Week 3: Thursday

EECS 281

Midterm Exam Review

Mon 5/25, time and location TBA

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.

Sorting Algorithms

Bubble sort

Selection sort

Insertion sort

Heapsort

Merge sort

Quicksort

Bubble Sort

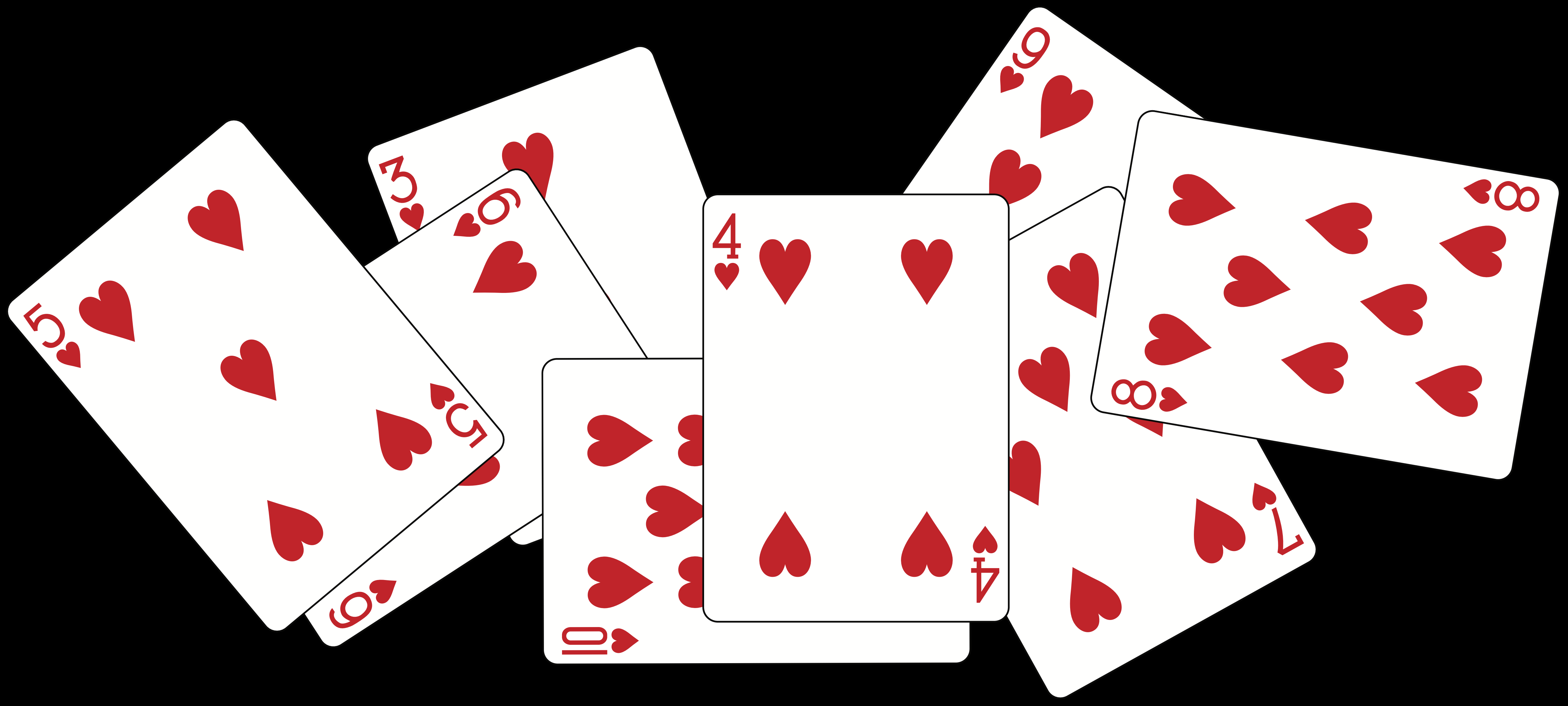
Reduce the number of inversions by going through the array and swapping adjacent elements if they are an inversion pair.

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]
```

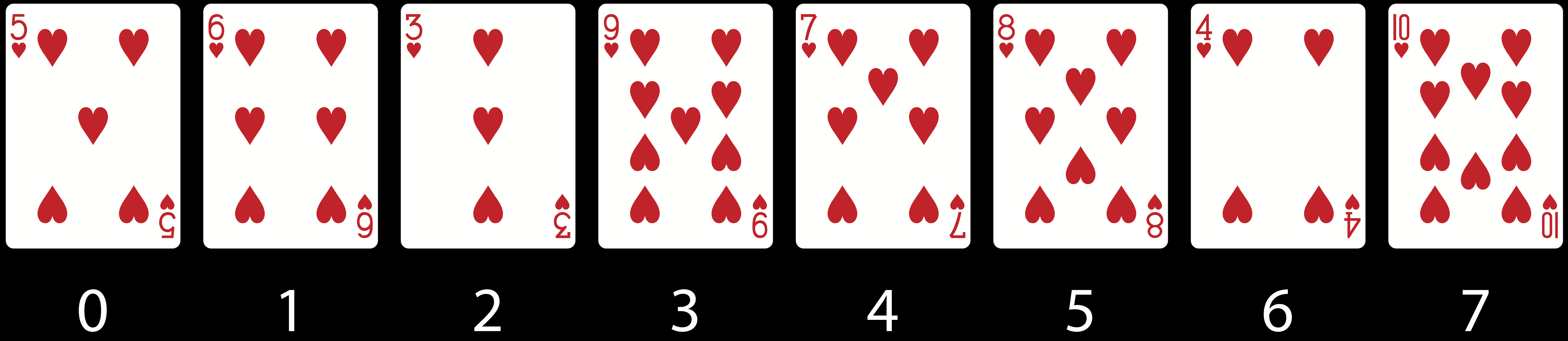
Bubble Sort

```
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]
```



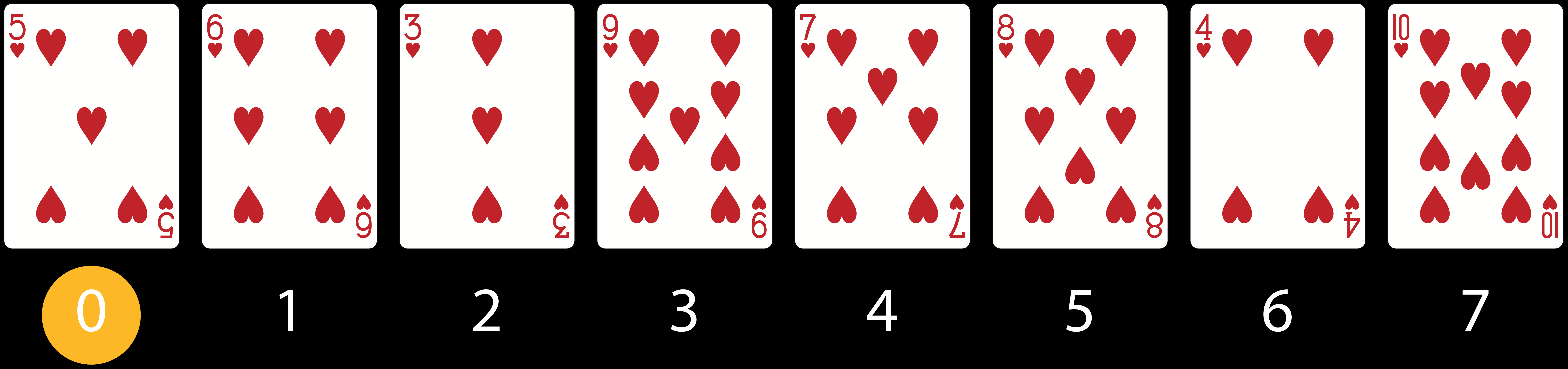
Bubble Sort

```
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]
```



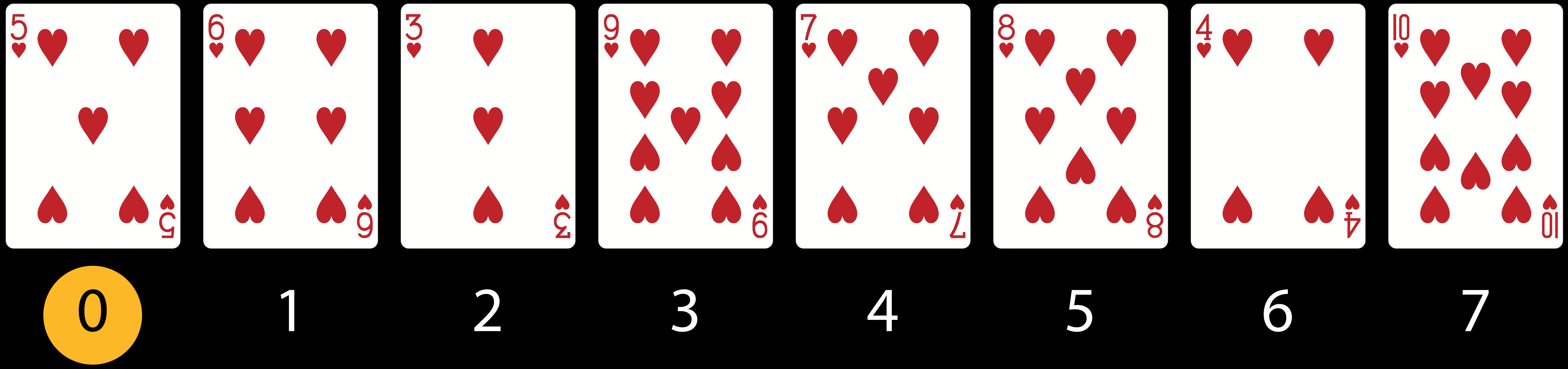
Bubble Sort

```
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]
```



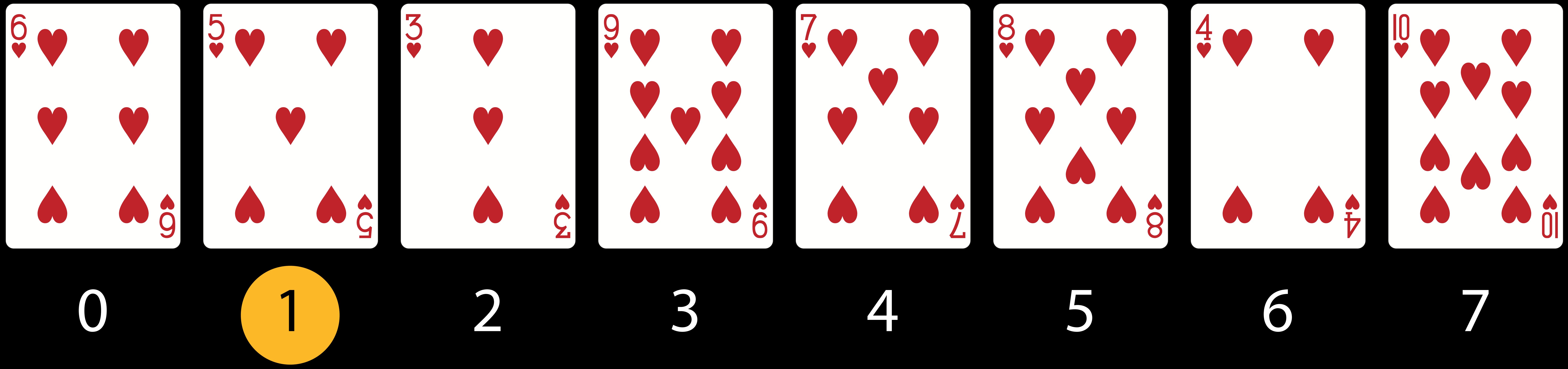
Bubble Sort

```
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]
```



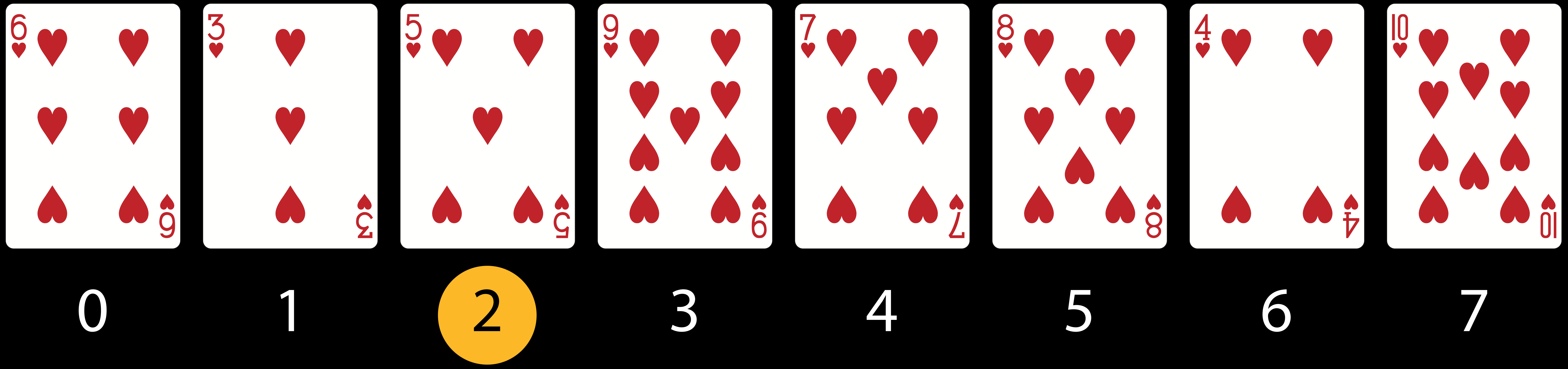
Bubble Sort

