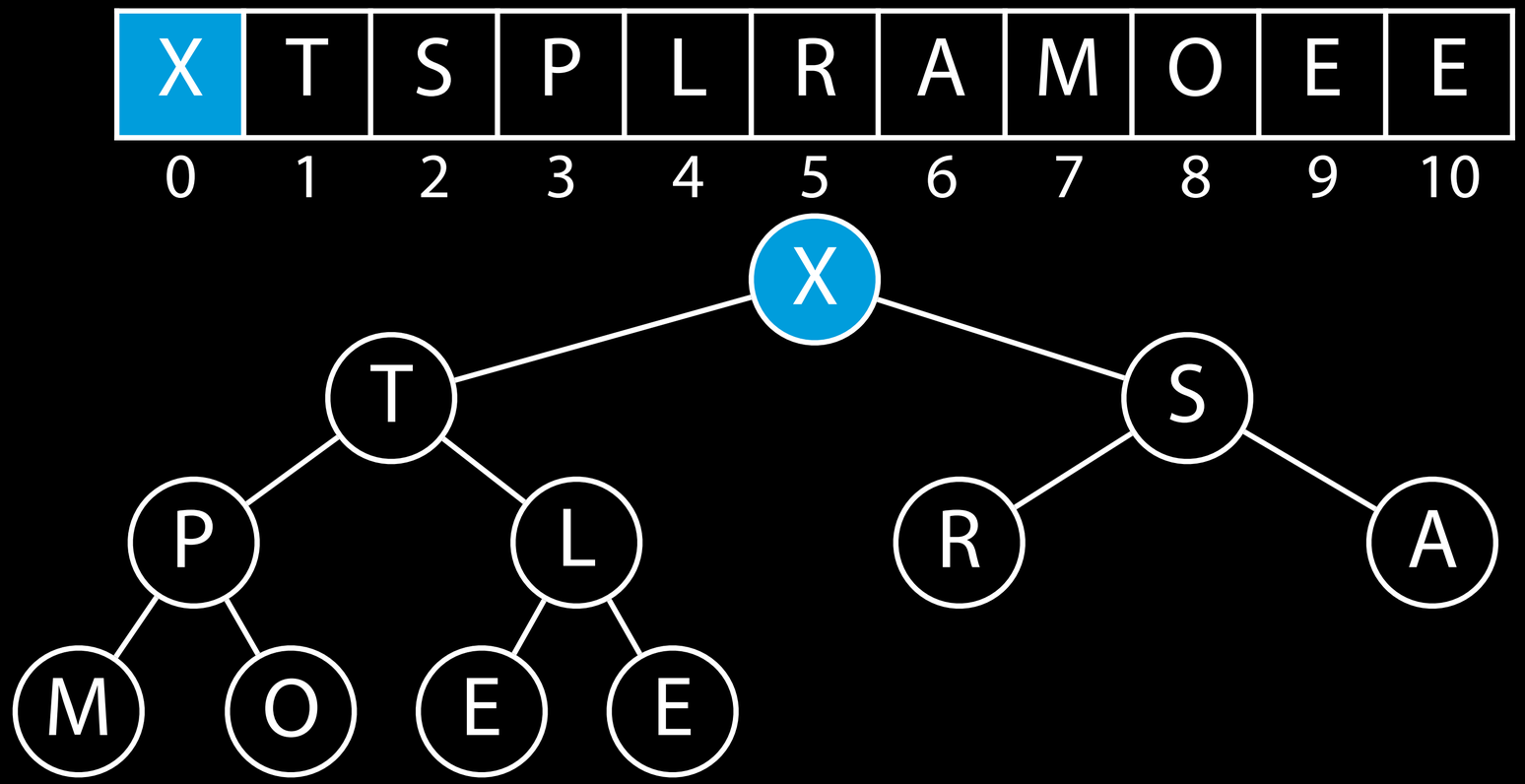
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



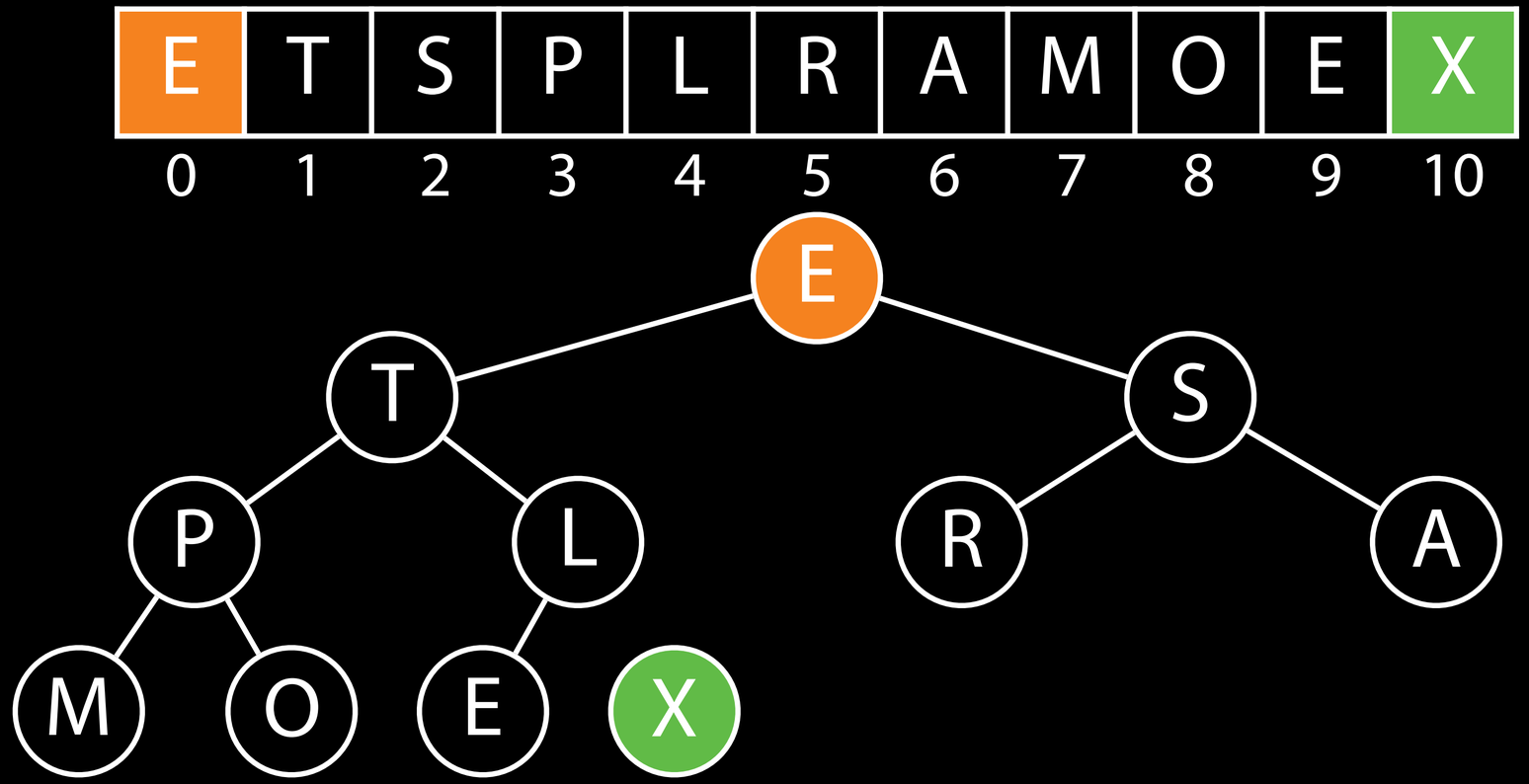
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



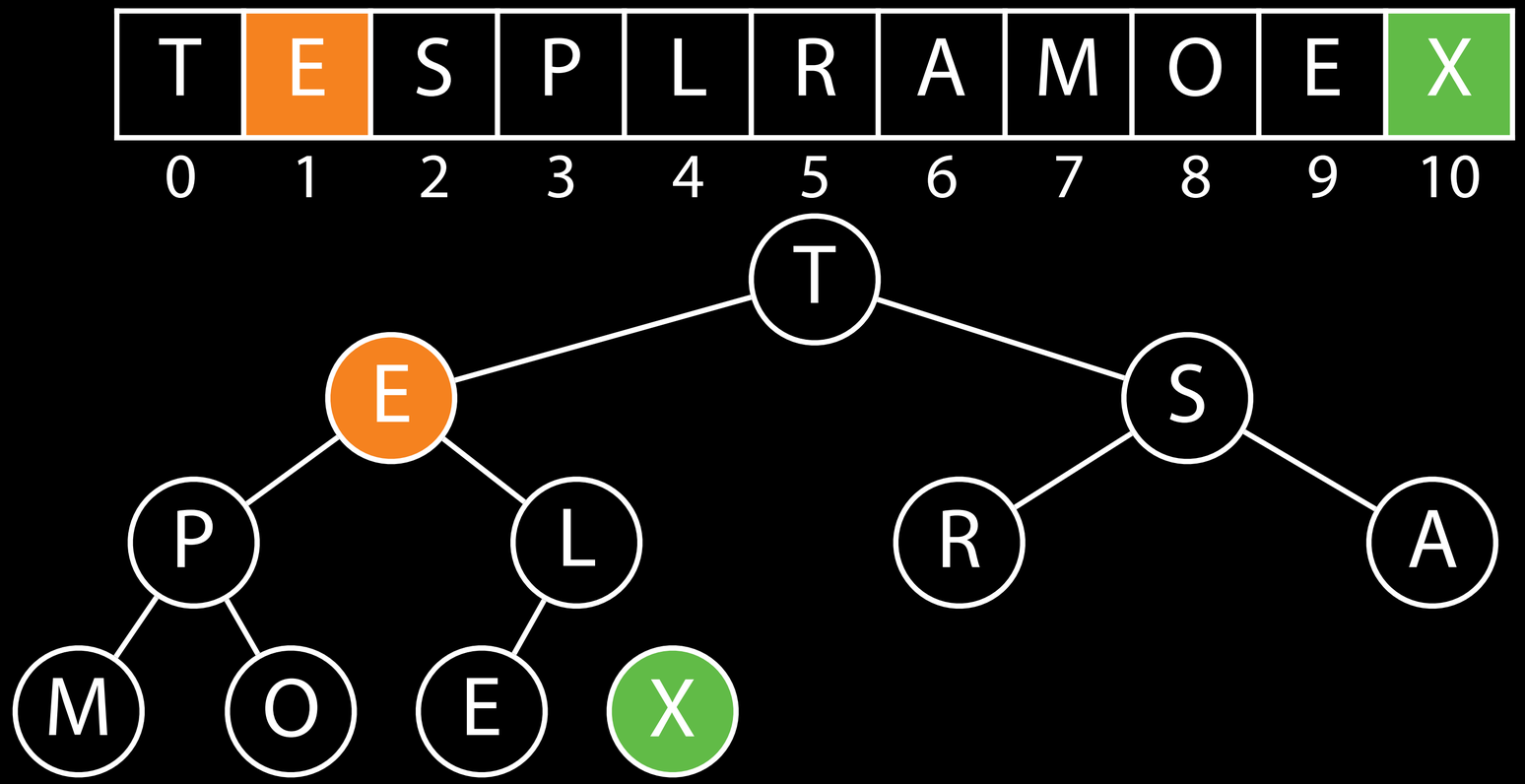
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



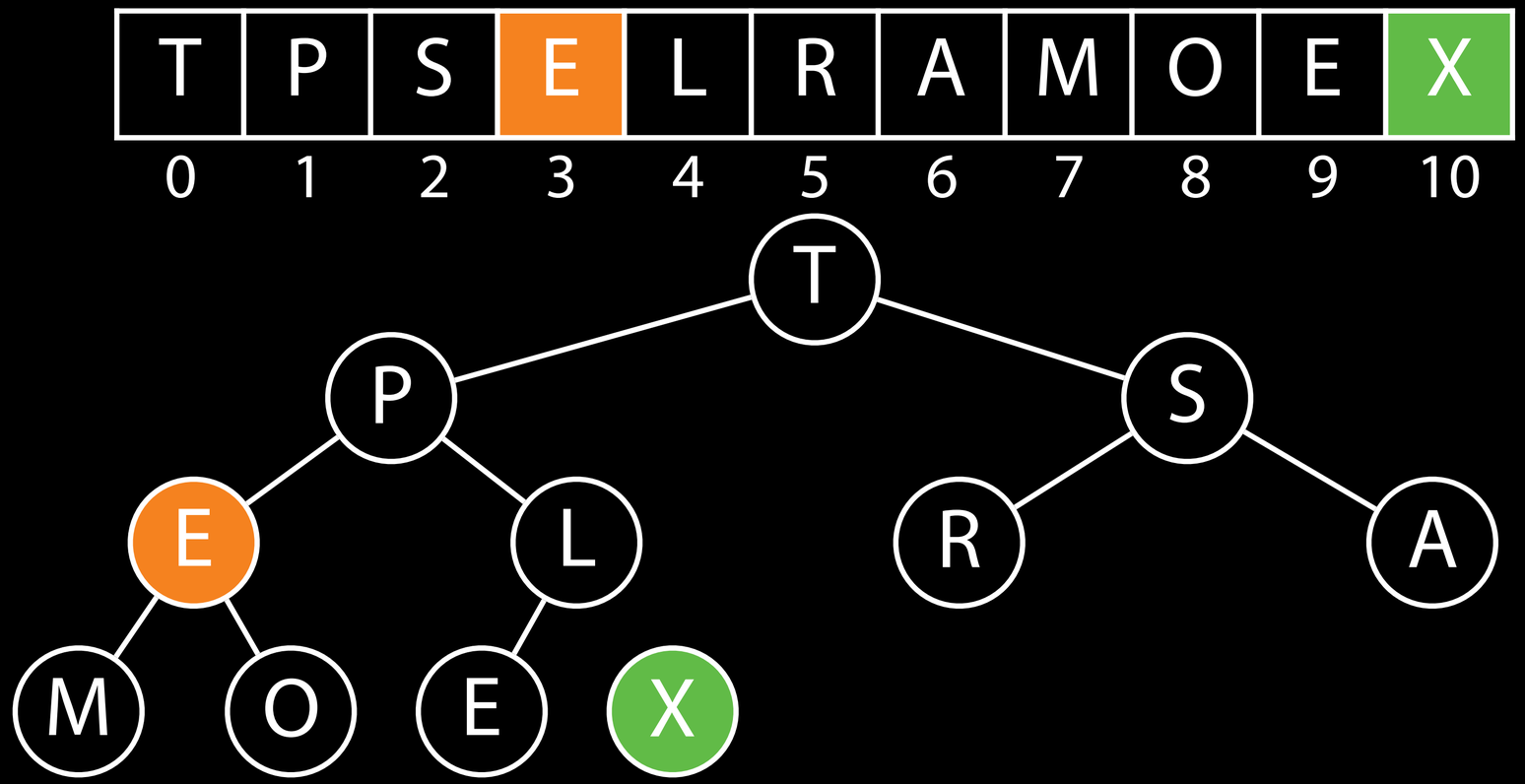
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



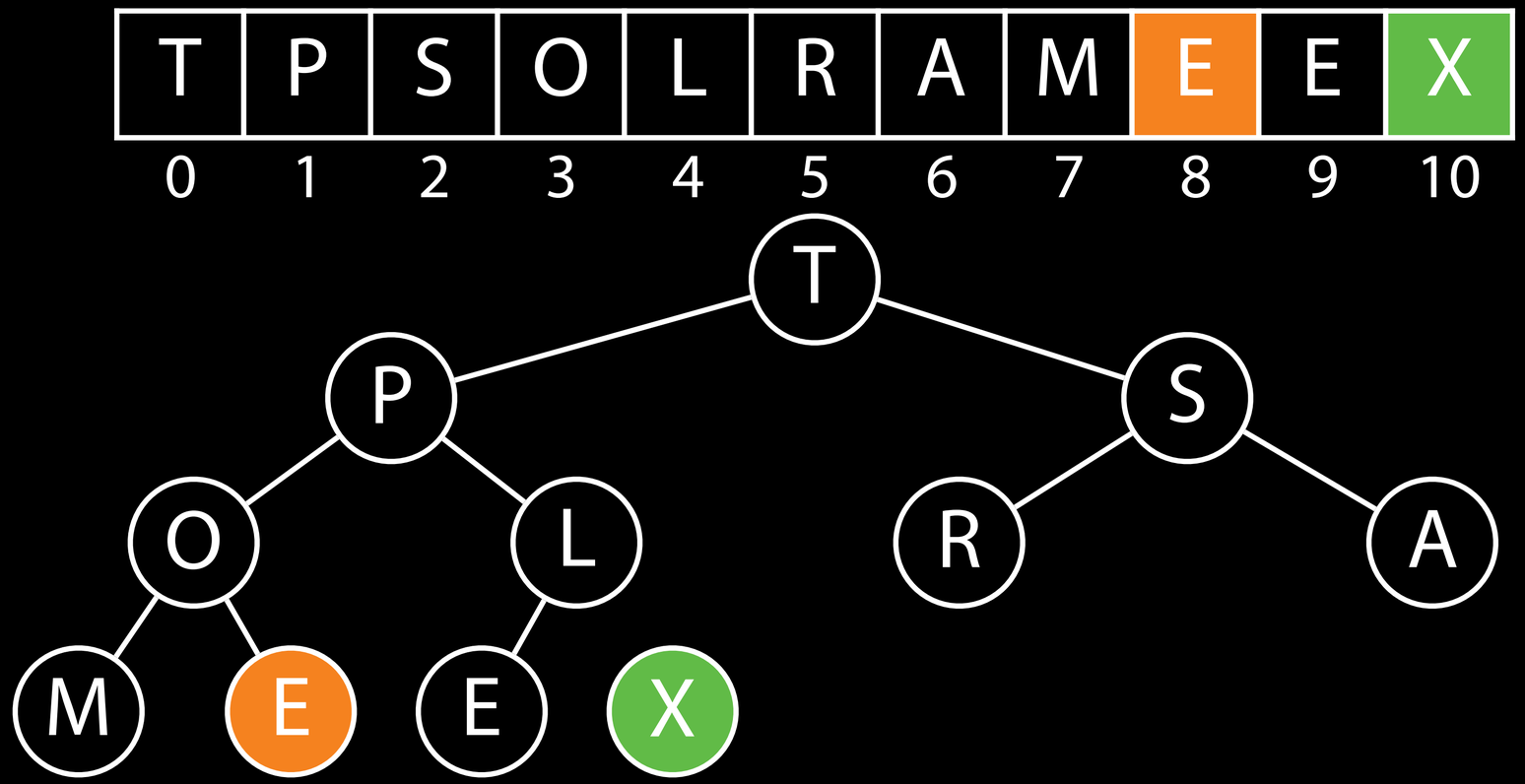
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



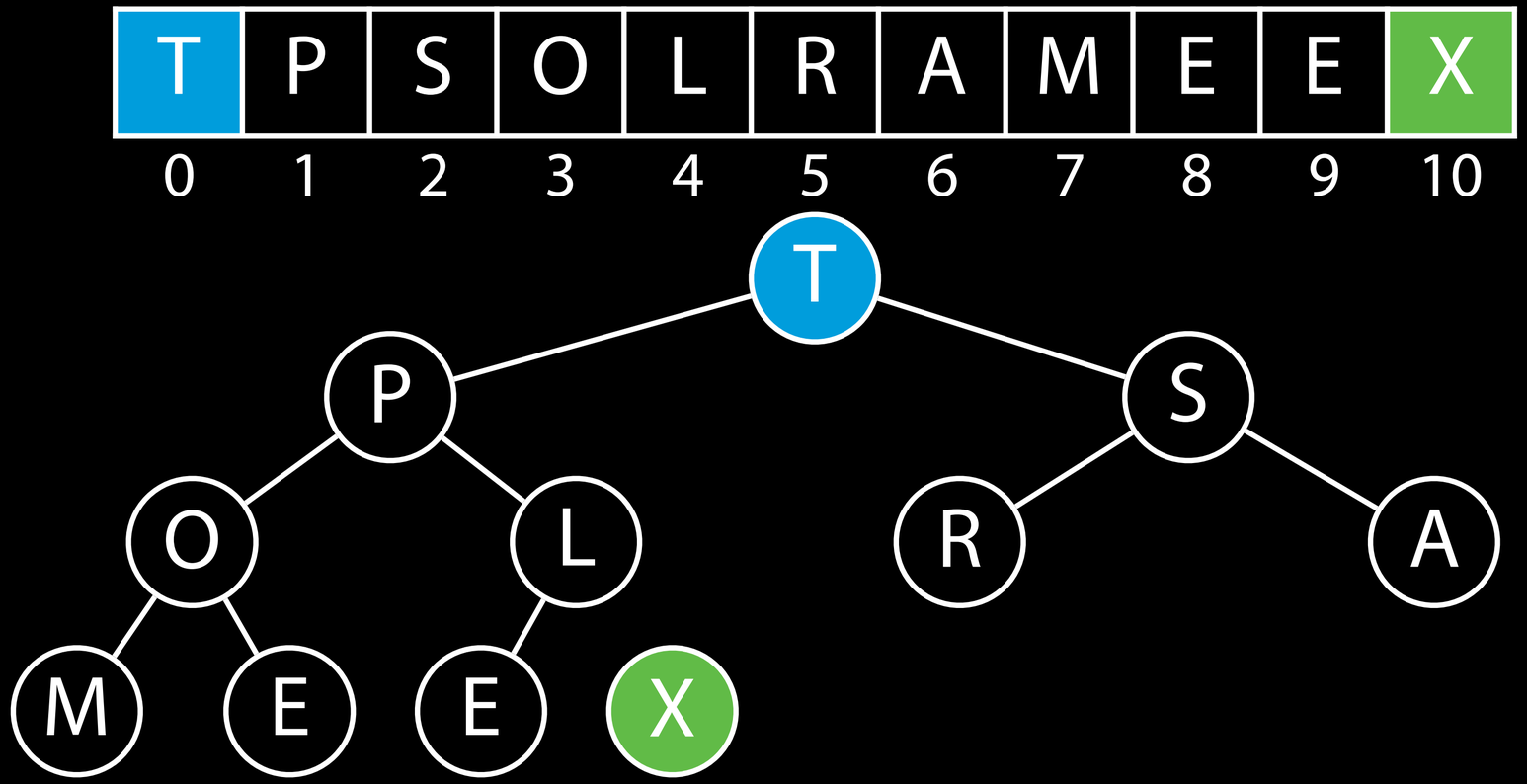
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



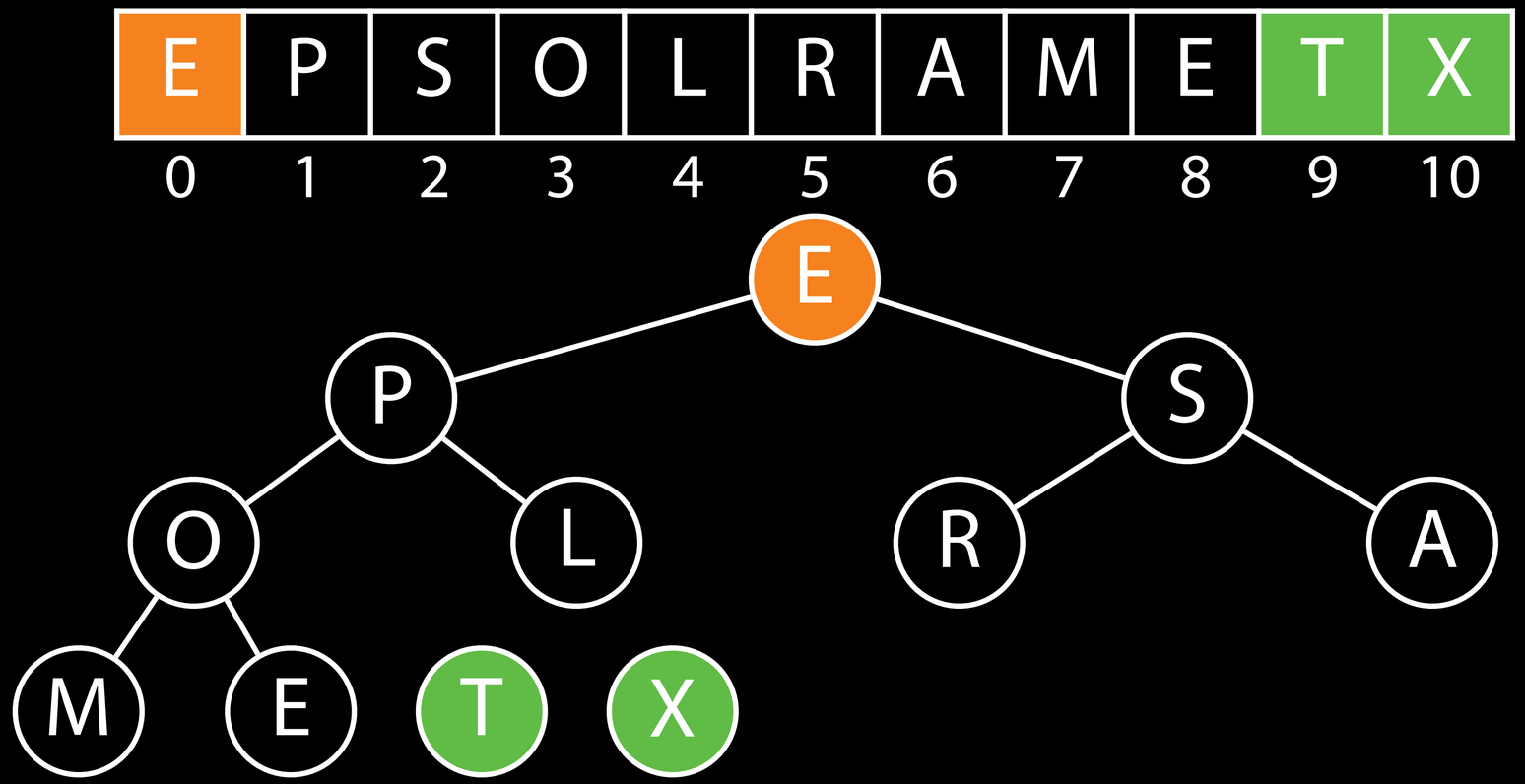
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



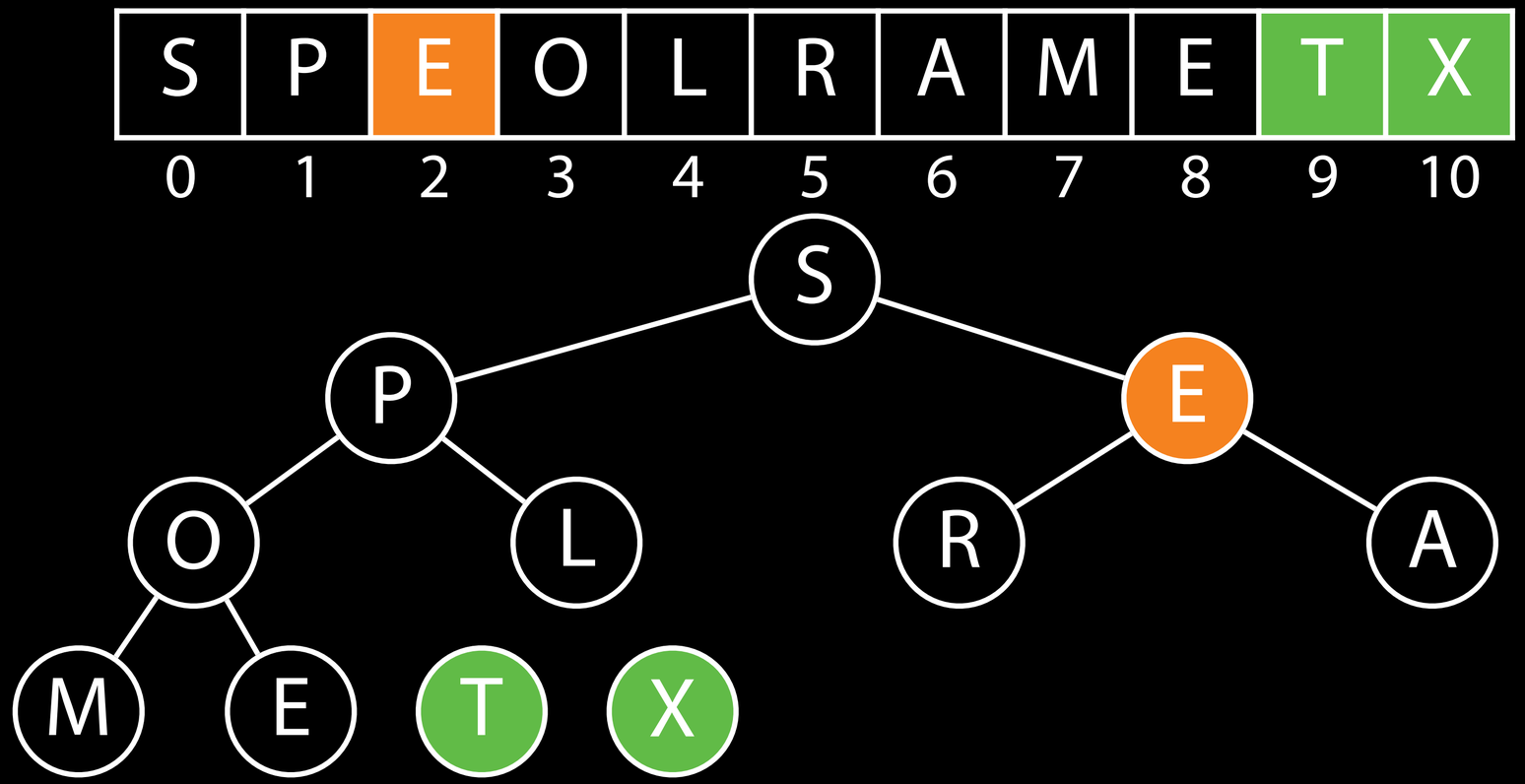
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



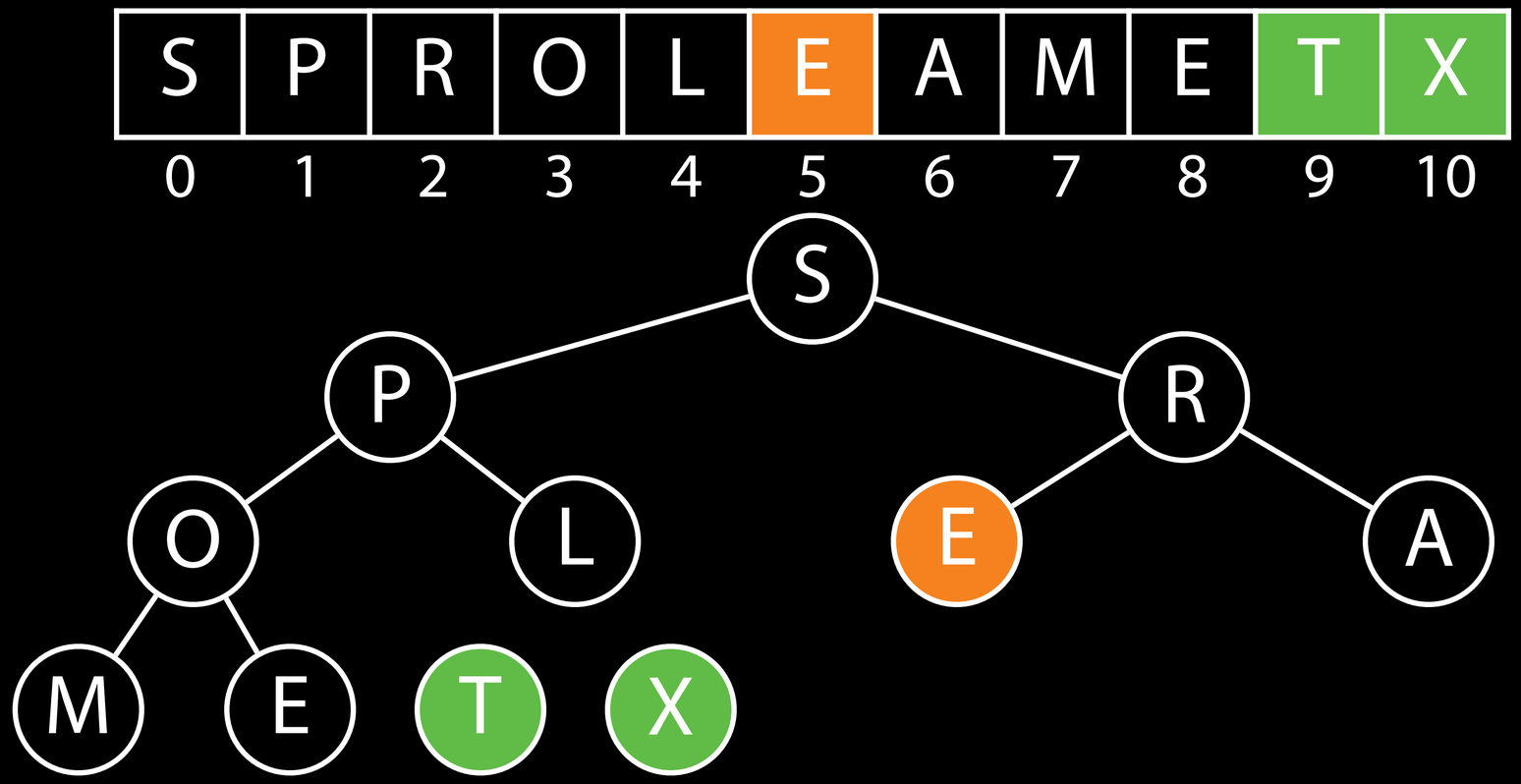
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



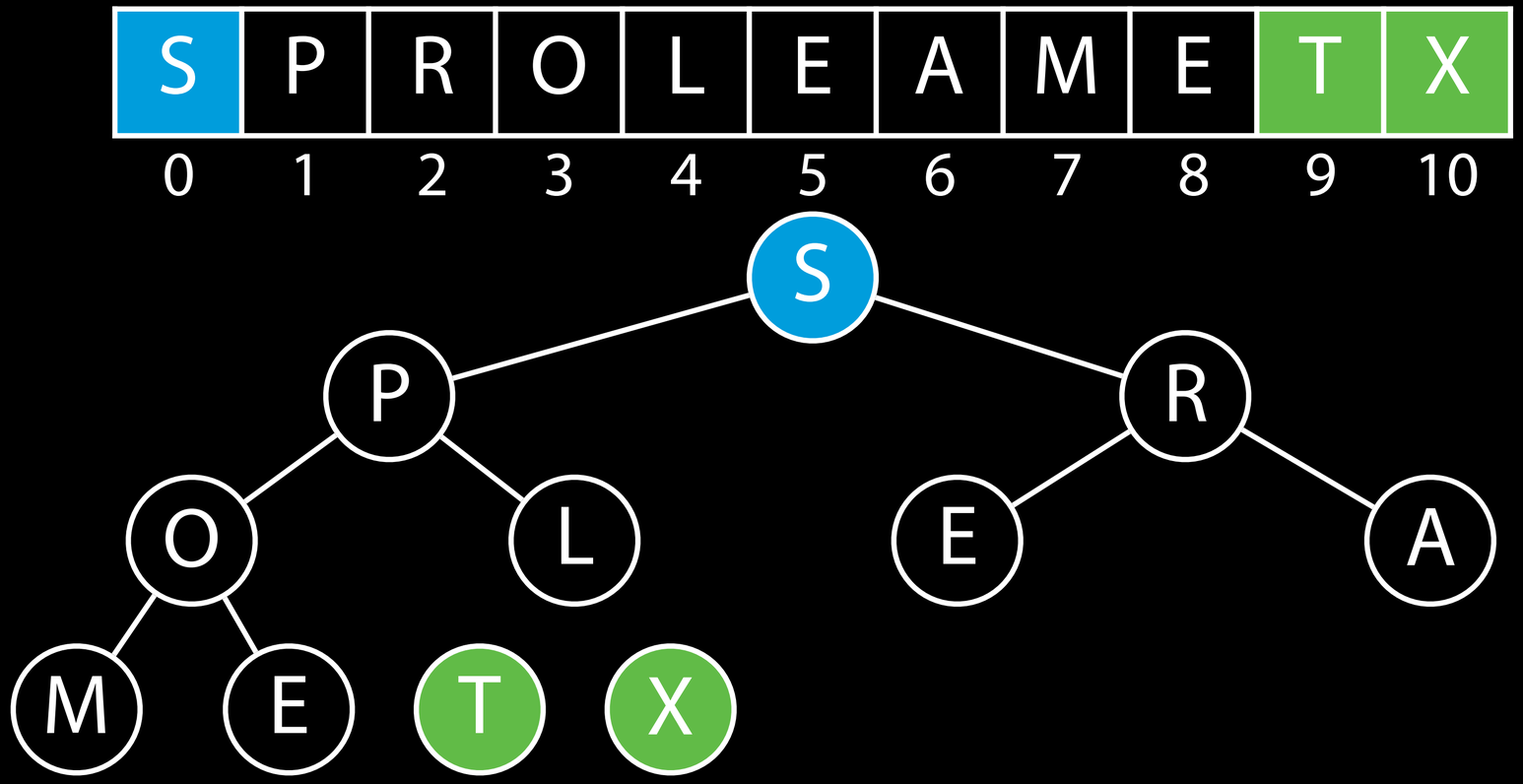
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



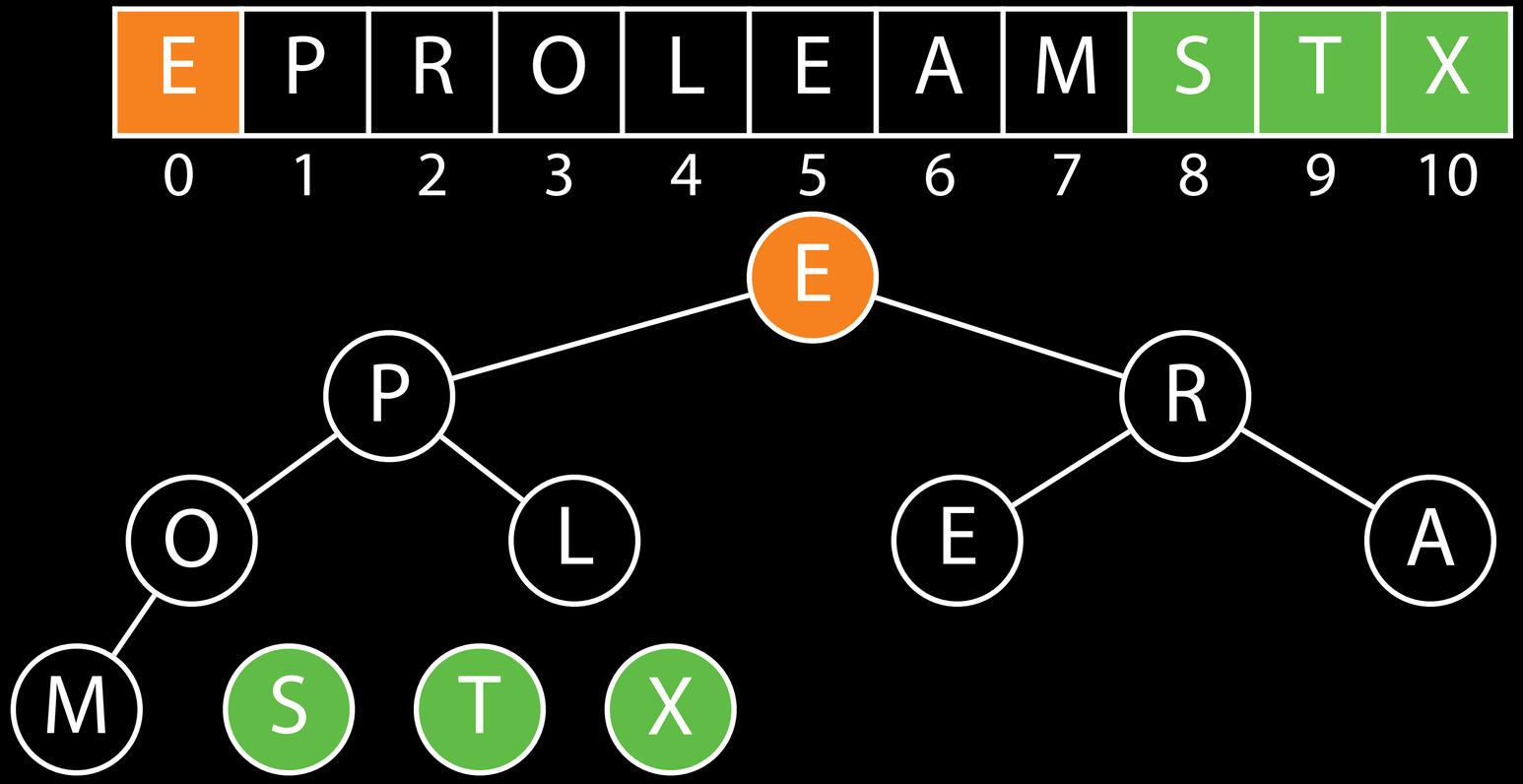
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