```
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]
```



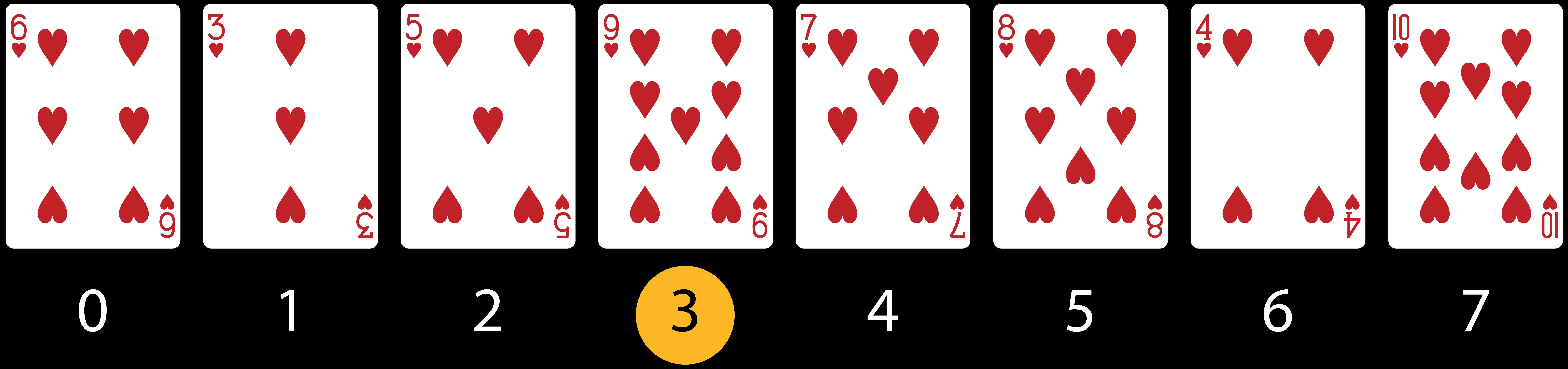
Bubble Sort

```
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]
```



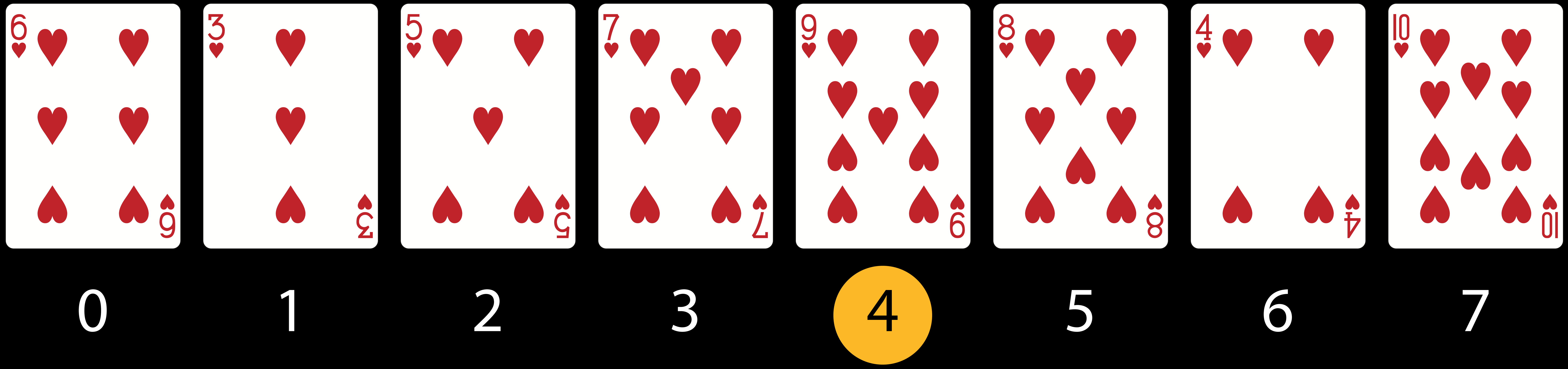
Bubble Sort

```
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]
```



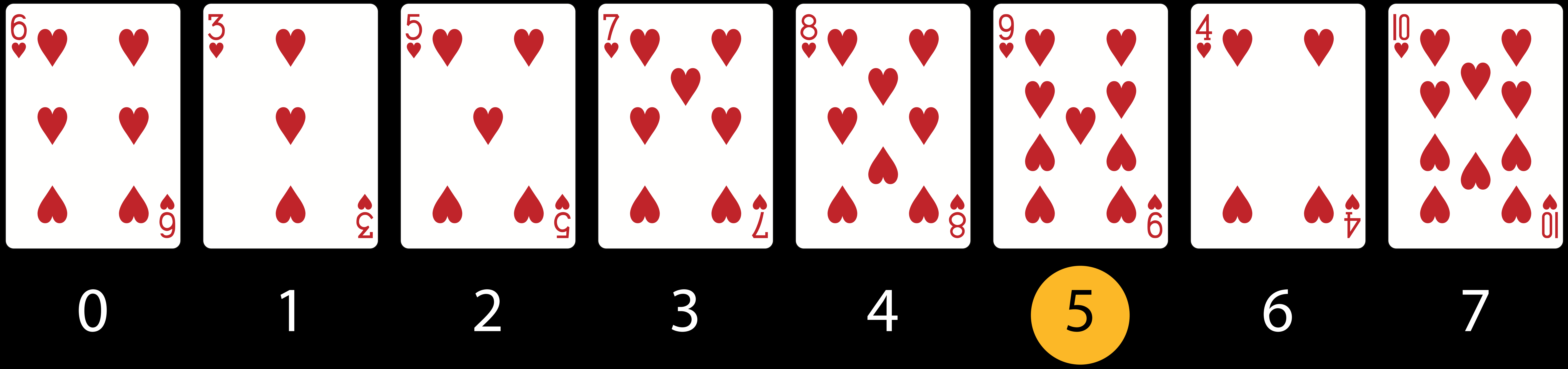
Bubble Sort

```
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]
```



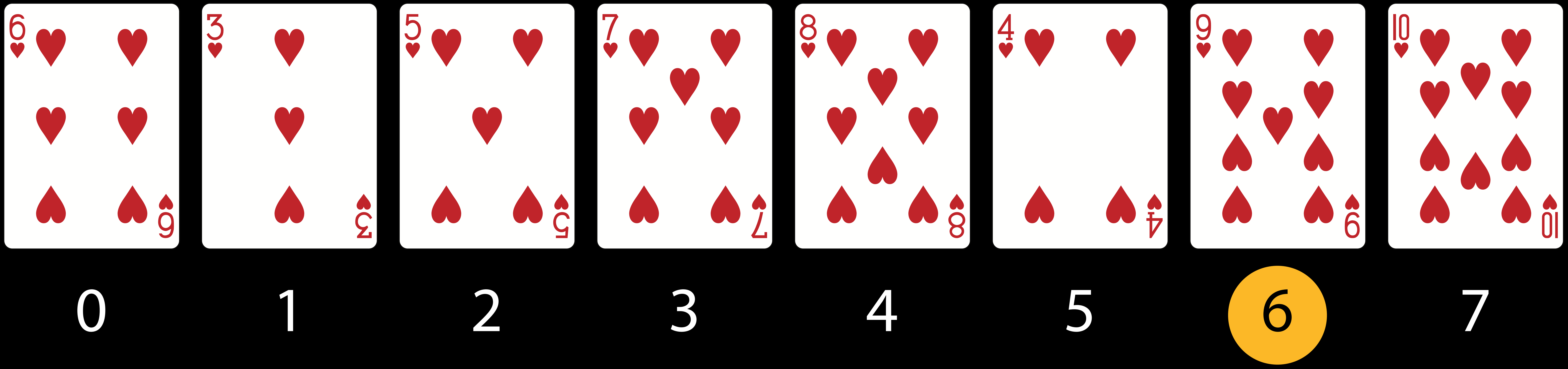
Bubble Sort

```
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]
```



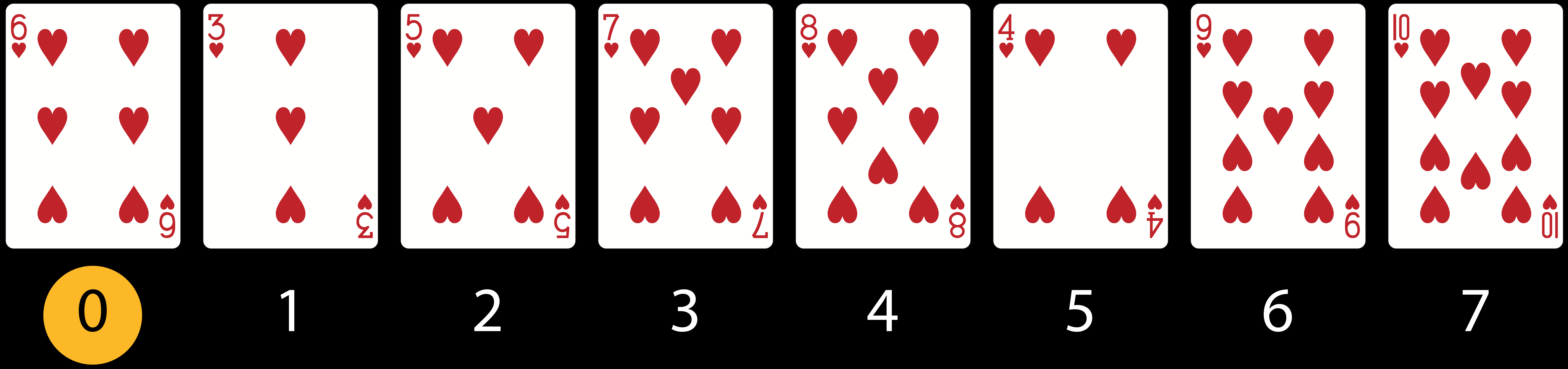
Bubble Sort

```
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]
```



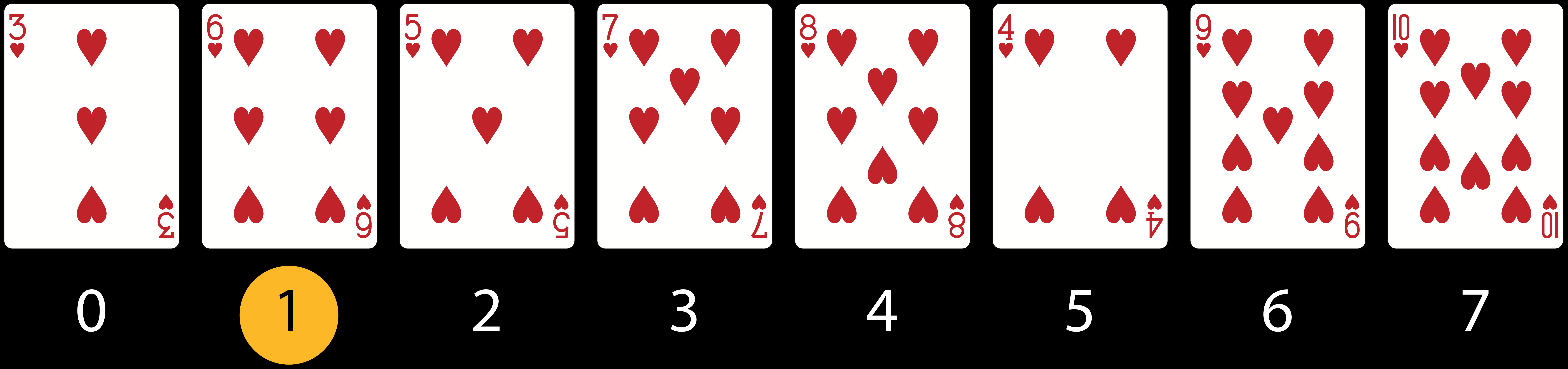
Bubble Sort

```
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]
```



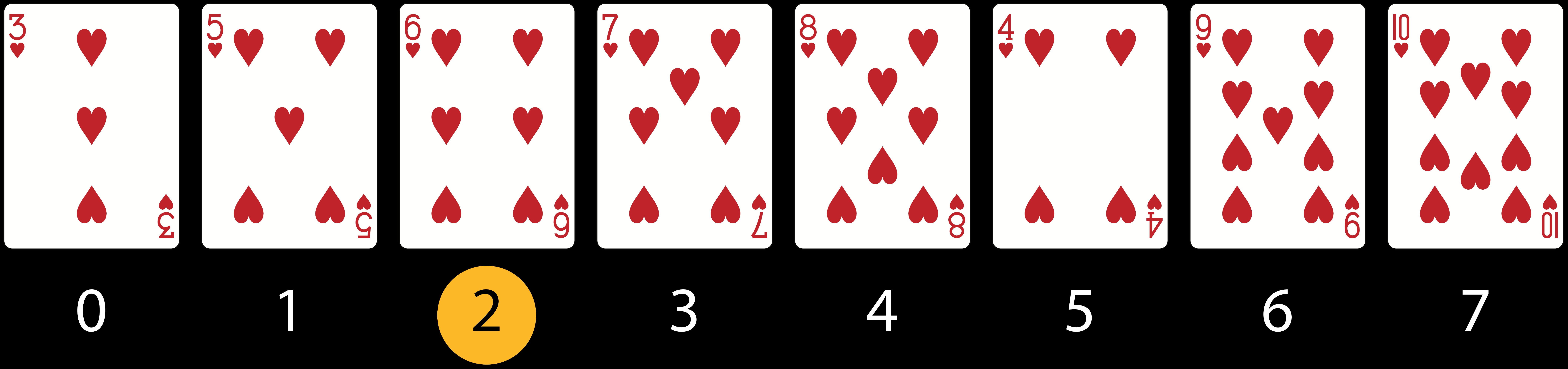
Bubble Sort