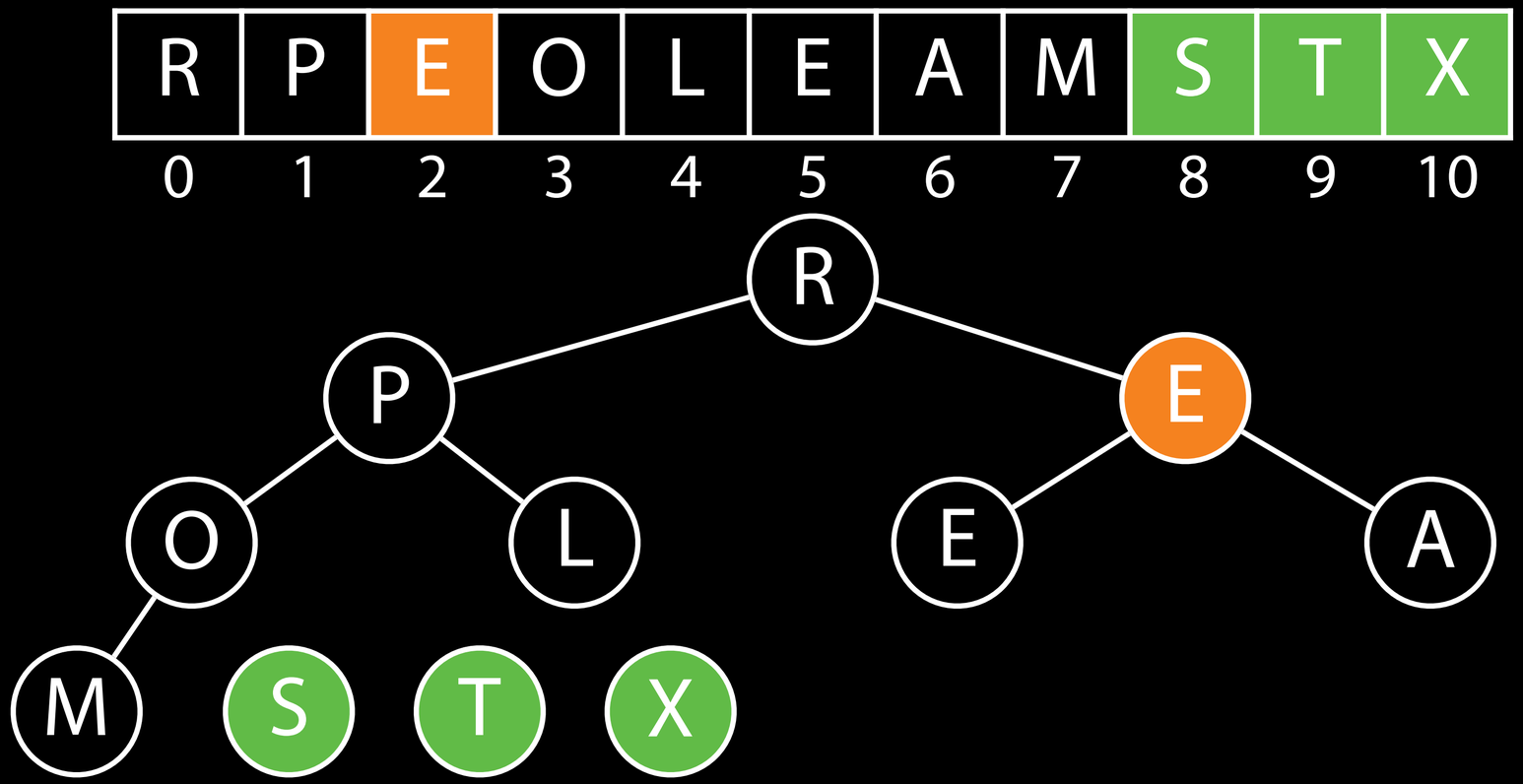
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



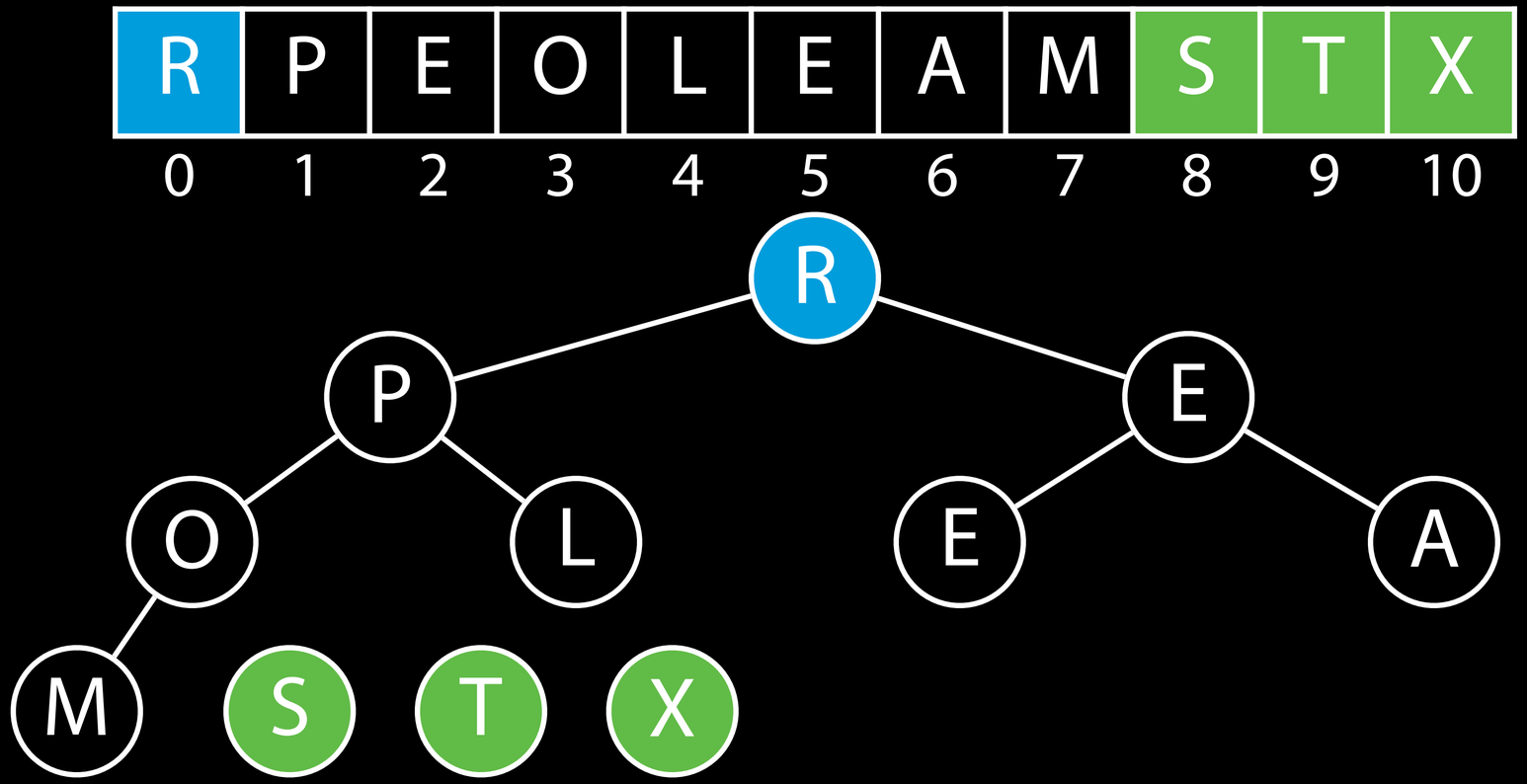
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



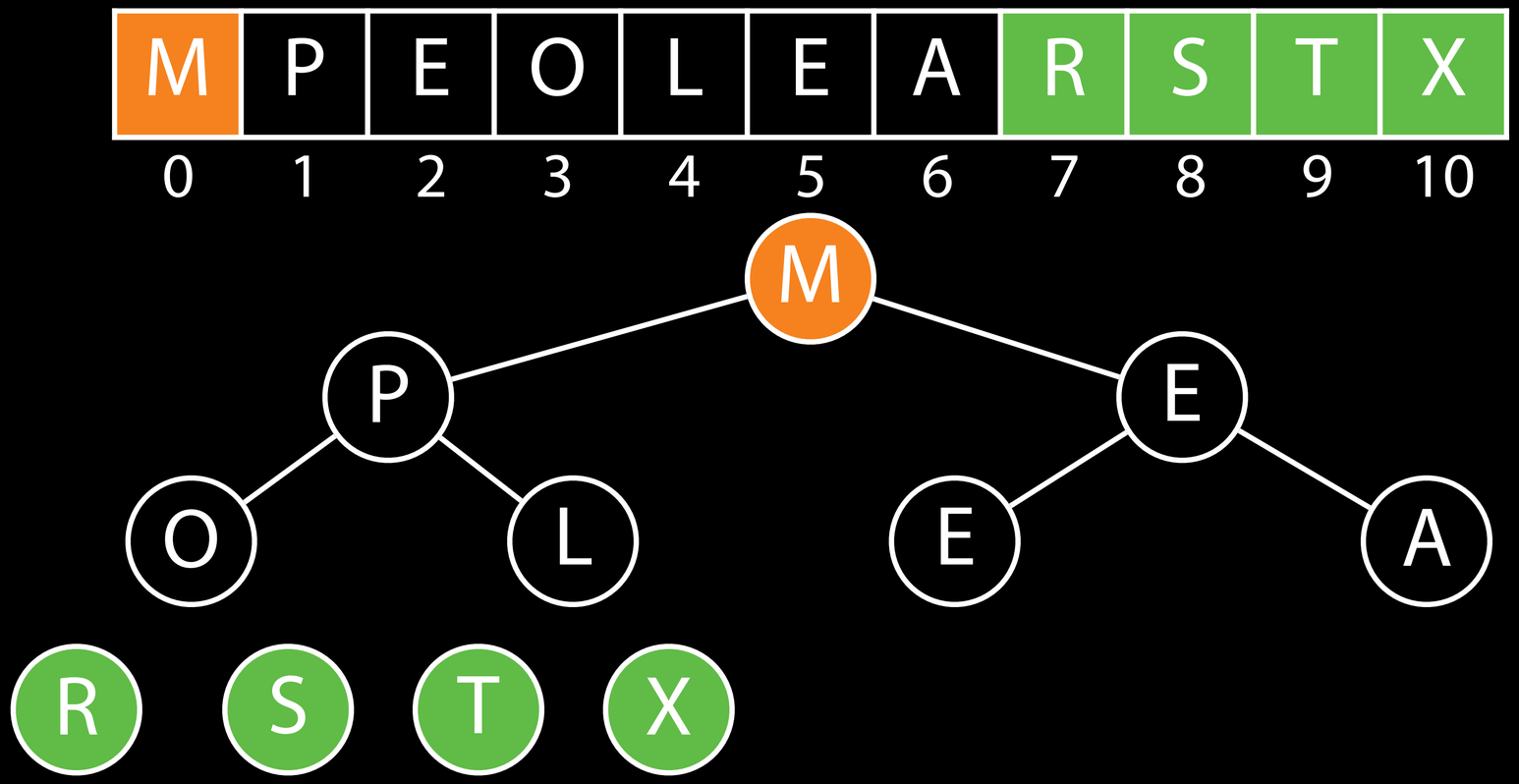
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



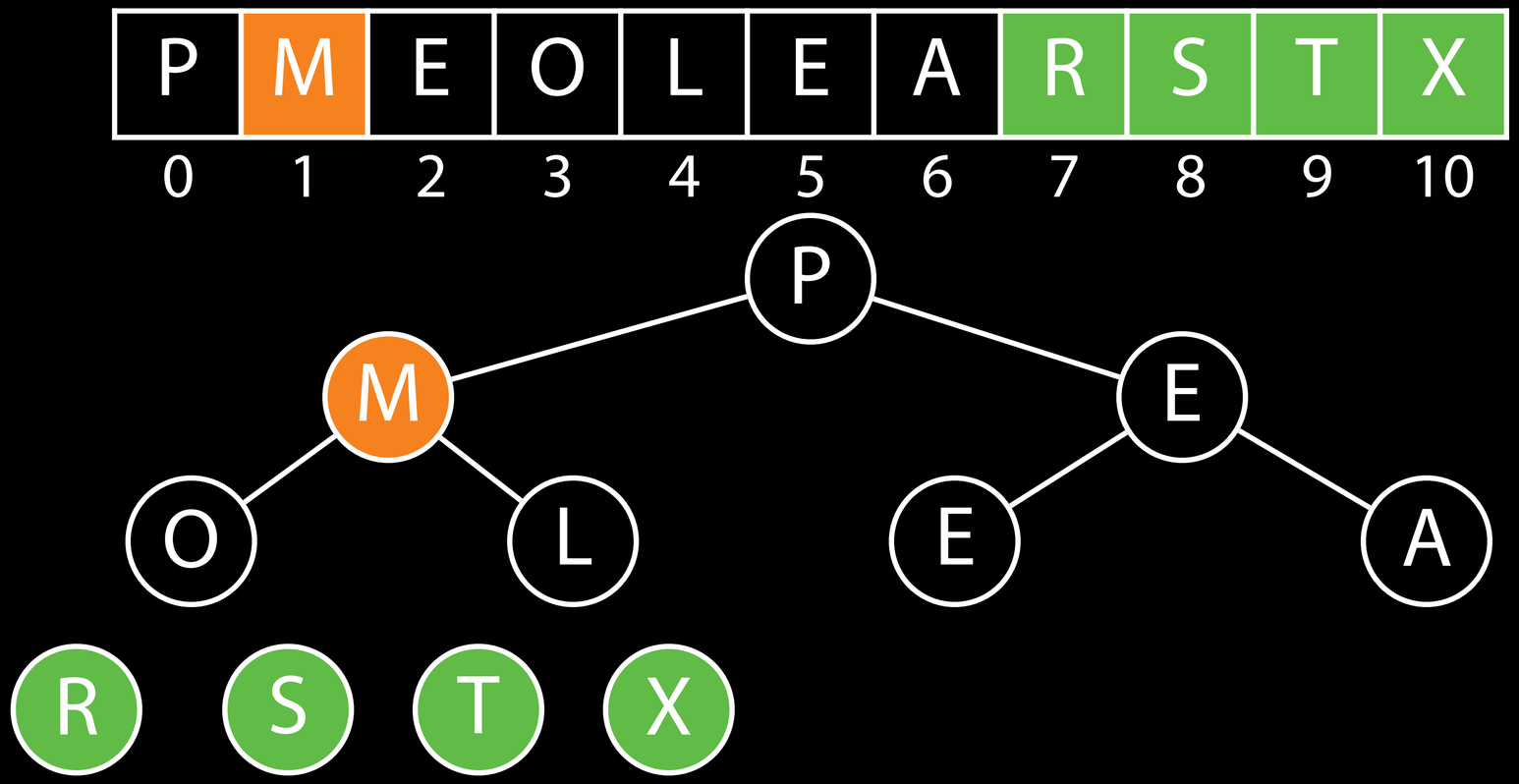
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



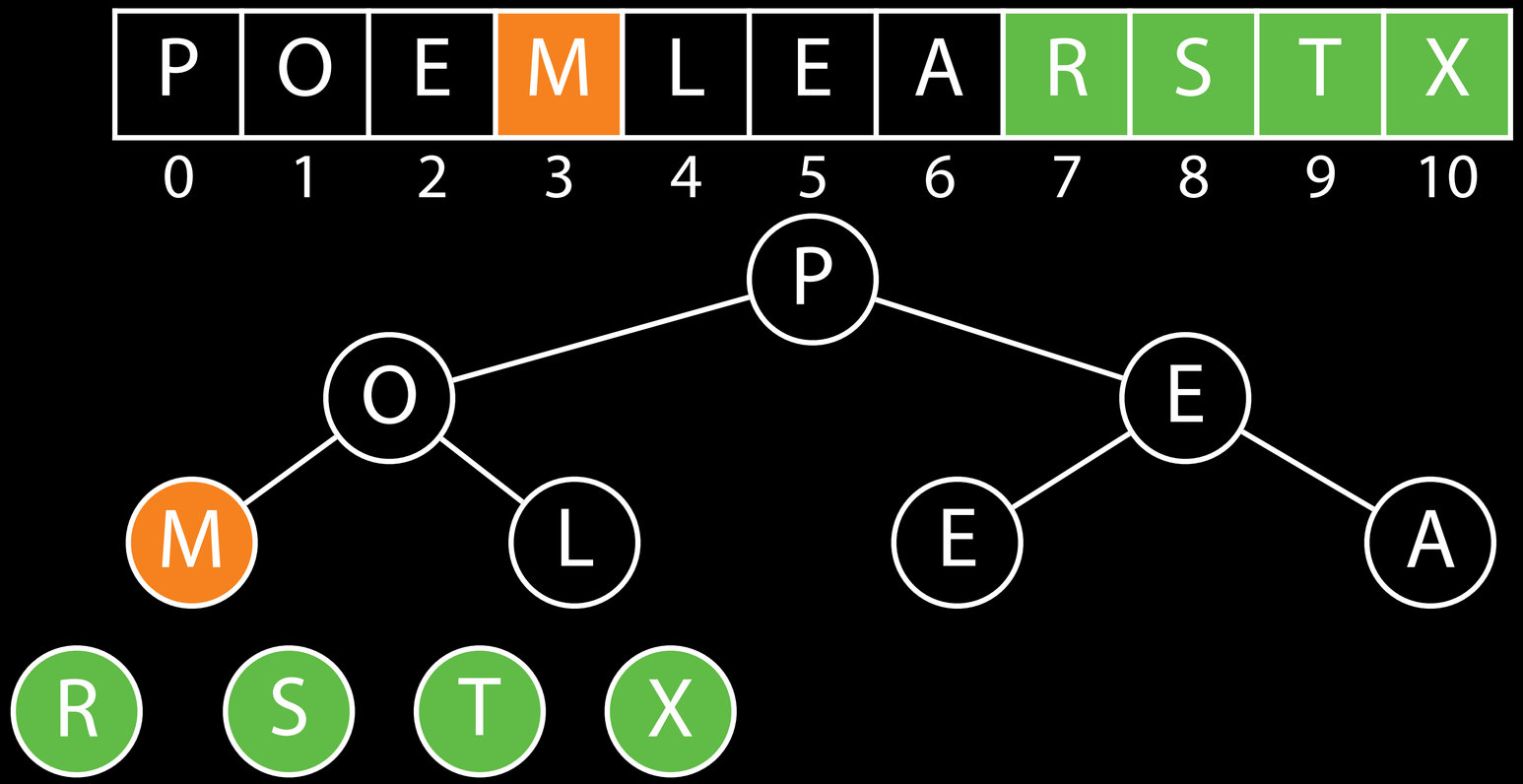
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



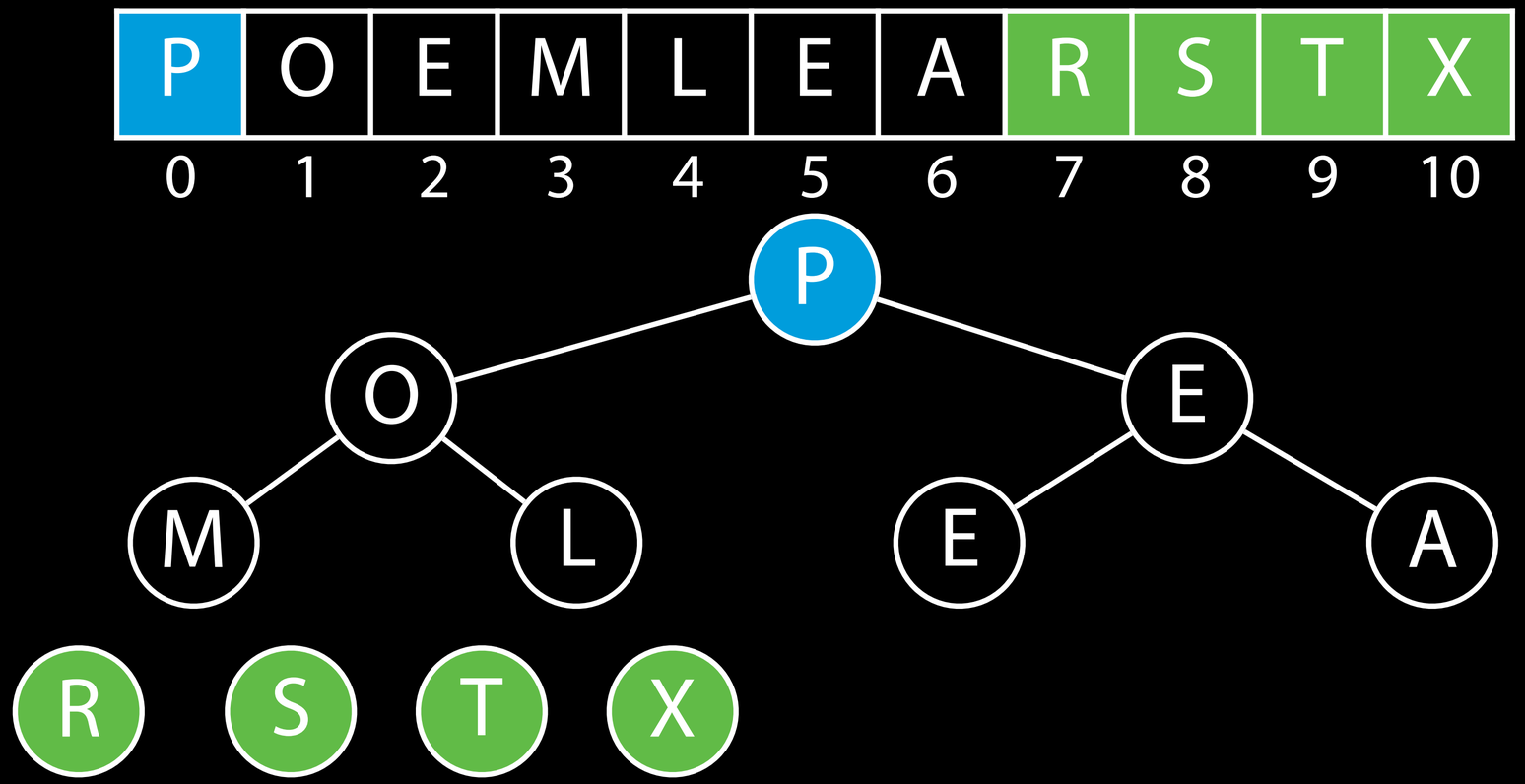
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



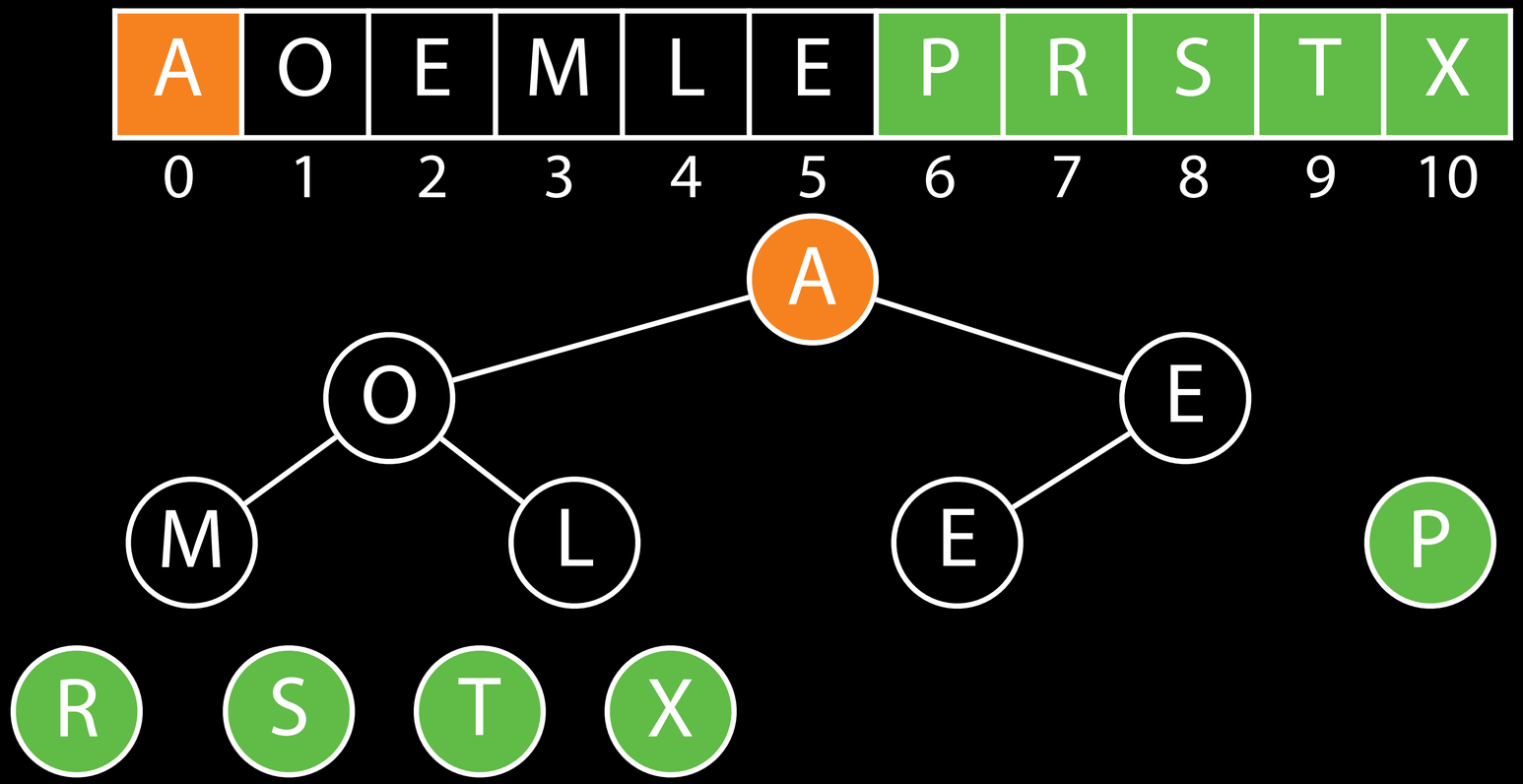
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



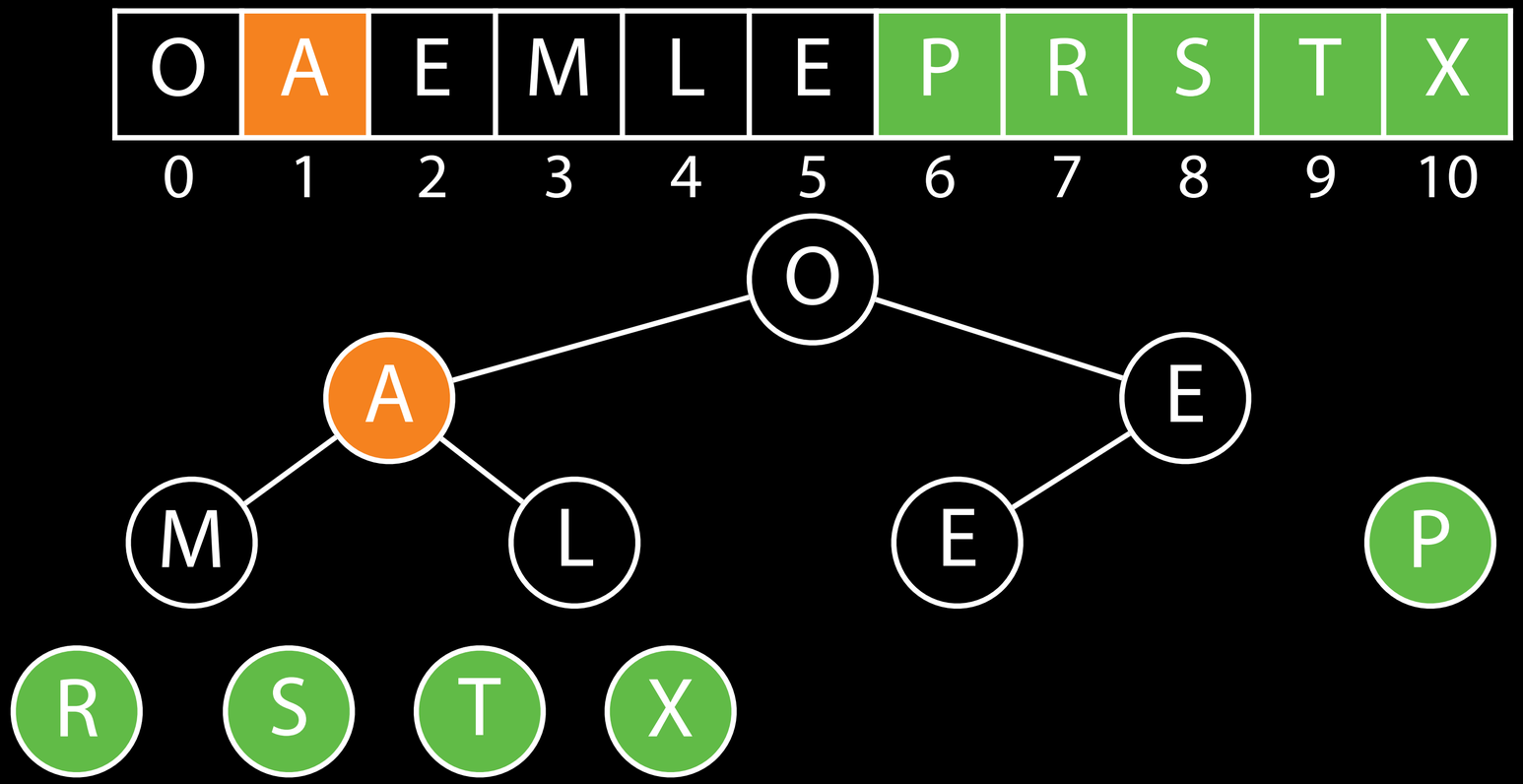
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



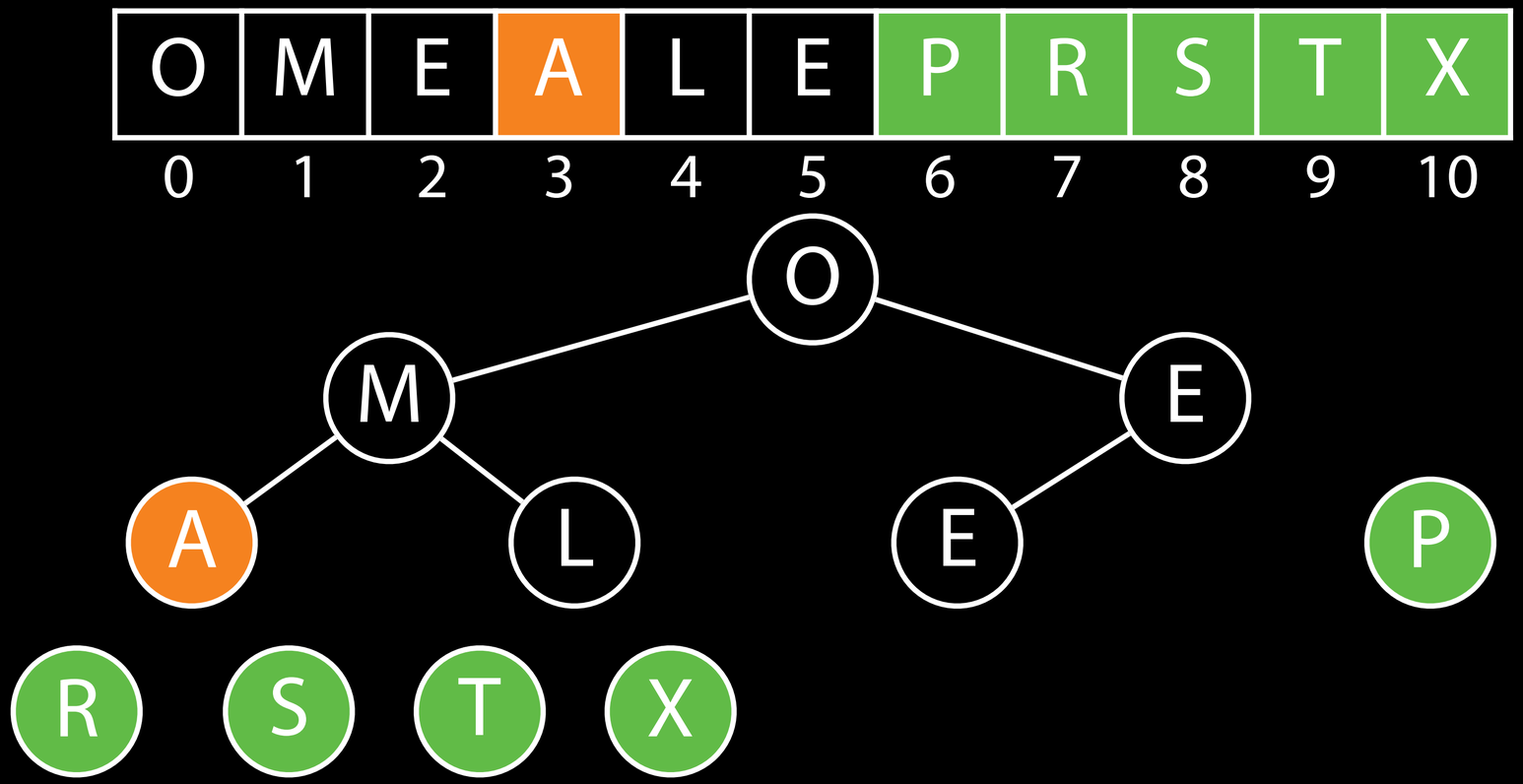
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



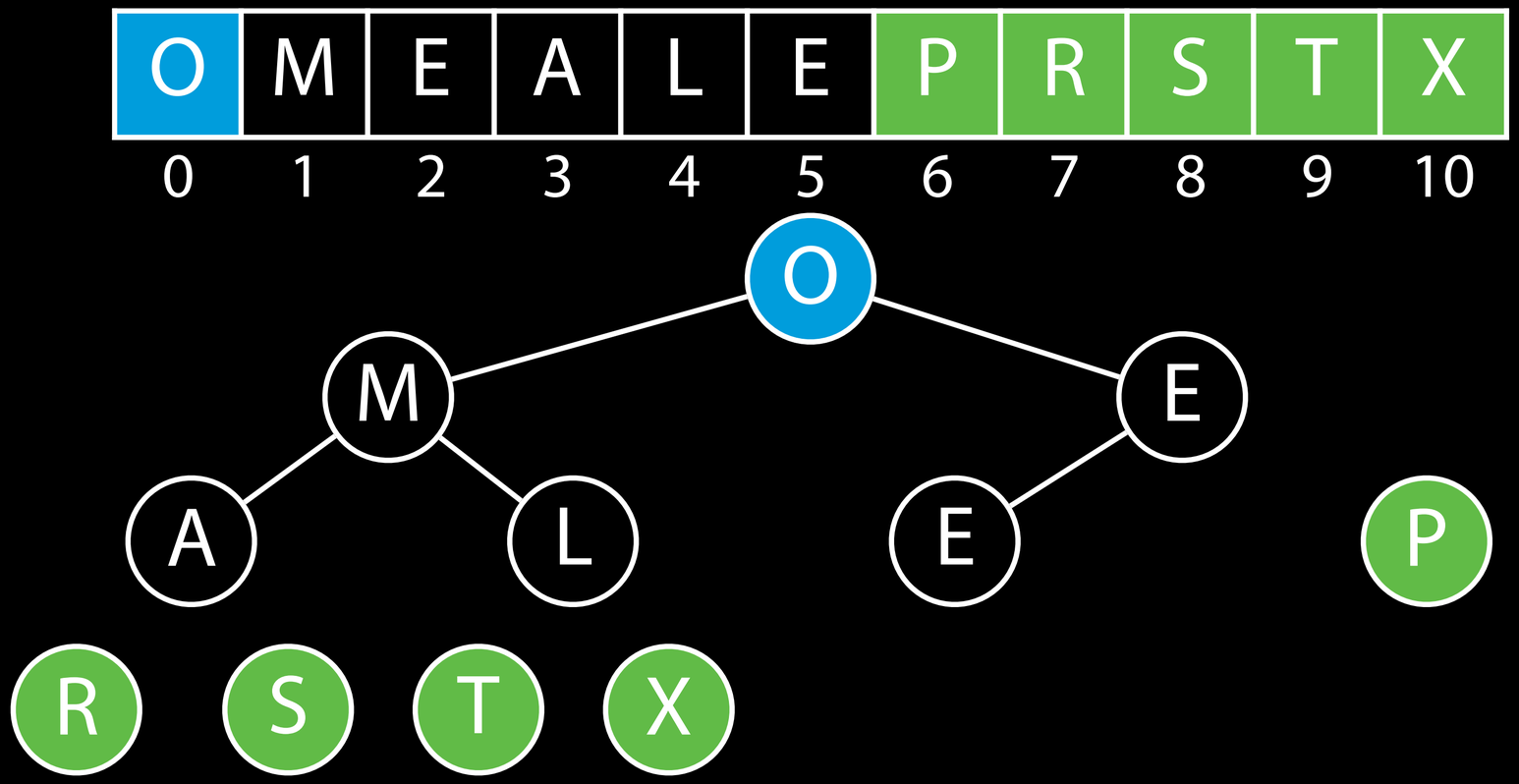
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



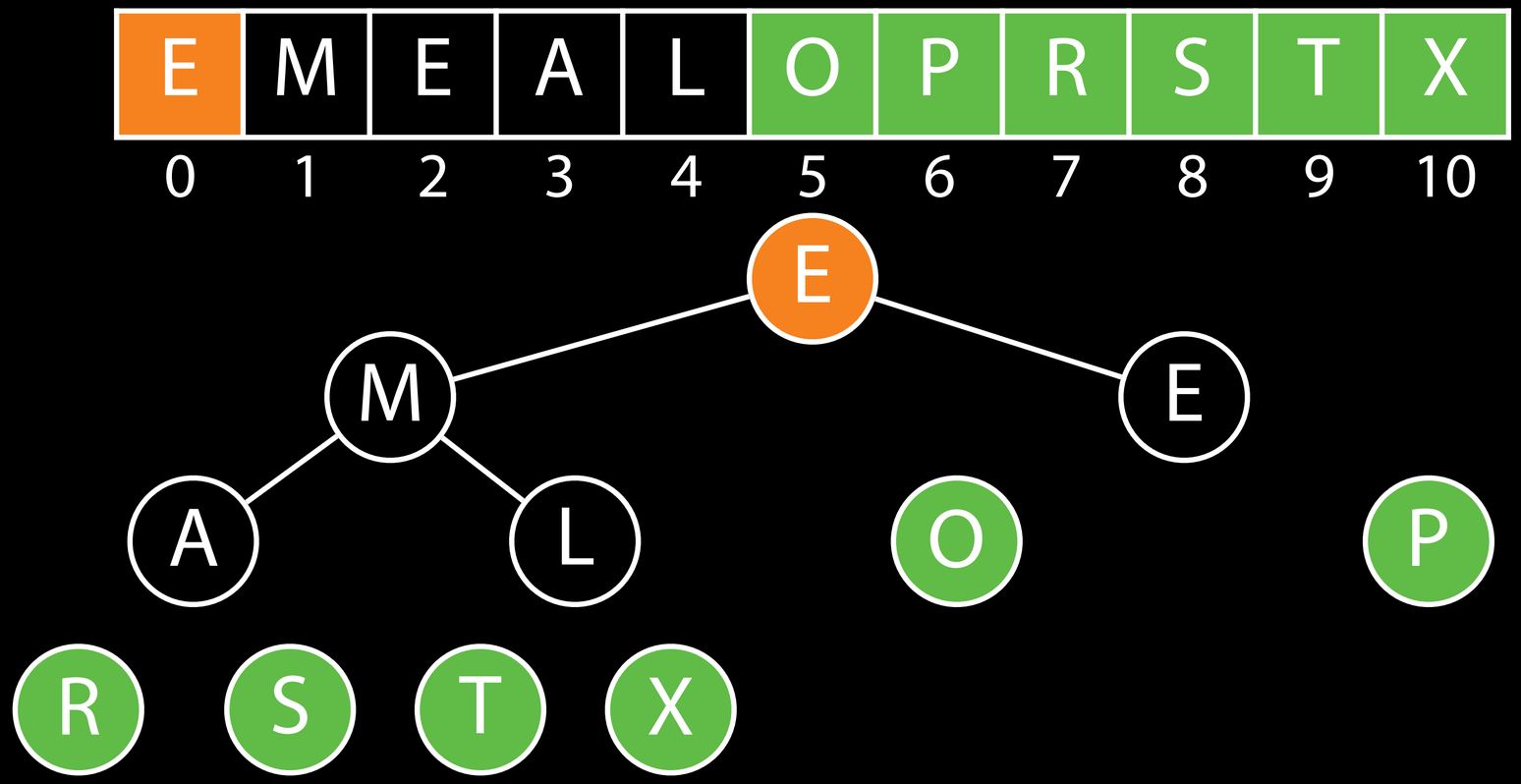
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



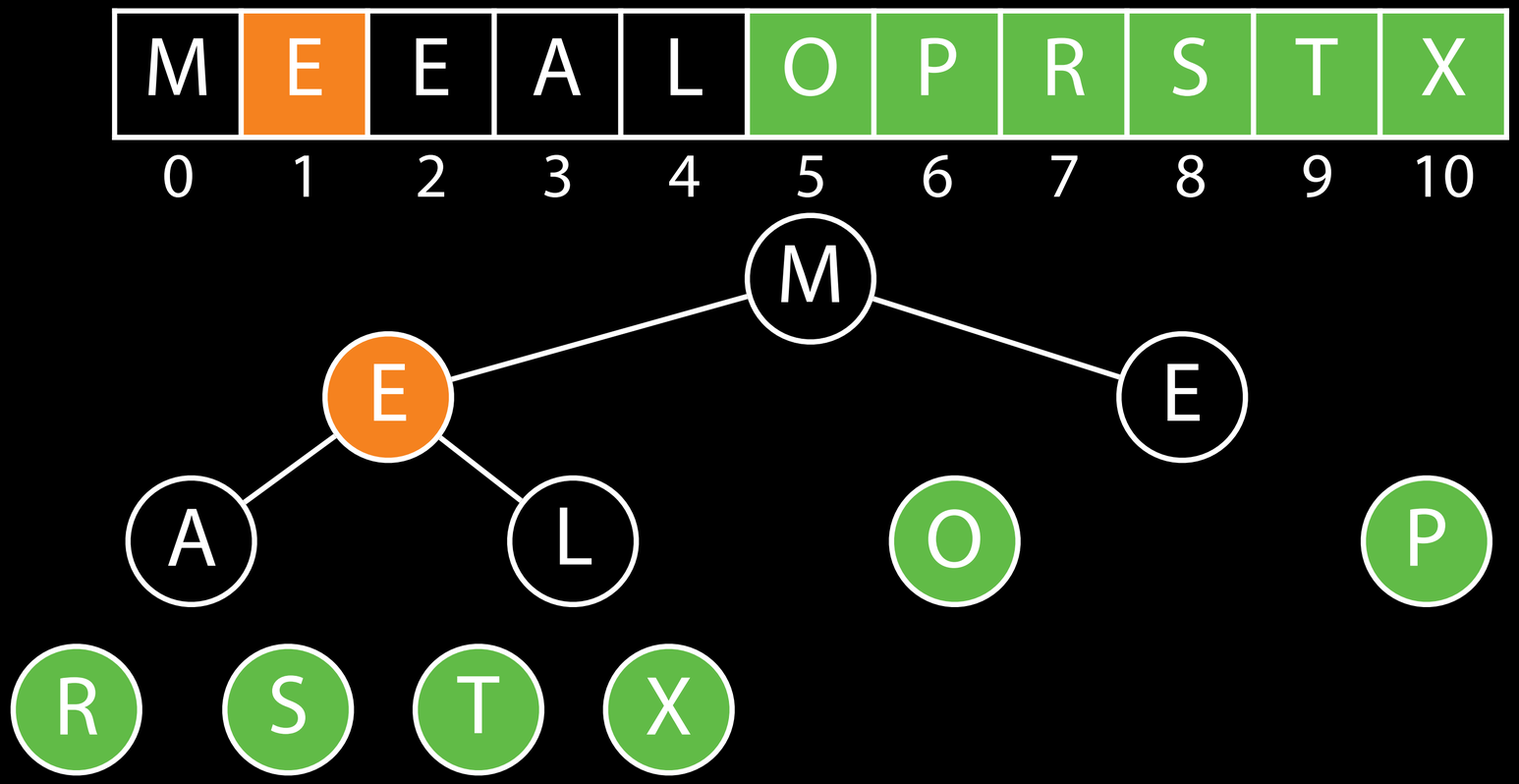
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



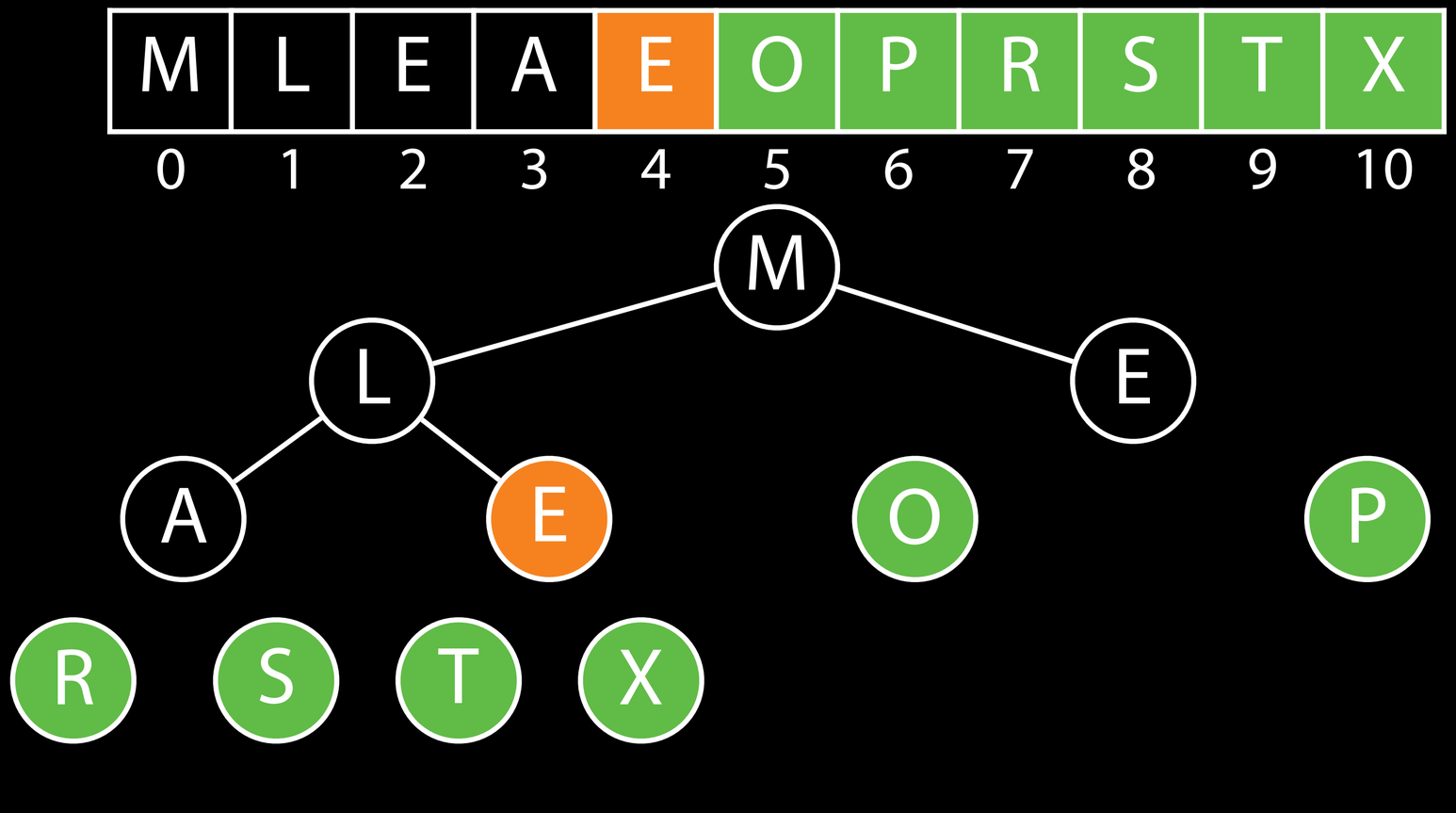
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



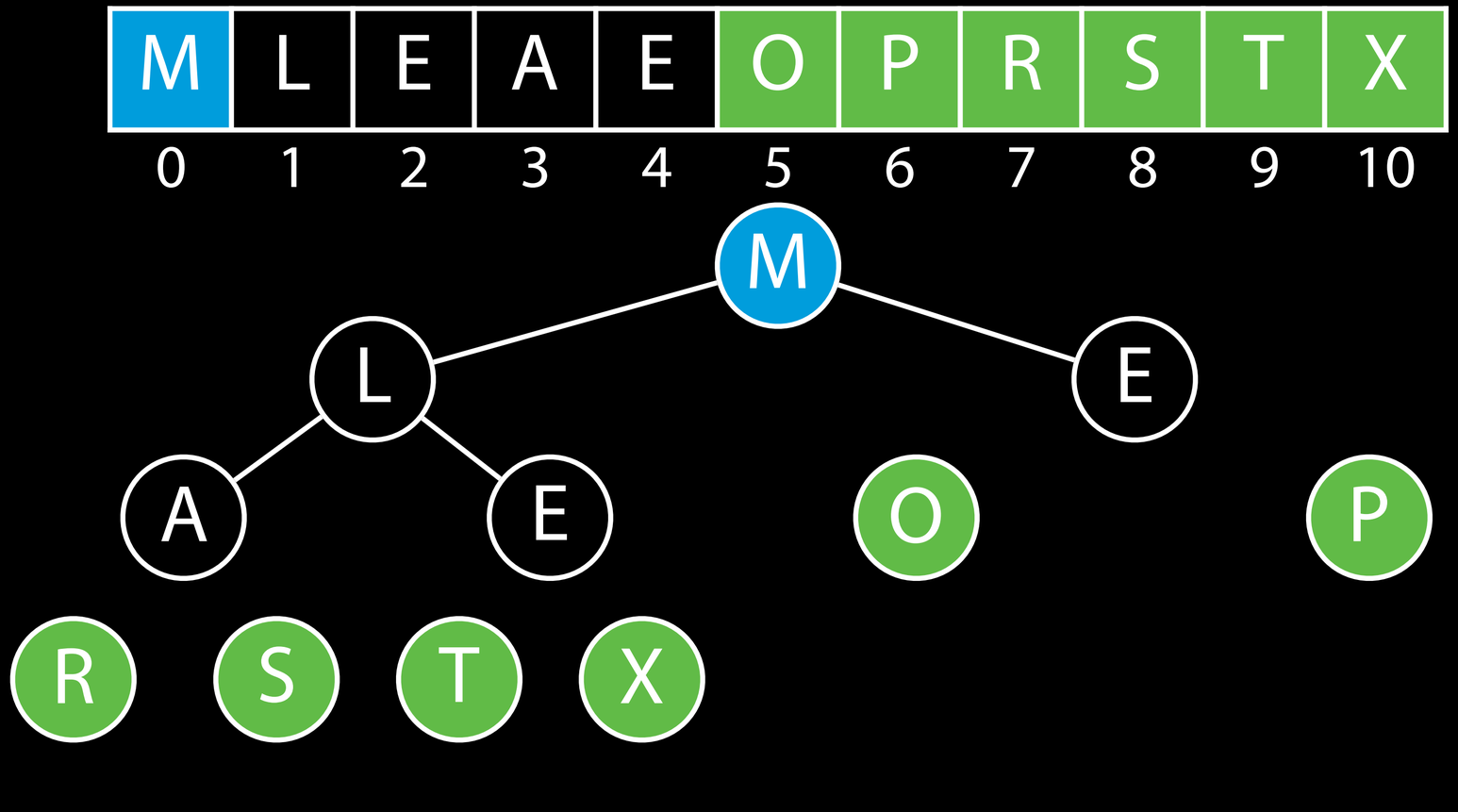
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



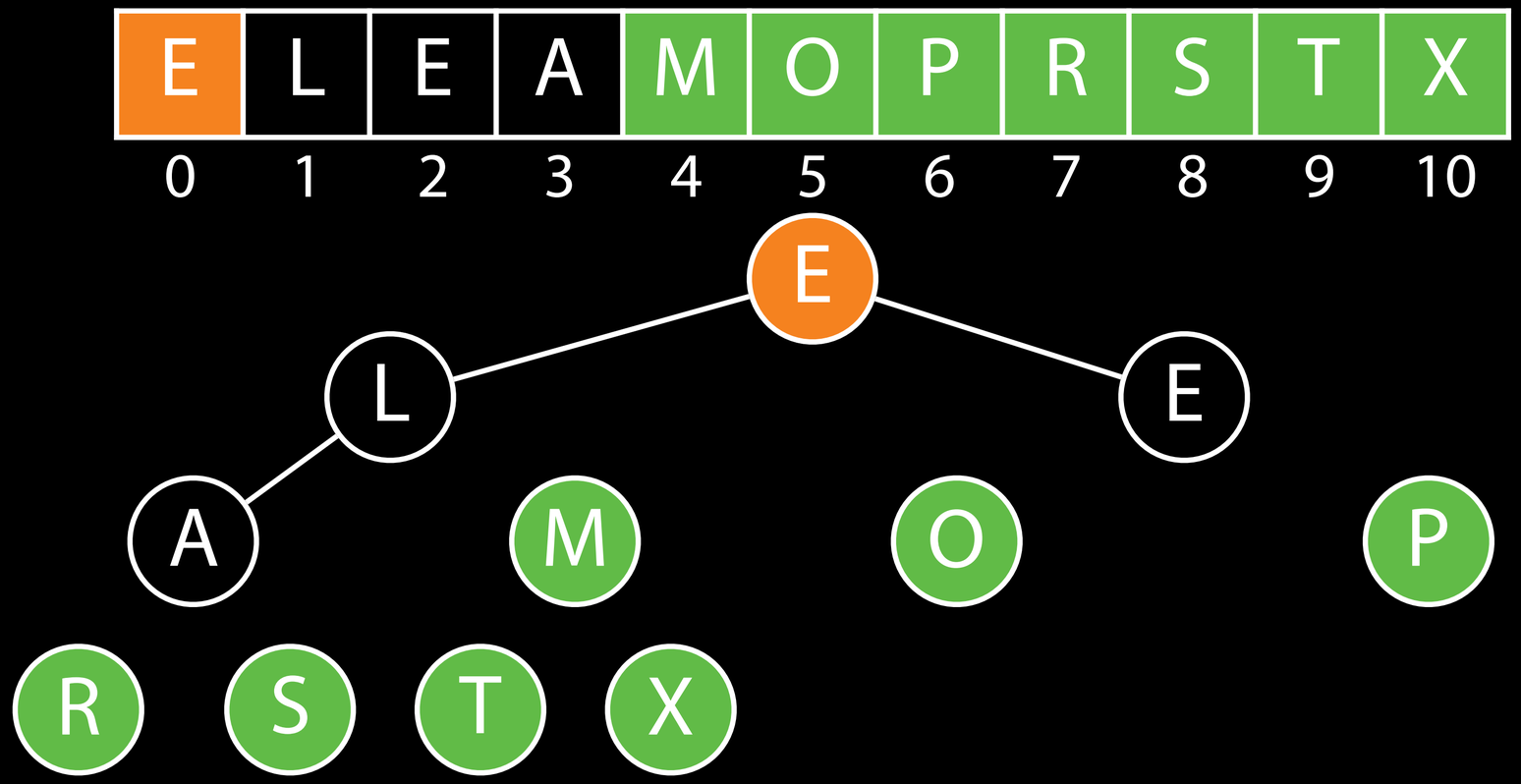
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



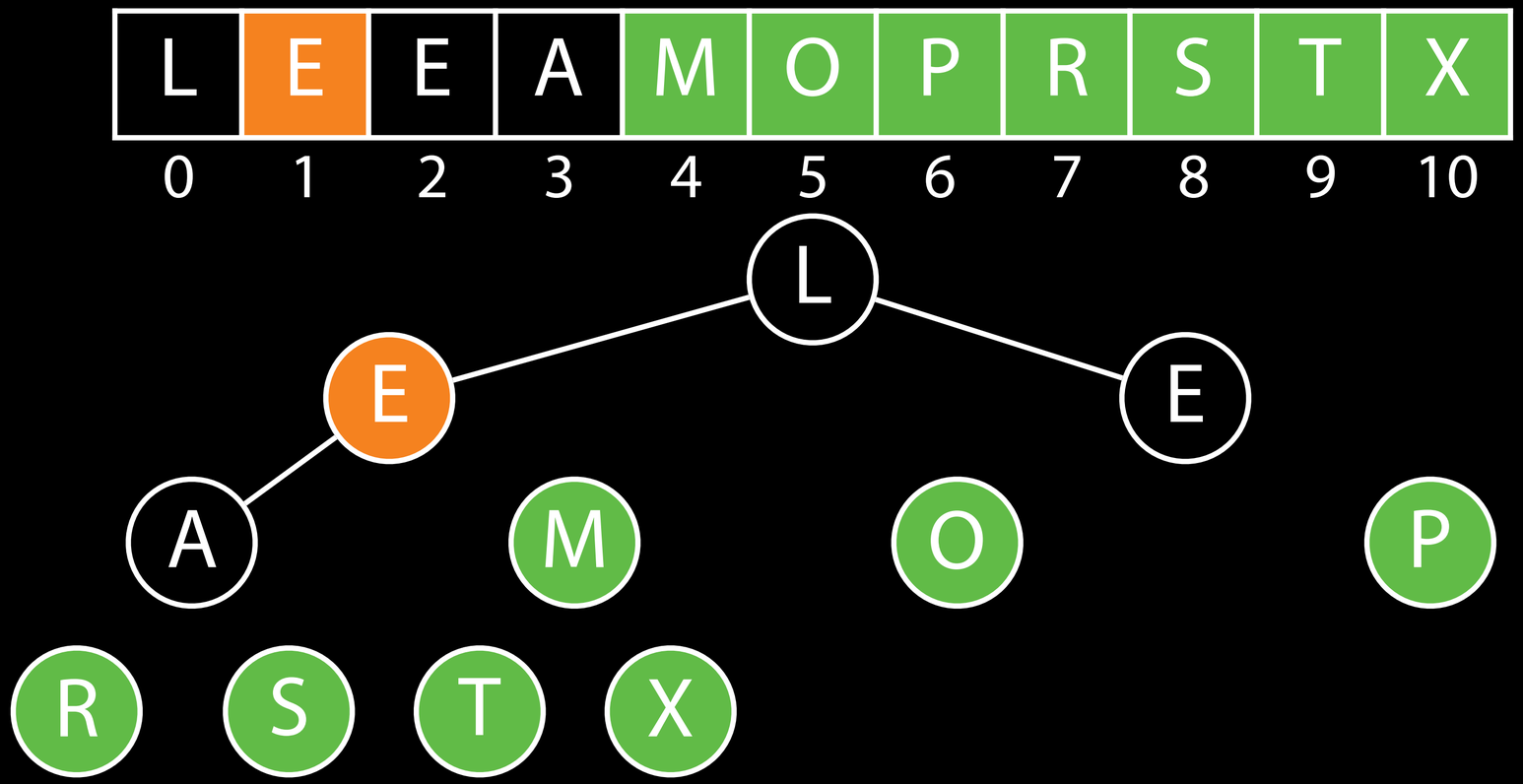
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



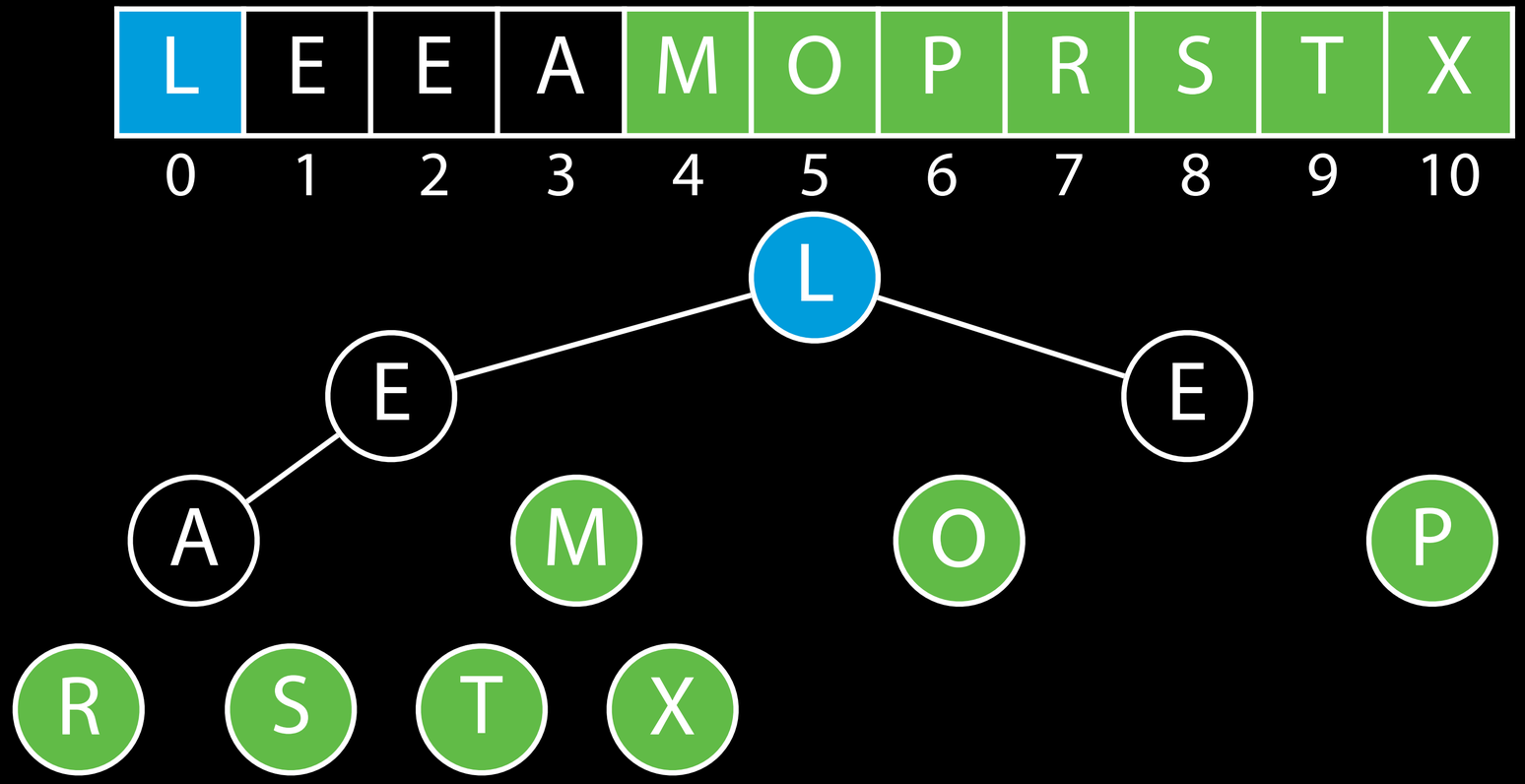
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



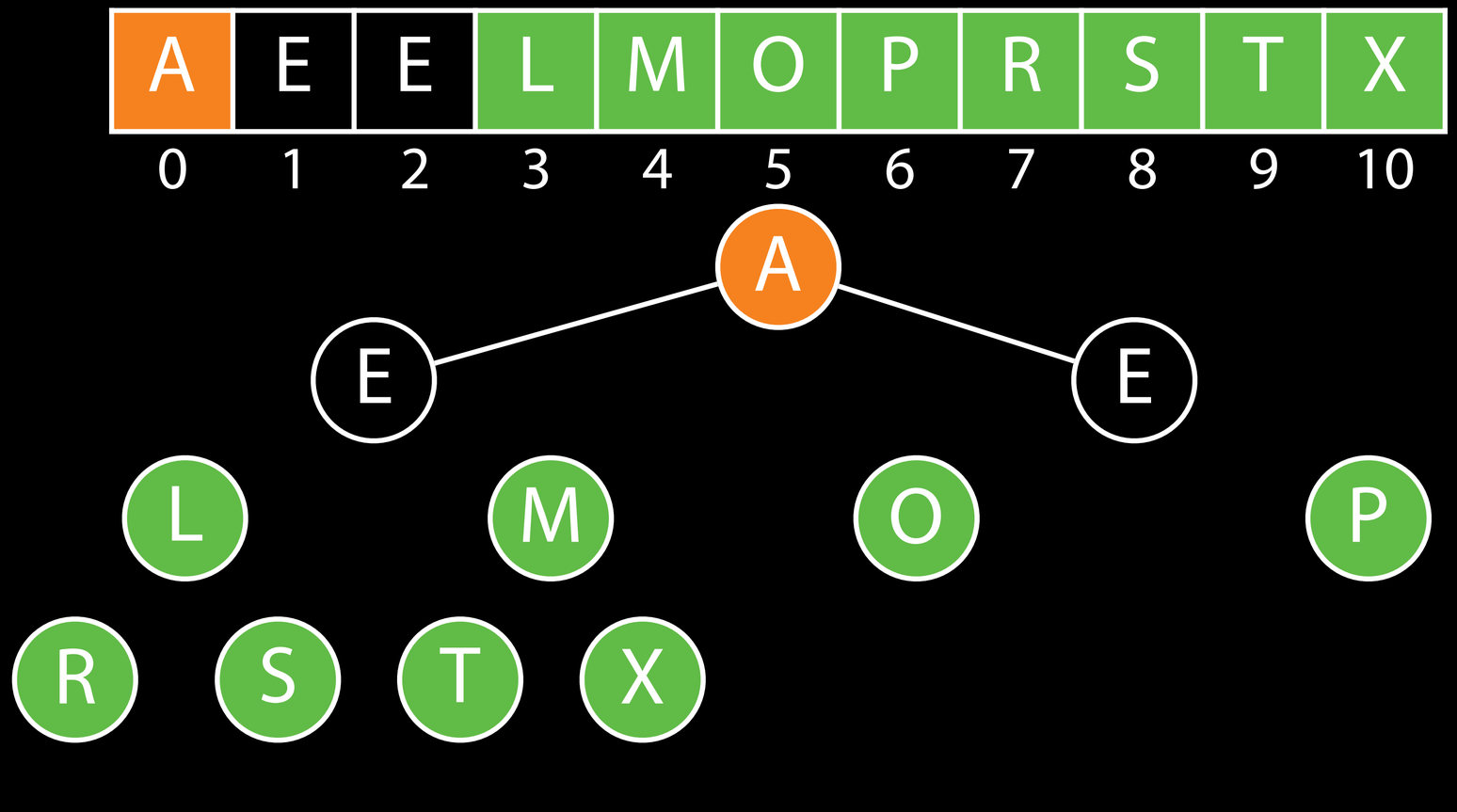
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



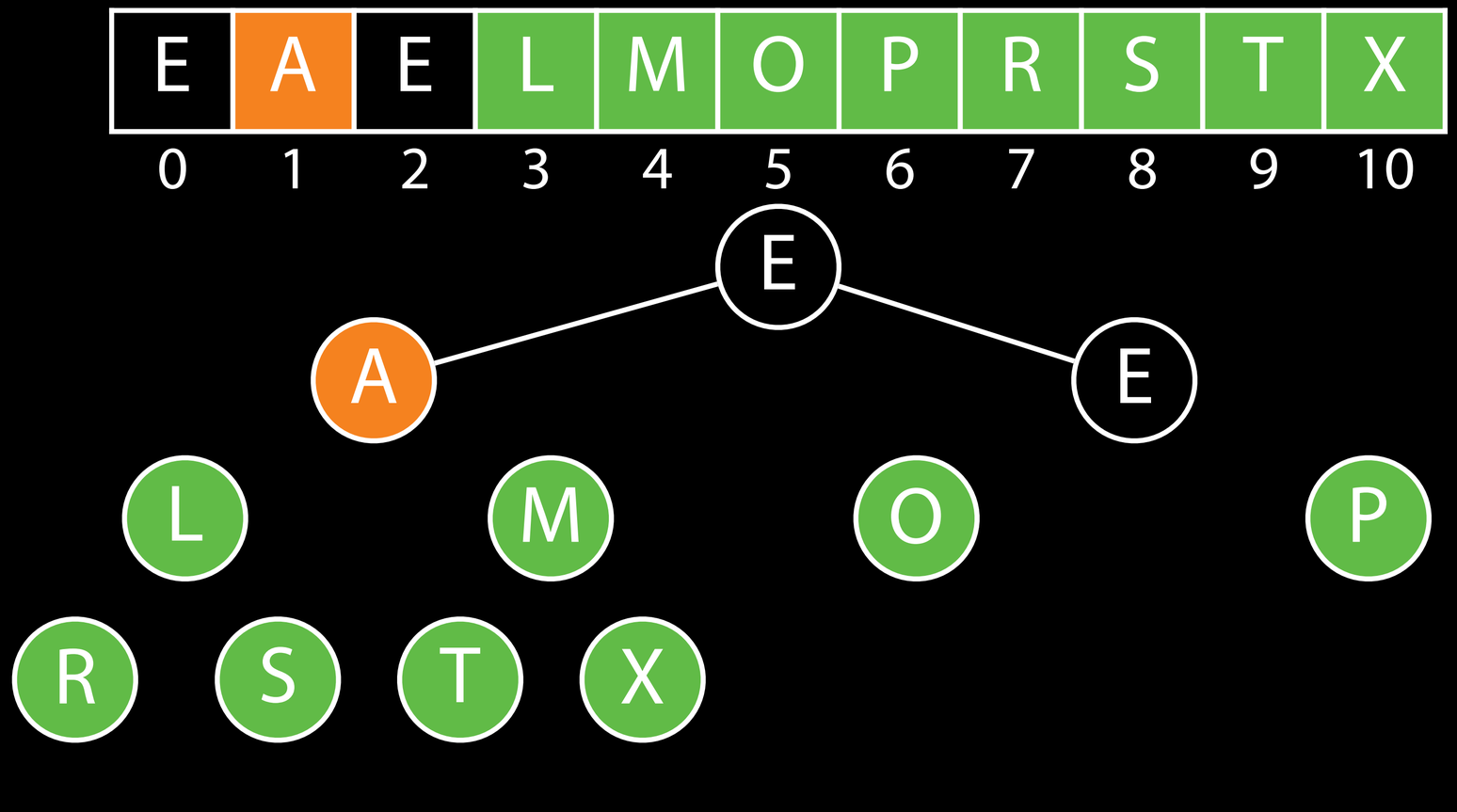
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



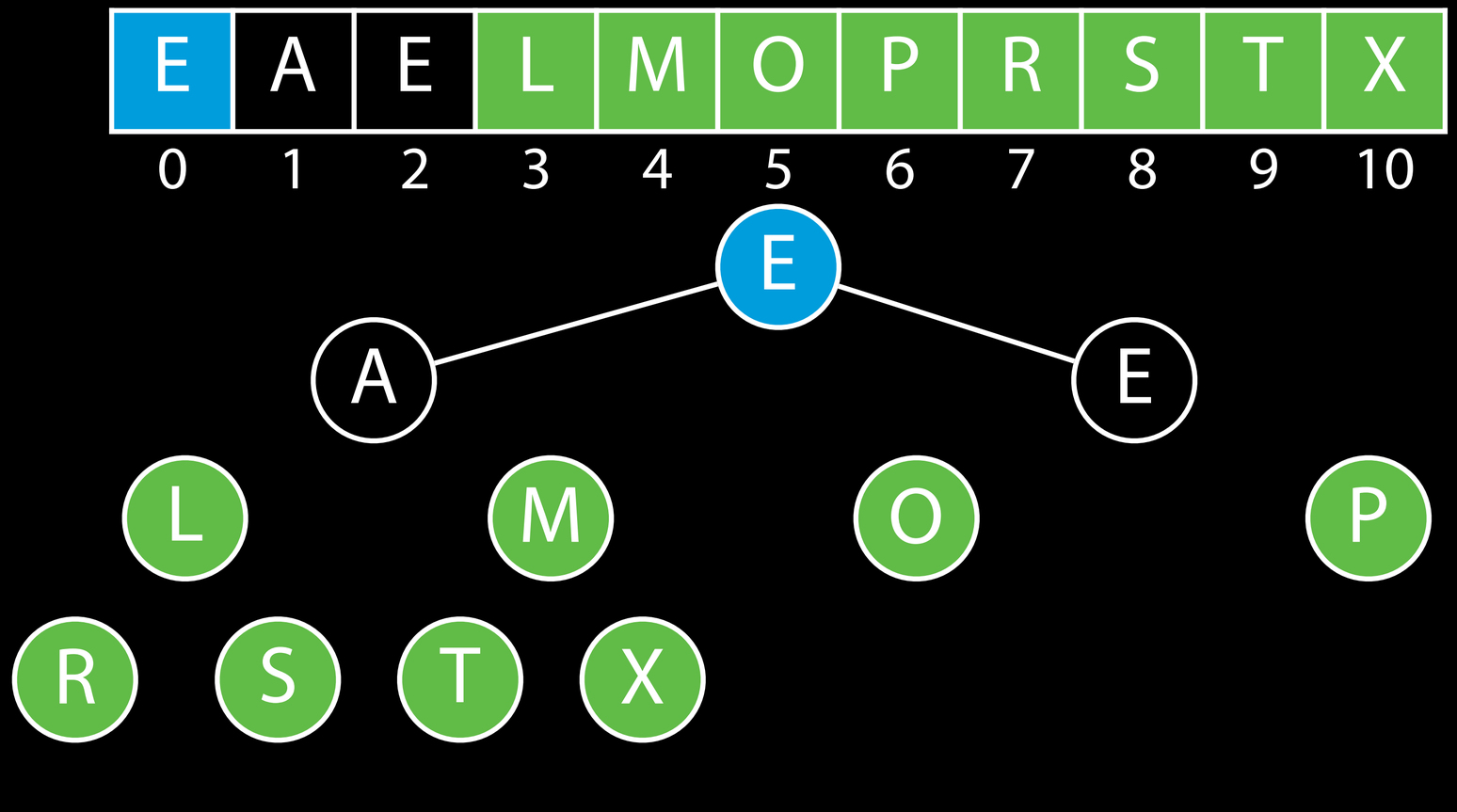
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



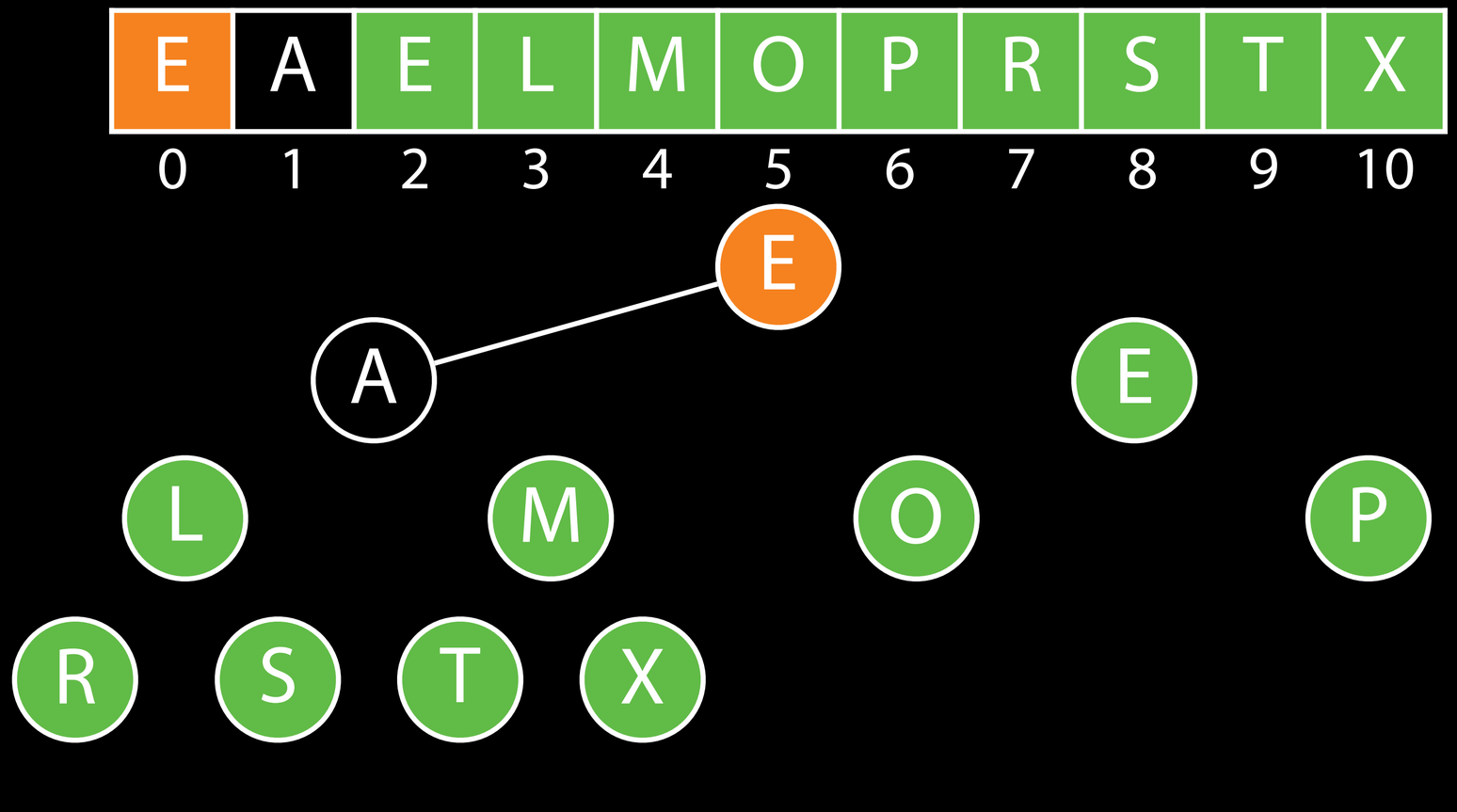
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