```
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]
```



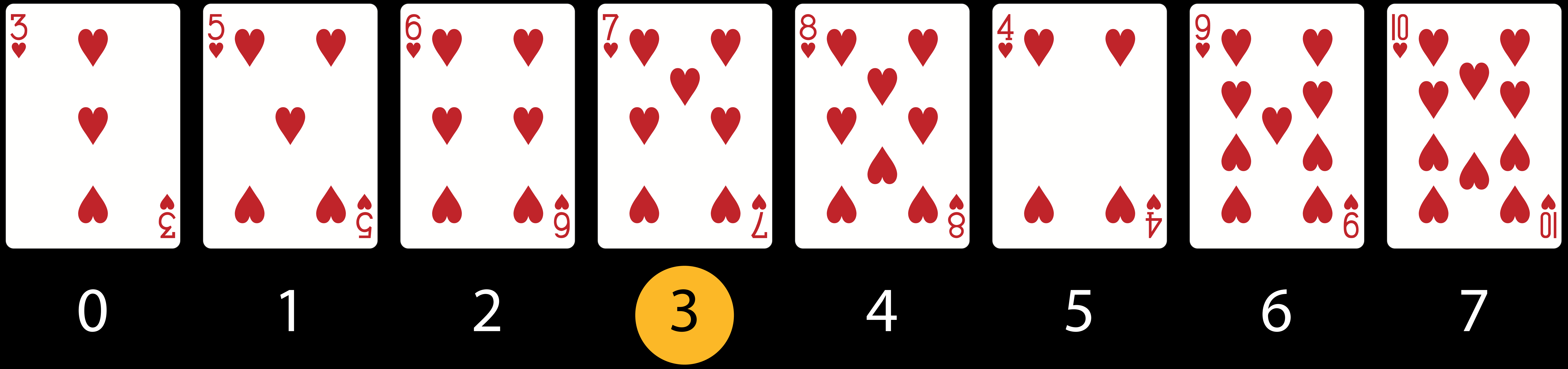
Bubble Sort

```
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]
```



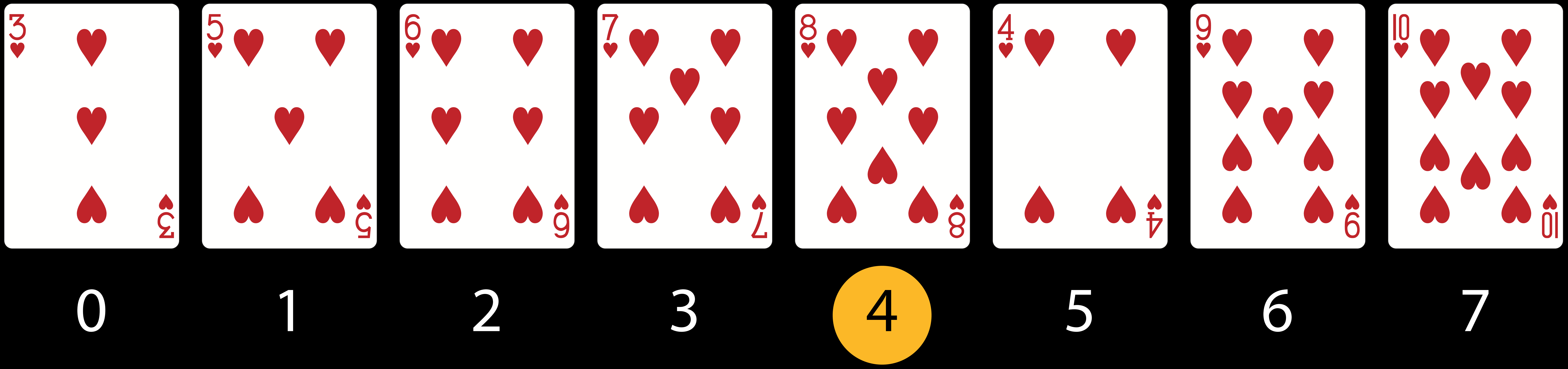
Bubble Sort

```
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]
```



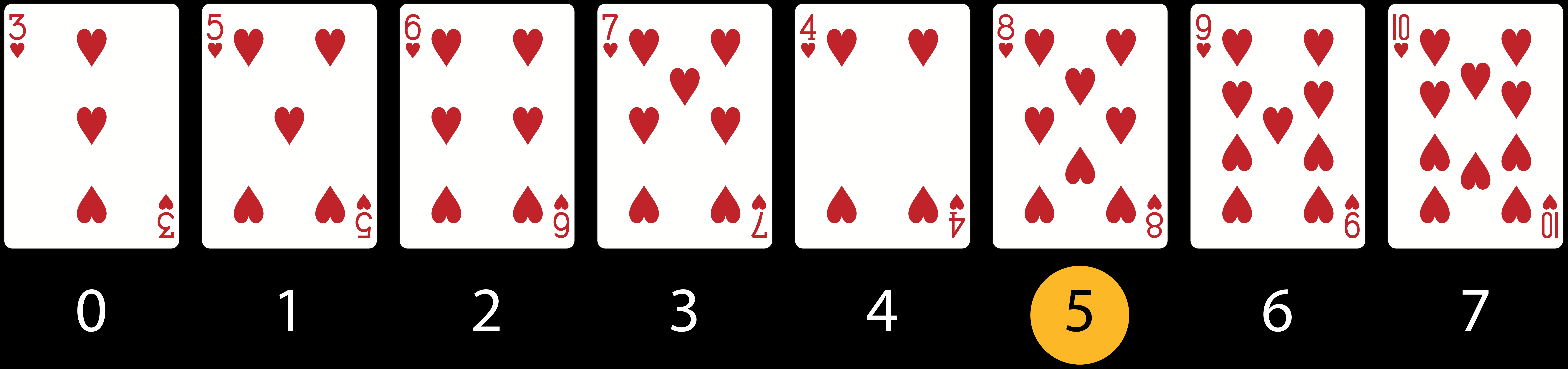
Bubble Sort

```
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]
```



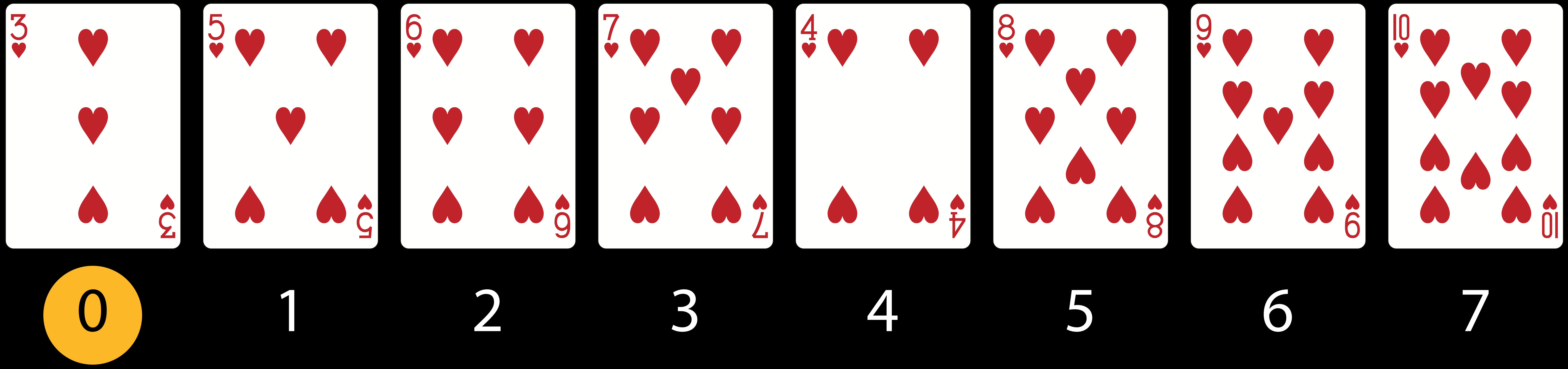
Bubble Sort

```
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]
```



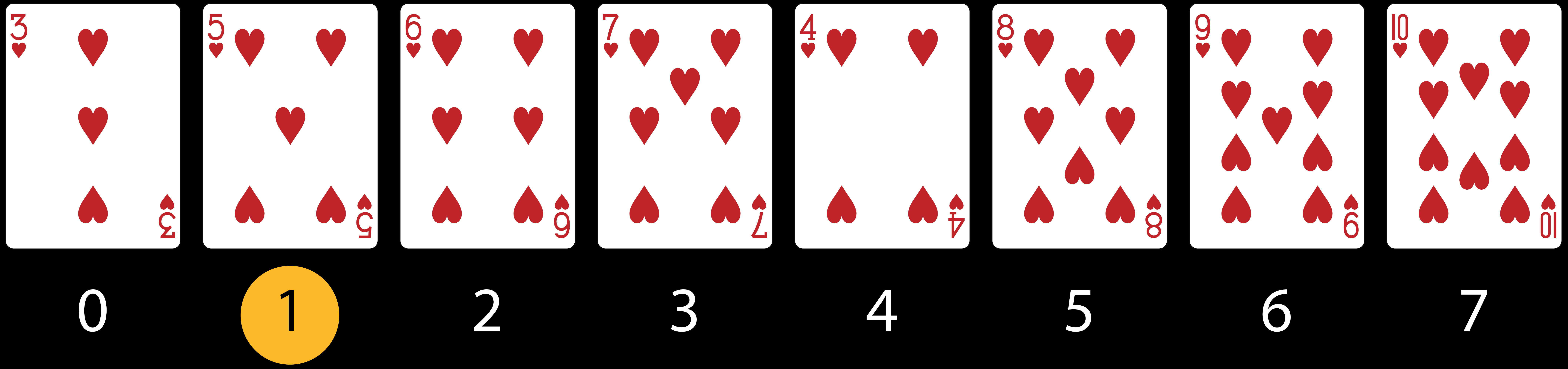
Bubble Sort

```
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]
```



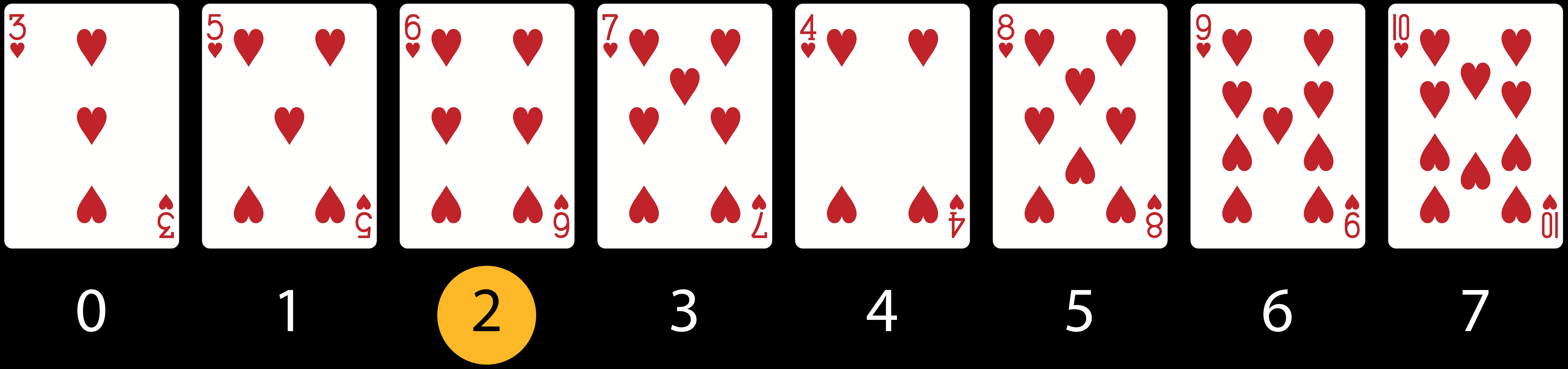
Bubble Sort

```
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]
```



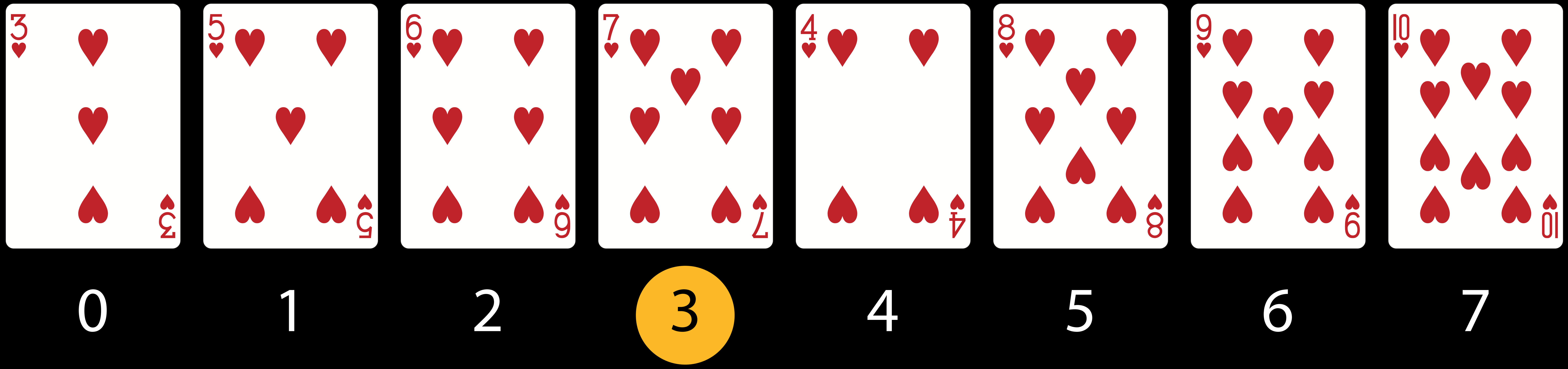
Bubble Sort