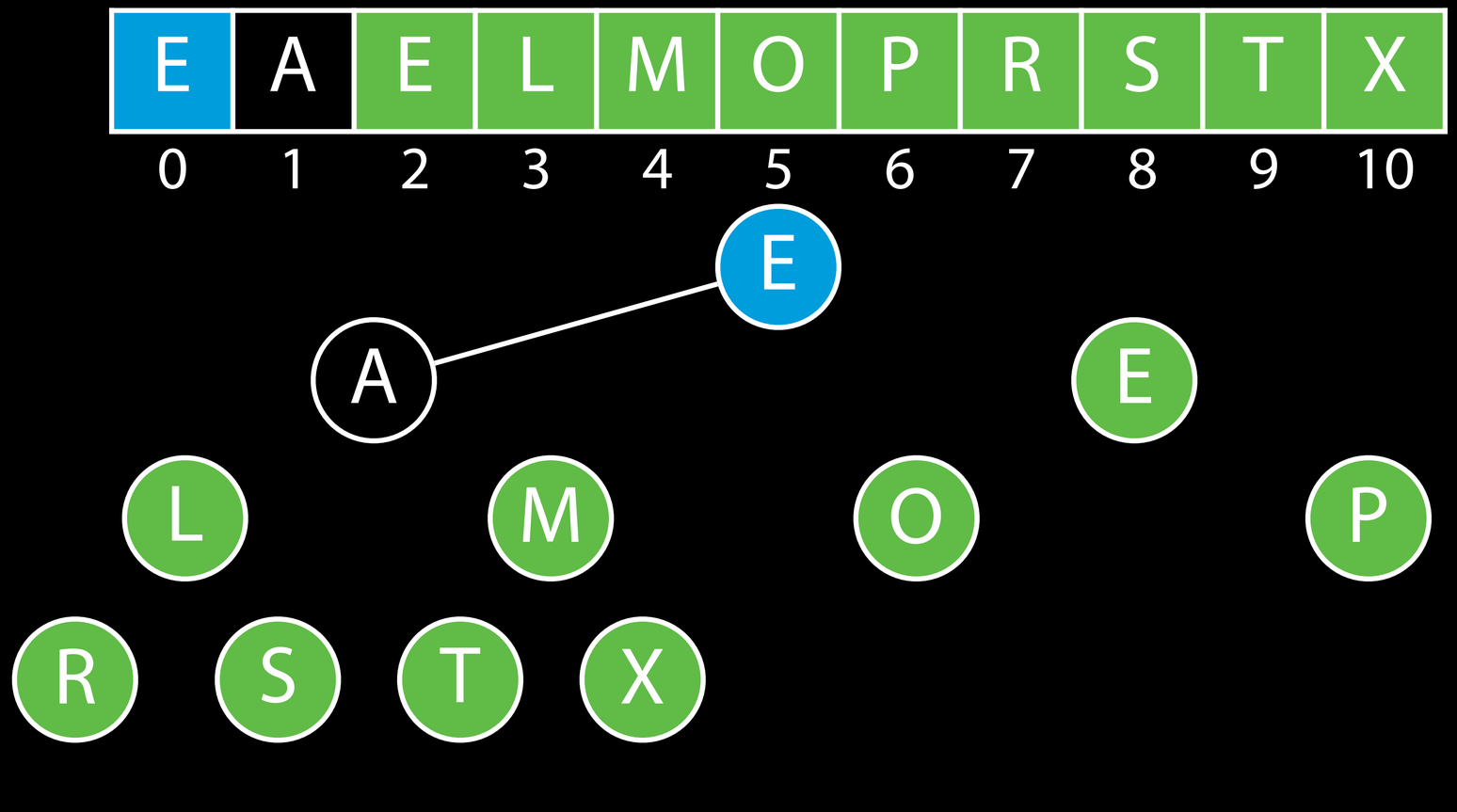
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



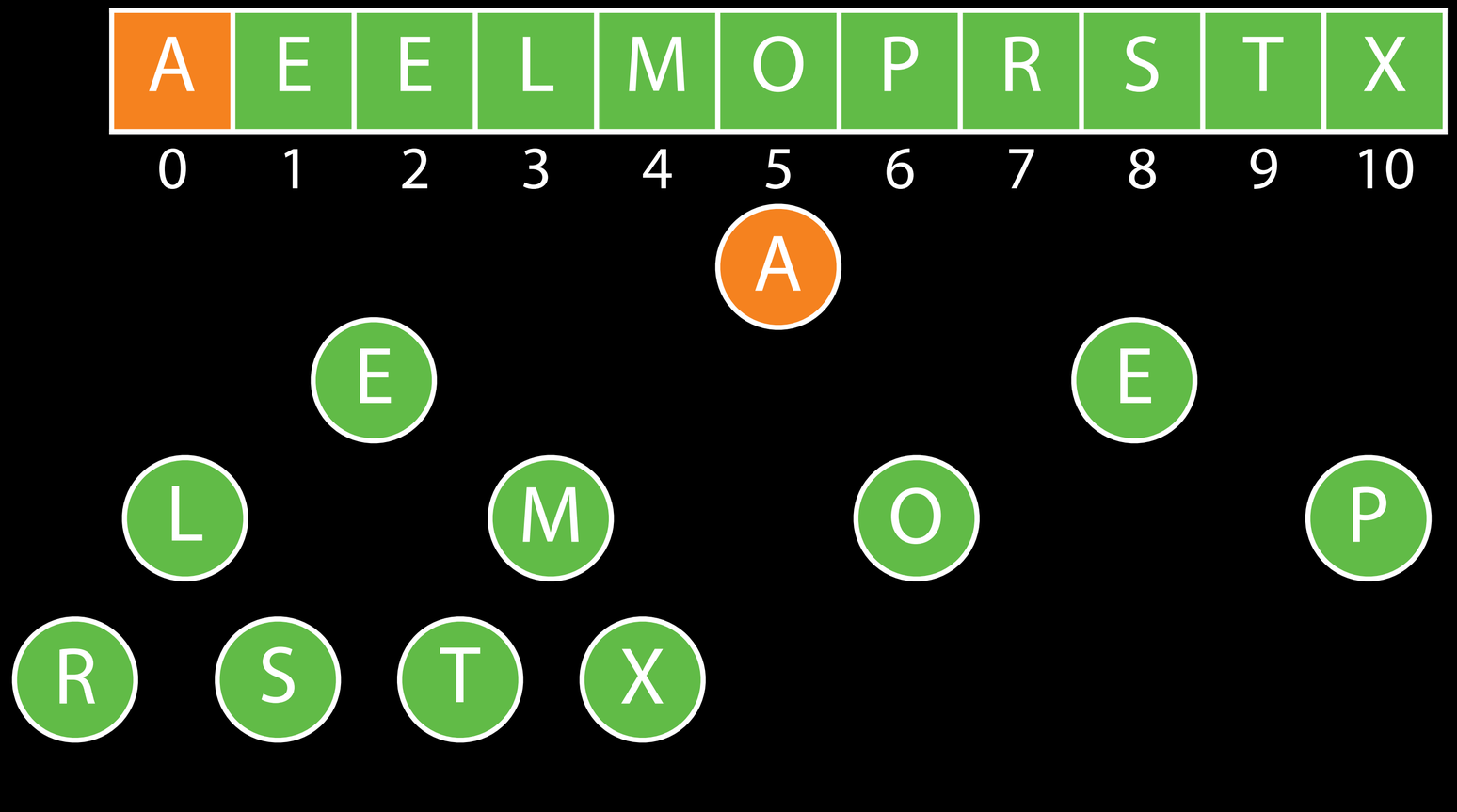
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



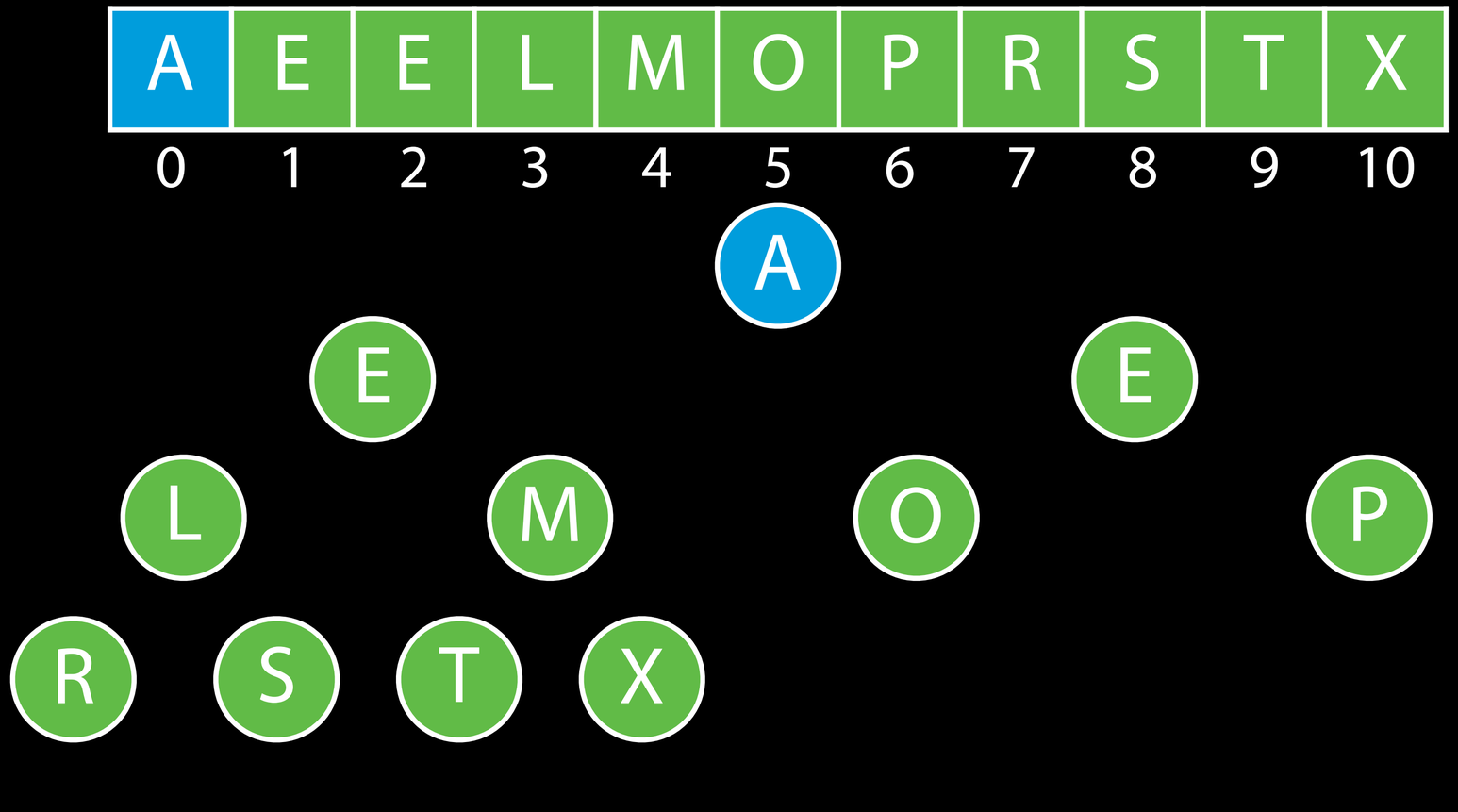
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



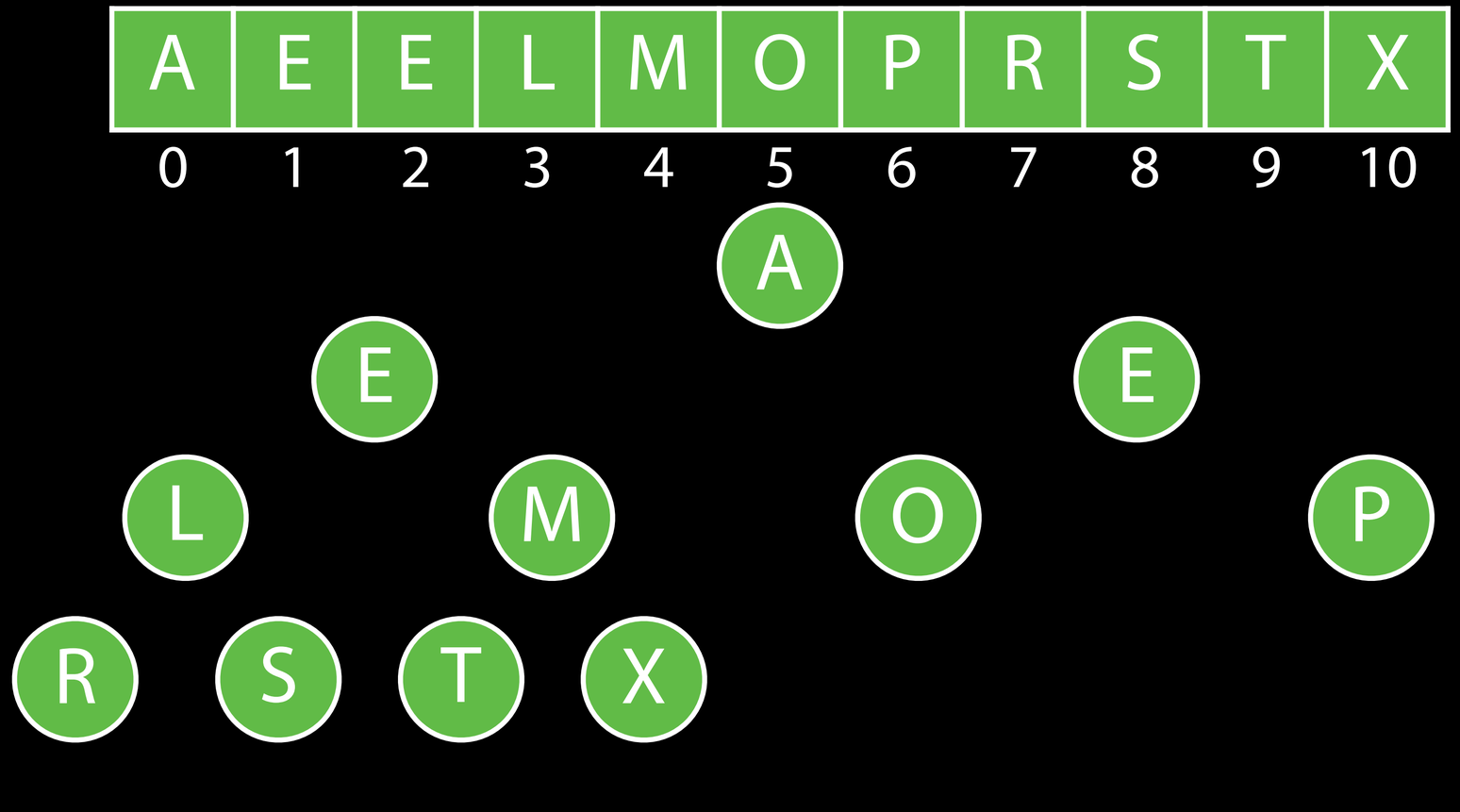
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



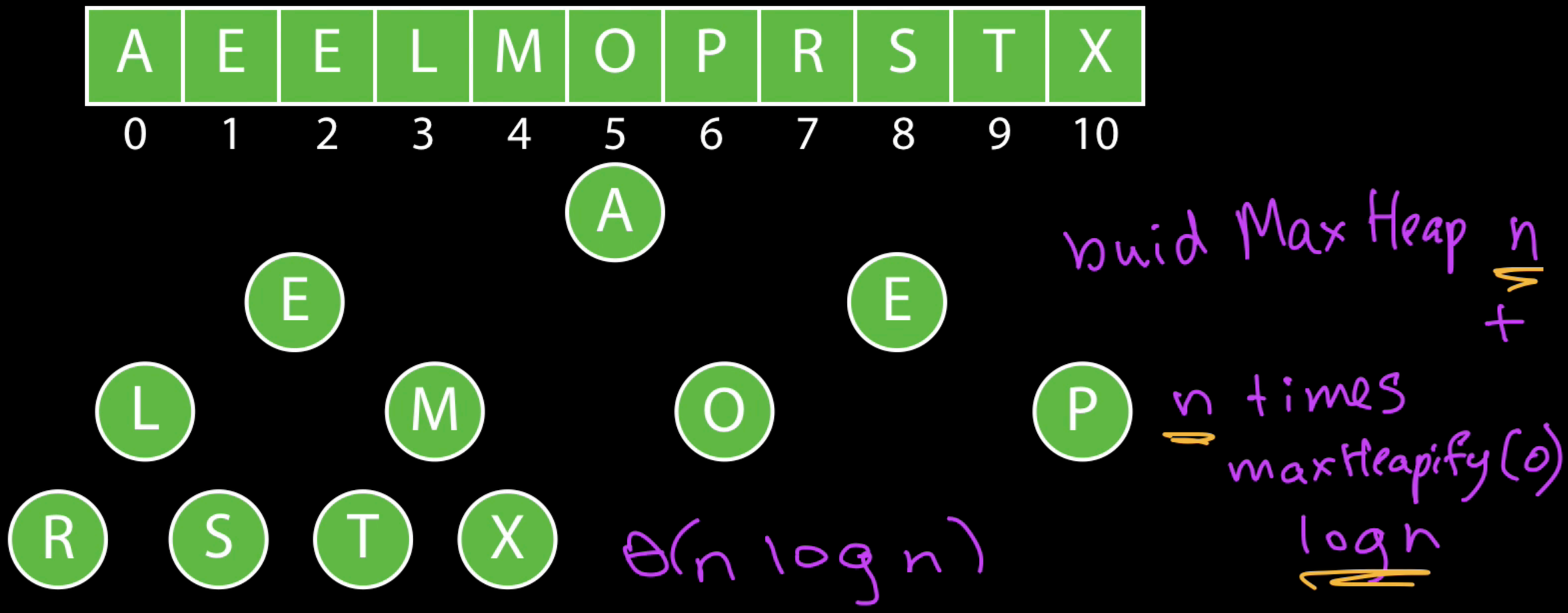
Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.



Heapsort

Repeatedly delete the largest remaining item by swapping with the last item.

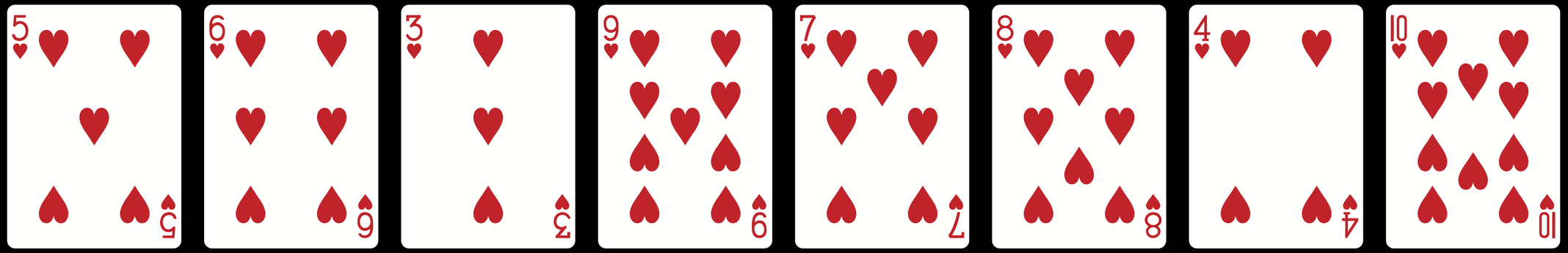


Merge sort

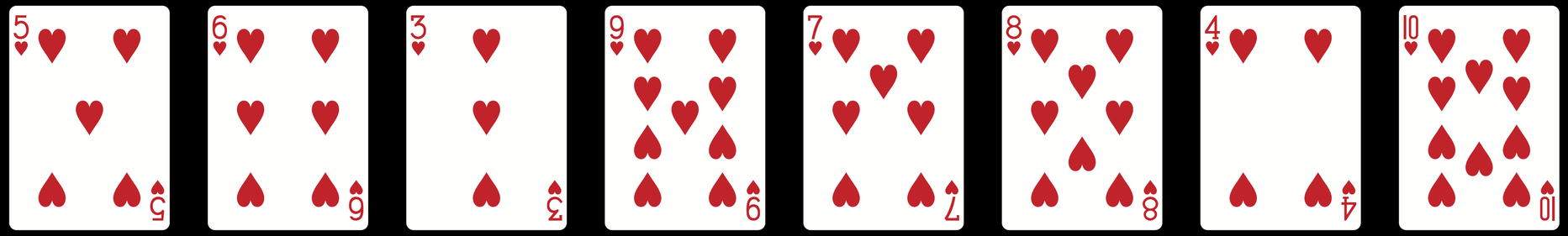
Sort the left half.

Sort the right half.

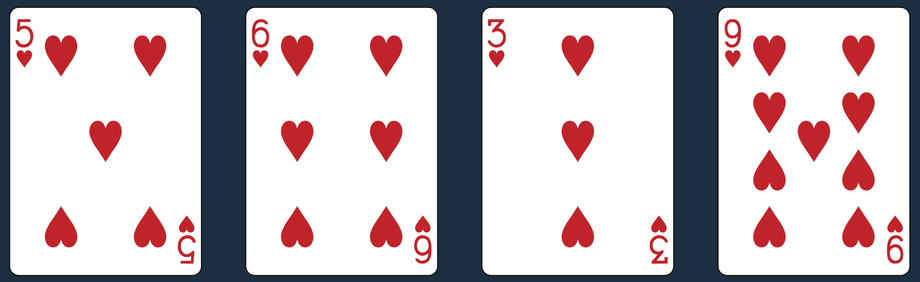
Merge the sorted halves.