```
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]
```



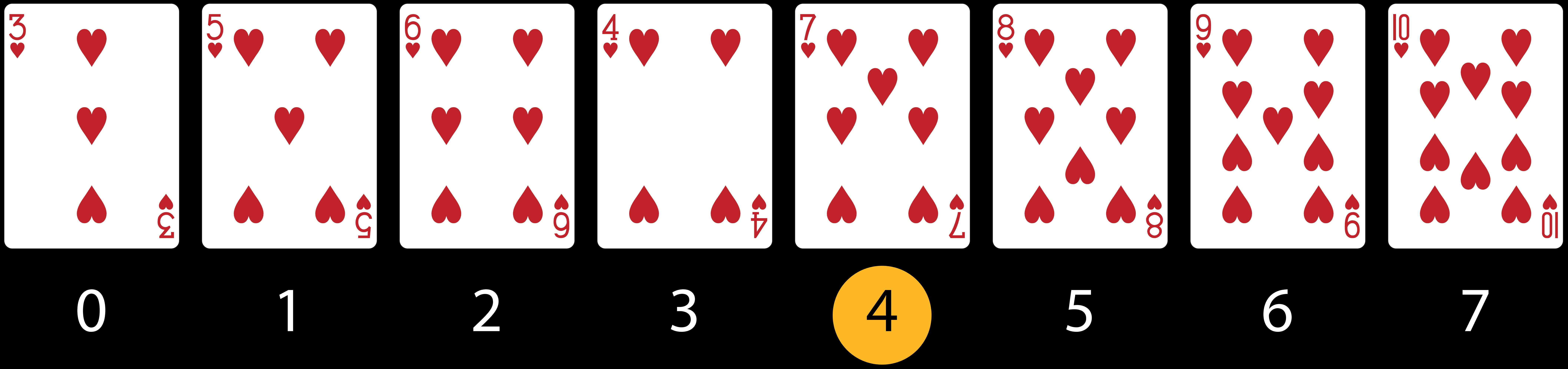
Bubble Sort

```
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]
```



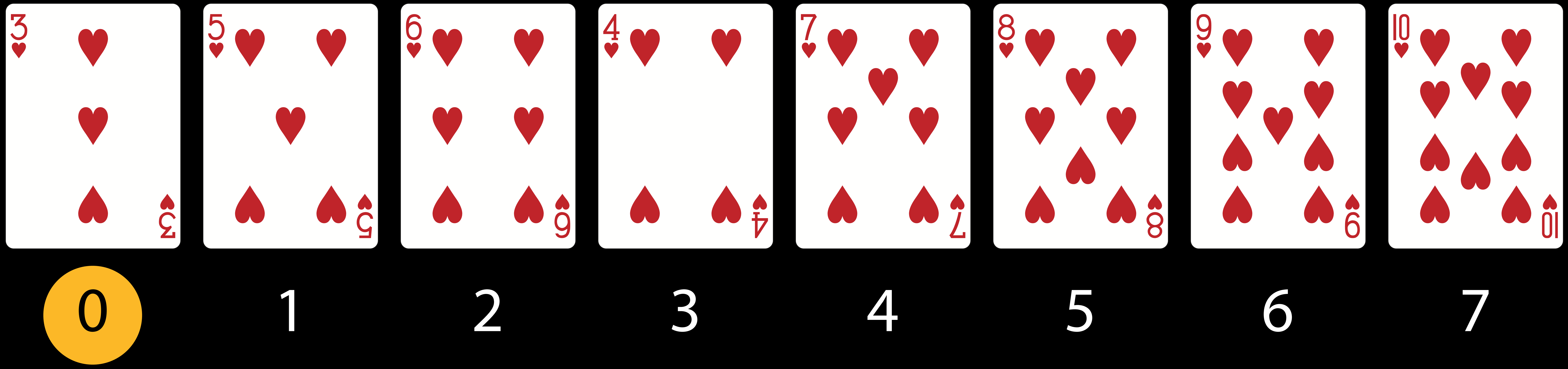
Bubble Sort

```
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]
```



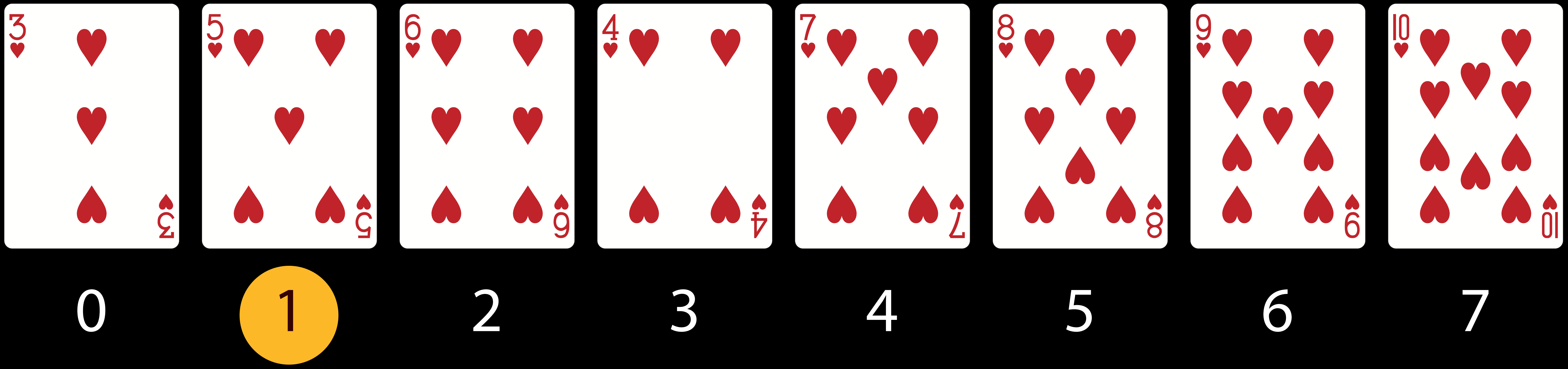
Bubble Sort

```
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]
```



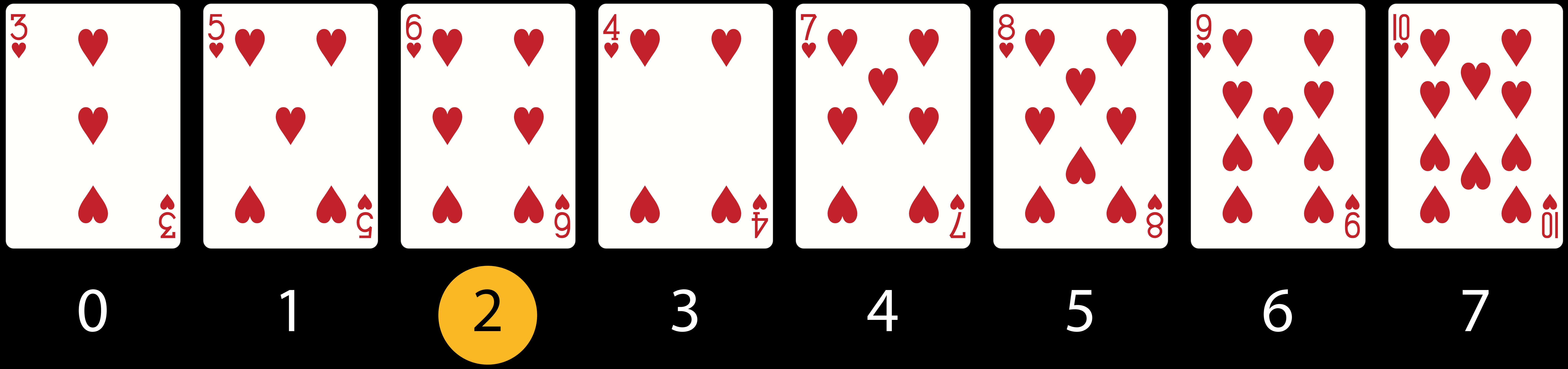
Bubble Sort

```
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]
```



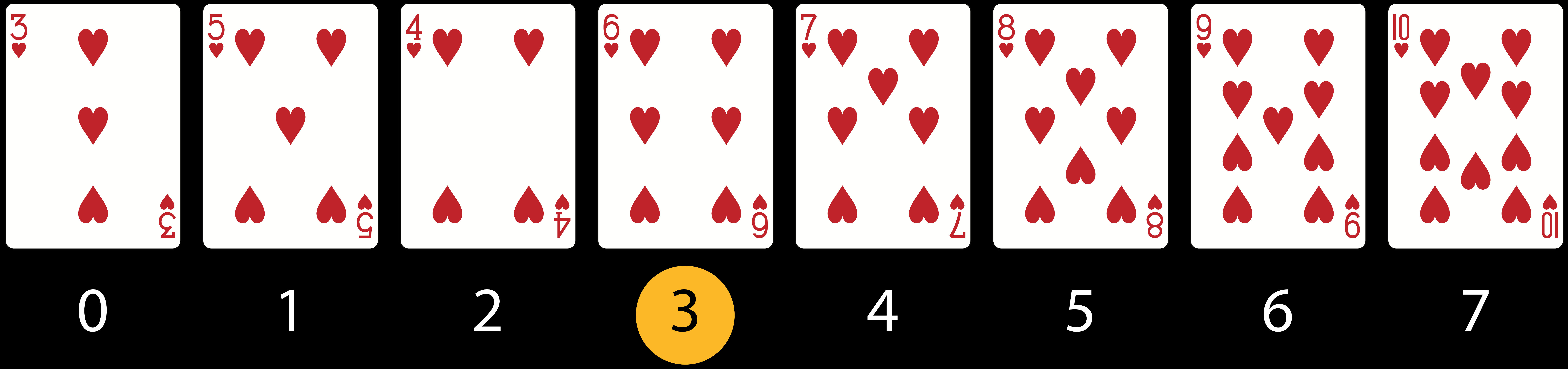
Bubble Sort

```
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]
```



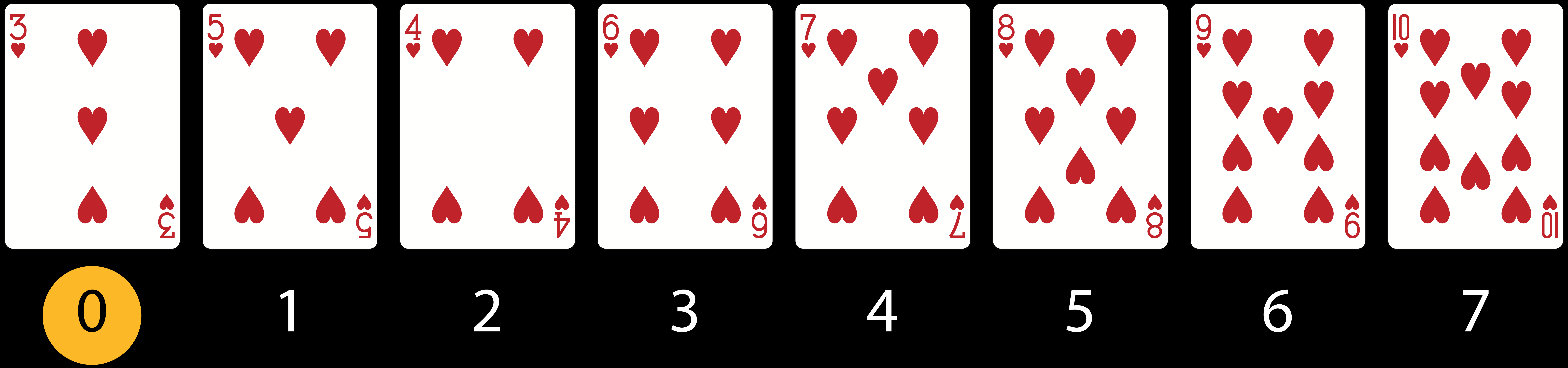
Bubble Sort

```
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]
```



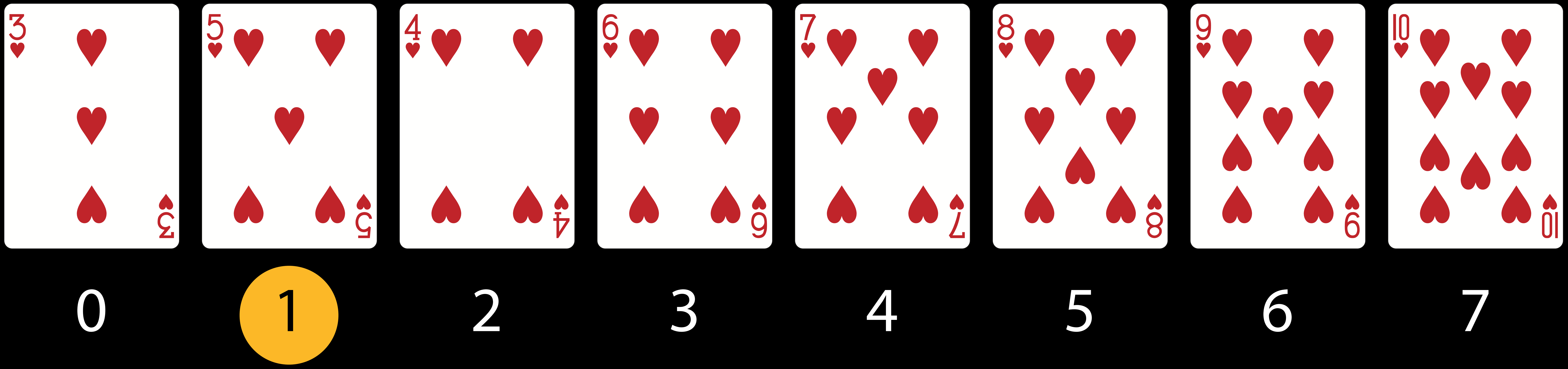
Bubble Sort

```
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]
```



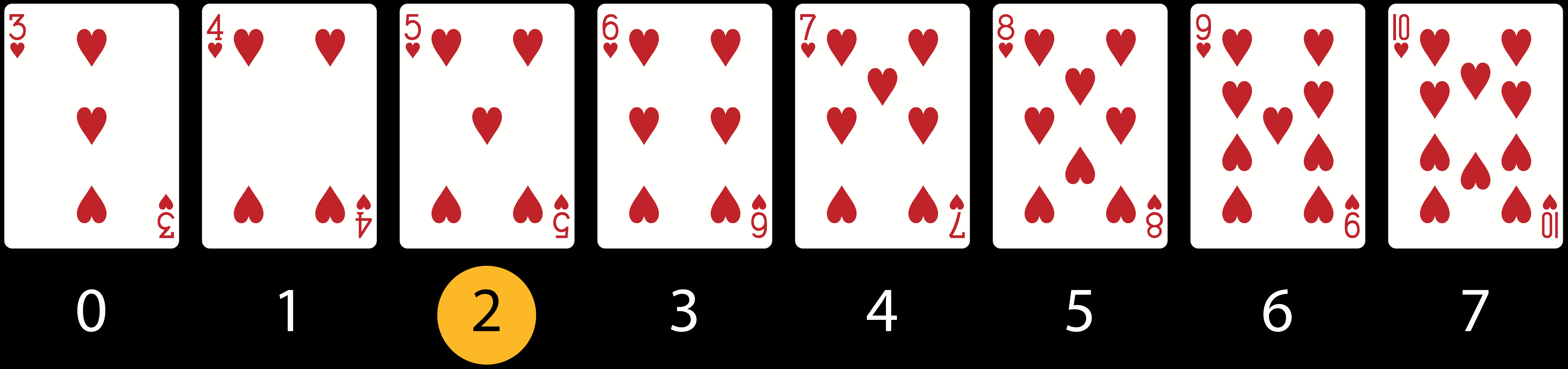
Bubble Sort