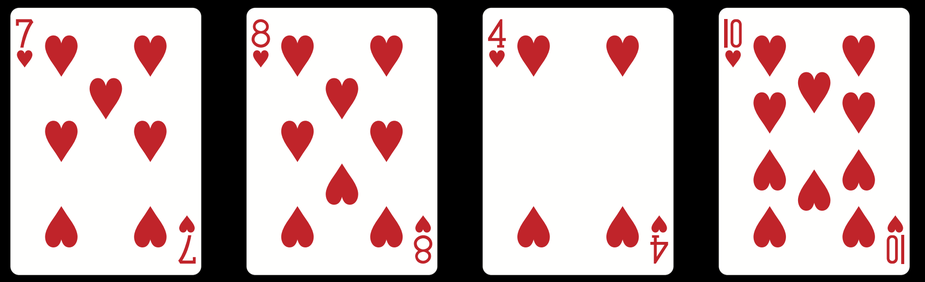
Merge sort



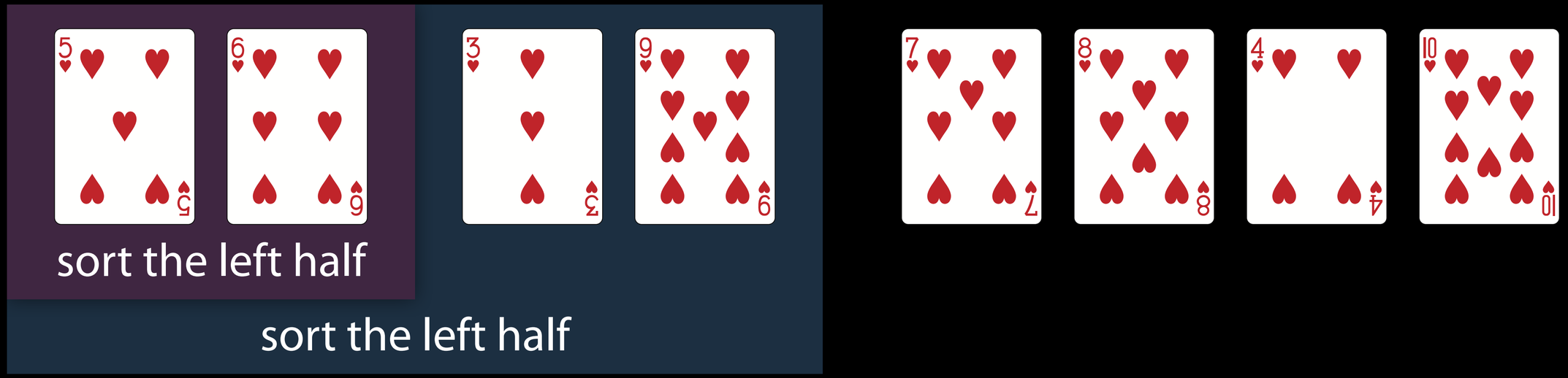
Merge sort



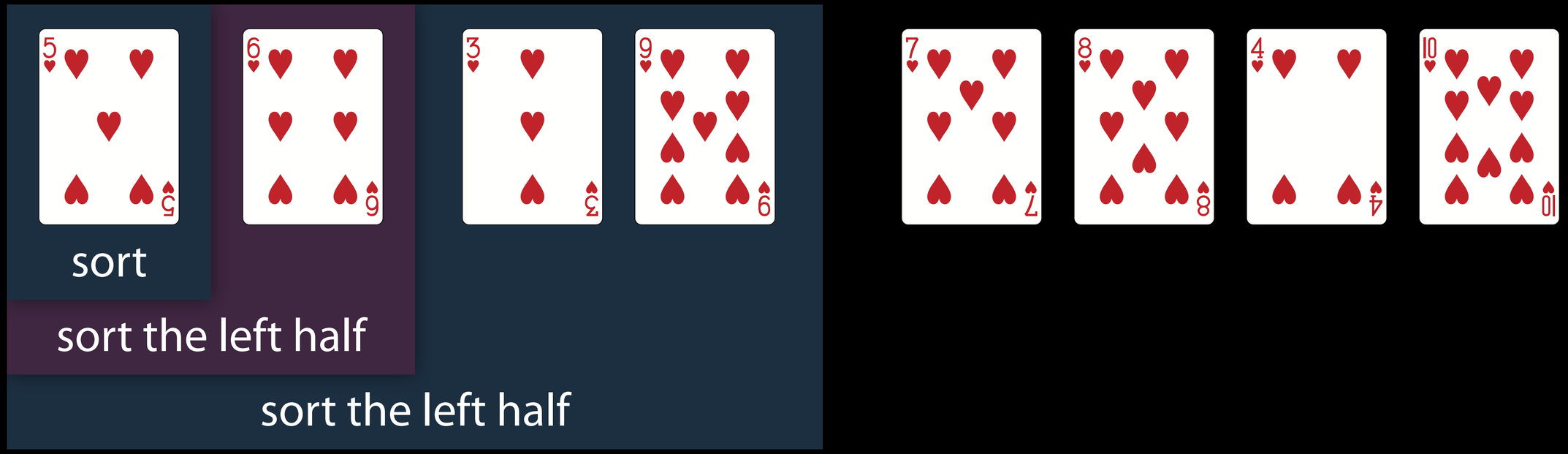
sort the left half



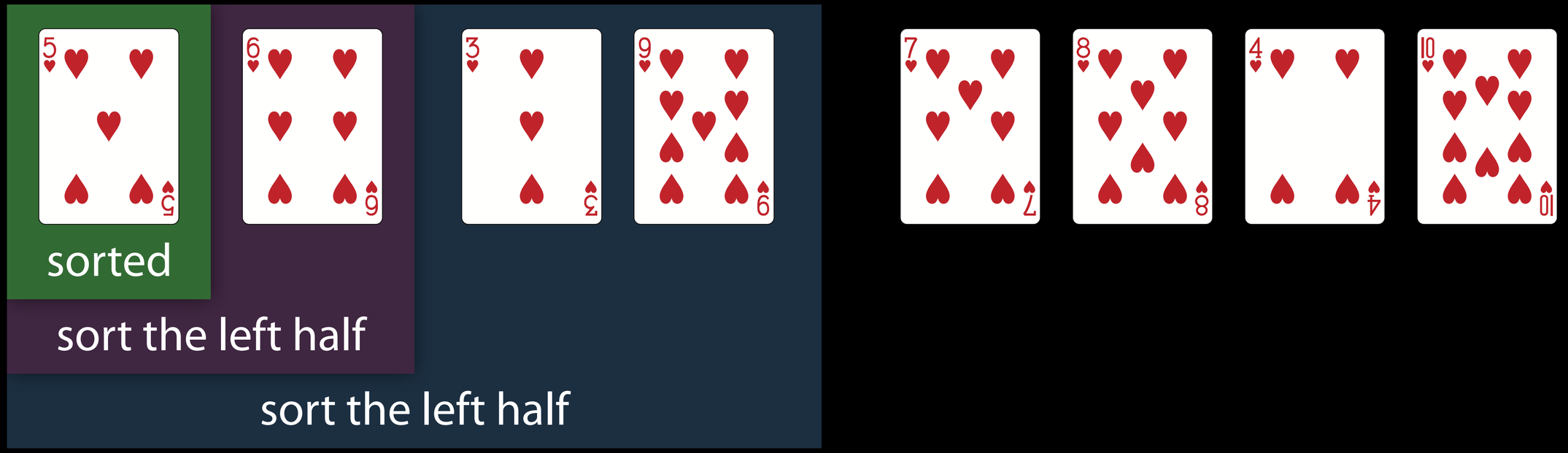
Merge sort



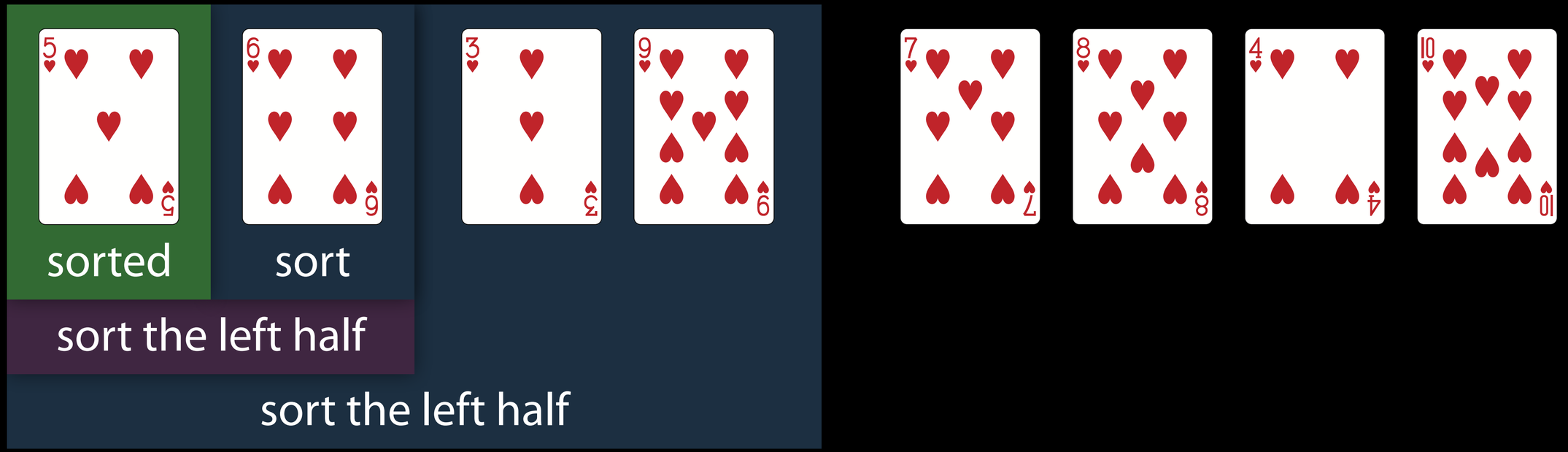
Merge sort



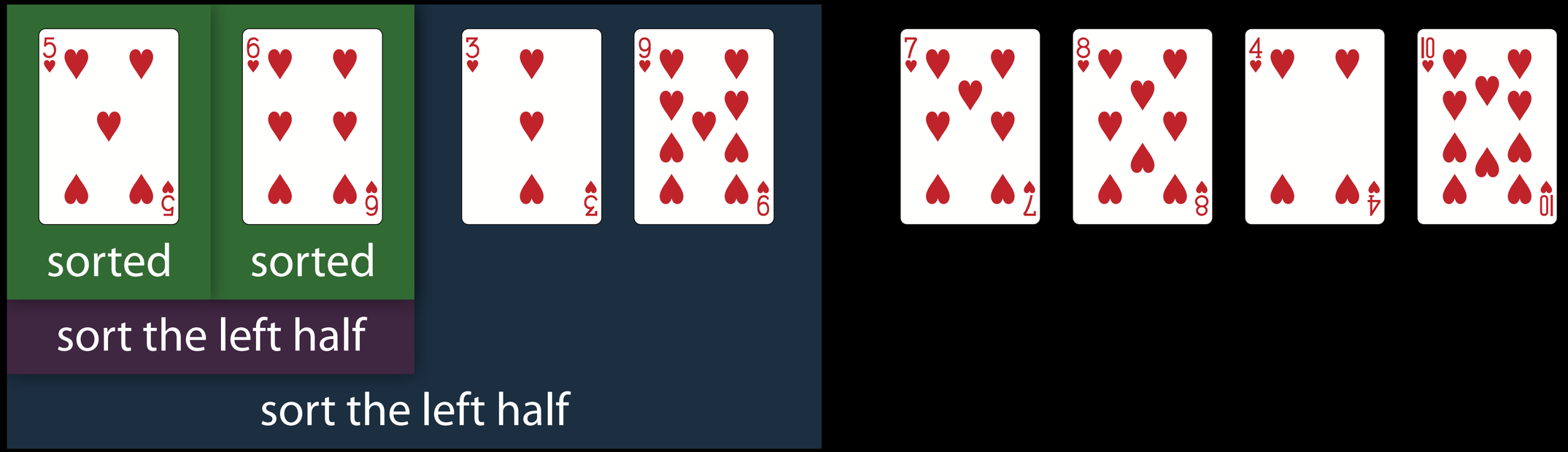
Merge sort



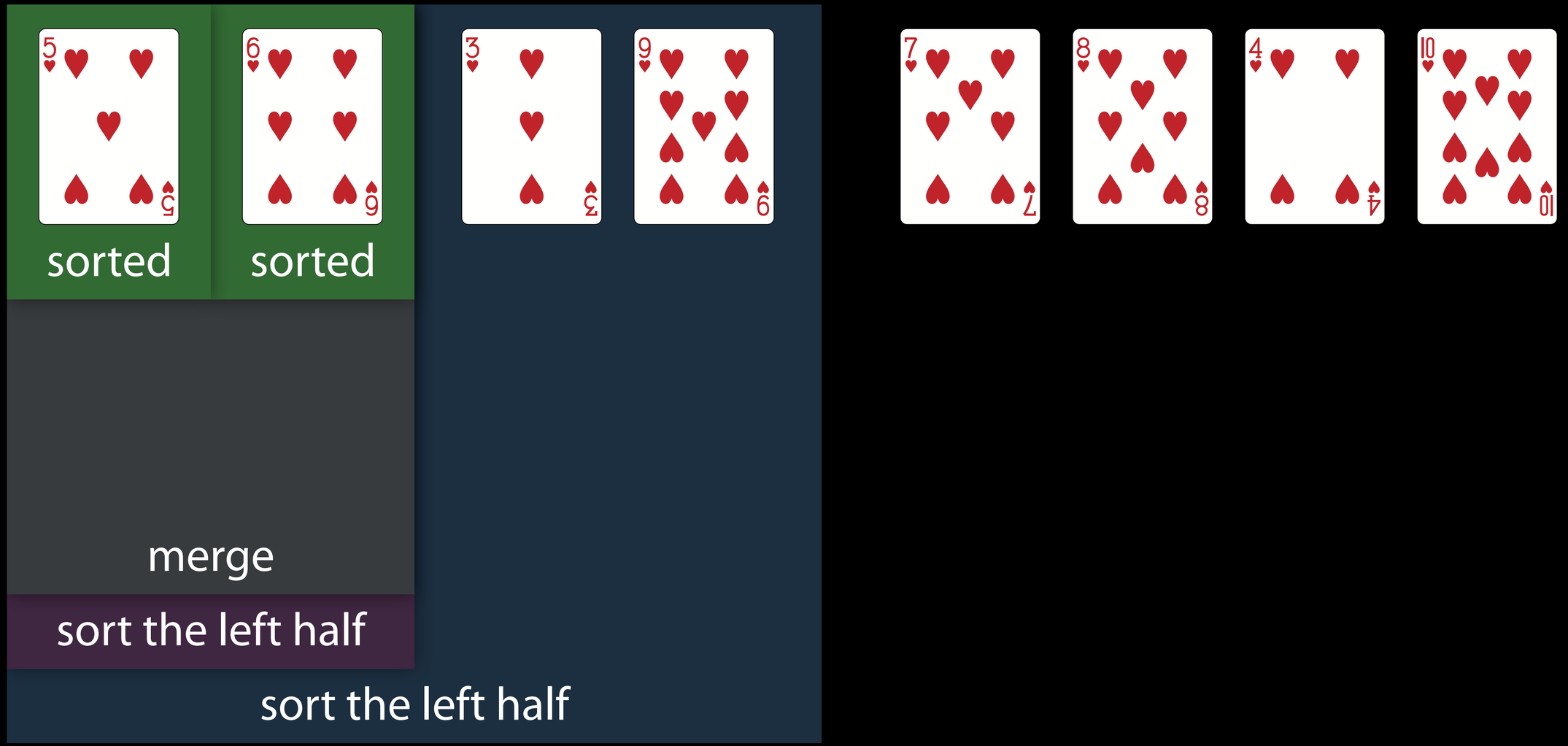
Merge sort



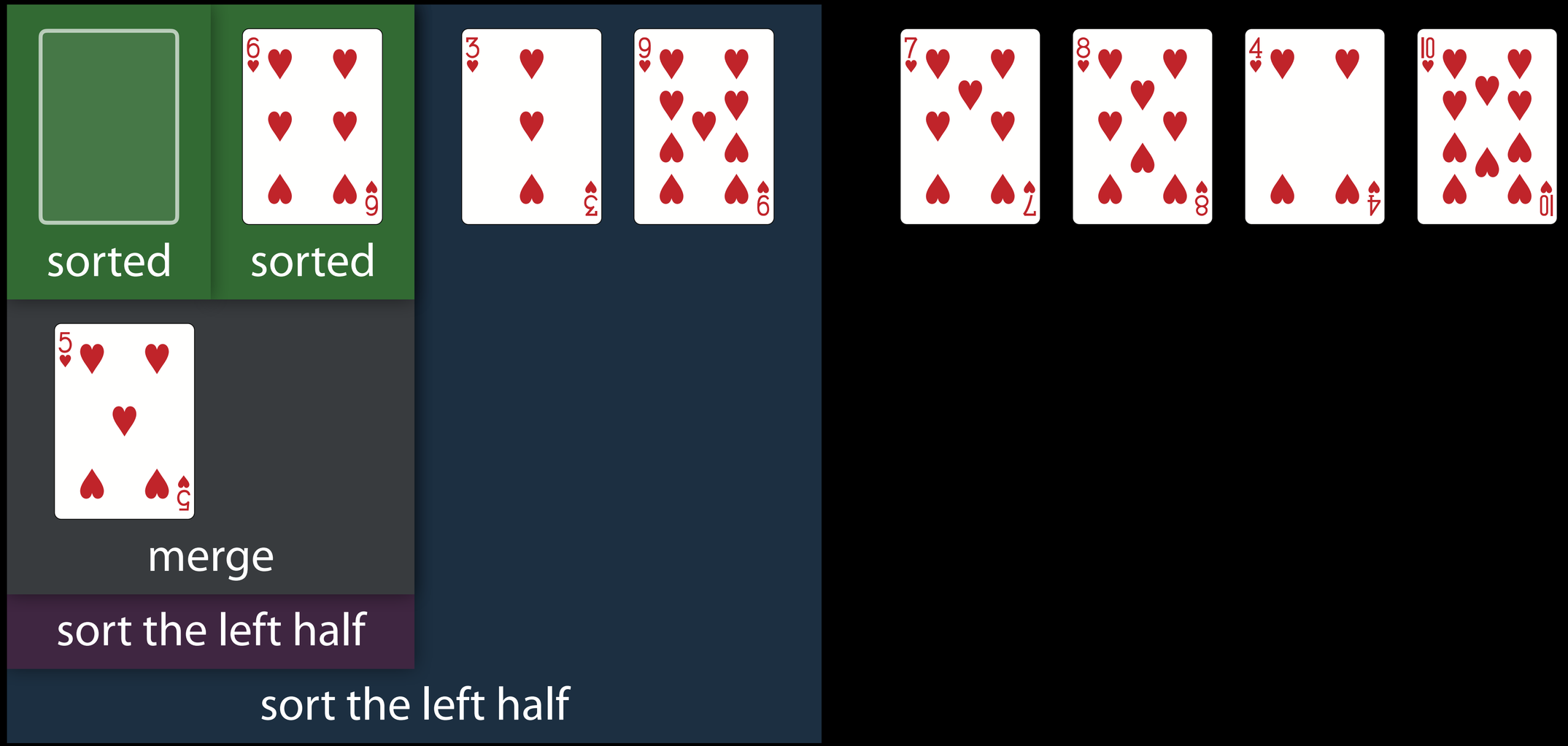
Merge sort



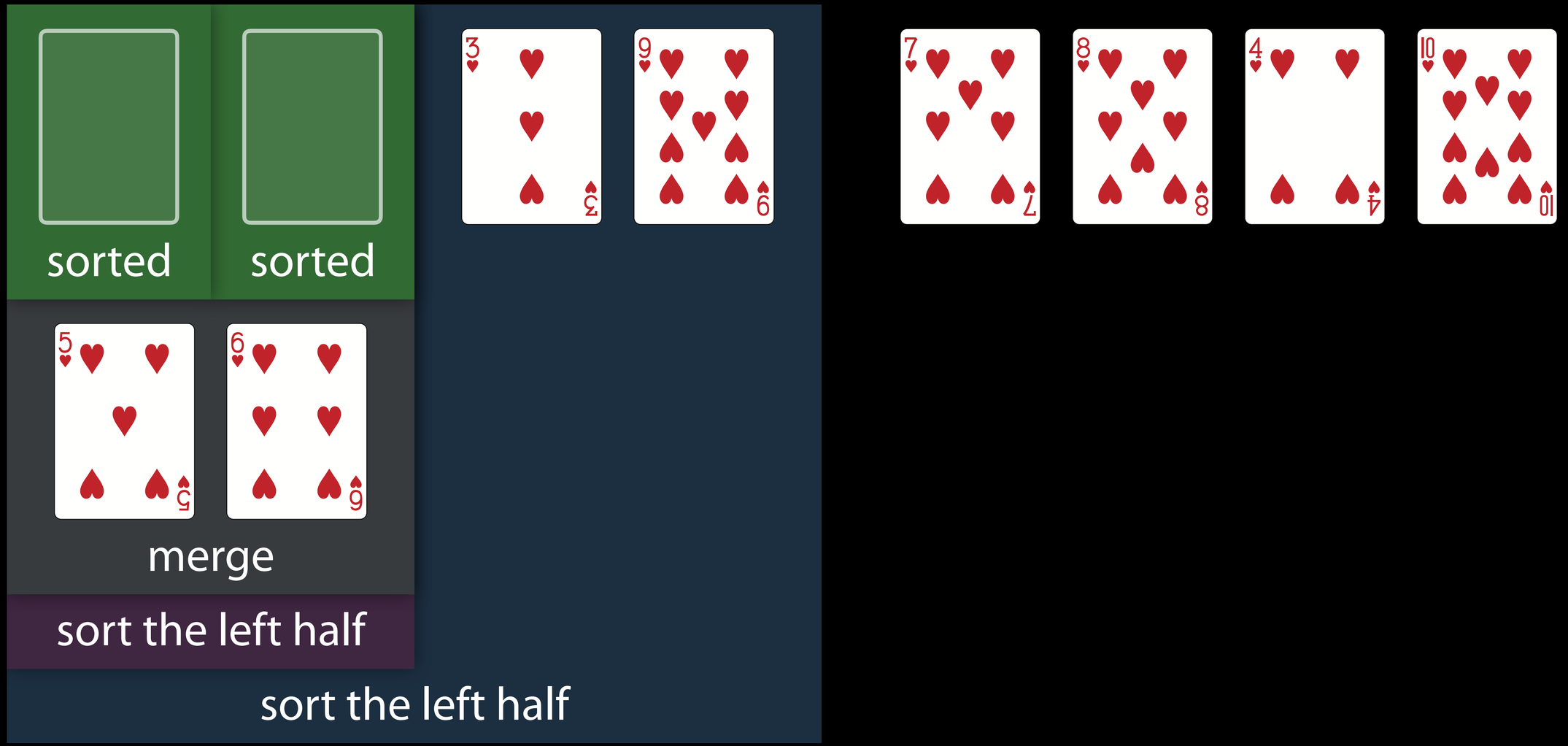
Merge sort



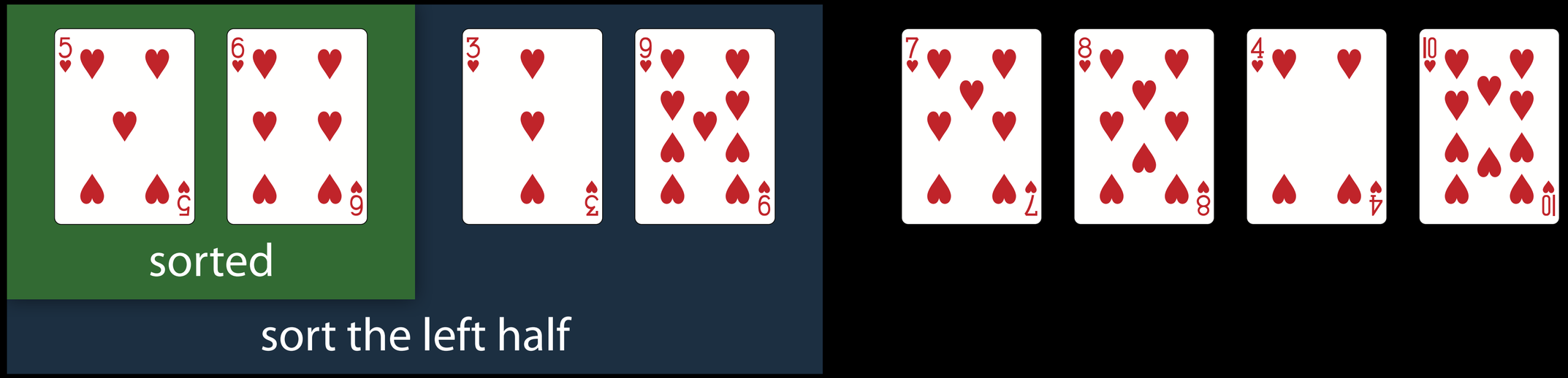
Merge sort



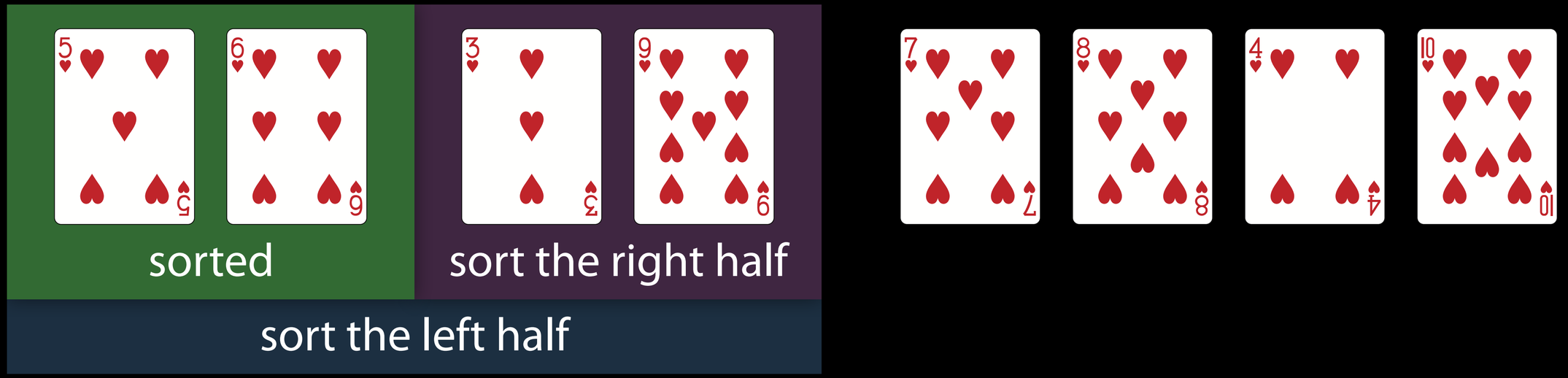
Merge sort



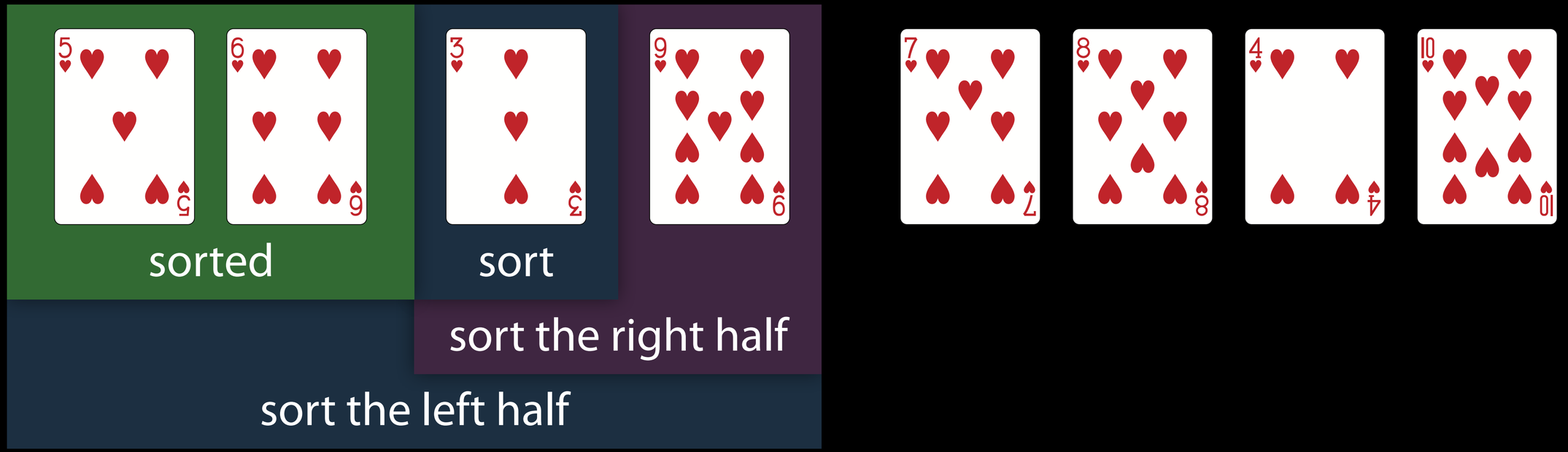
Merge sort



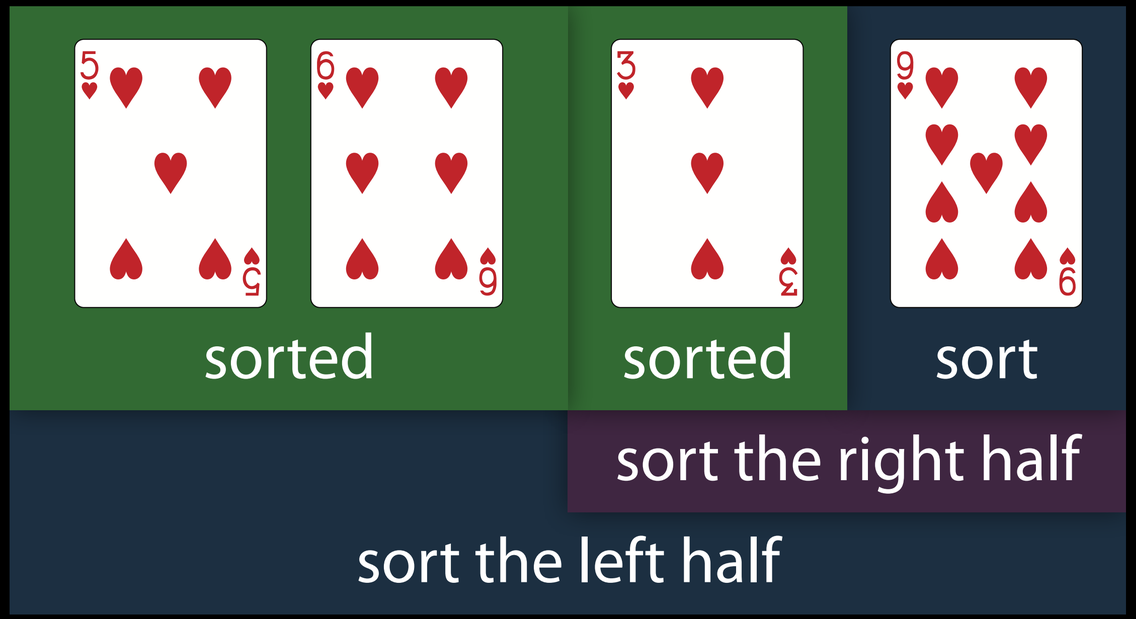
Merge sort

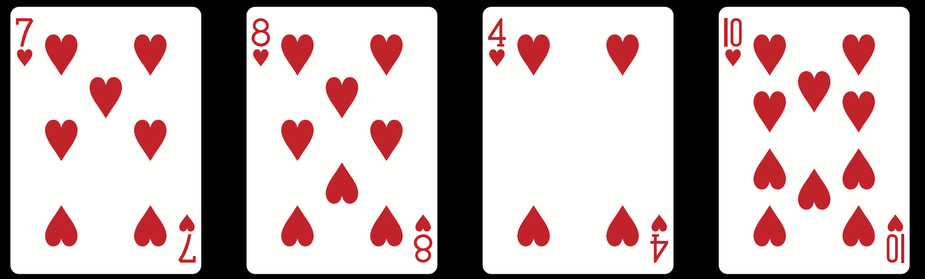


Merge sort

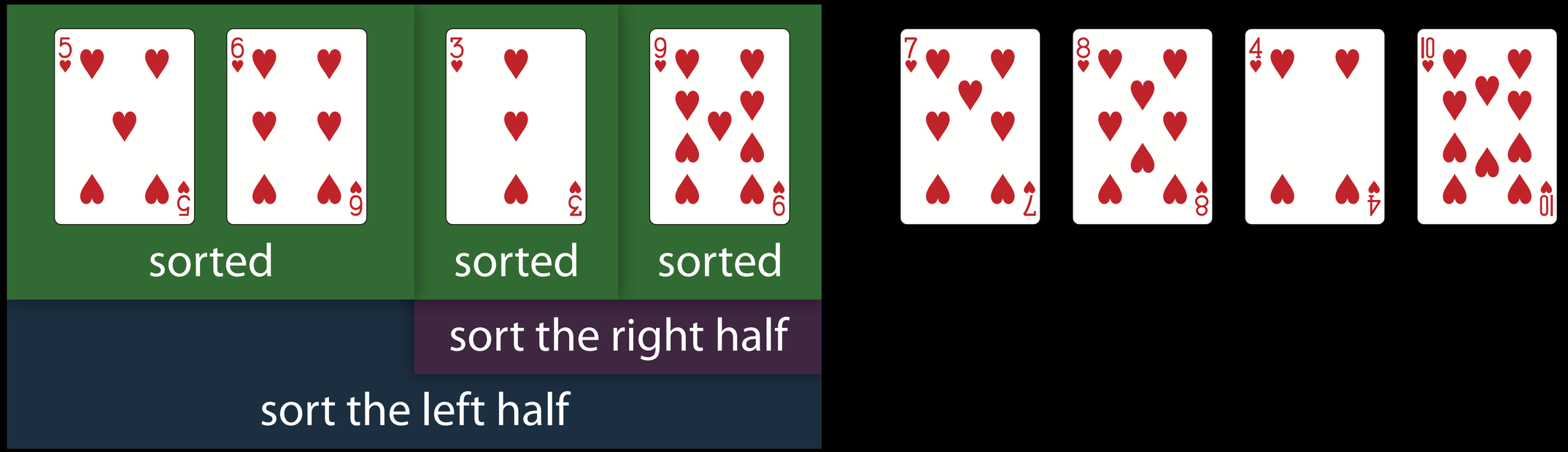


Merge sort

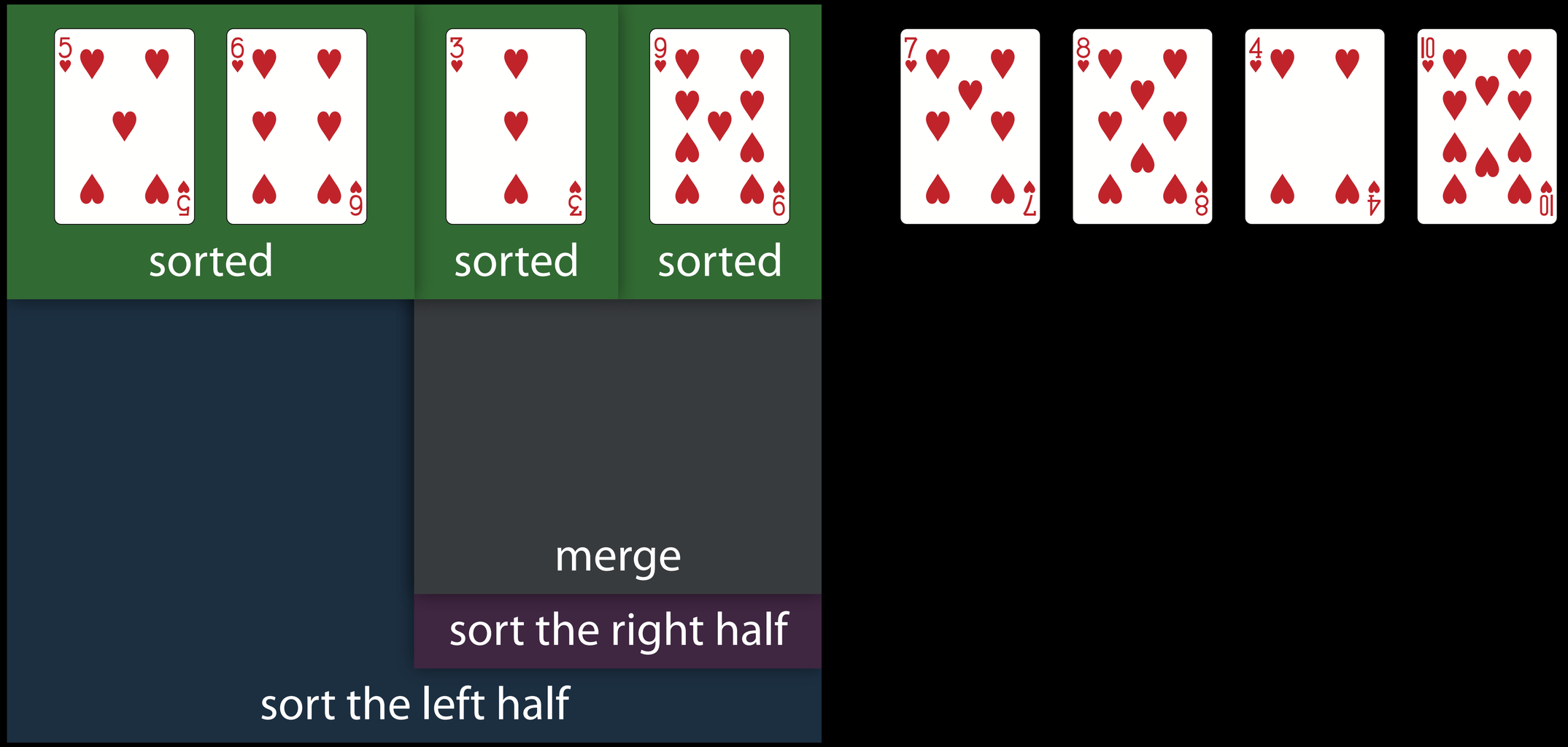




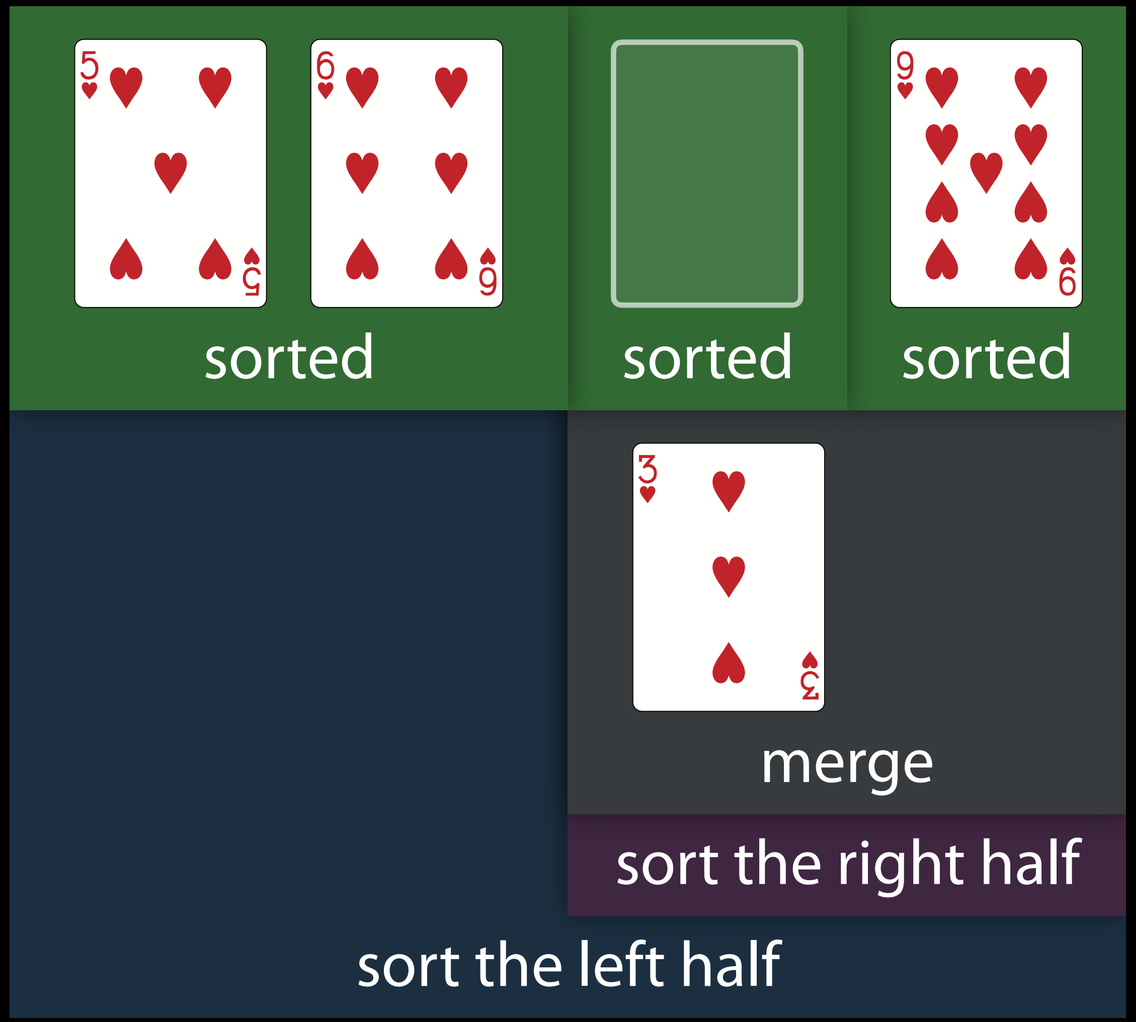
Merge sort

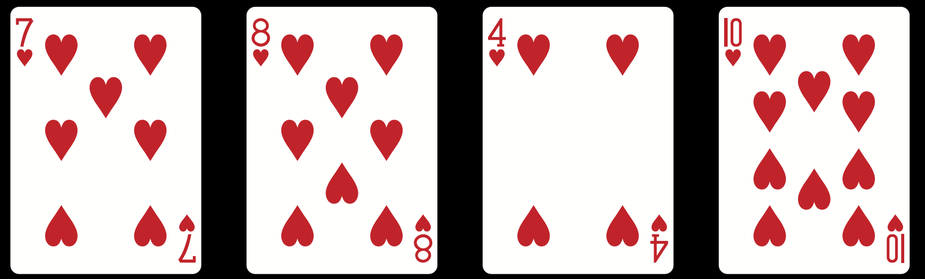


Merge sort

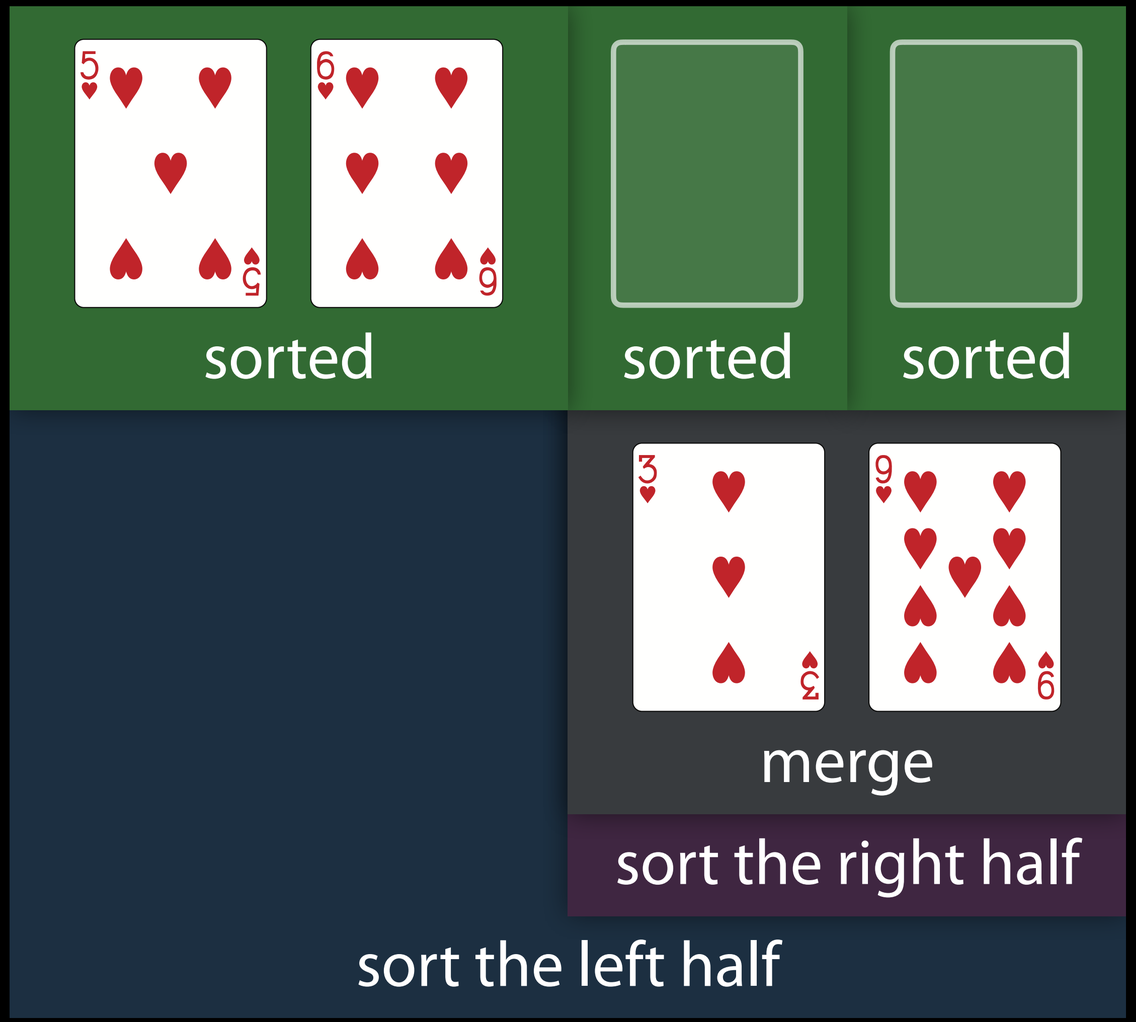

sorted
sorted
sorted
merge
sort the right half
sort the left half

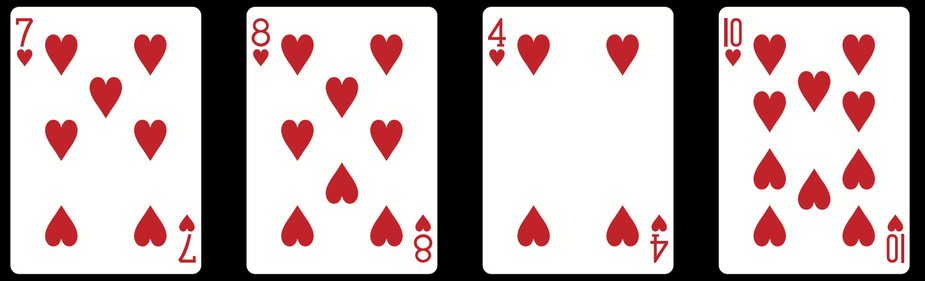
Merge sort


sorted
sorted
sorted
merge
sort the right half
sort the left half

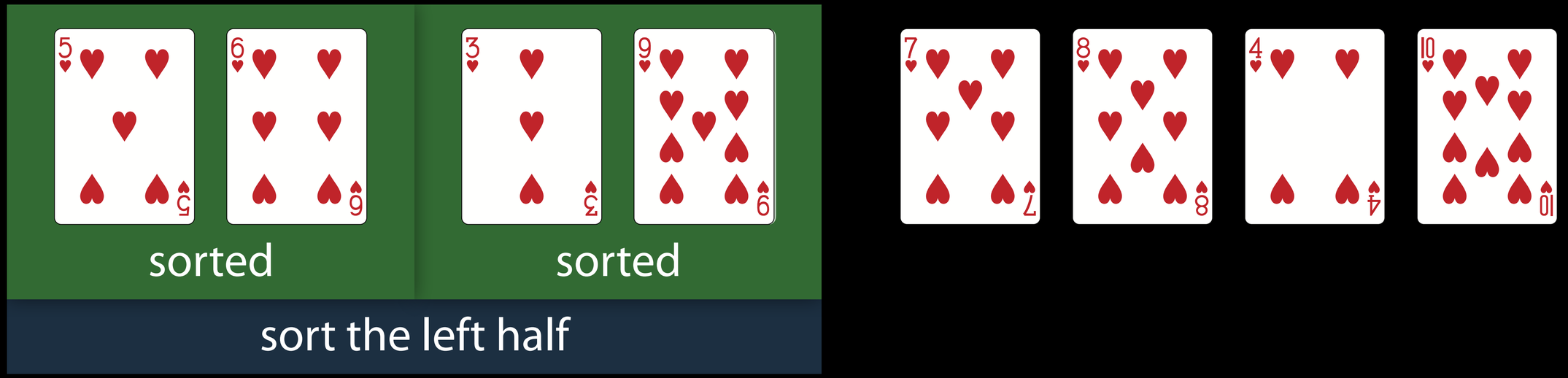


Merge sort

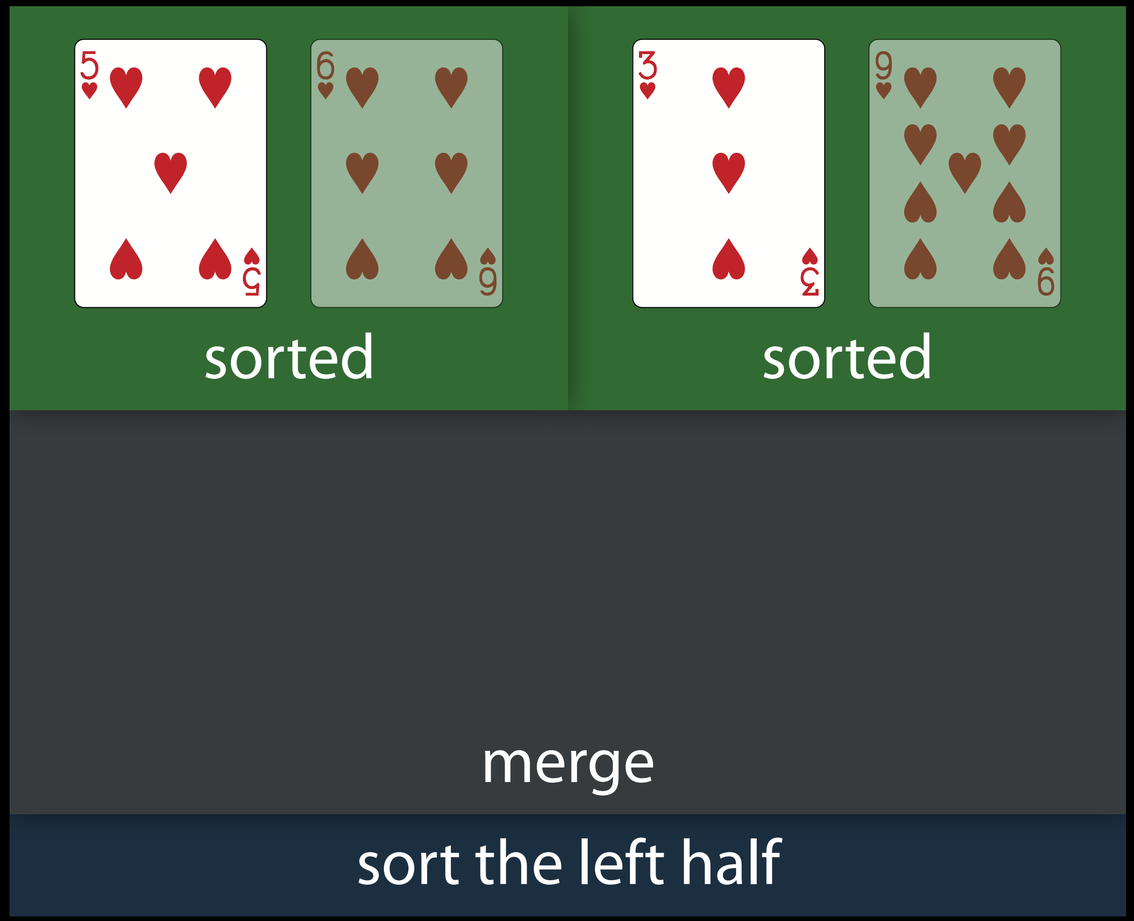

sorted
sorted
sorted
merge
sort the right half
sort the left half

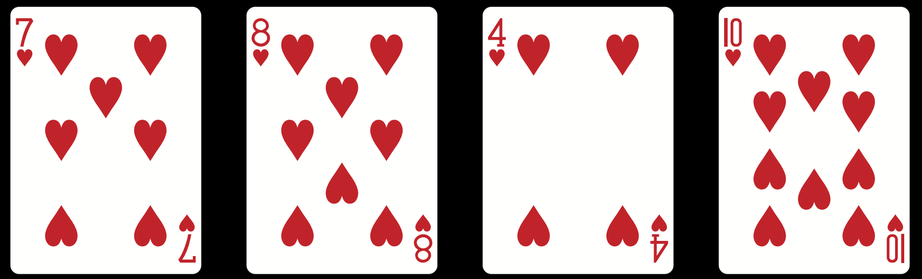


Merge sort

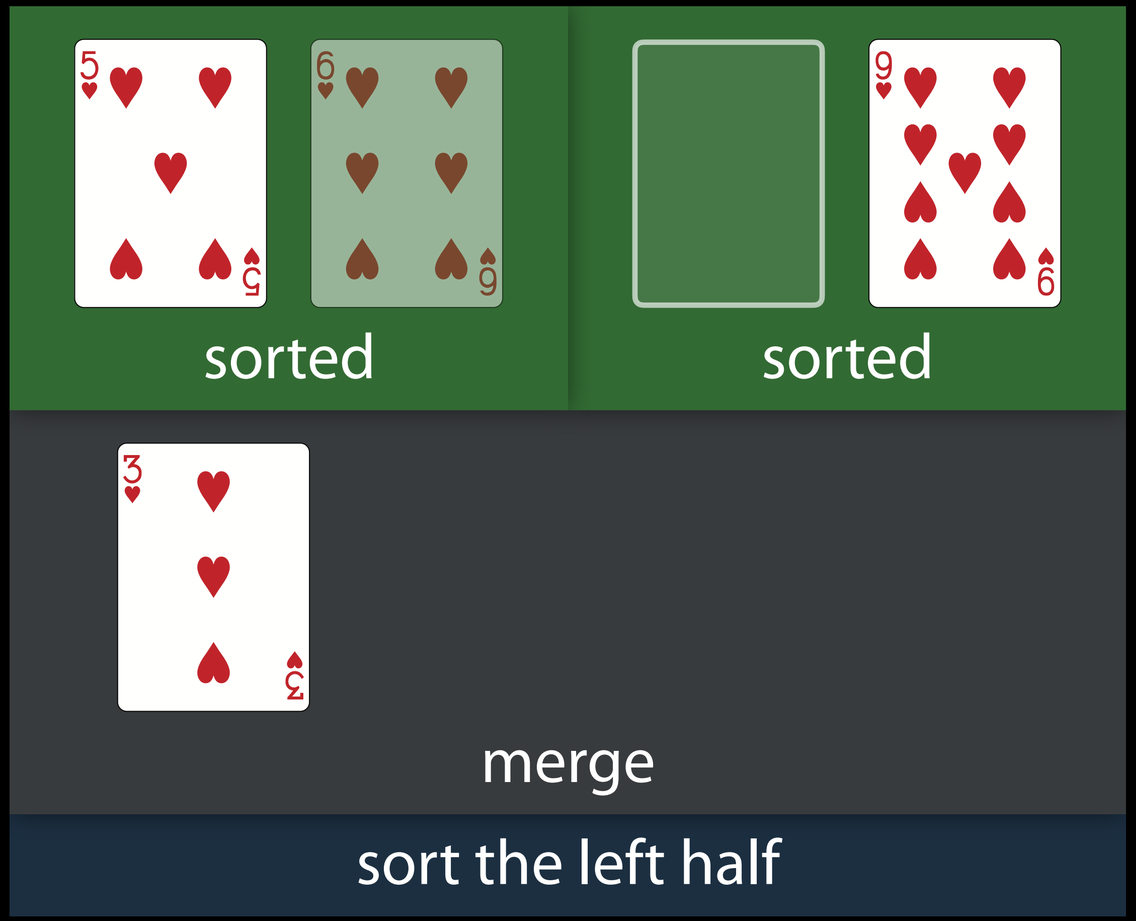


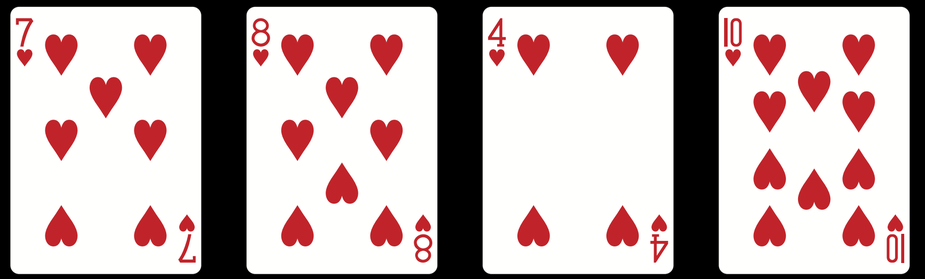
Merge sort



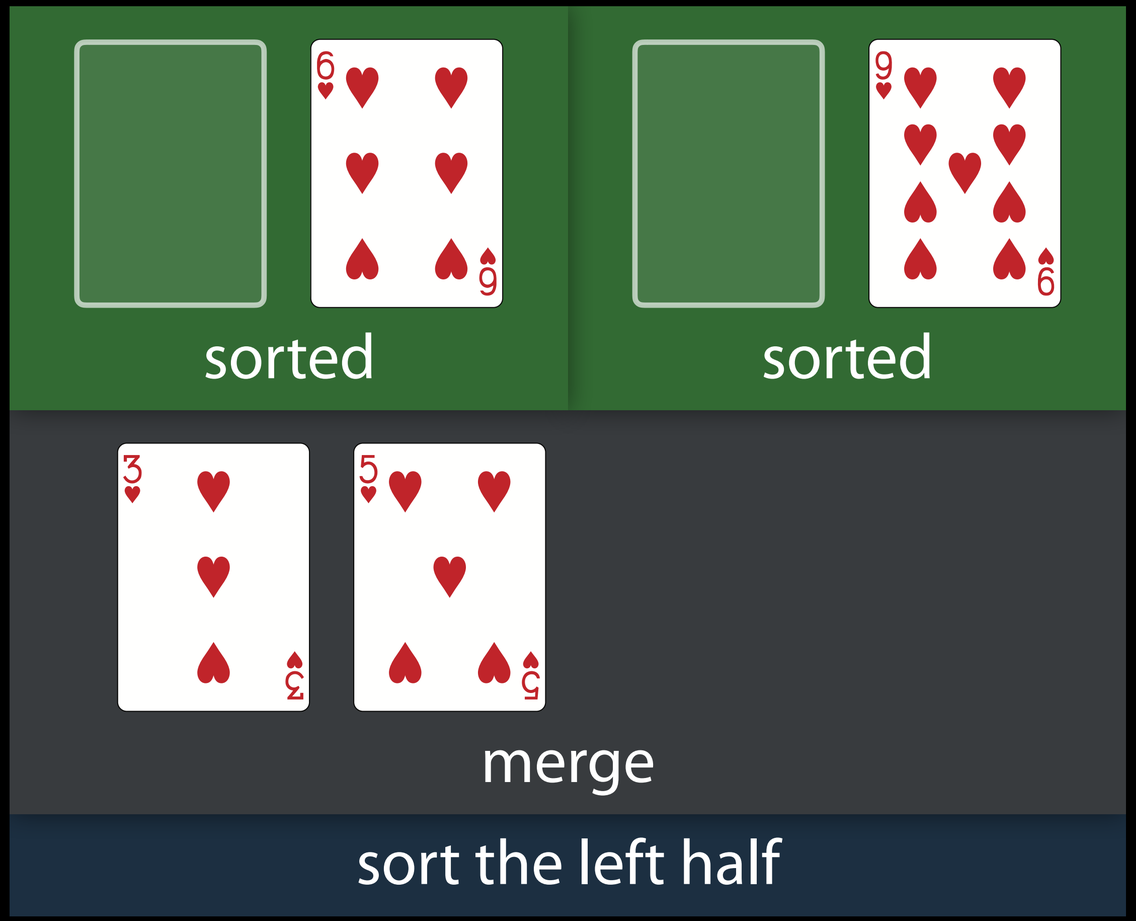


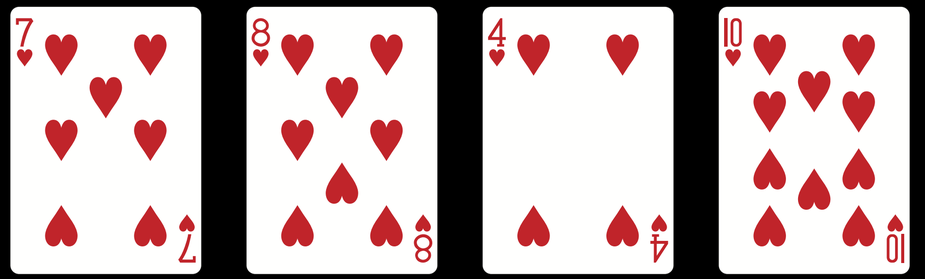
Merge sort



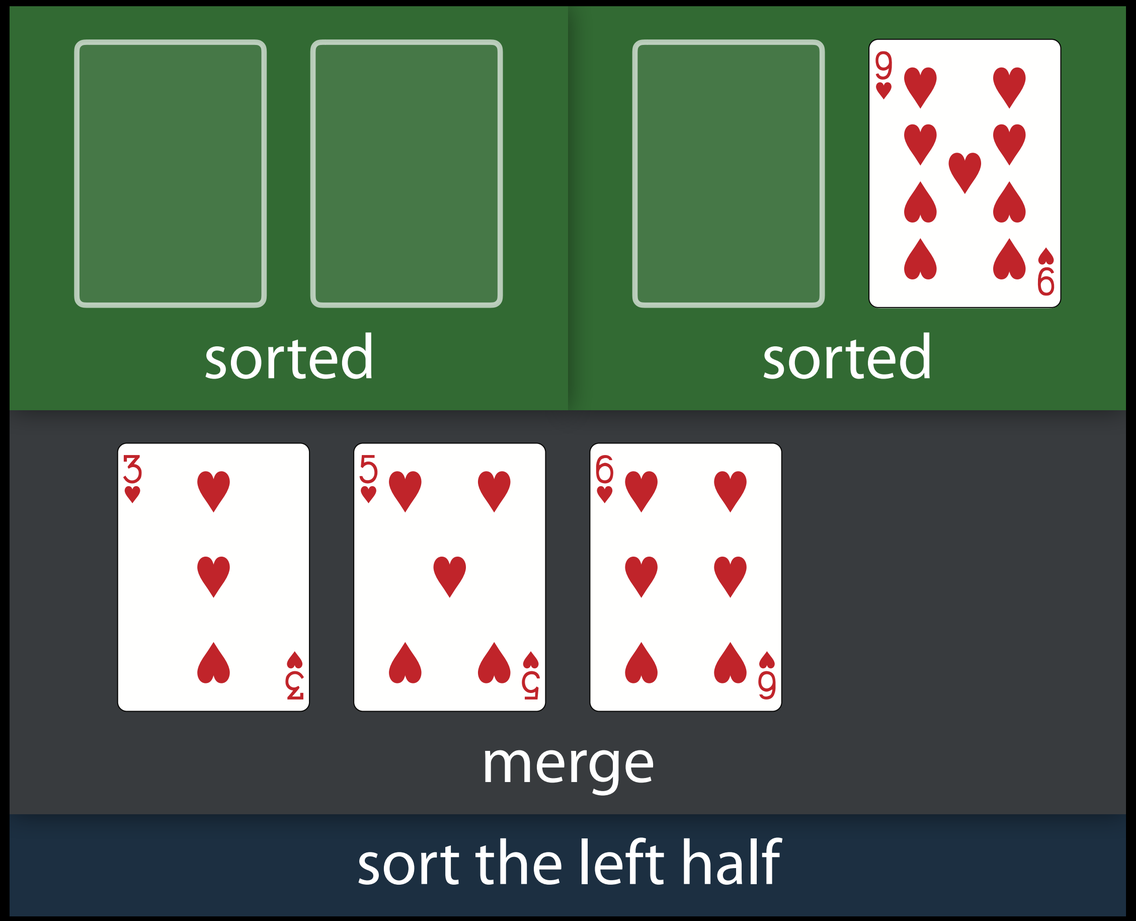


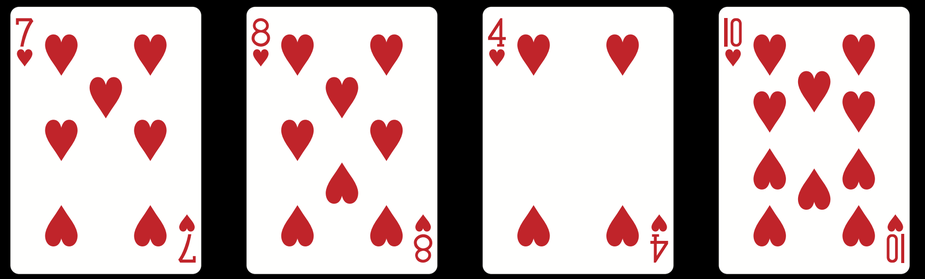
Merge sort



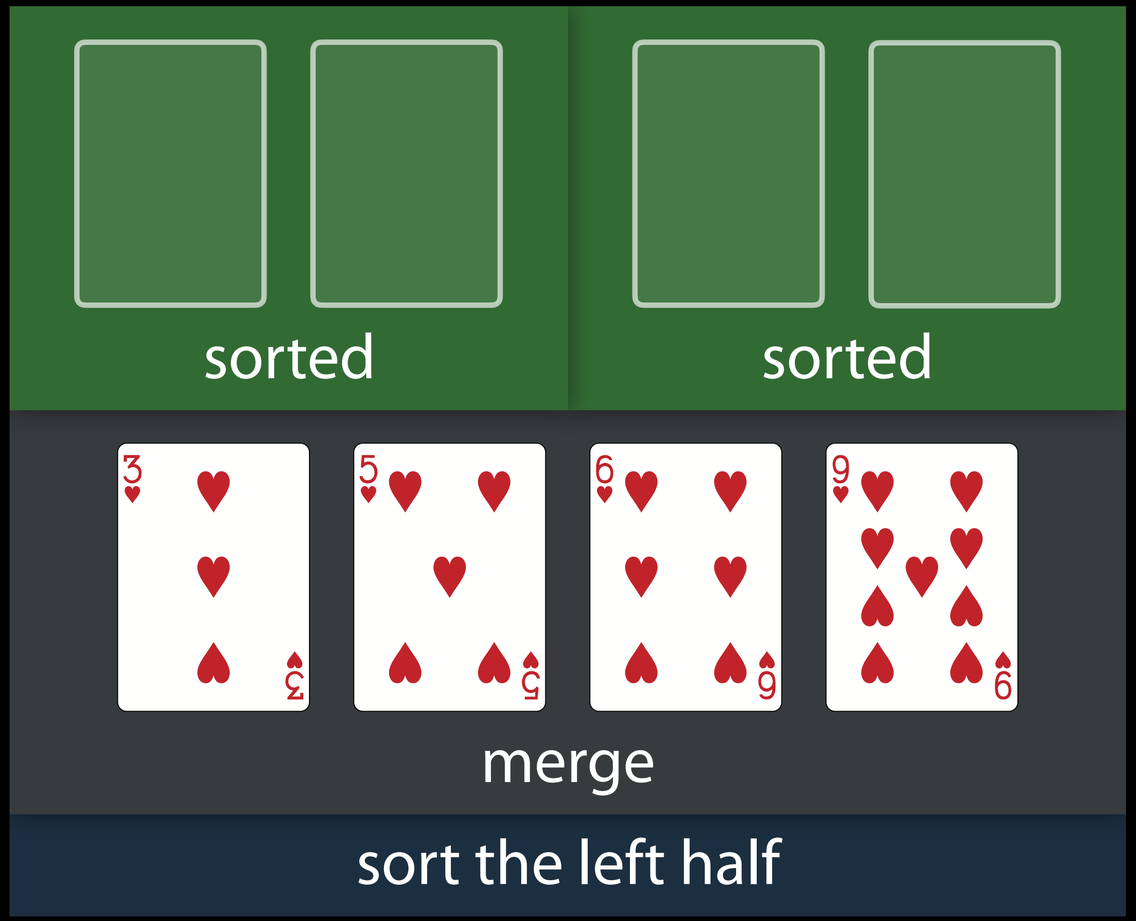


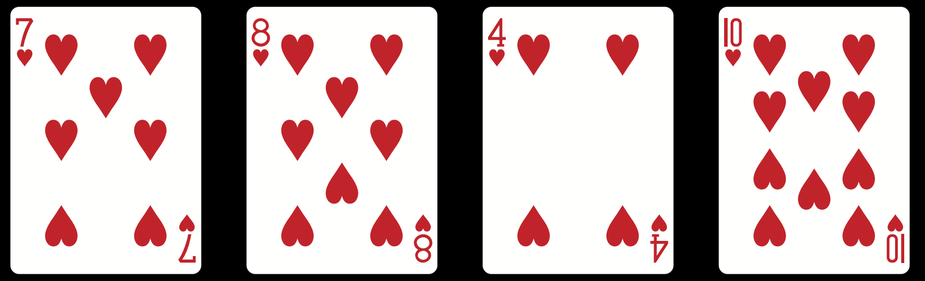
Merge sort



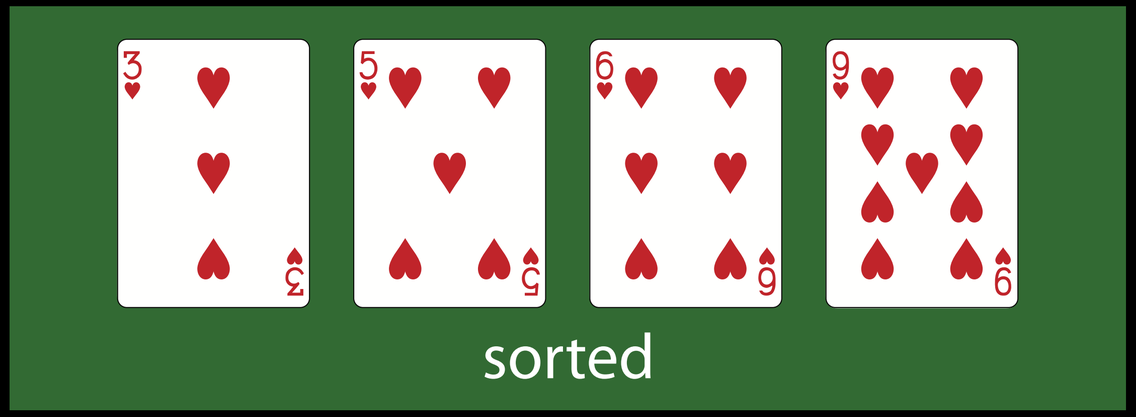


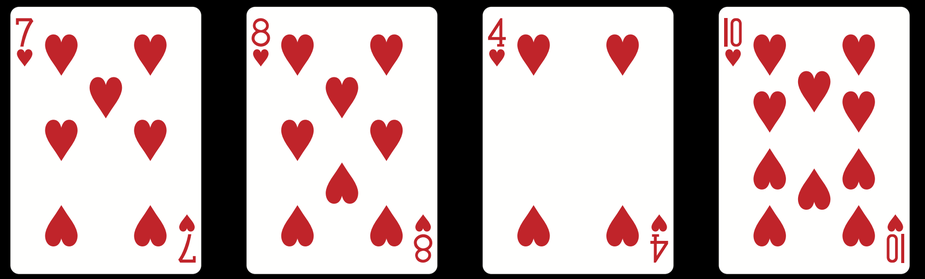
Merge sort



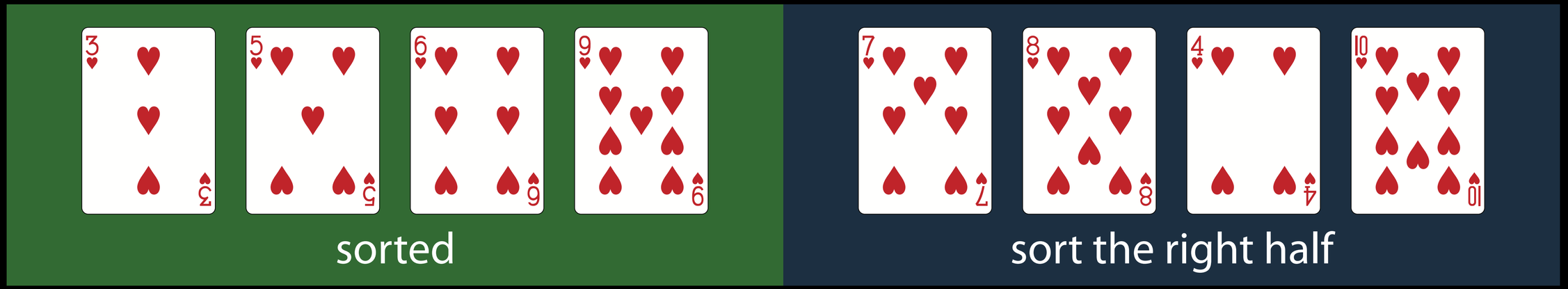


Merge sort

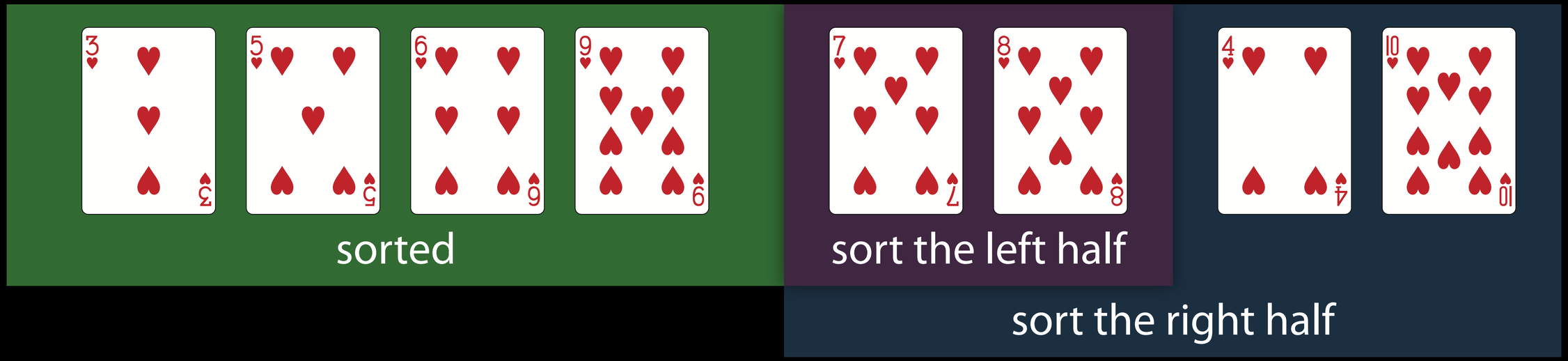




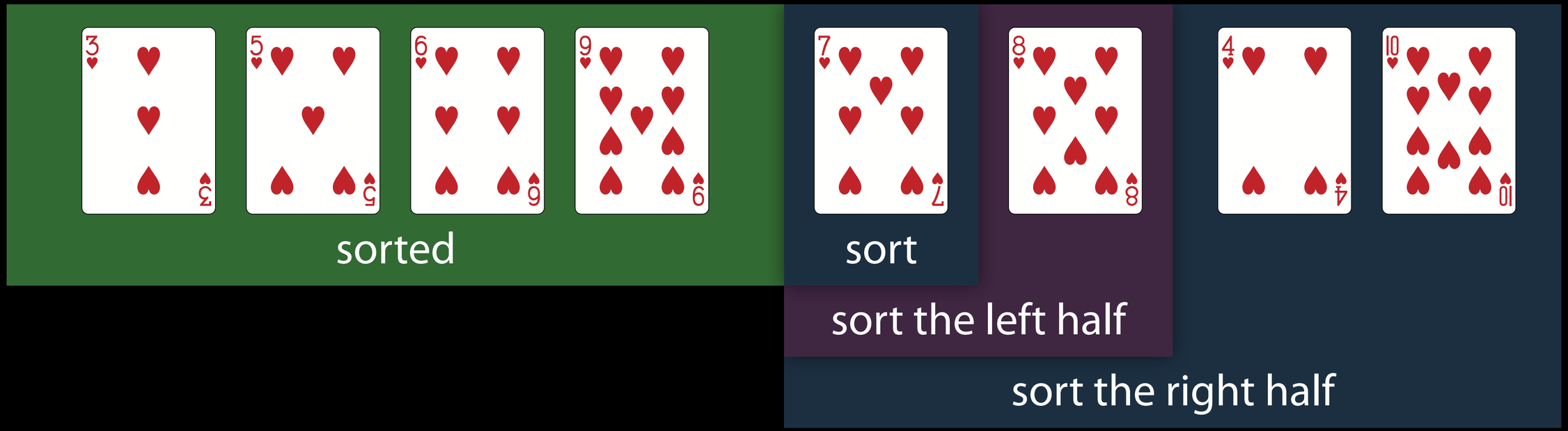
Merge sort



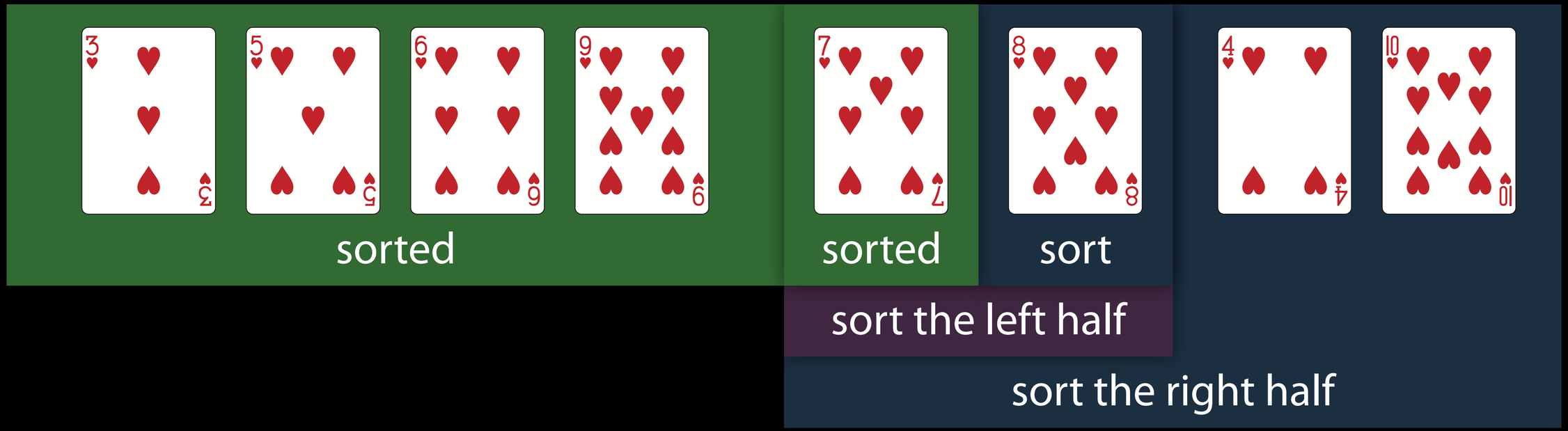
Merge sort



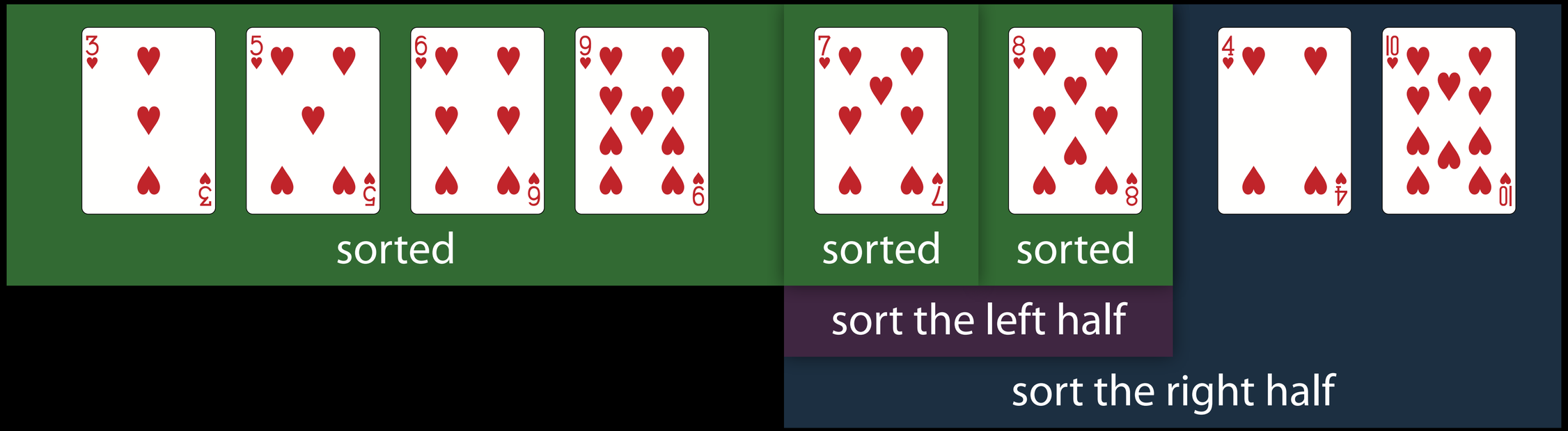
Merge sort



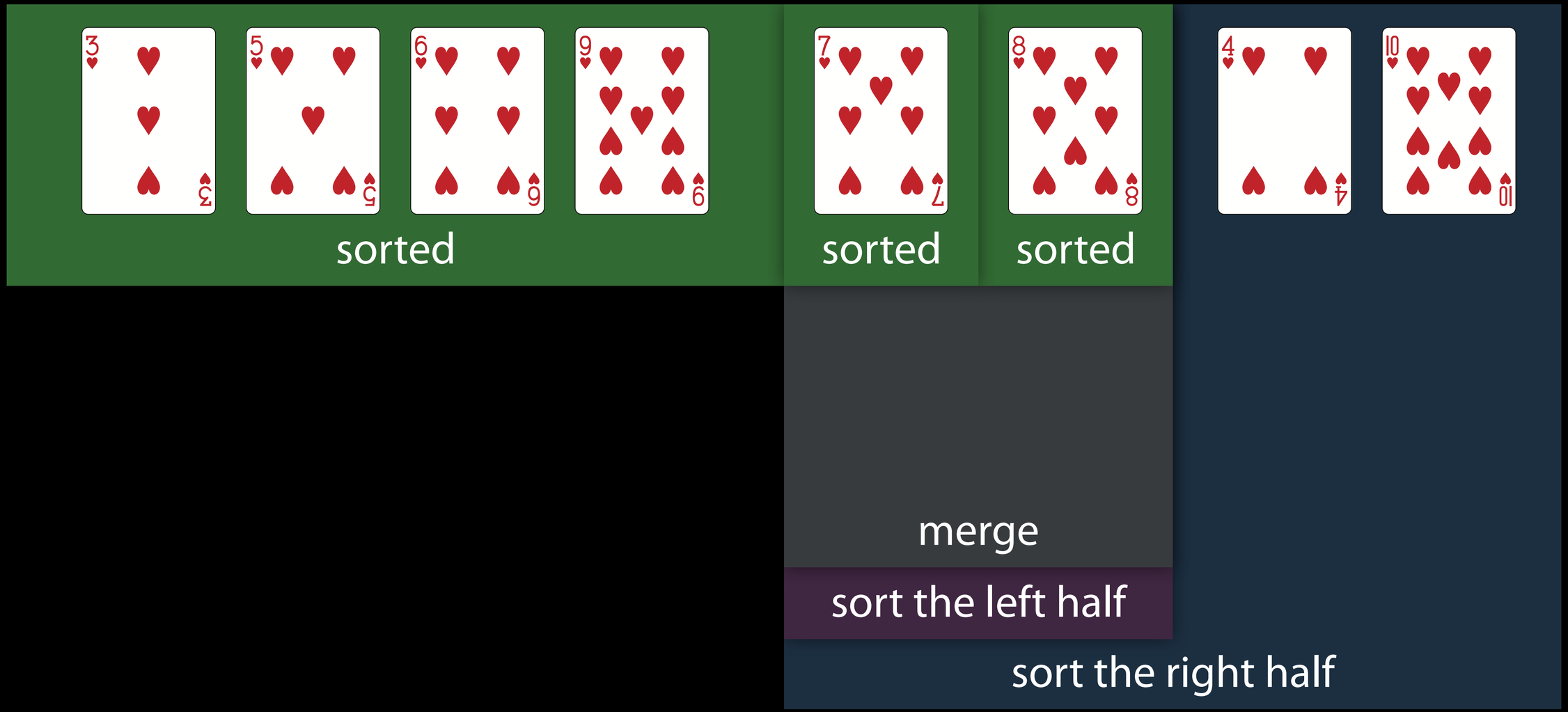