```
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]
```



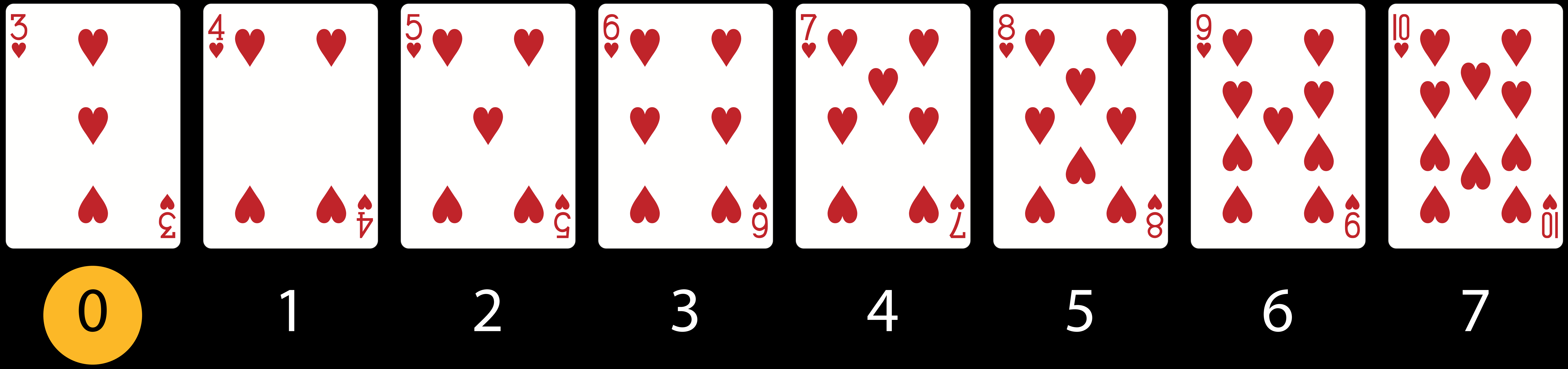
Bubble Sort

```
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]
```



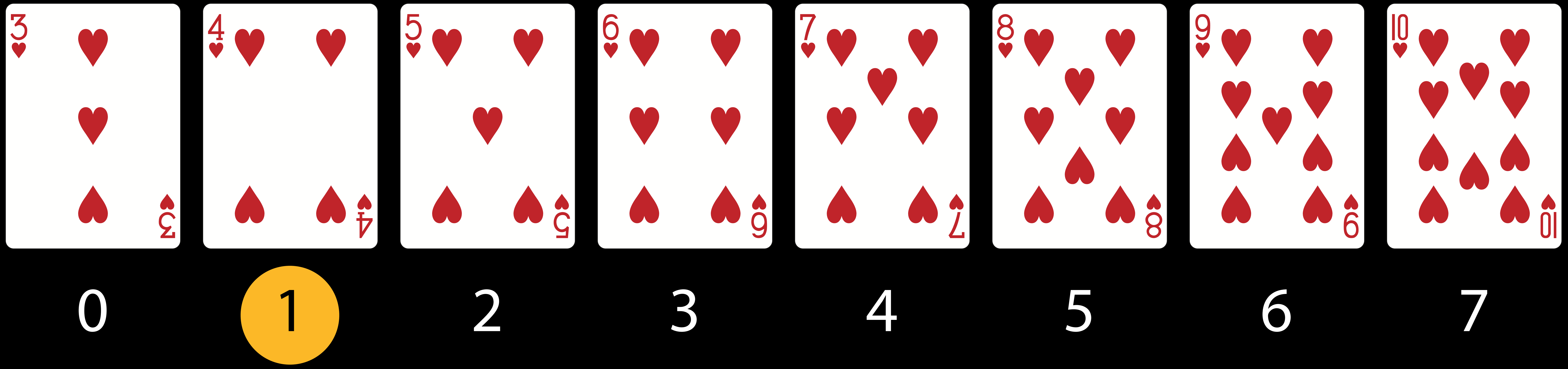
Bubble Sort

```
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]
```



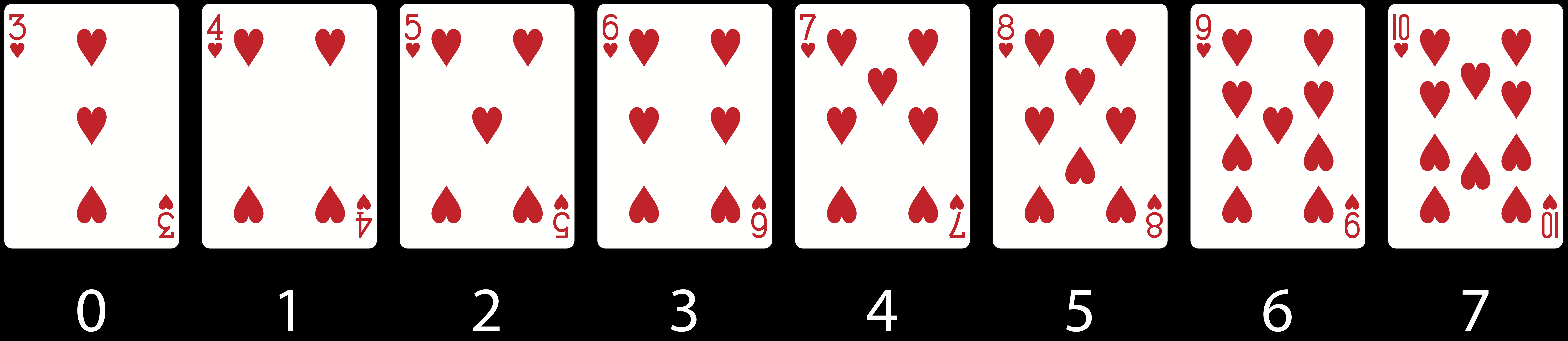
Bubble Sort

```
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]
```



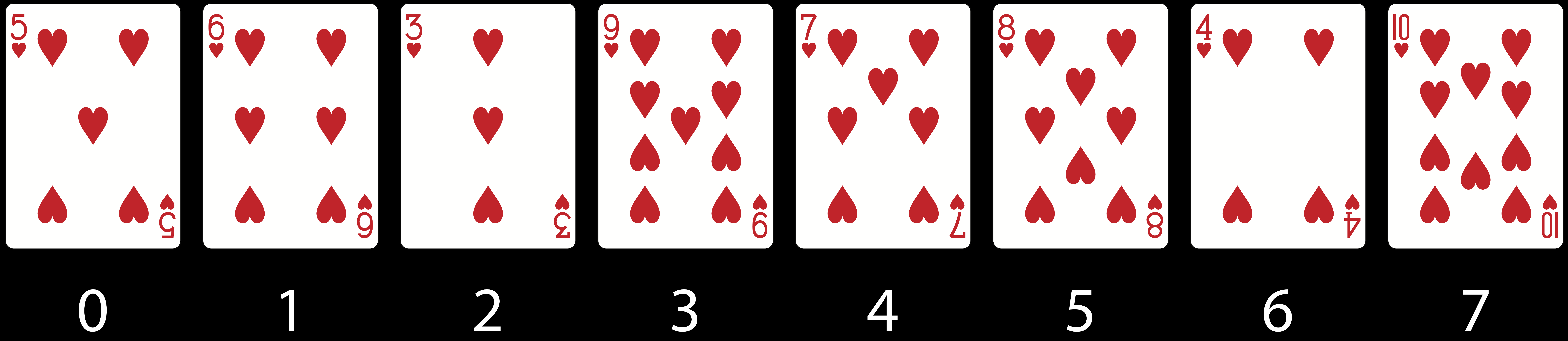
Bubble Sort

```
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]
```



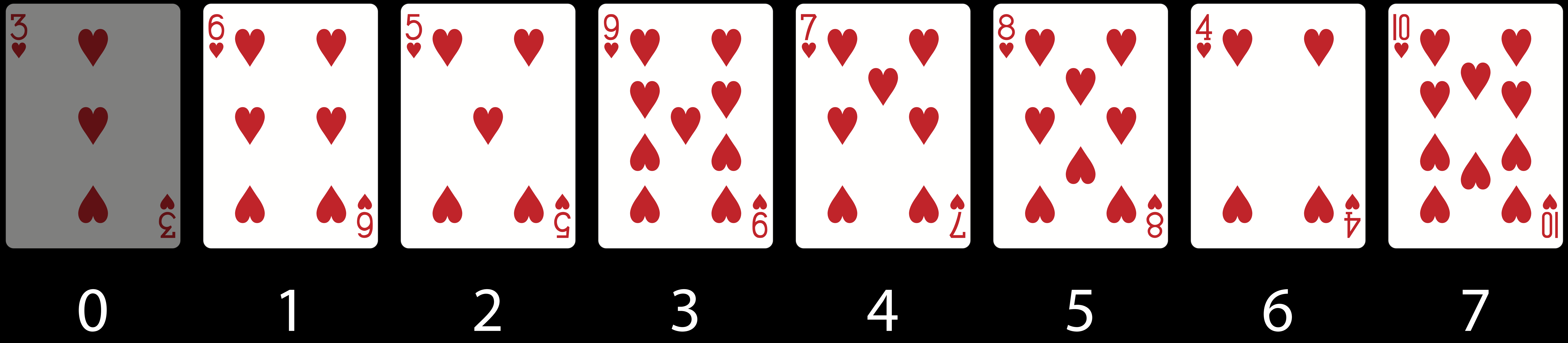
Selection Sort

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.



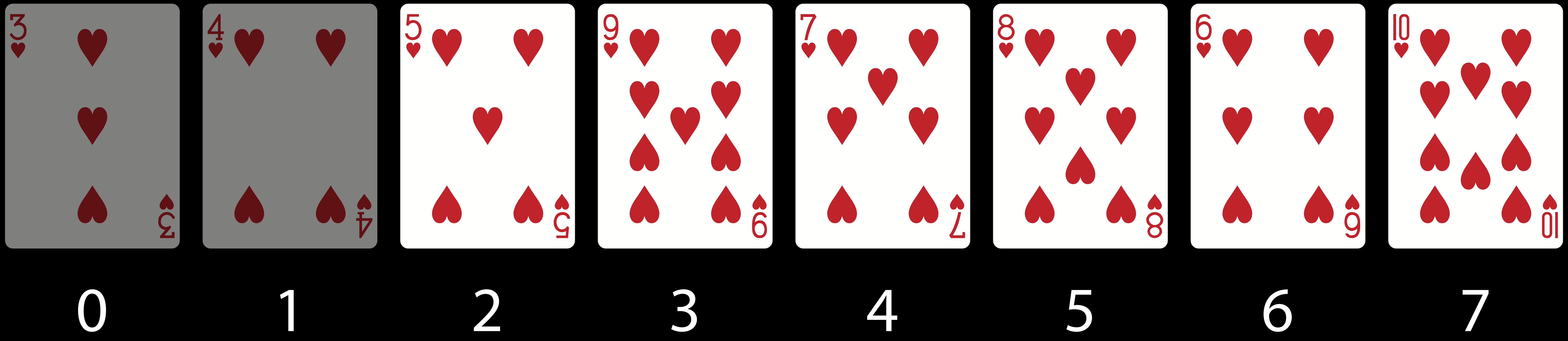
Selection Sort

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.



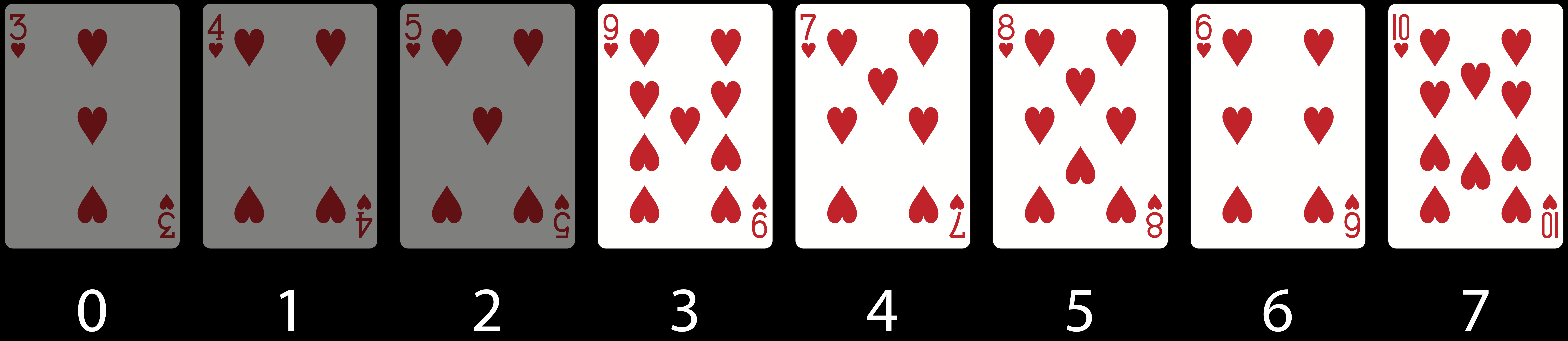
Selection Sort

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.



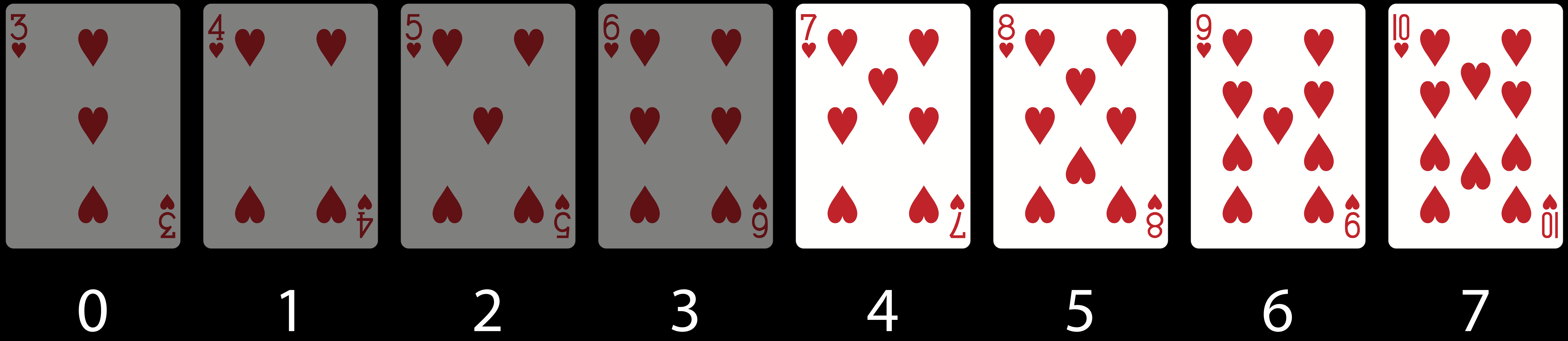
Selection Sort

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.



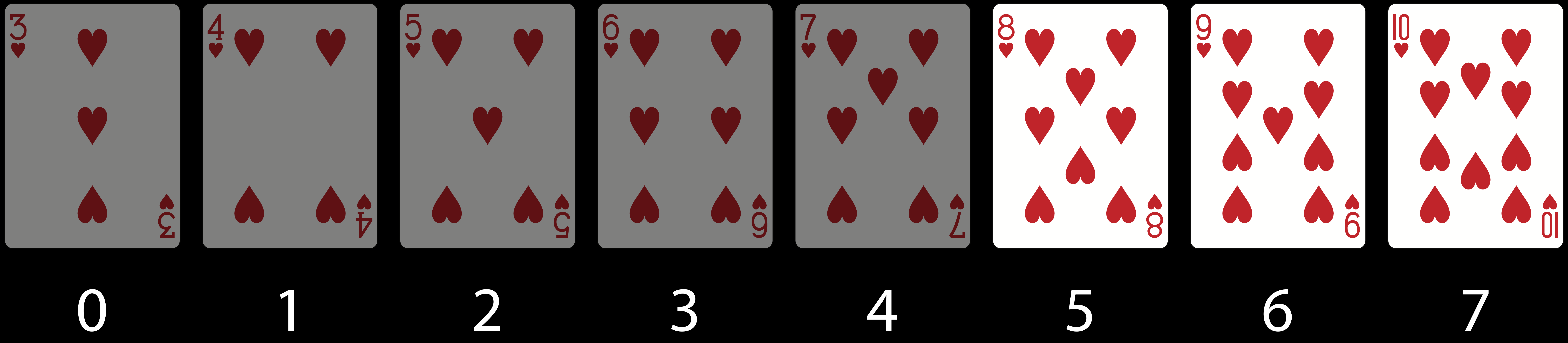
Selection Sort

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.



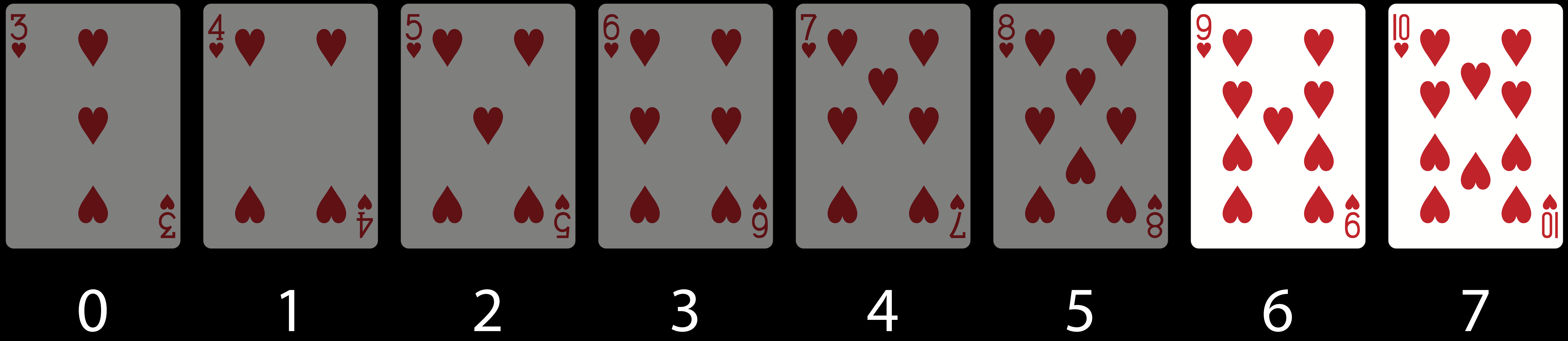
Selection Sort

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.



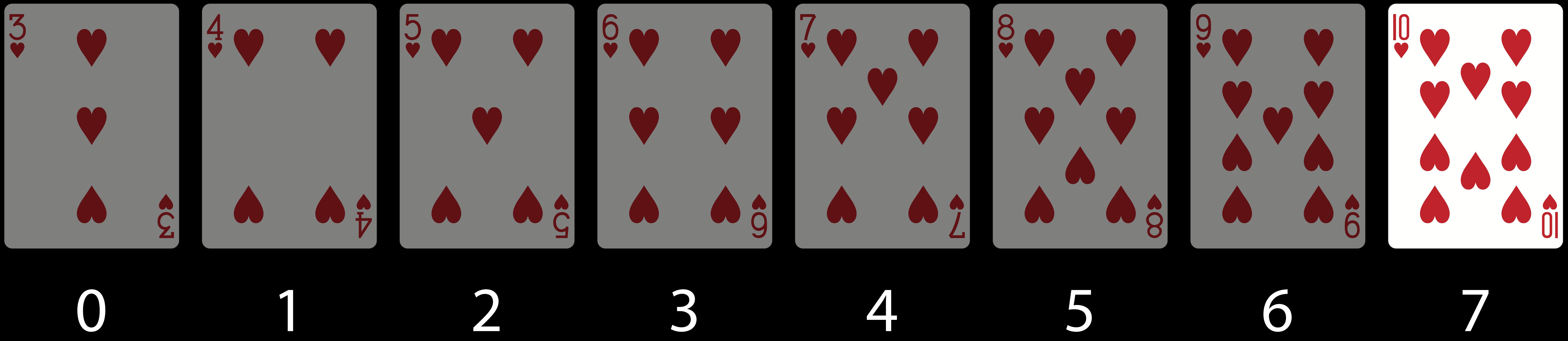
Selection Sort