Merge sort



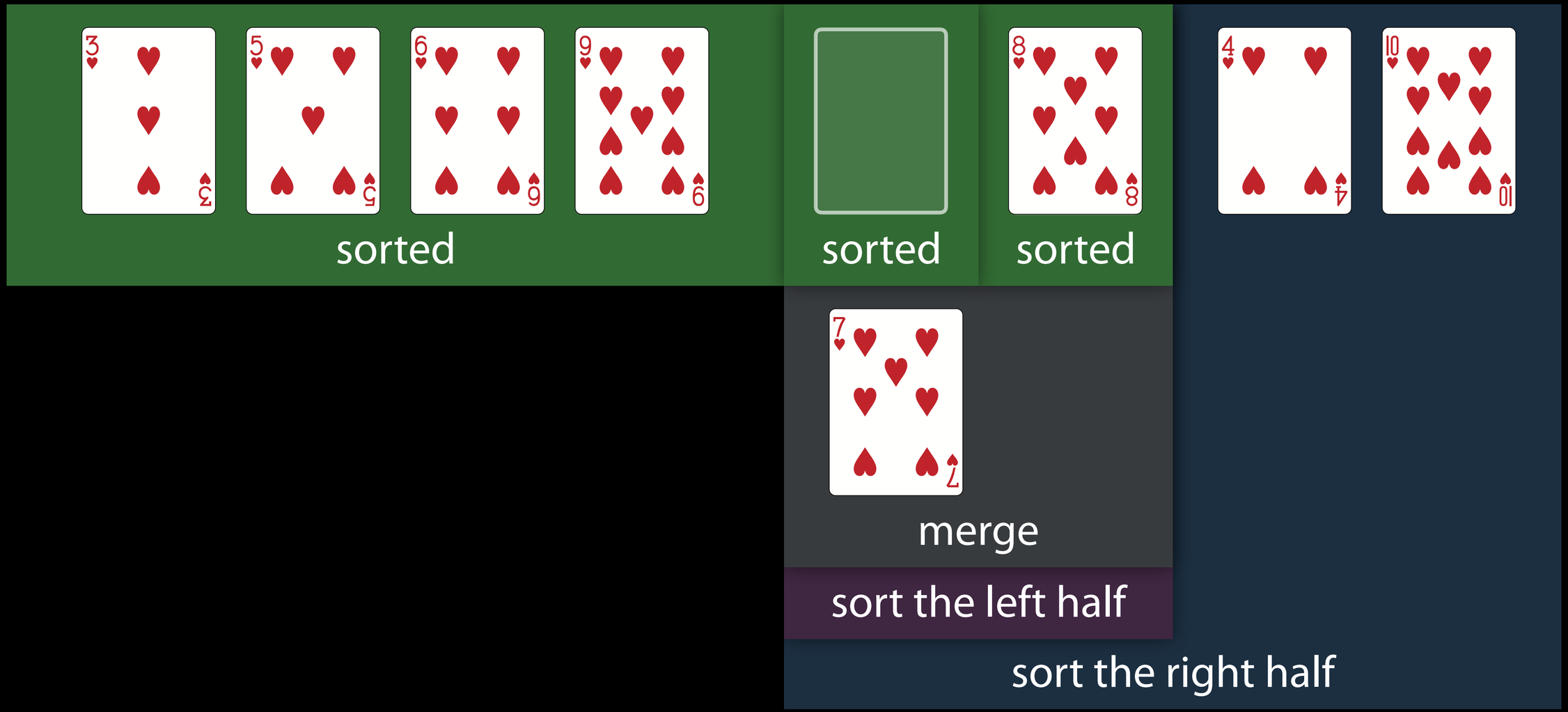
Merge sort



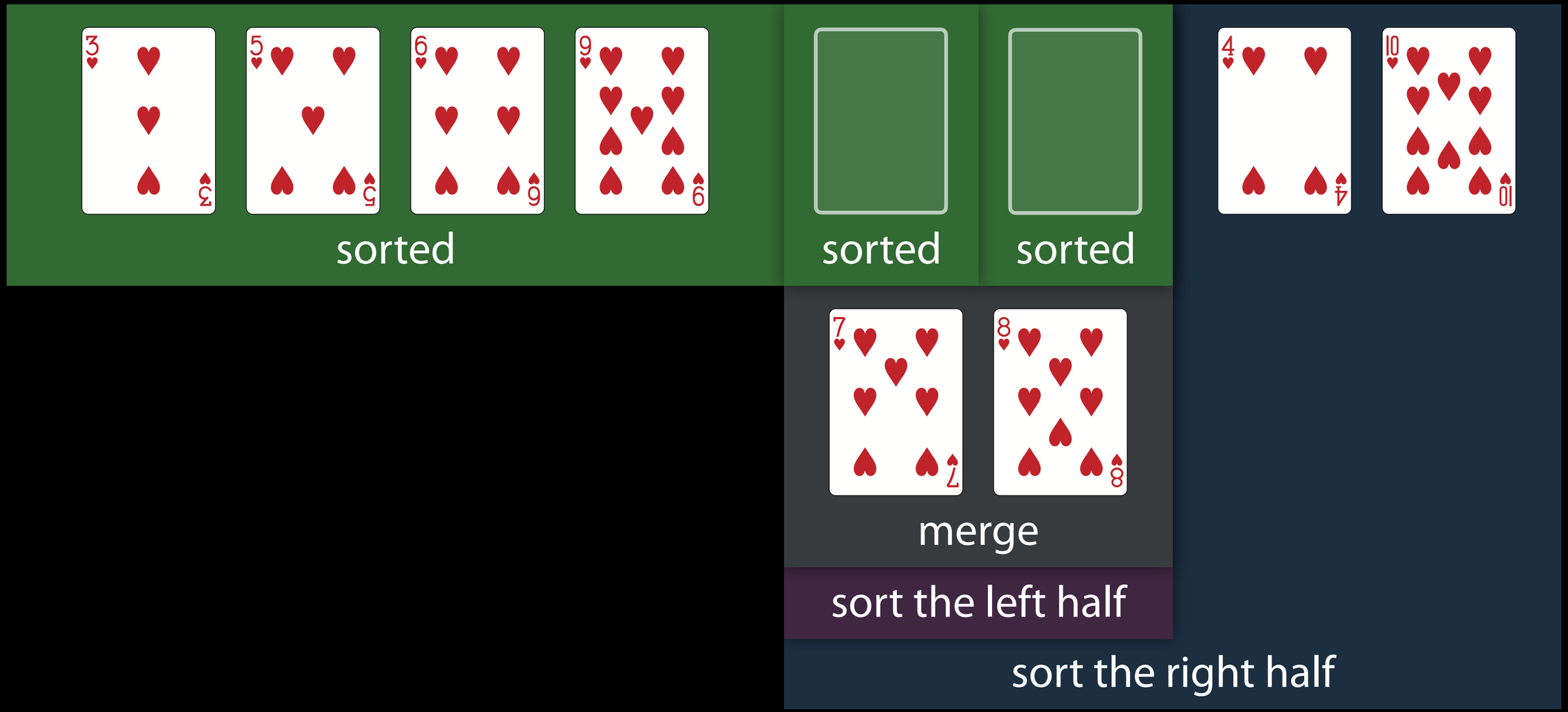
Merge sort


sorted
sorted
sorted
merge
sort the left half
sort the right half

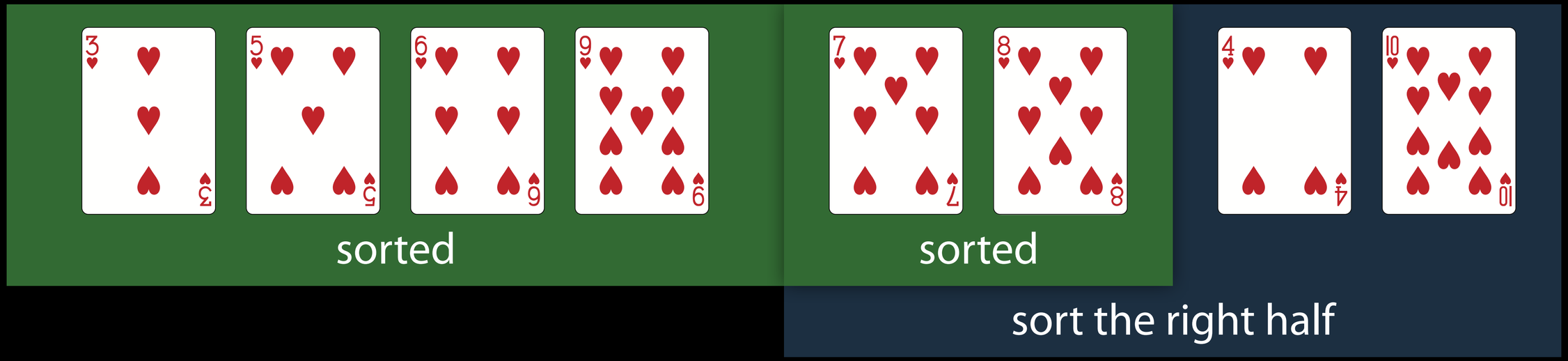
Merge sort


sorted
sorted
sorted
merge
sort the left half
sort the right half

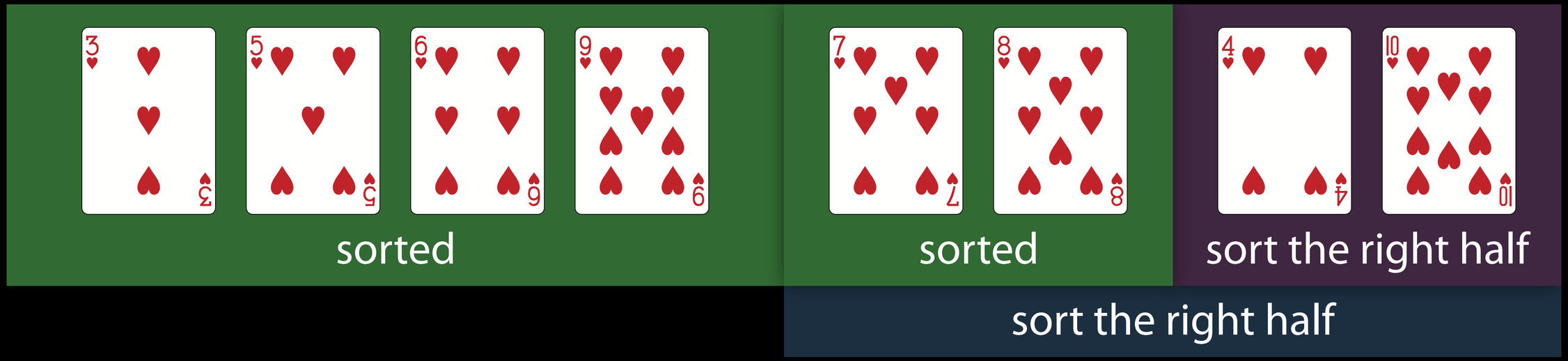
Merge sort


sorted
sorted
sorted
merge
sort the left half
sort the right half

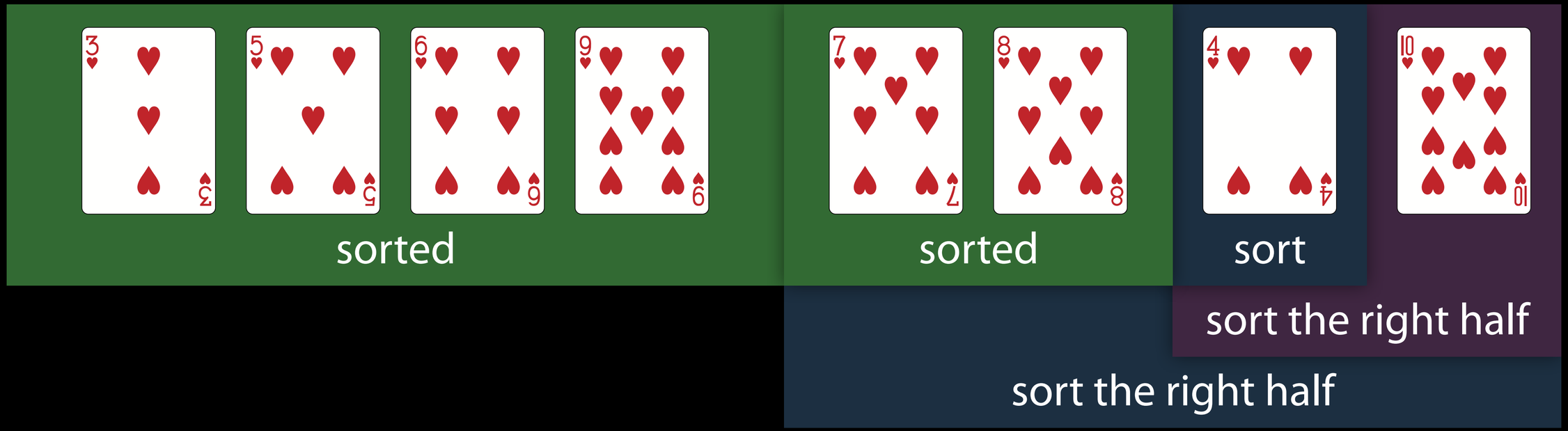
Merge sort



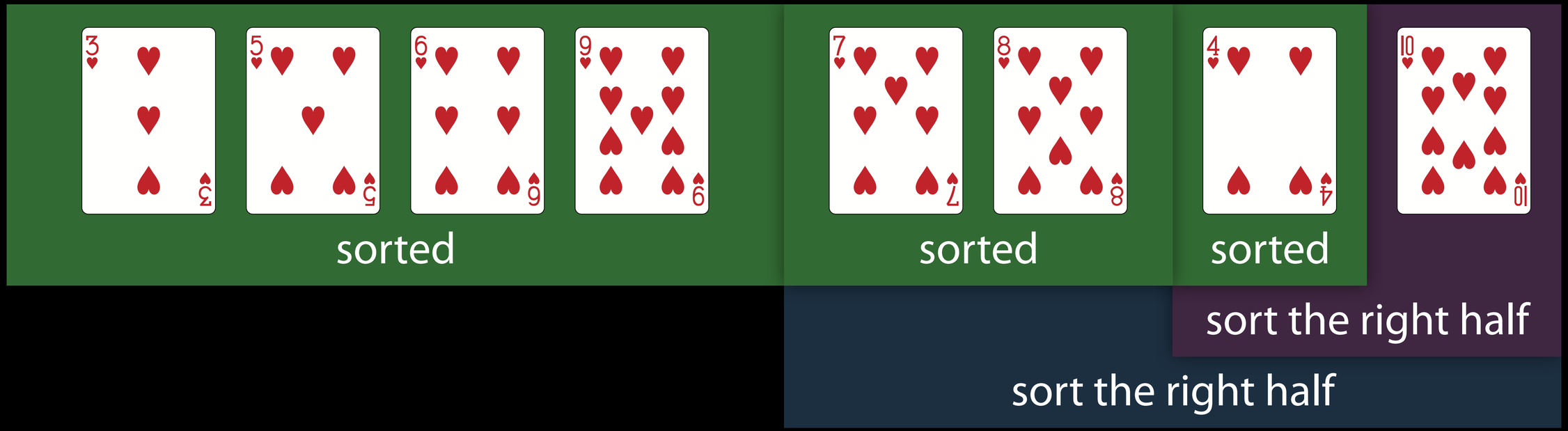
Merge sort



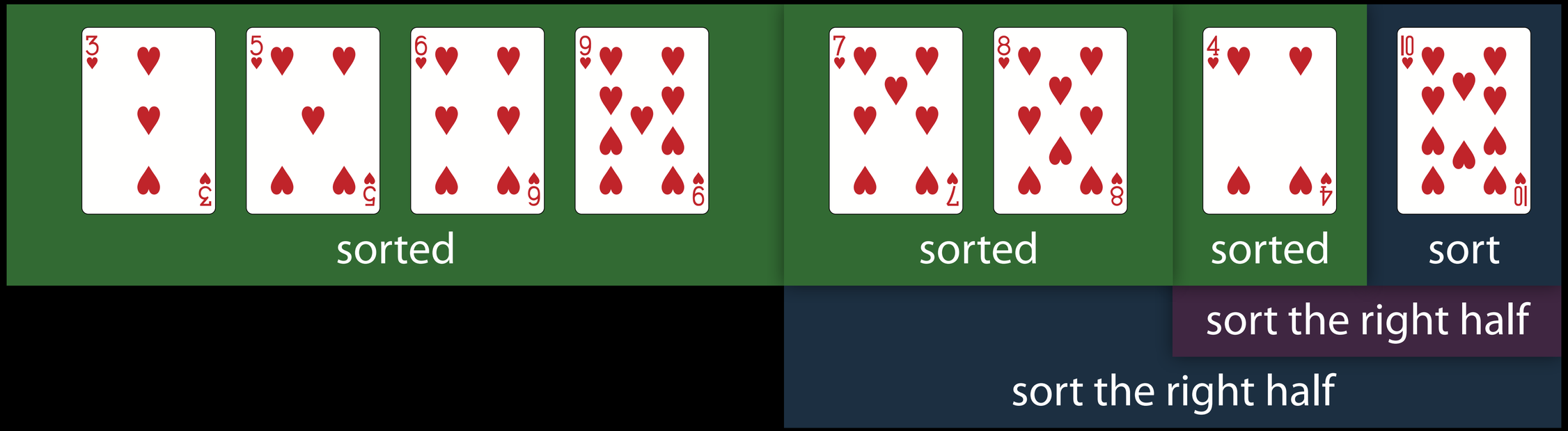
Merge sort



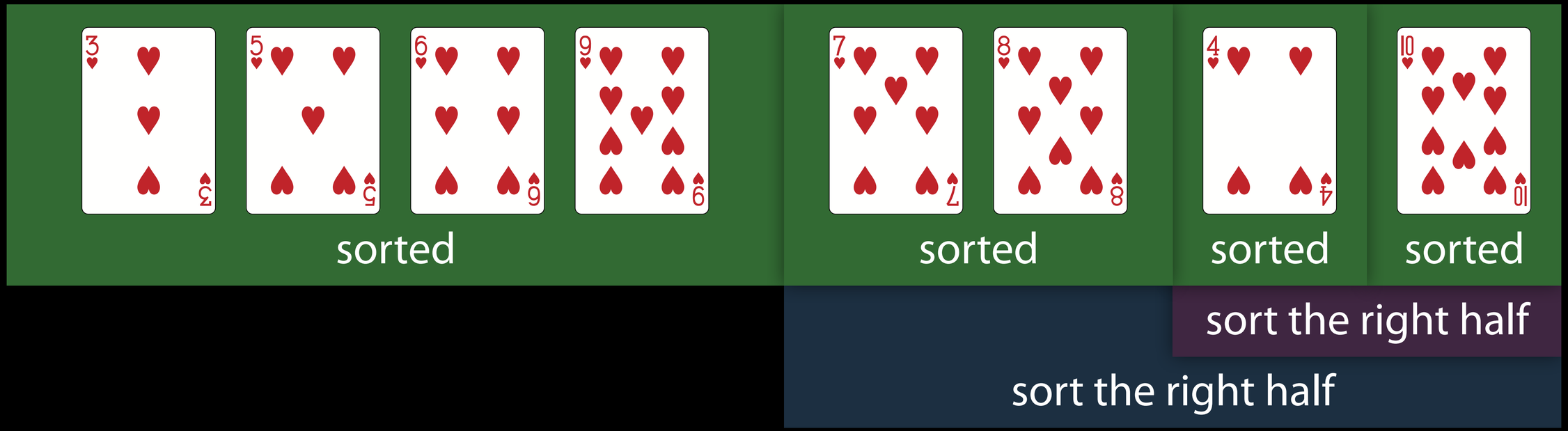
Merge sort


sorted
sorted
sorted
sort the right half
sort the right half

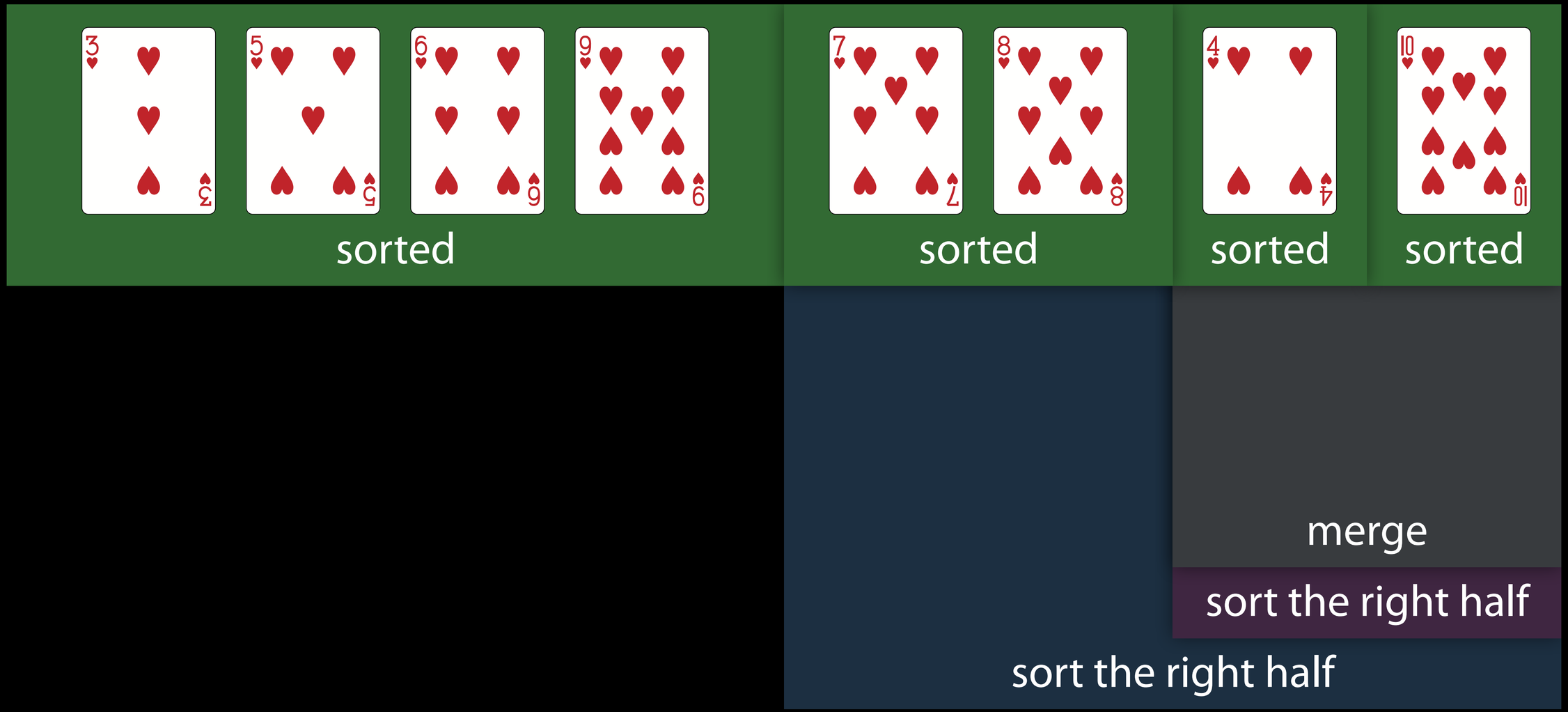
Merge sort


sorted
sorted
sorted
sort
sort the right half
sort the right half

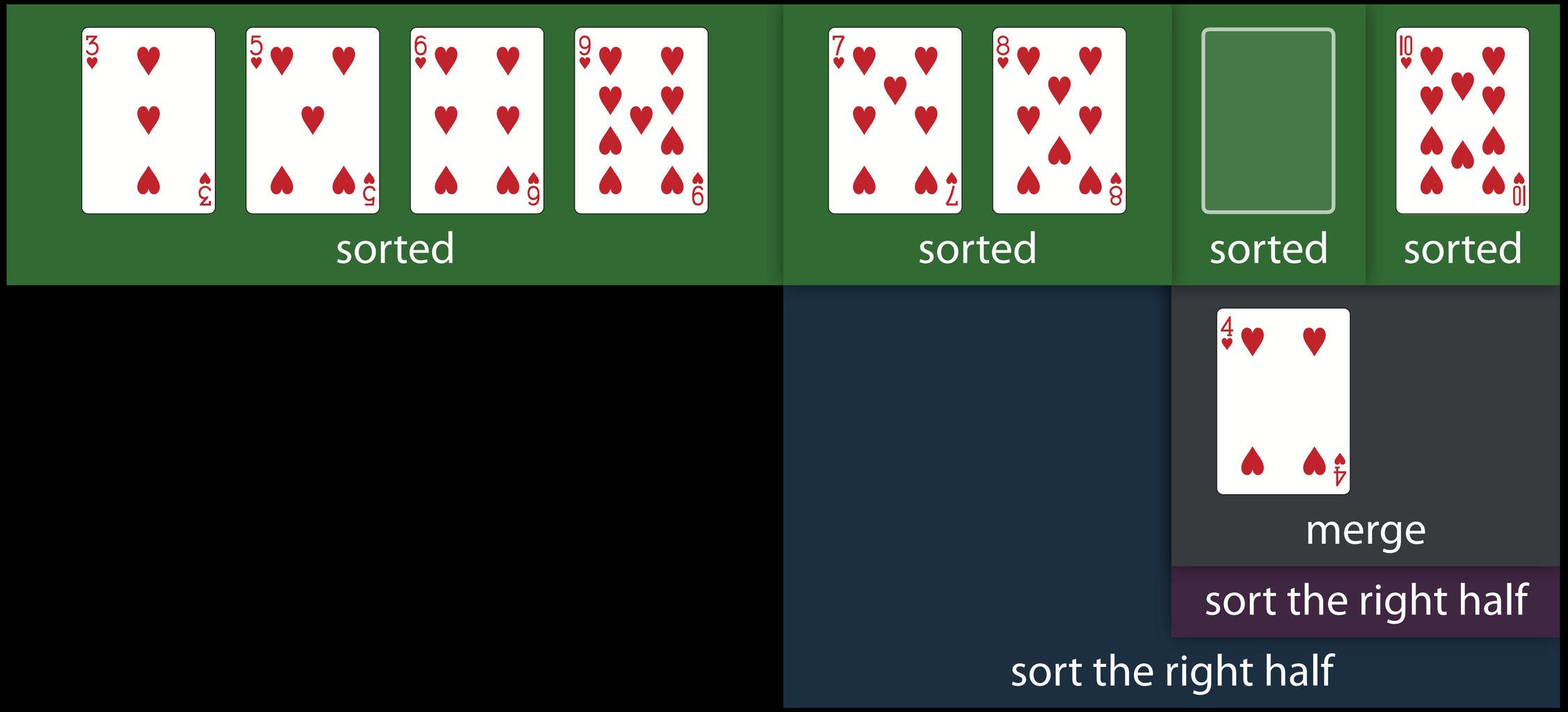
Merge sort


sorted
sorted
sorted
sorted
sort the right half
sort the right half

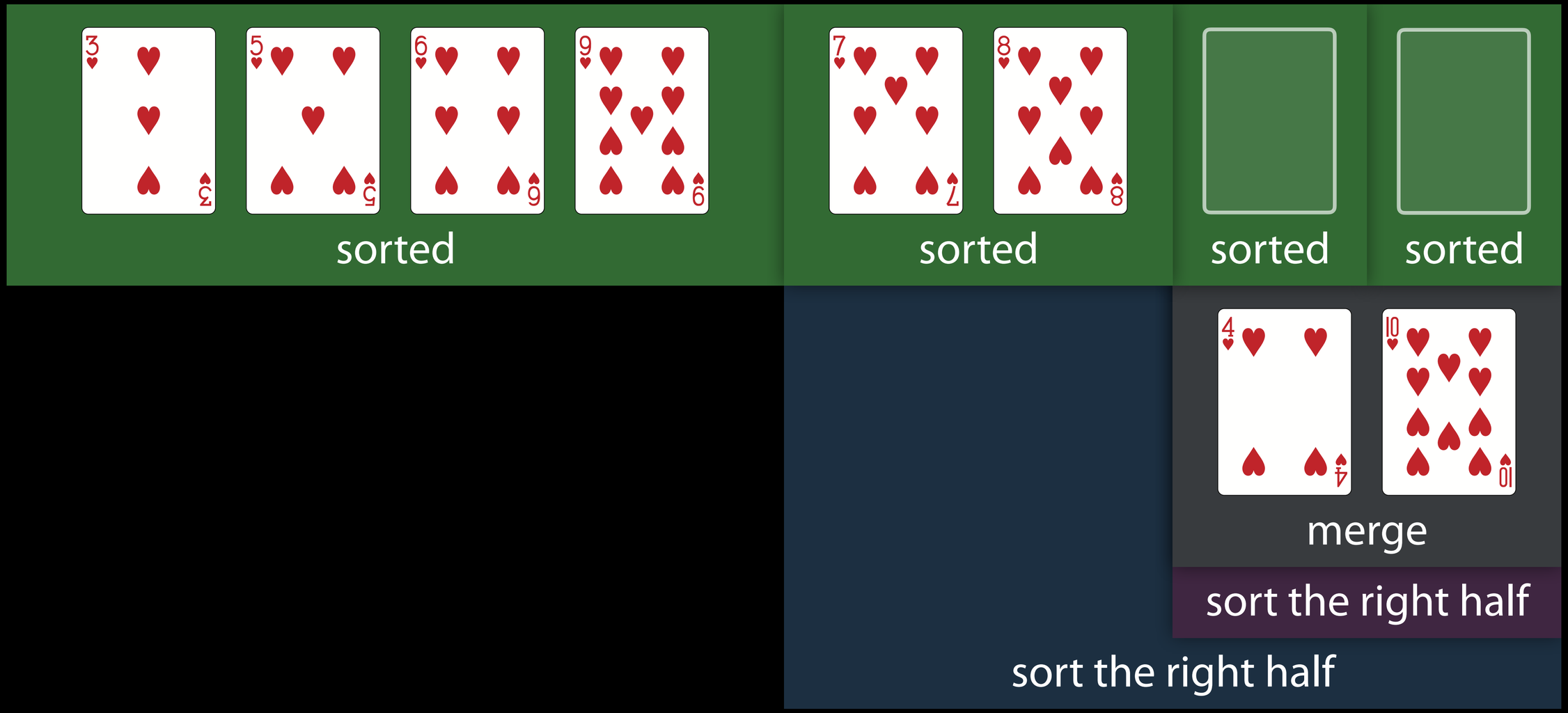
Merge sort


sorted
sorted
sorted
sorted
merge
sort the right half
sort the right half

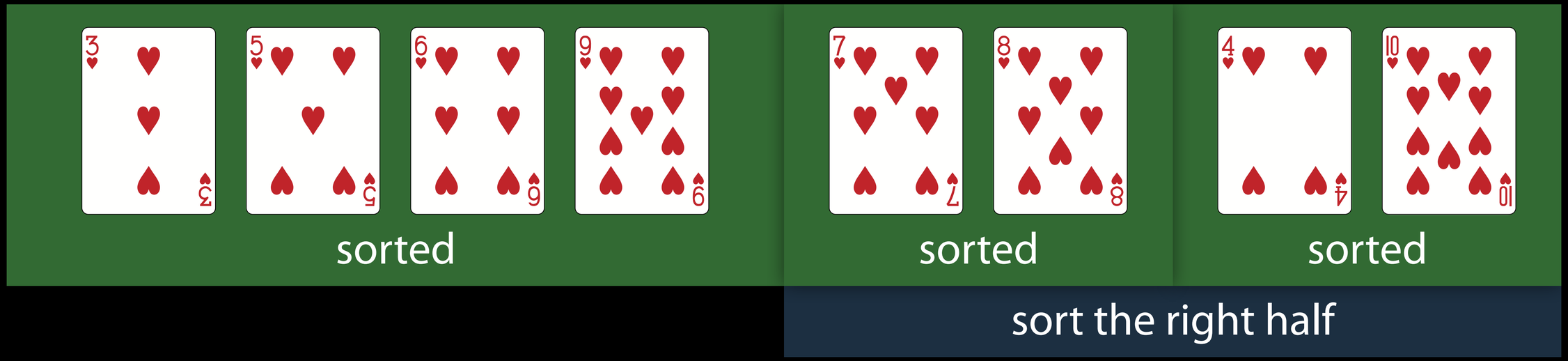
Merge sort


sorted
sorted
sorted
sorted
merge
sort the right half
sort the right half

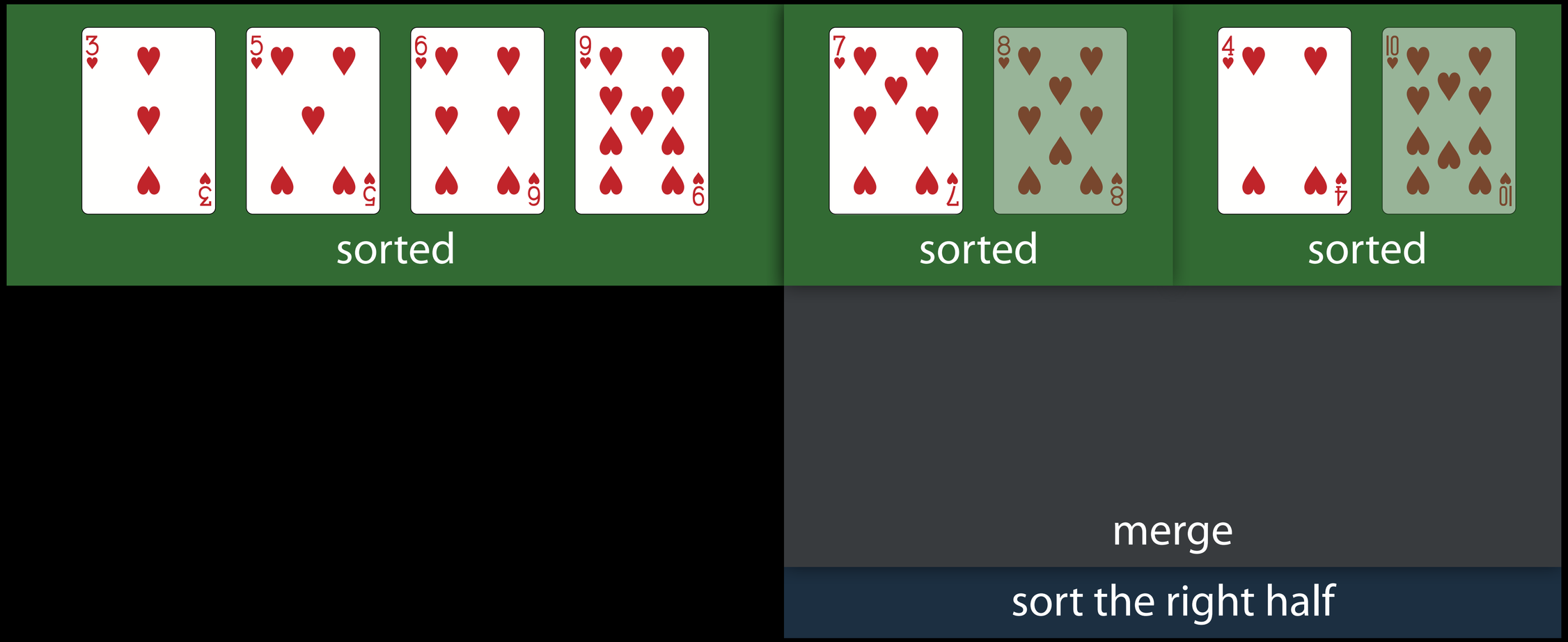
Merge sort


sorted
sorted
sorted
sorted
merge
sort the right half
sort the right half

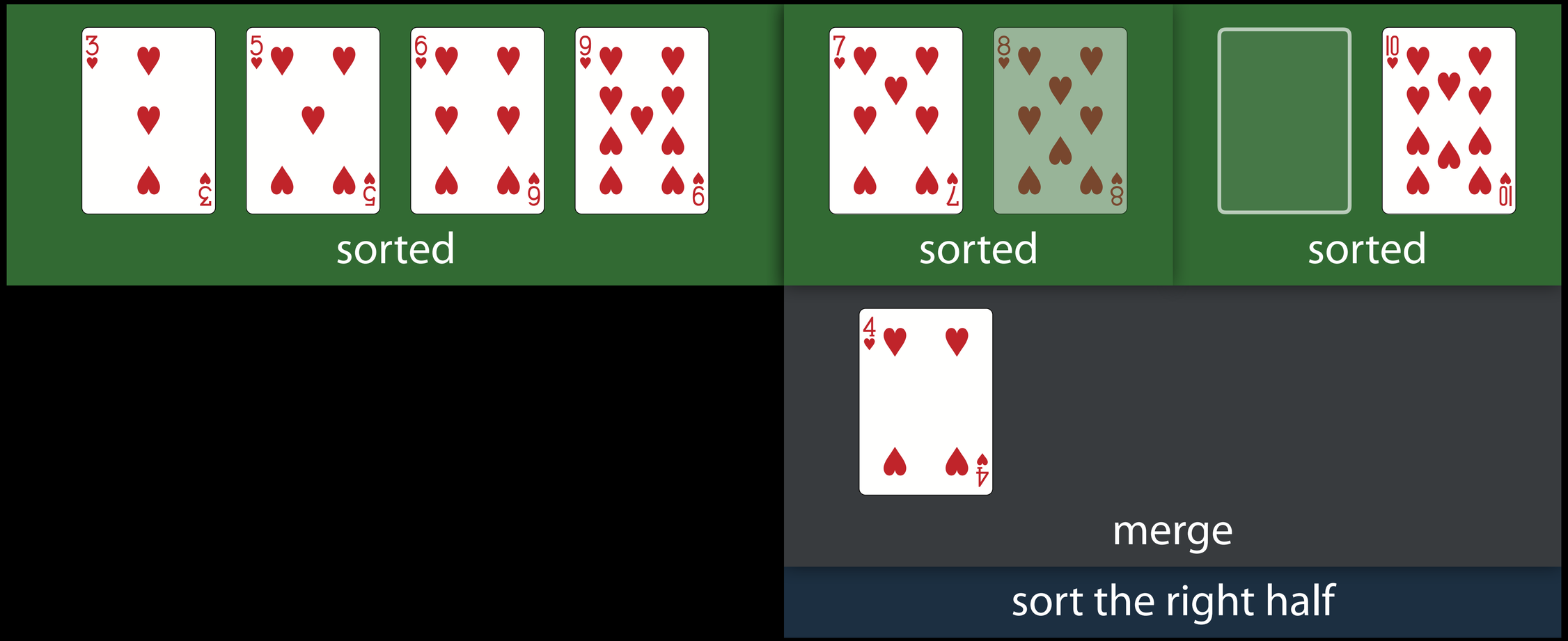
Merge sort



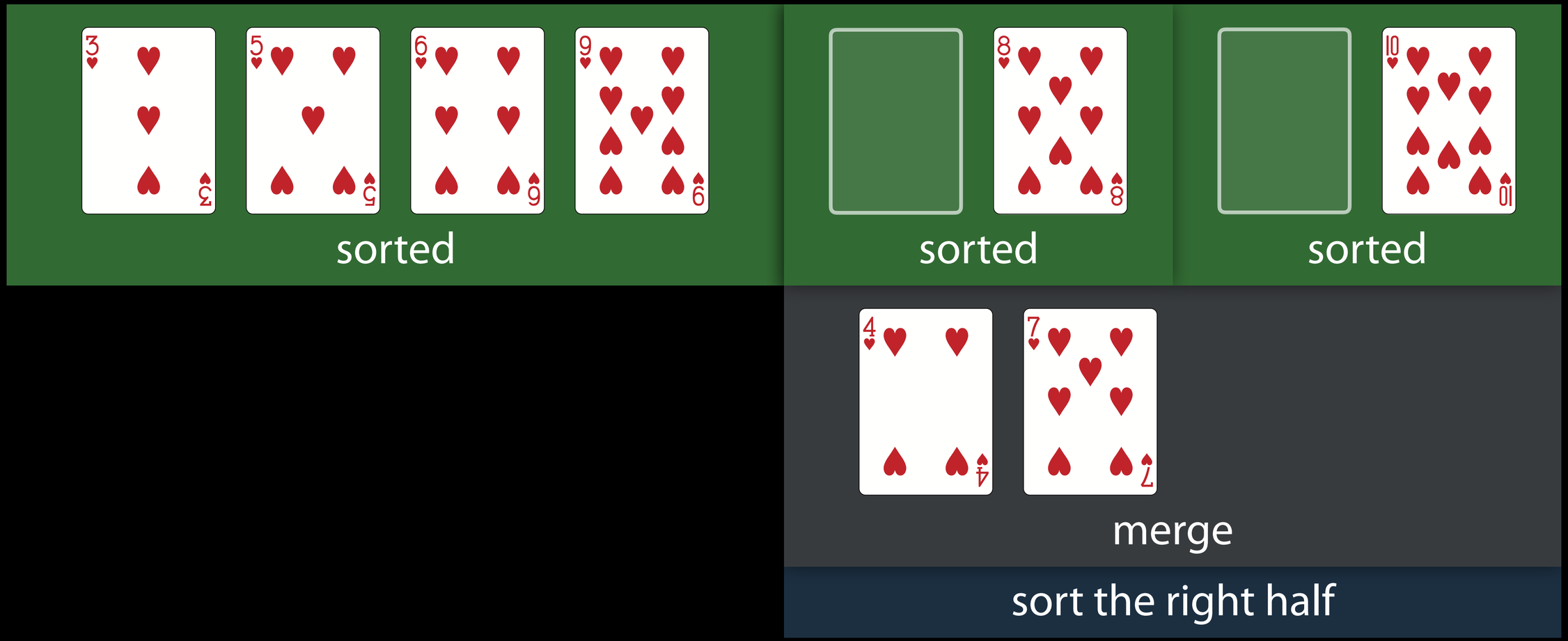
Merge sort



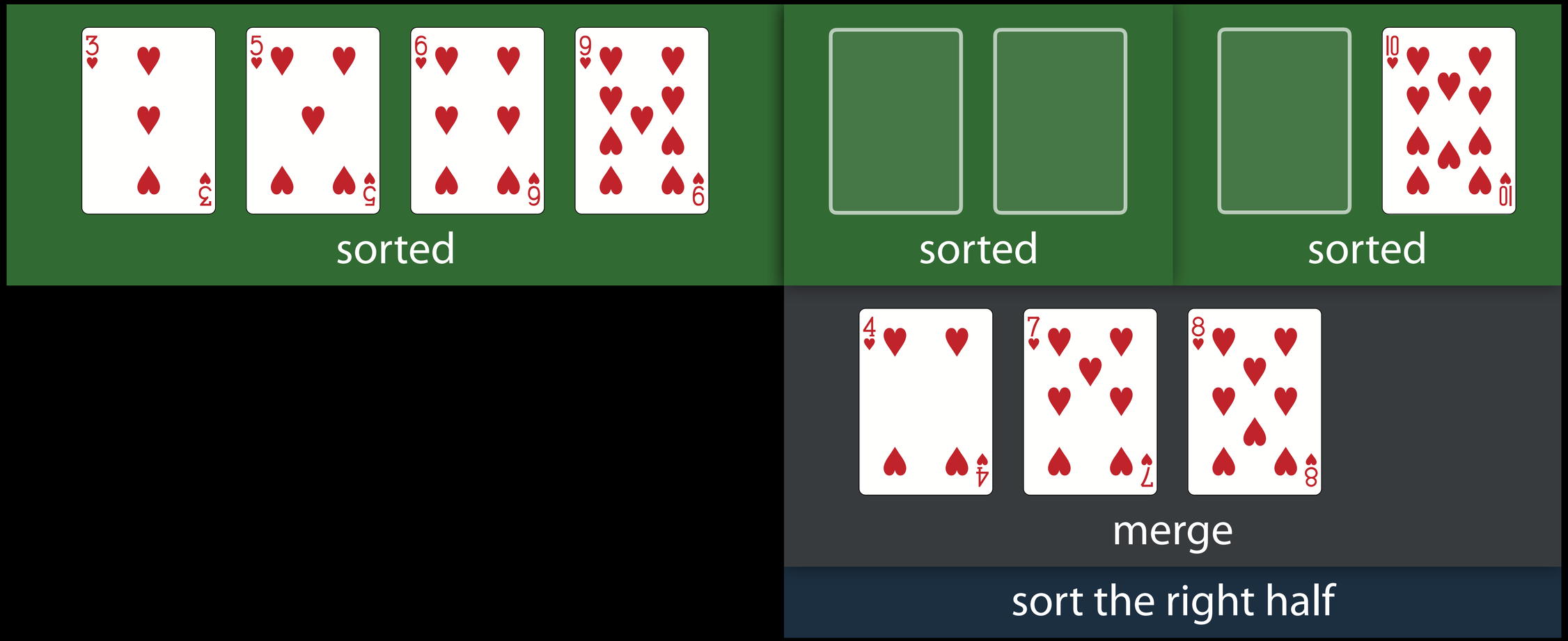
Merge sort



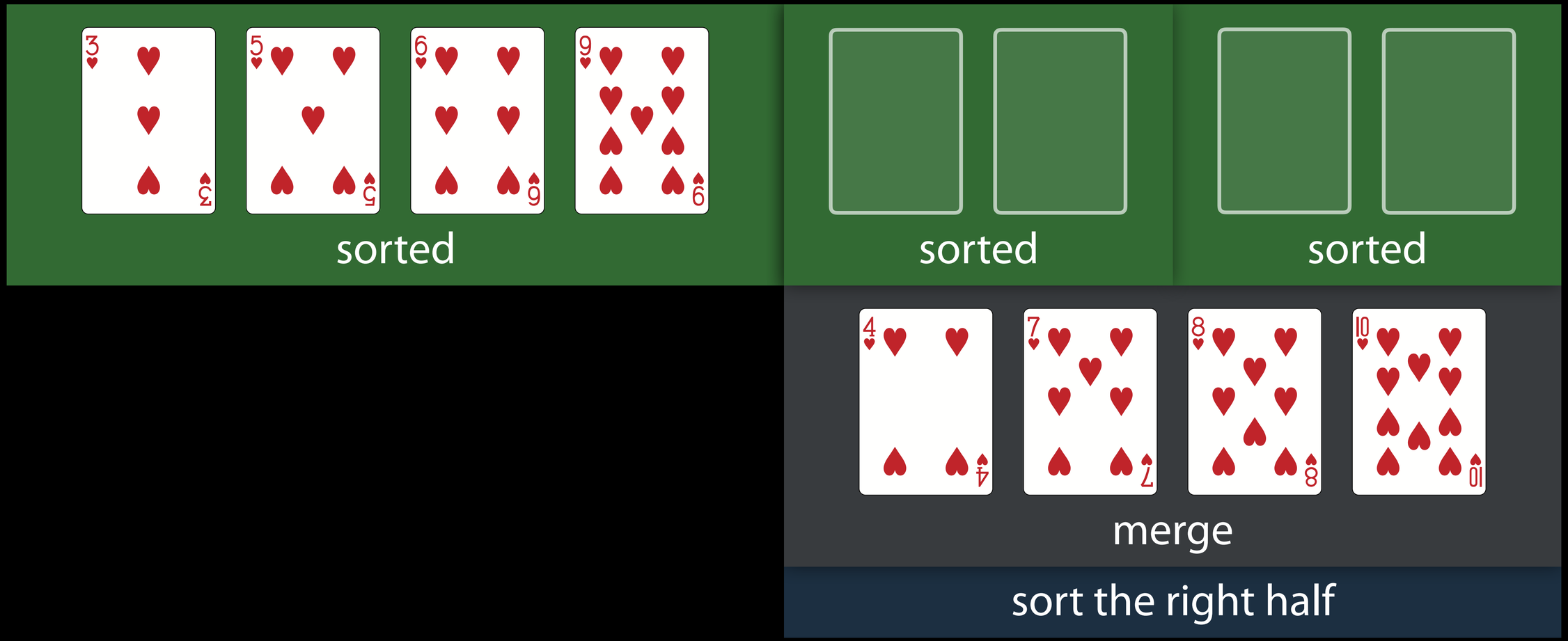
Merge sort



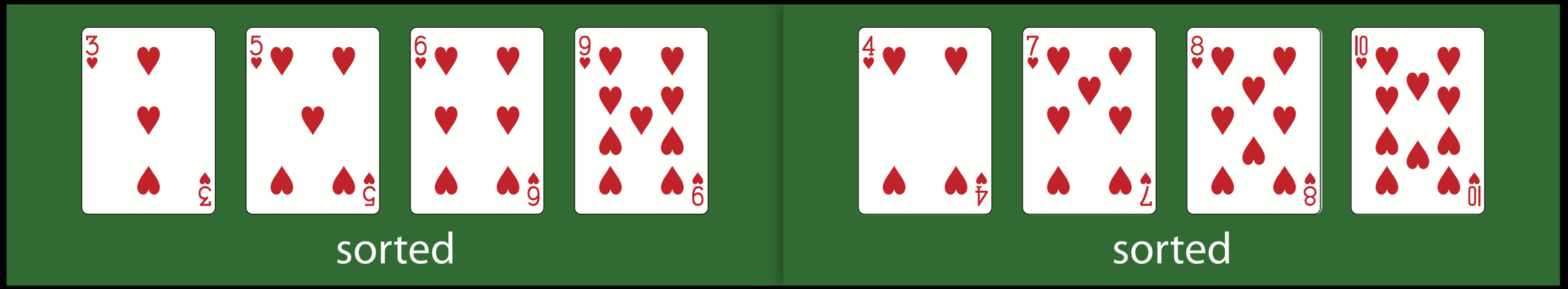
Merge sort



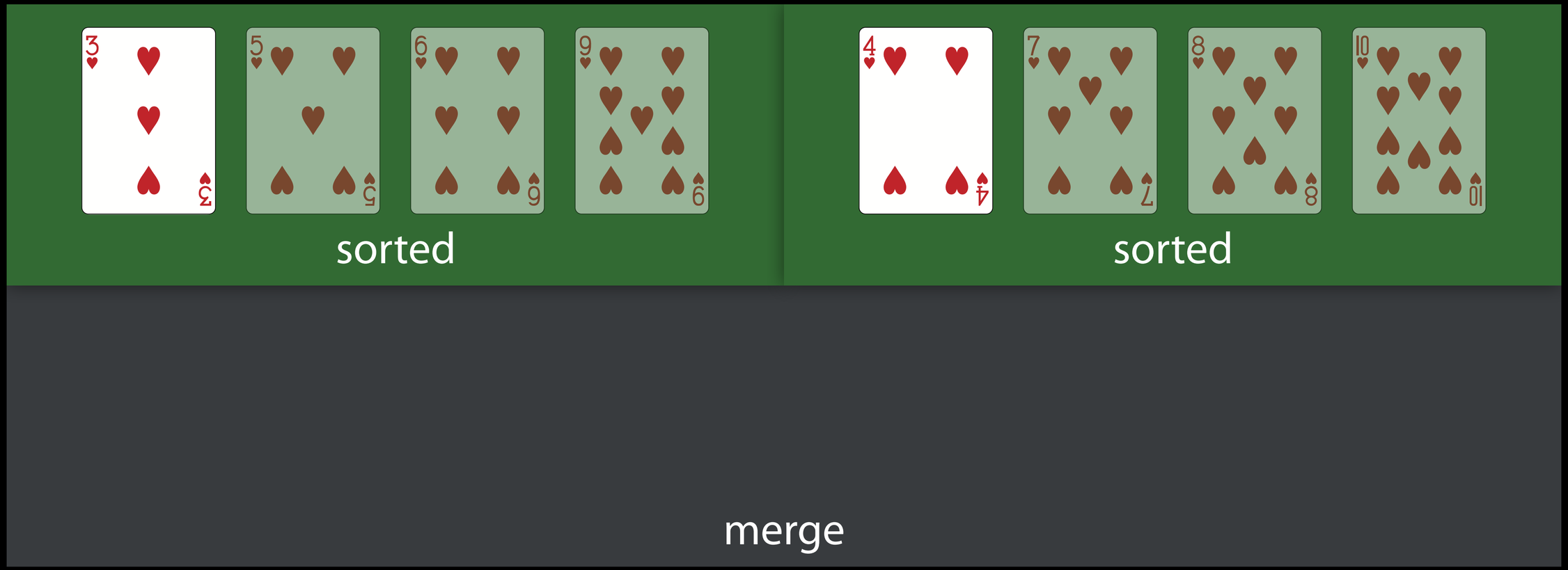
Merge sort



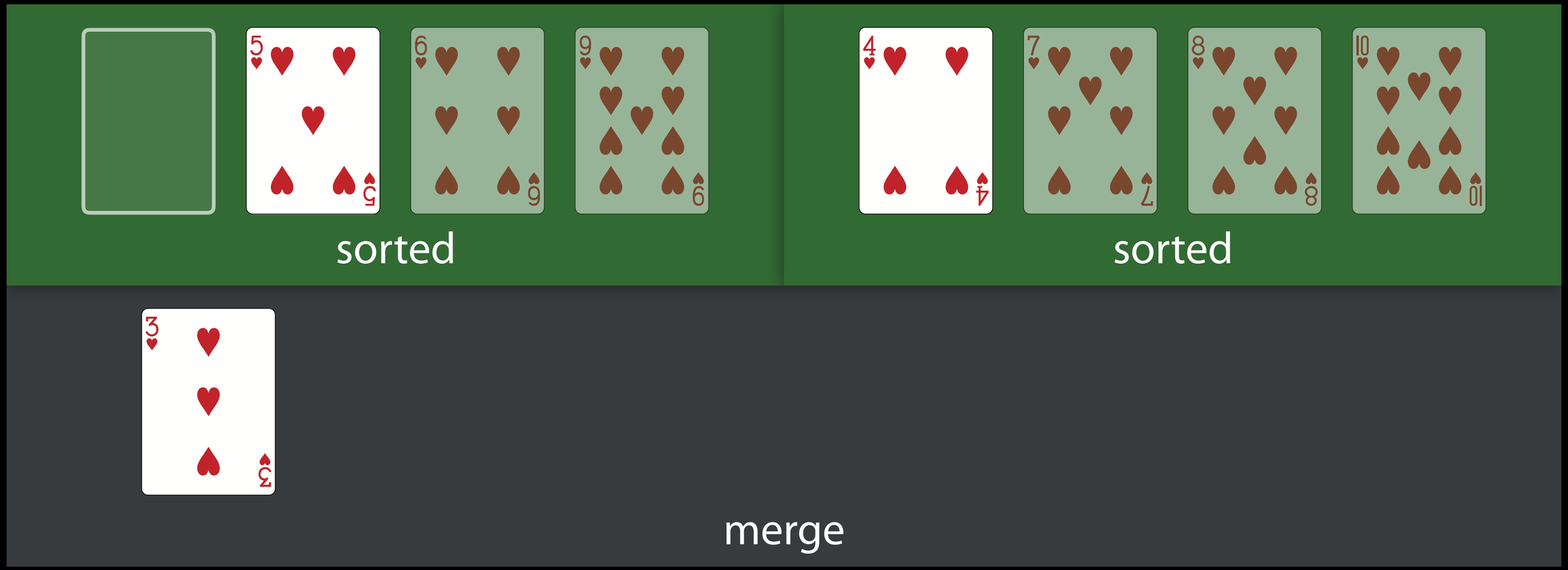
Merge sort



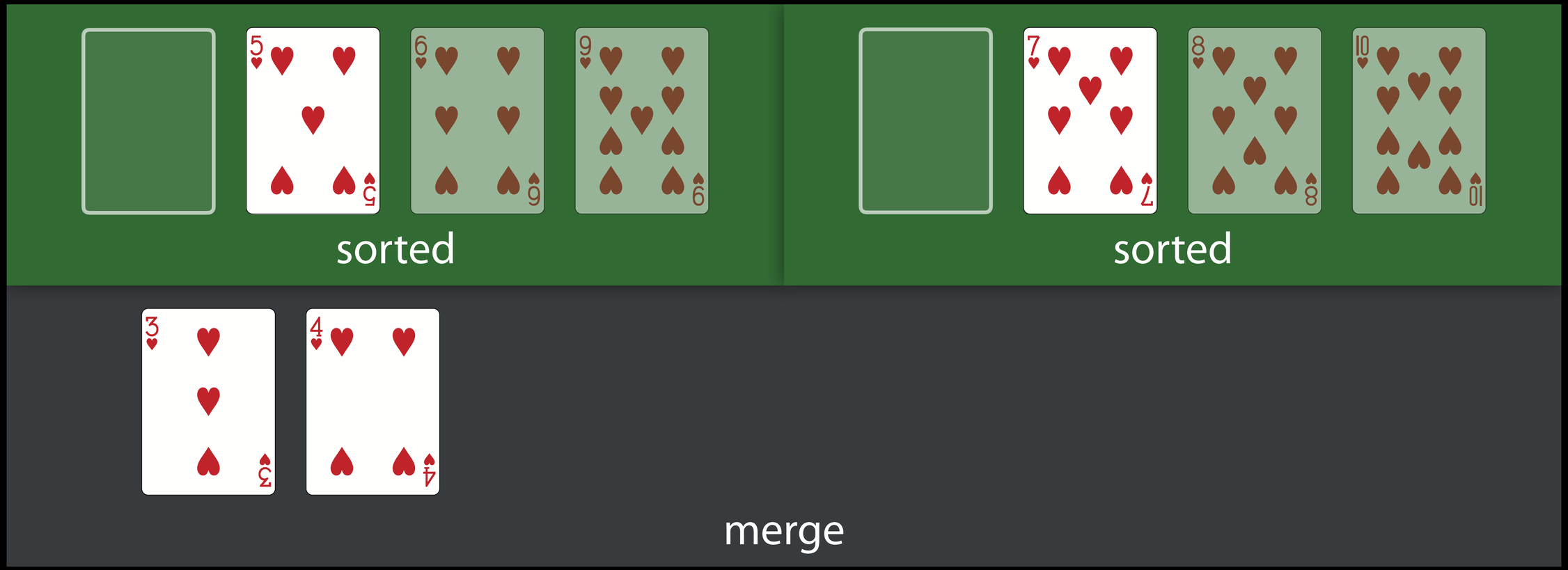
Merge sort



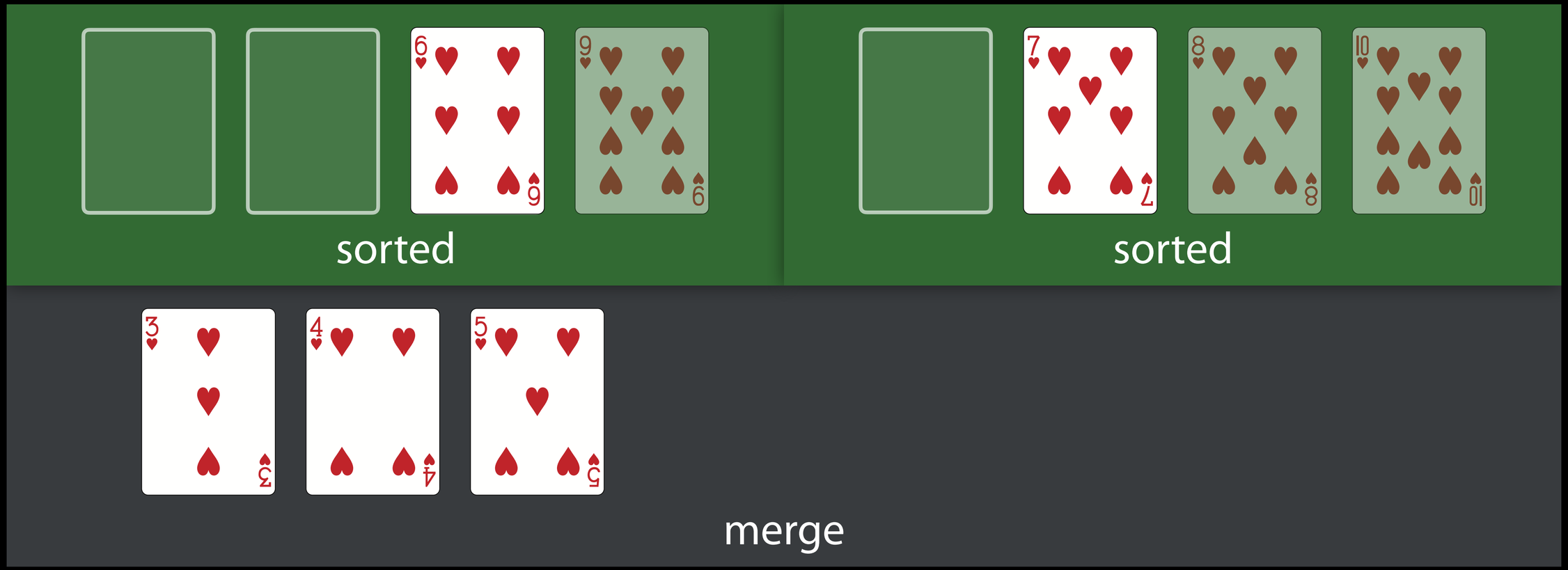
Merge sort



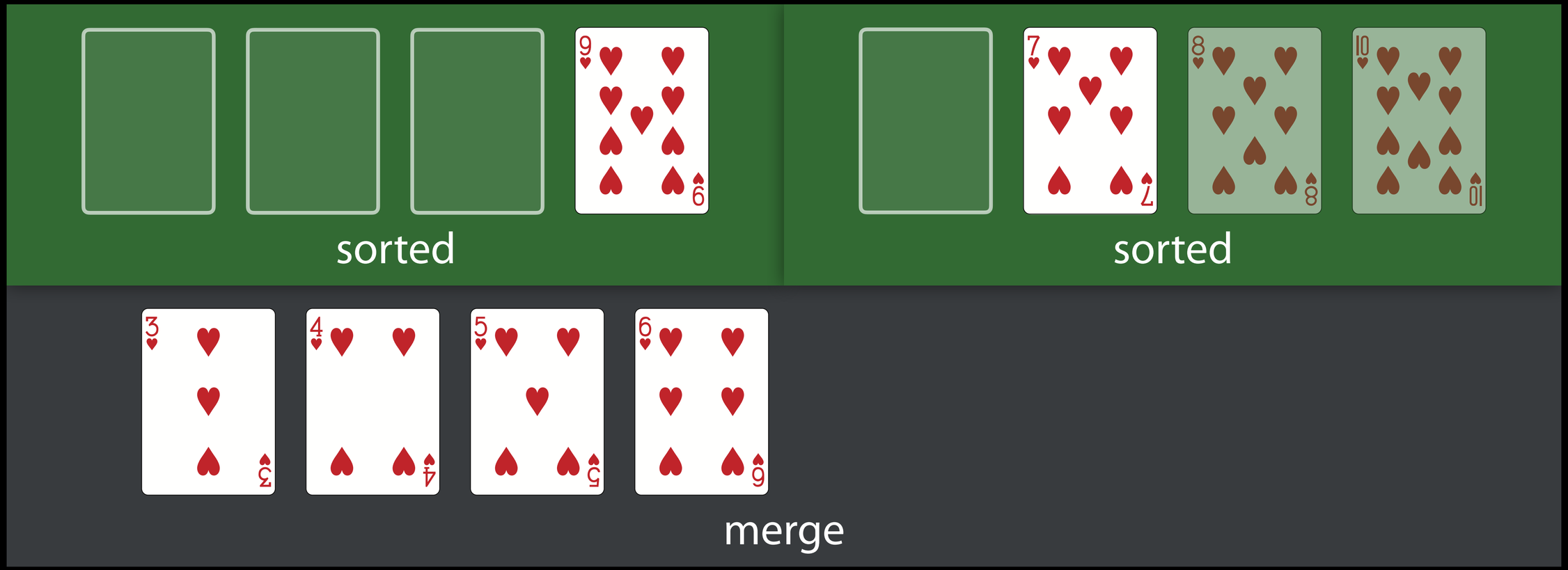
Merge sort



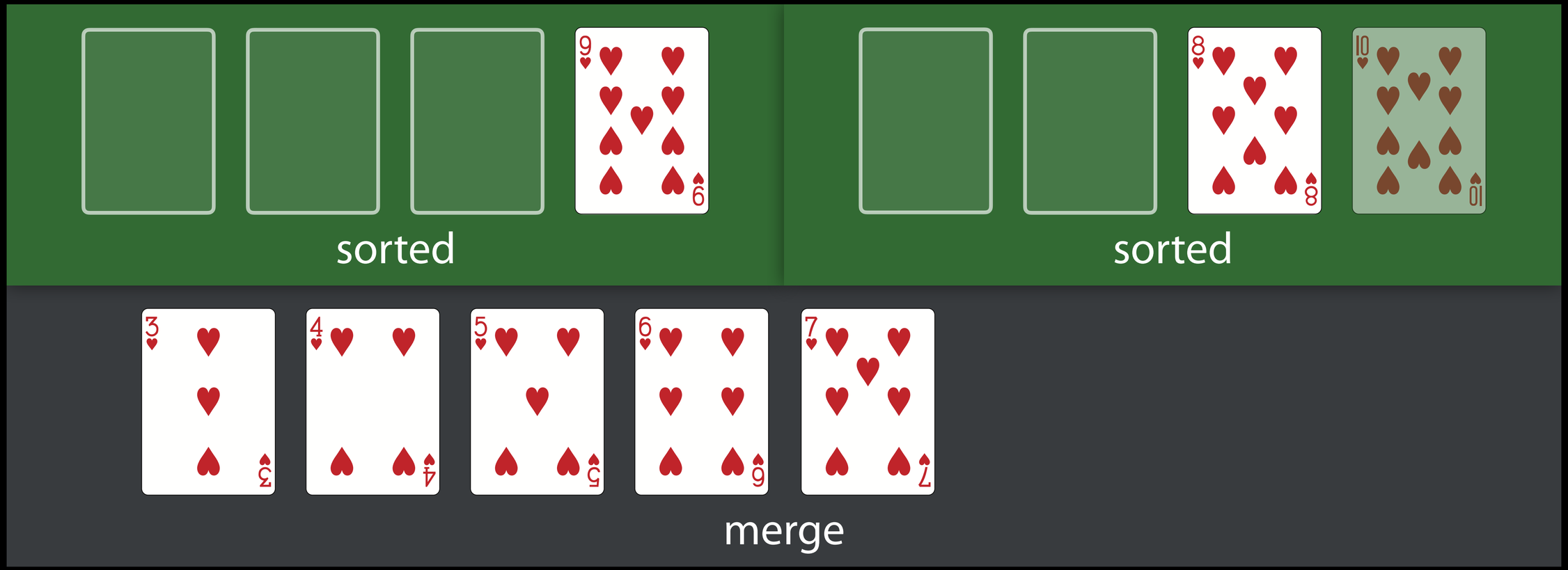
Merge sort



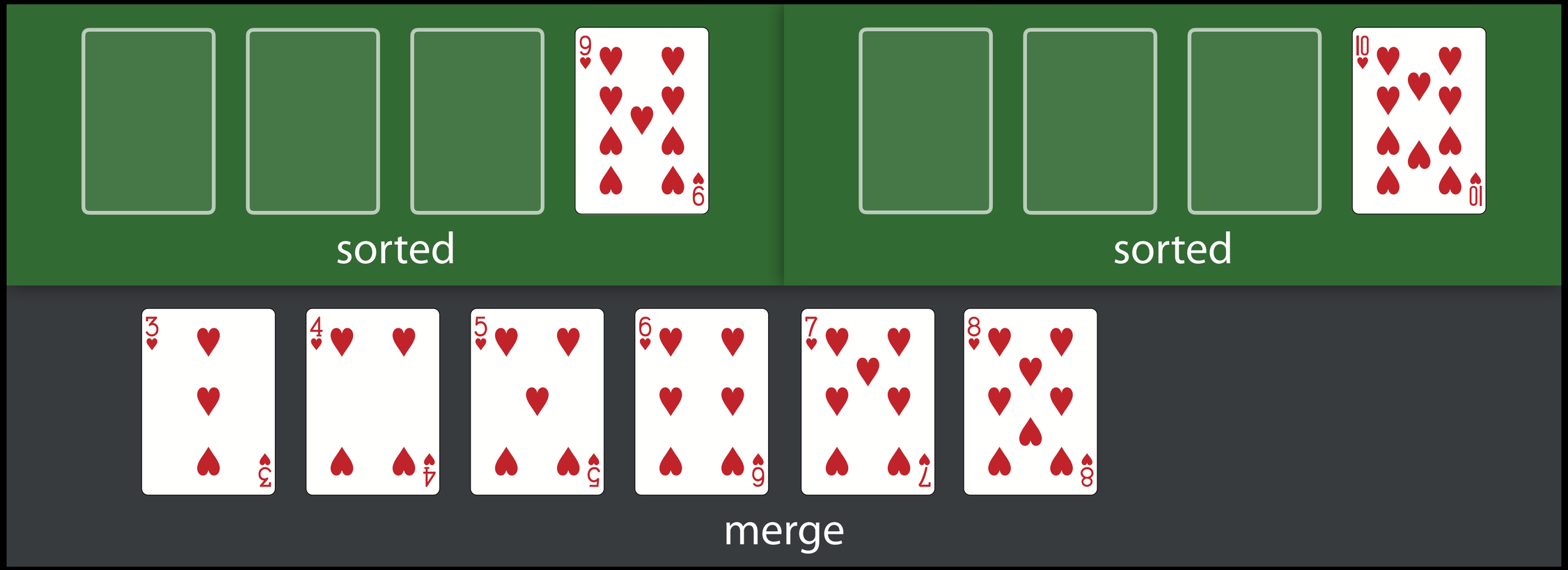
Merge sort



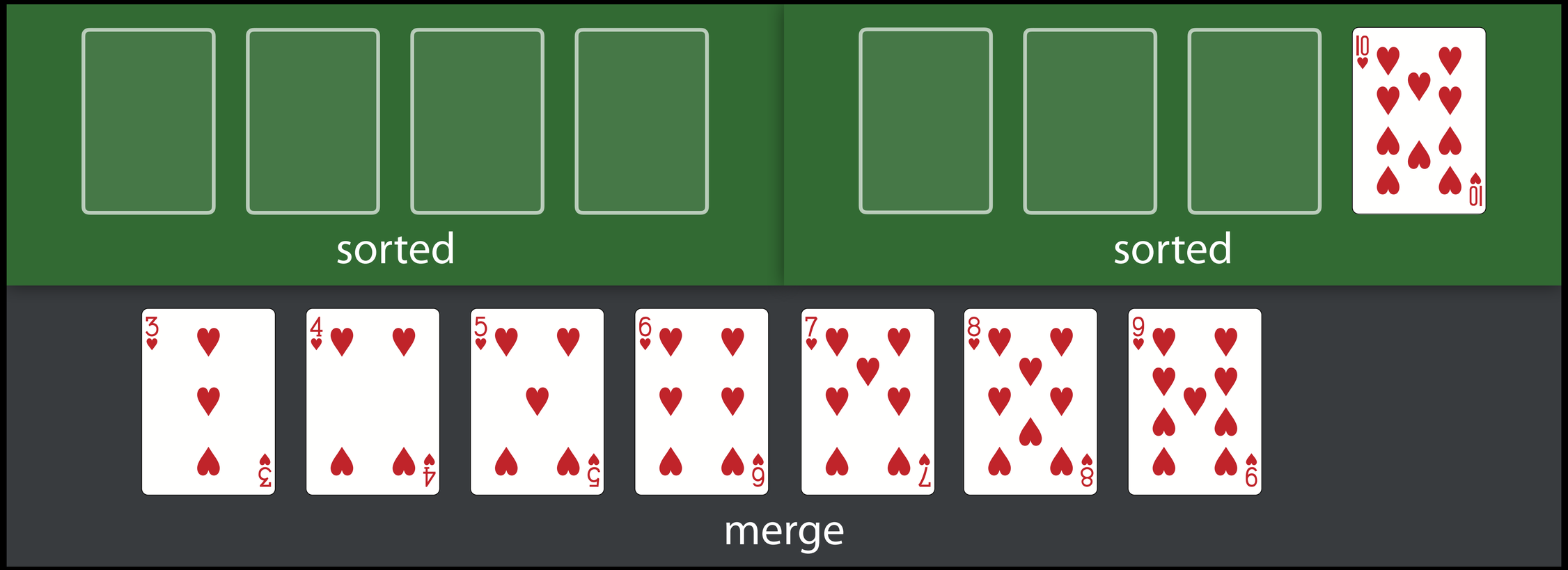
Merge sort



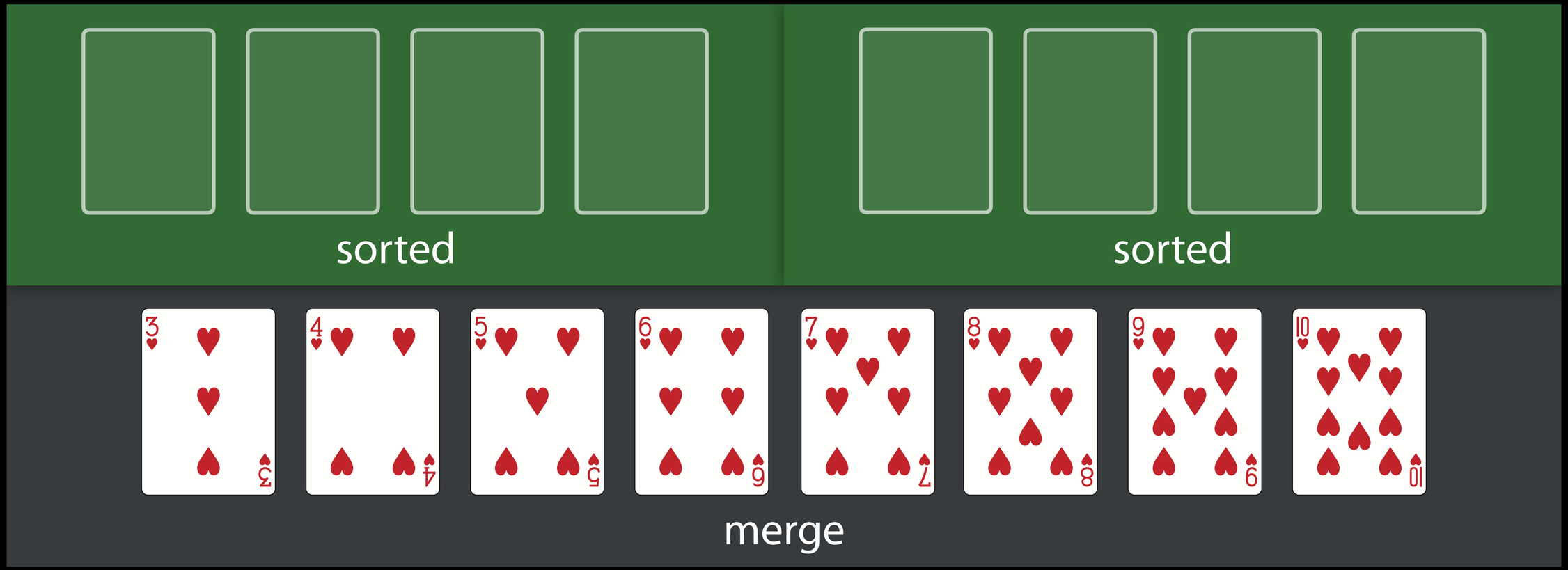
Merge sort



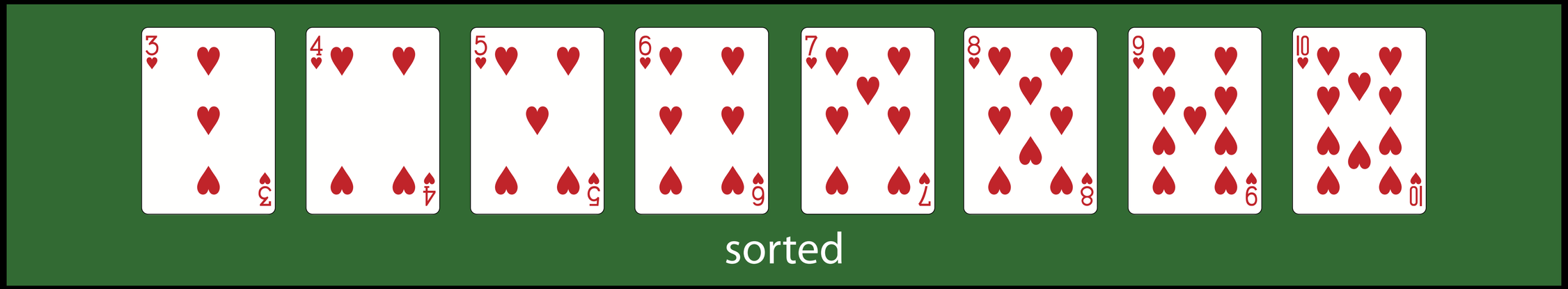
Merge sort



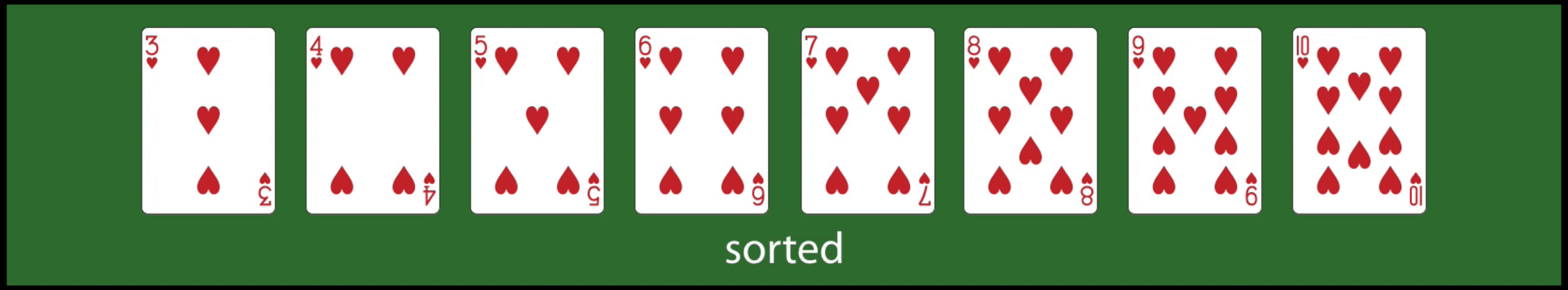