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.



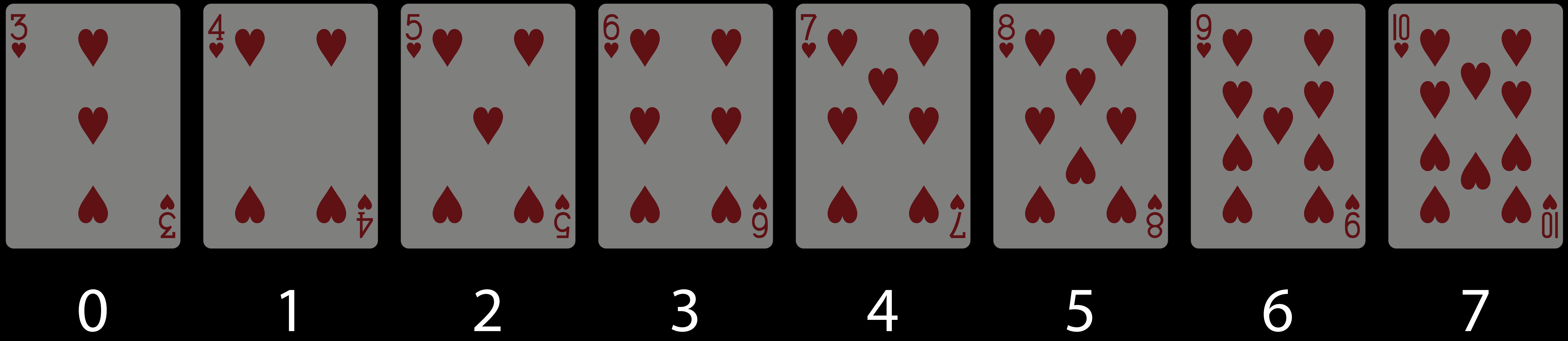
Selection Sort

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.



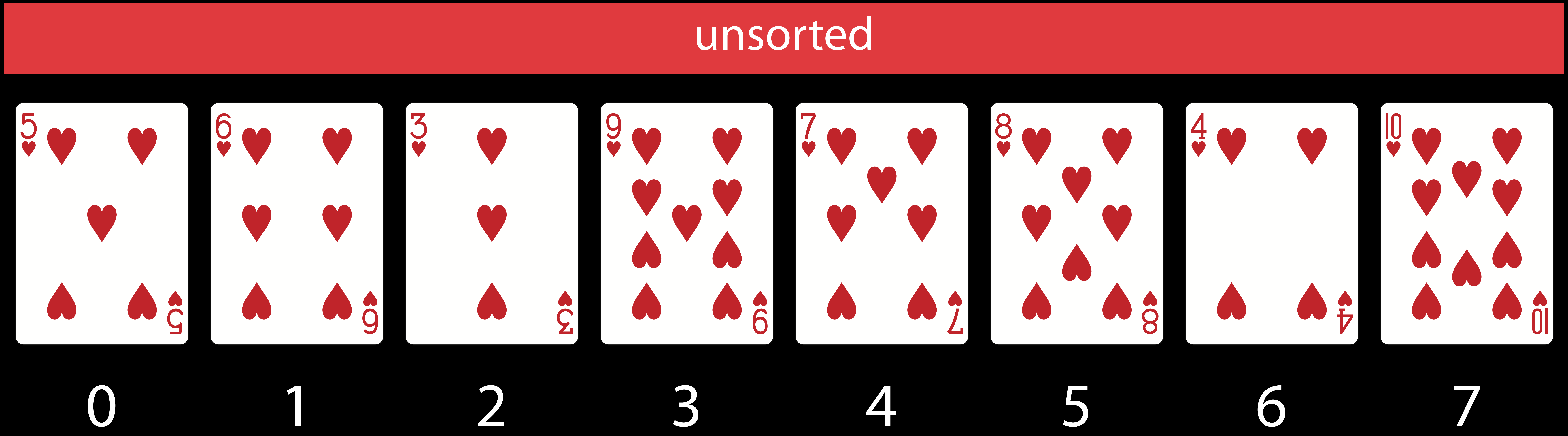
Selection Sort

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.



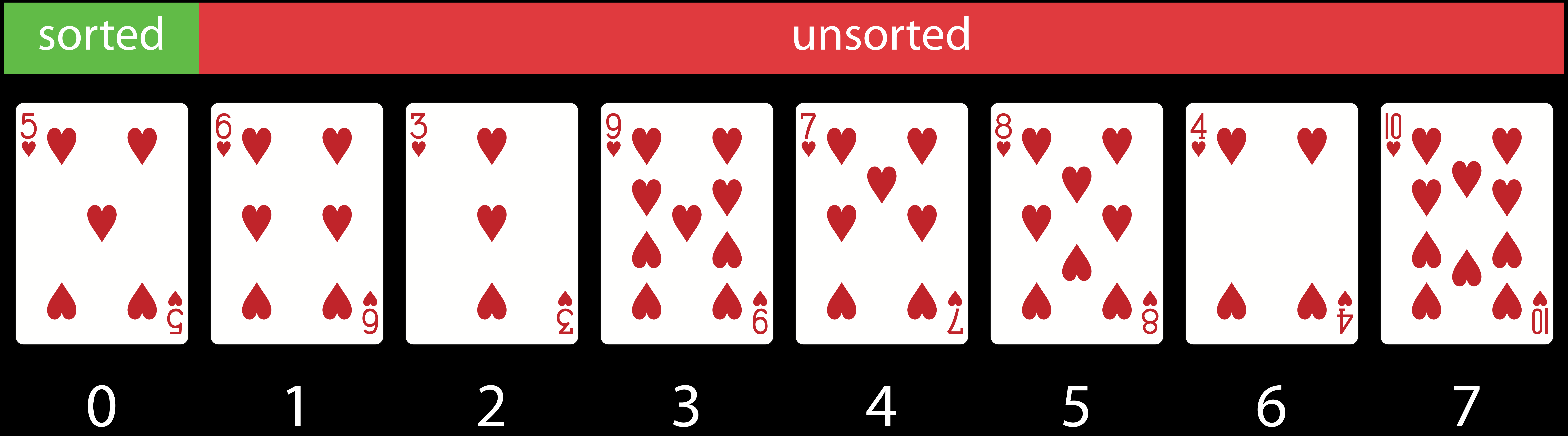
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



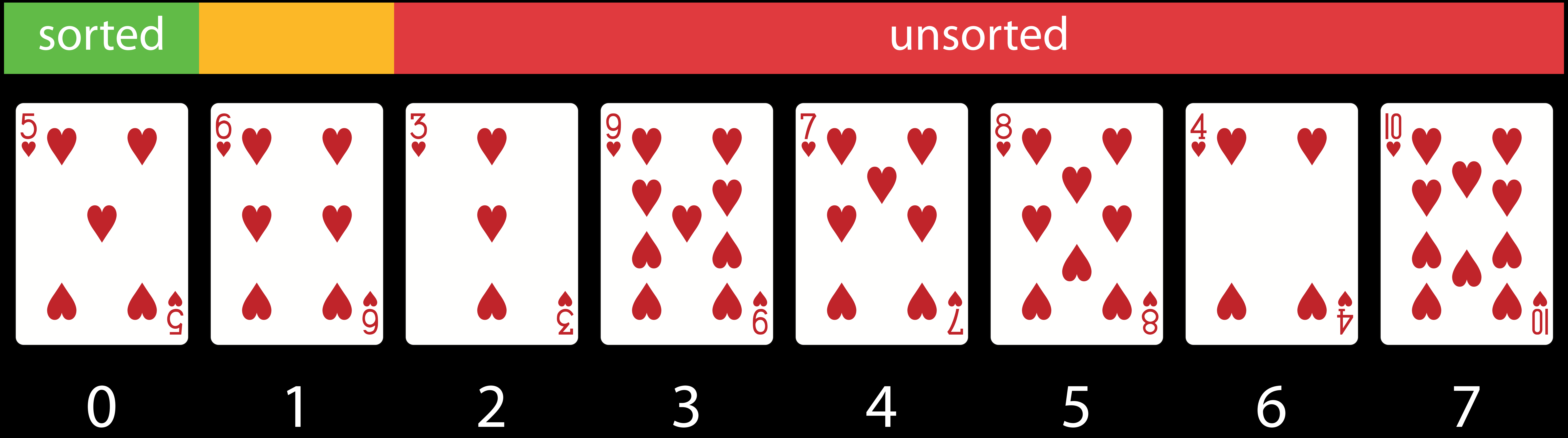
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



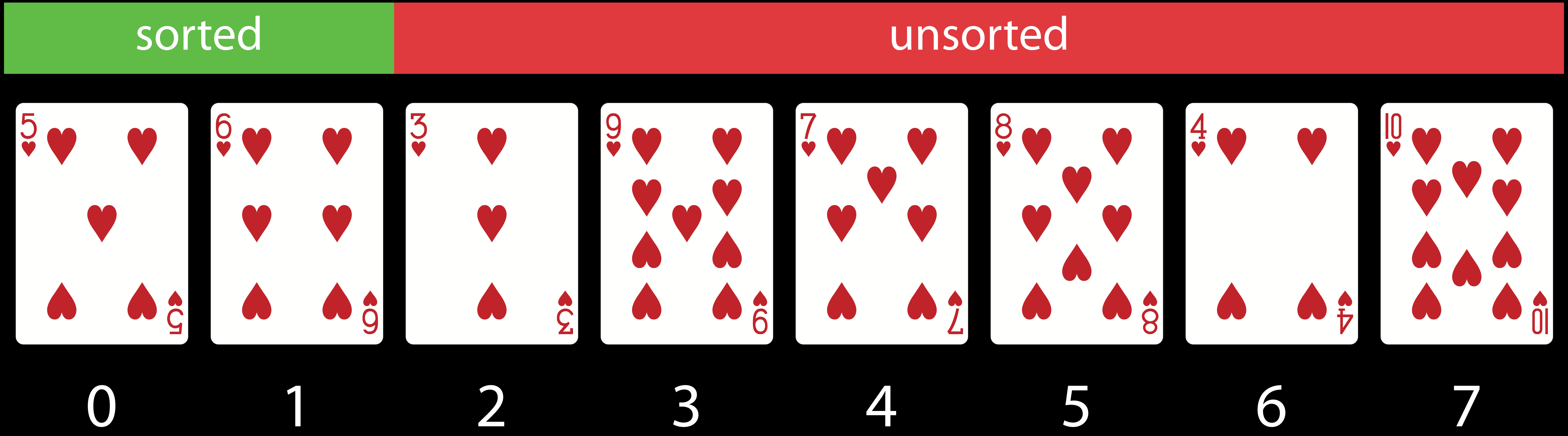
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



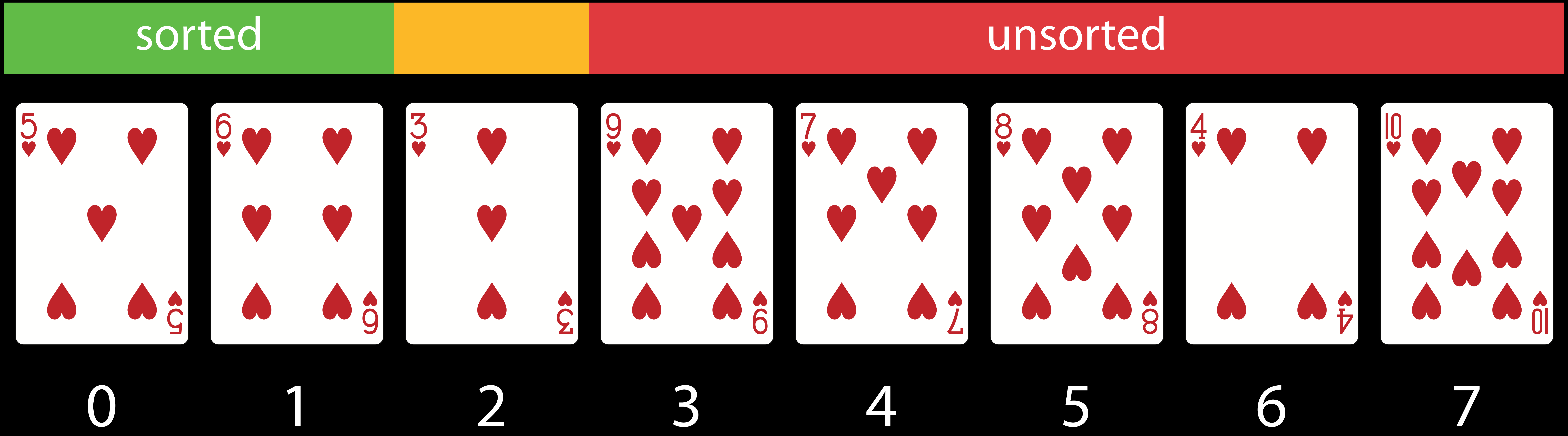
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



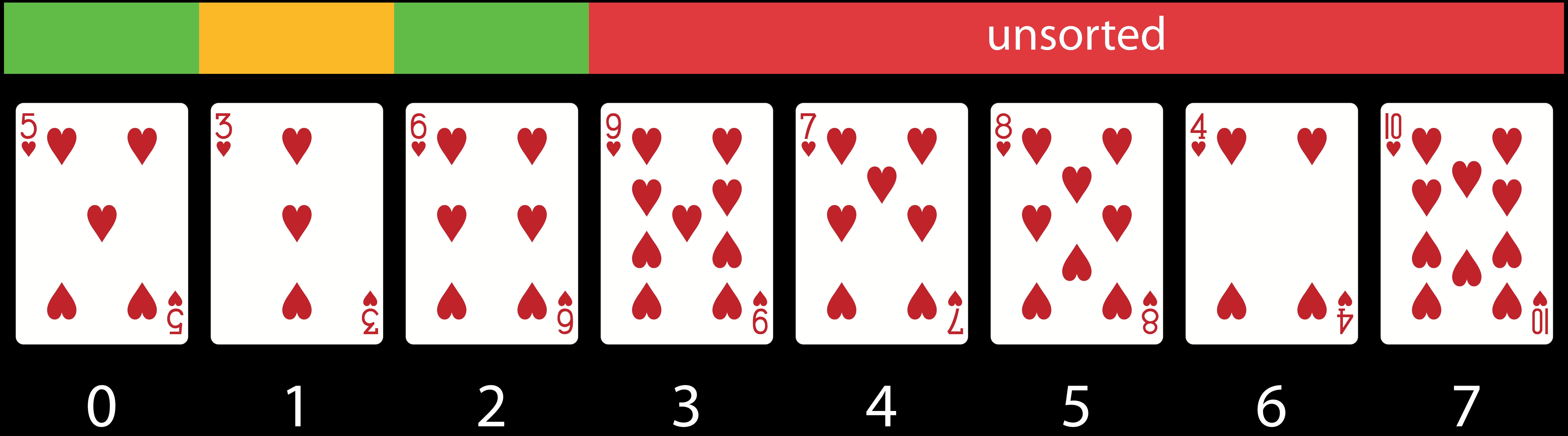
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



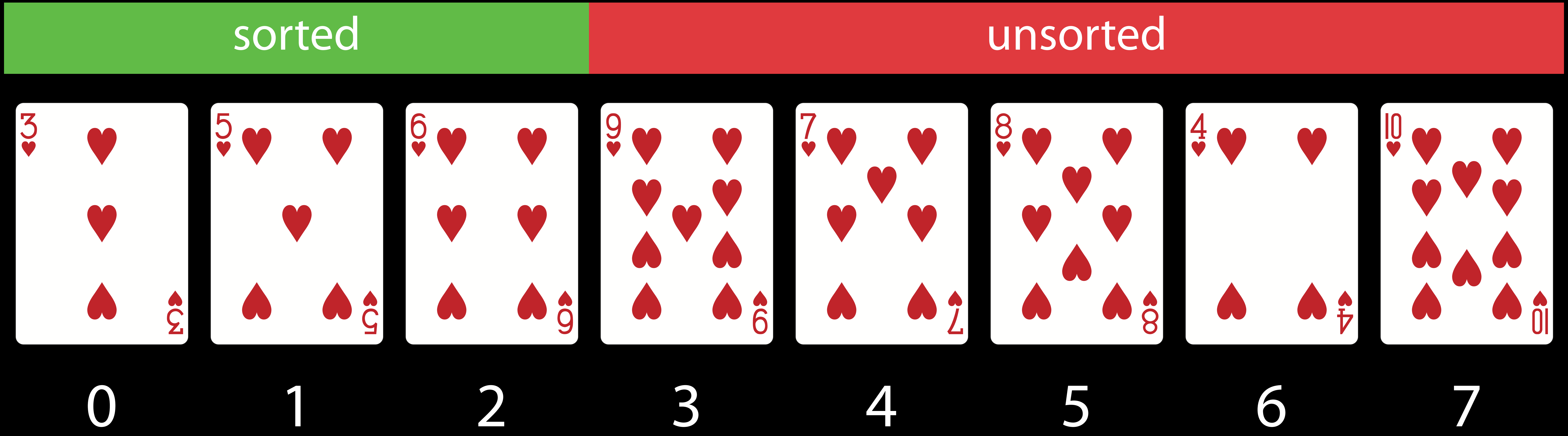