Merge sort



Merge sort



Merge sort



$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\hookrightarrow \Theta(n \log n)$$

Quicksort

Partitioning

To partition an array `A` on element `x = A[i]` is to rearrange `A[]` so that:

- `x` moves to position `j` (may be the same as `i`)
- All entries to the left of `x` are `≤ x`.
- All entries to the right of `x` are `≥ x`.

Partitioning

To partition an array `A` on element `x = A[i]` is to rearrange `A[]` so that:

- `x` moves to position `j` (may be the same as `i`)
- All entries to the left of `x` are ≤ `x`.
- All entries to the right of `x` are ≥ `x`.

`A[i]` (pivot)

5	550	10	4	10	9	330

Which partitions are valid?

A.

4	5	9	10	330	10	550

B.

5	9	10	4	10	550	330

C.

5	4	9	10	10	550	330

D.

5	9	10	4	10	550	330

Partitioning

To partition an array `A` on element `x = A[i]` is to rearrange `A[]` so that:

- `x` moves to position `j` (may be the same as `i`)
- All entries to the left of `x` are ≤ `x`.
- All entries to the right of `x` are ≥ `x`.

`A[i]` (pivot)

5	550	10	4	10	9	330

Which partitions are valid?

A.

4	5	9	10	330	10	550

B.

5	9	10	4	10	550	330

C.

5	4	9	10	10	550	330

D.

5	9	10	4	10	550	330

Quicksort

Partitioning puts pivot in the correct position.

Keep partitioning until all items become pivots.

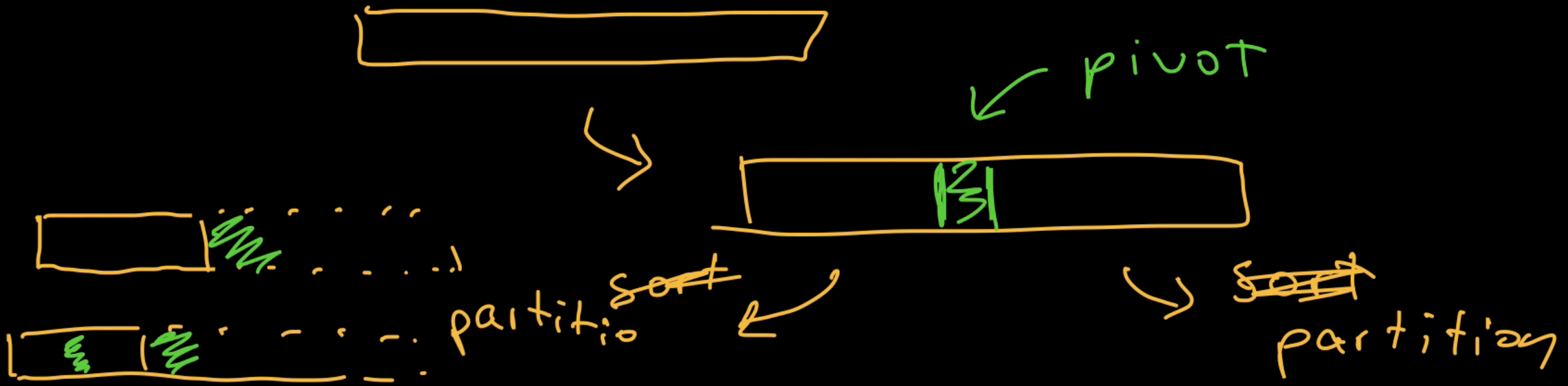
For most common situations, it is empirically the fastest sort.

Quicksort

Partitioning puts pivot in the correct position.

Keep partitioning until all items become pivots.

For most common situations, it is empirically the fastest sort.



Partitioning

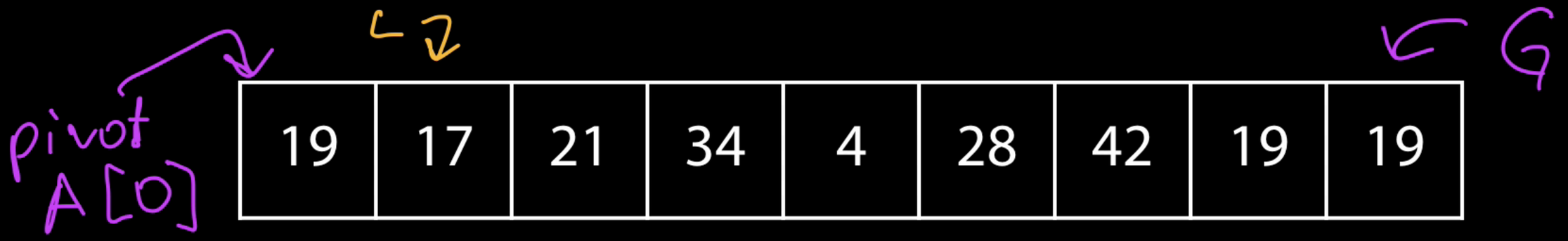
Put pivot at position 0
Create L and G pointers at left and right ends
L pointer is a friend to small items and hates large or equal items
G pointer is a friend to large items and hates small or equal items
Repeat until pointers cross:
    Walk pointers toward each other stopping on hated items
    When pointers have stopped, swap items and move pointers by one
Swap pivot and element pointed to by G

19	17	21	34	4	28	42	19	19

Partitioning

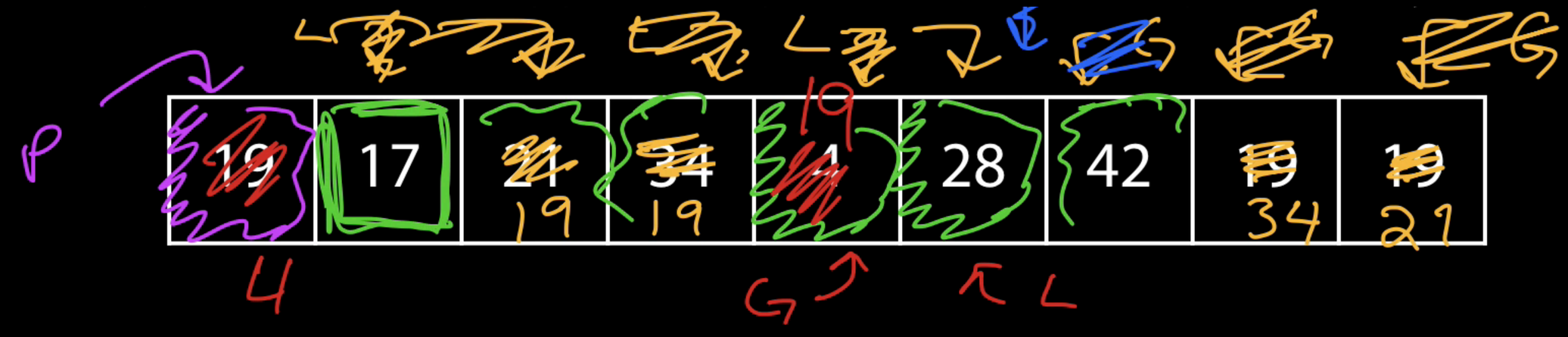
$\Theta(n)$ time
$\Theta(1)$ space

Put pivot at position 0
Create L and G pointers at left and right ends
L pointer is a friend to small items and hates large or equal items
G pointer is a friend to large items and hates small or equal items
Repeat until pointers cross:
    Walk pointers toward each other stopping on hated items
    When pointers have stopped, swap items and move pointers by one
Swap pivot and element pointed to by G



Partitioning

Put pivot at position 0
Create L and G pointers at left and right ends
L pointer is a friend to small items and hates large or equal items
G pointer is a friend to large items and hates small or equal items
Repeat until pointers cross:
    Walk pointers toward each other stopping on hated items
    When pointers have stopped, swap items and move pointers by one
Swap pivot and element pointed to by G



Pick the best sort

Suppose we do the following:

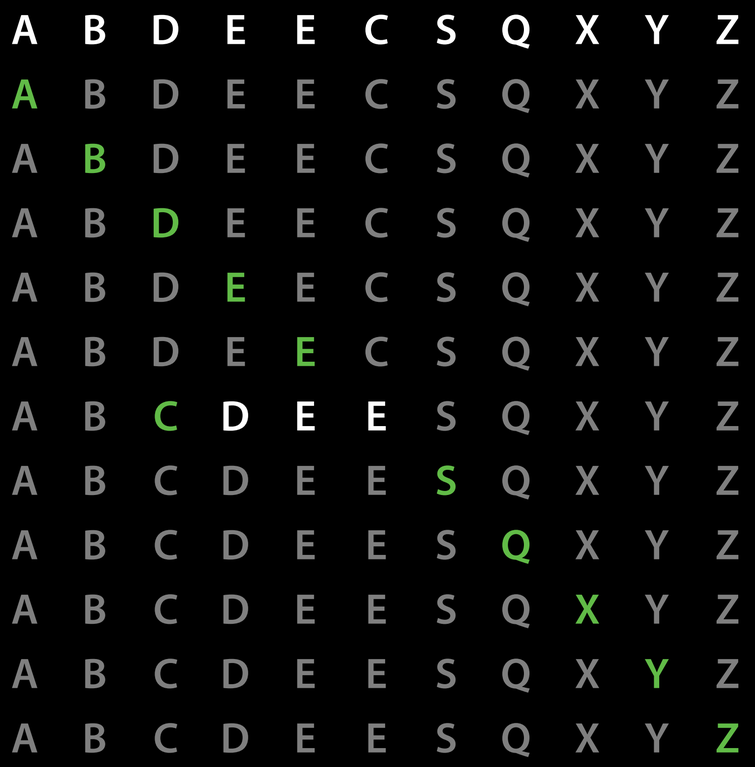
- Read 1,000,000 integers from a file into an array of length 1,000,000.
- Use merge sort to sort these integers.
- Randomly select one integer and change it.
- Sort using algorithm **S** of your choice.

Which sorting algorithm would be the fastest choice for **S**?

A. Selection sort
B. Heap sort
C. Merge sort
D. Insertion sort

Almost sorted arrays

For arrays that are almost sorted, insertion sort does very little work.



Find Sum

Interview Question

Given an integer **`sum`** and a sorted array **`numbers`** of **`N`** distinct integers, implement a function to find if there exist indices **`i`** and **`j`** such that **`numbers[i] + numbers[j] == x`**.

```
bool findSum(const vector<int> &numbers, int sum) {
    for (int i = 0; i < numbers.size(); i += 1) {
        for (int j = 0; j < numbers.size(); j += 1) {
            if (numbers[i] + numbers[j] == sum) {
                return true;
            }
        }
    }
    return false;
}
```

Find Sum

Interview Question

Given an integer **`sum`** and a sorted array **`numbers`** of **`N`** distinct integers, implement a function to find if there exist indices **`i`** and **`j`** such that **`numbers[i] + numbers[j] == x`**.

```
bool betterFindSum(const vector<int> &numbers, int sum);
```

Implement a better, more efficient version of the algorithm.