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



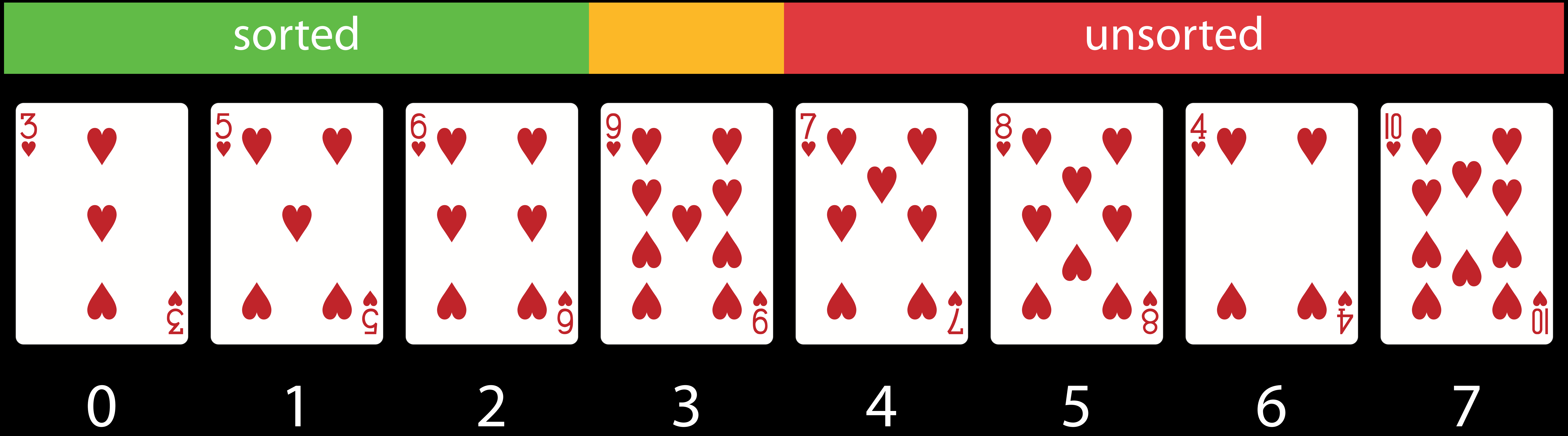
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



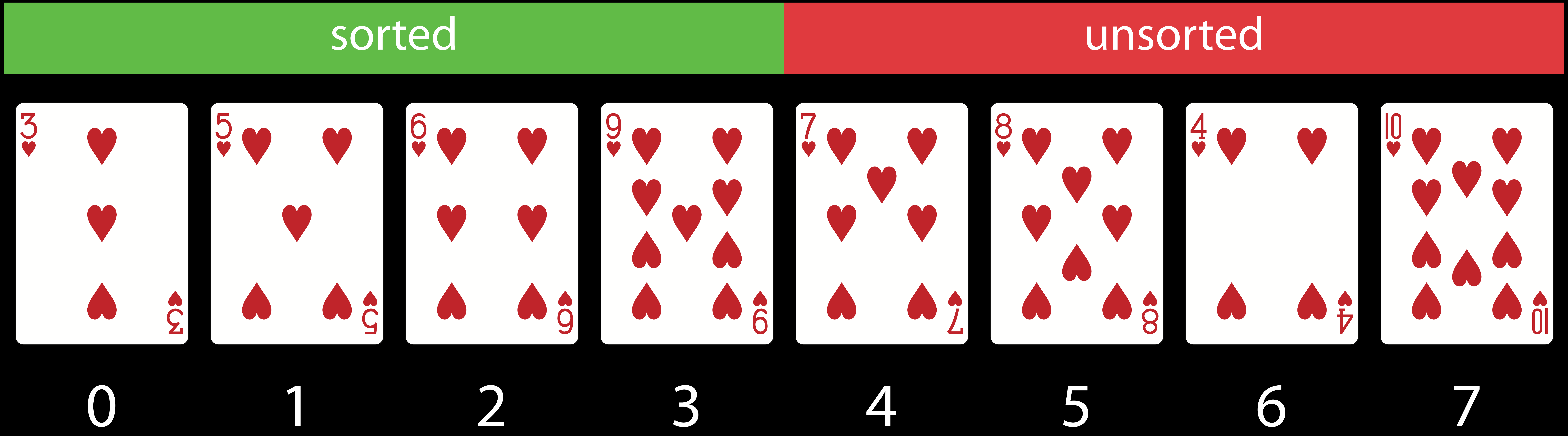
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



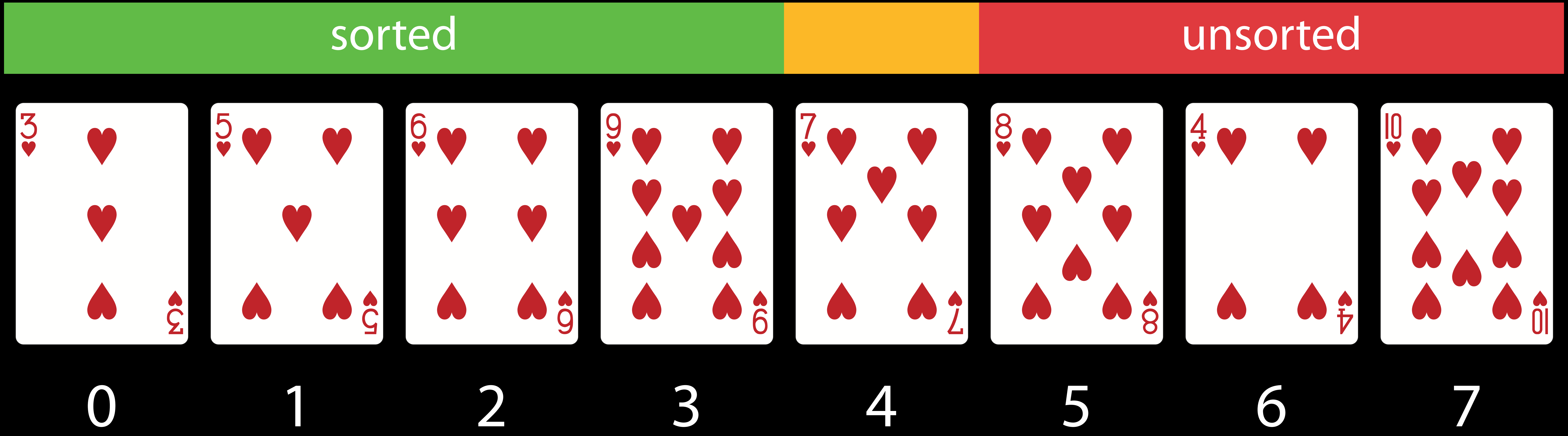
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



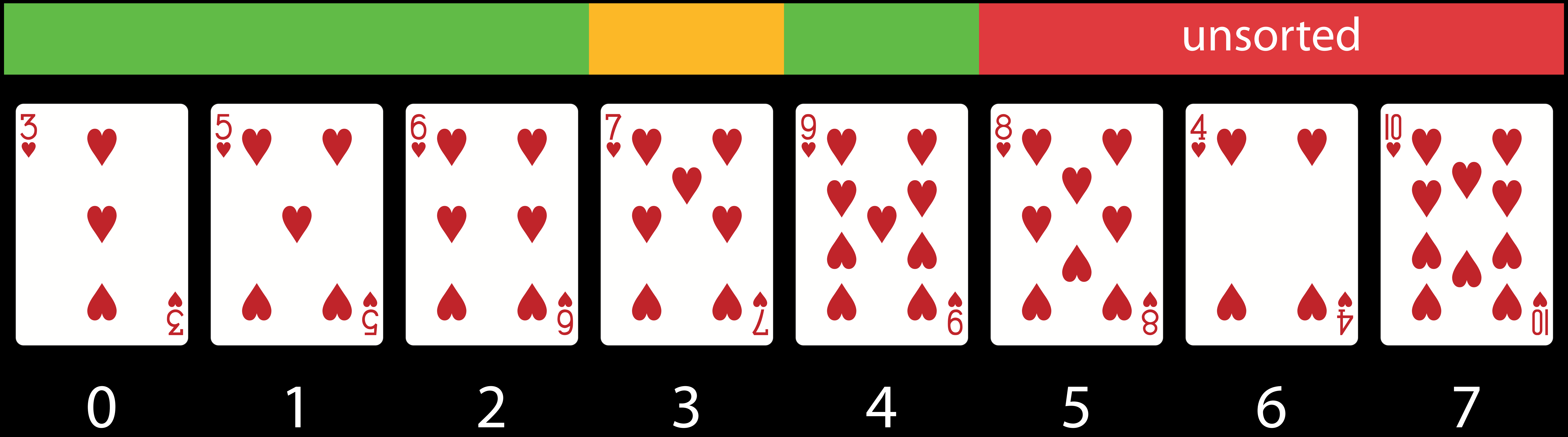
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



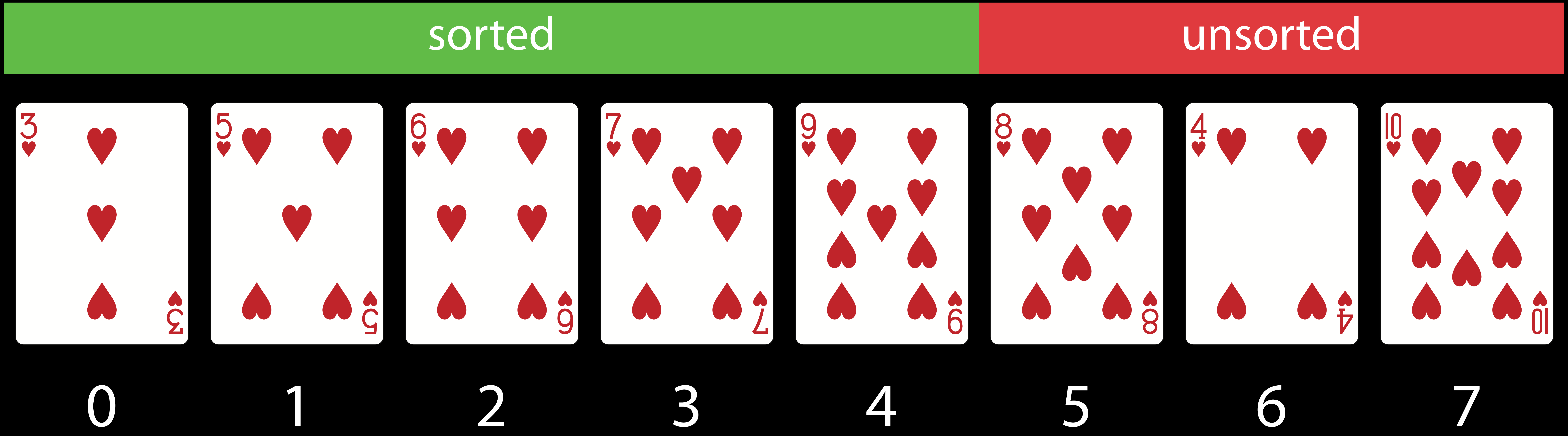
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



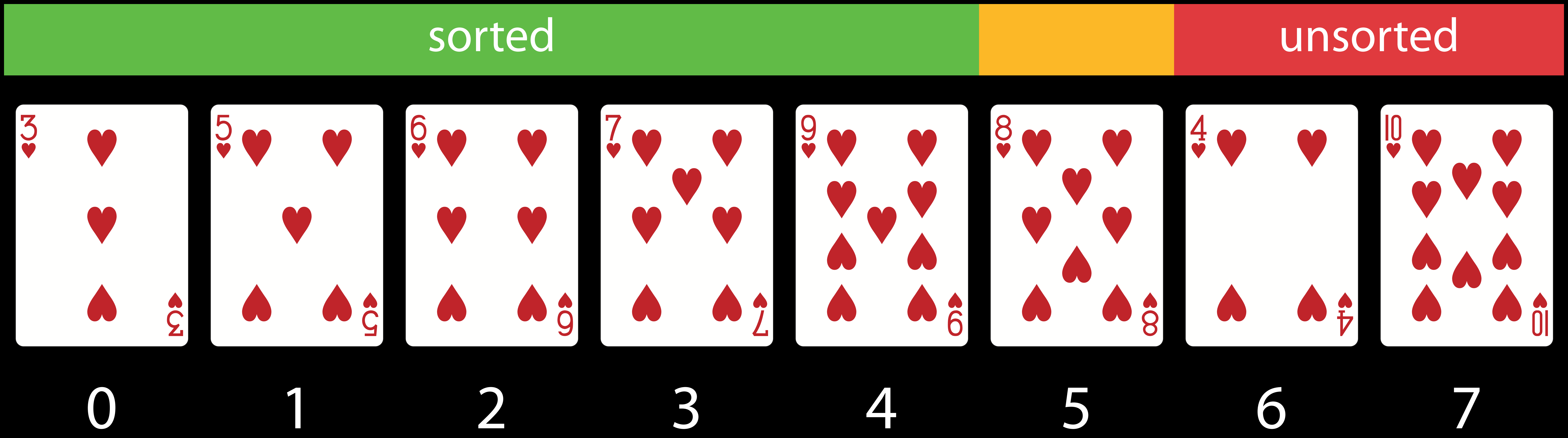
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



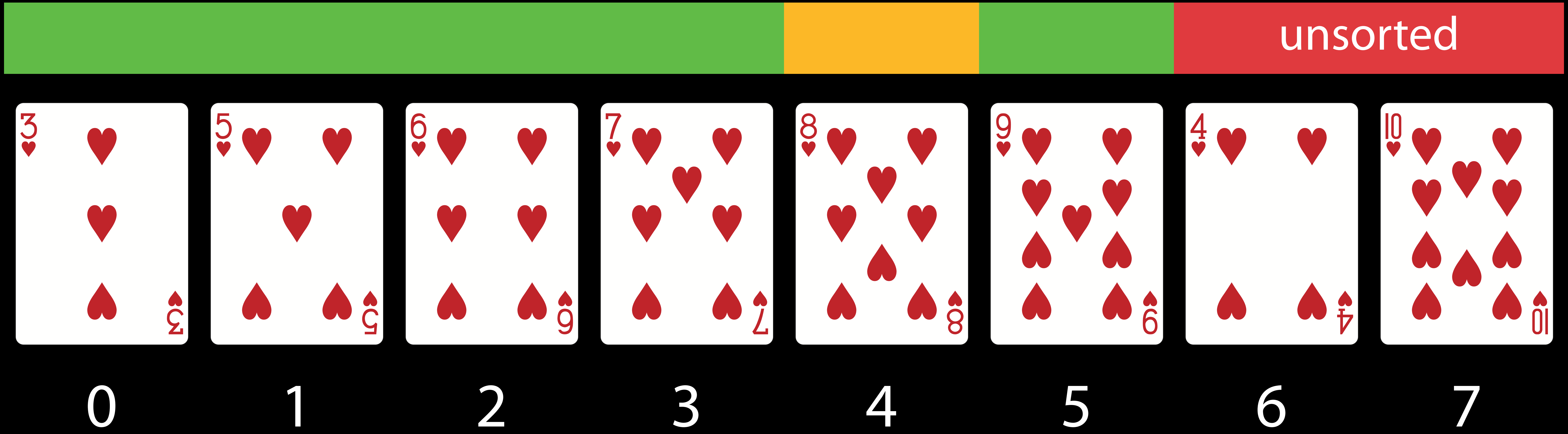
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



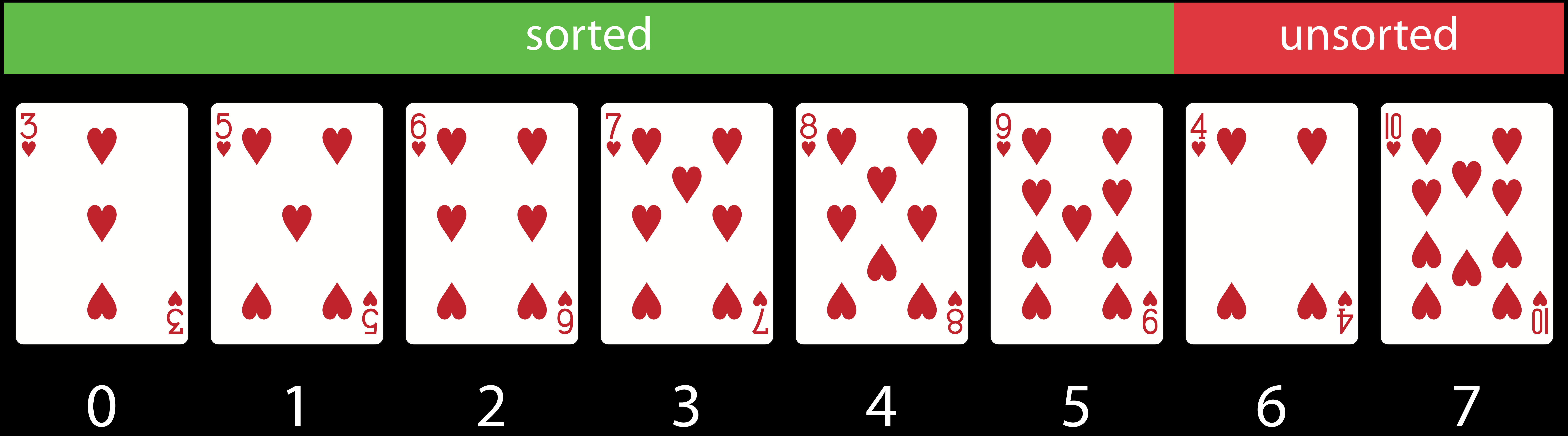
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



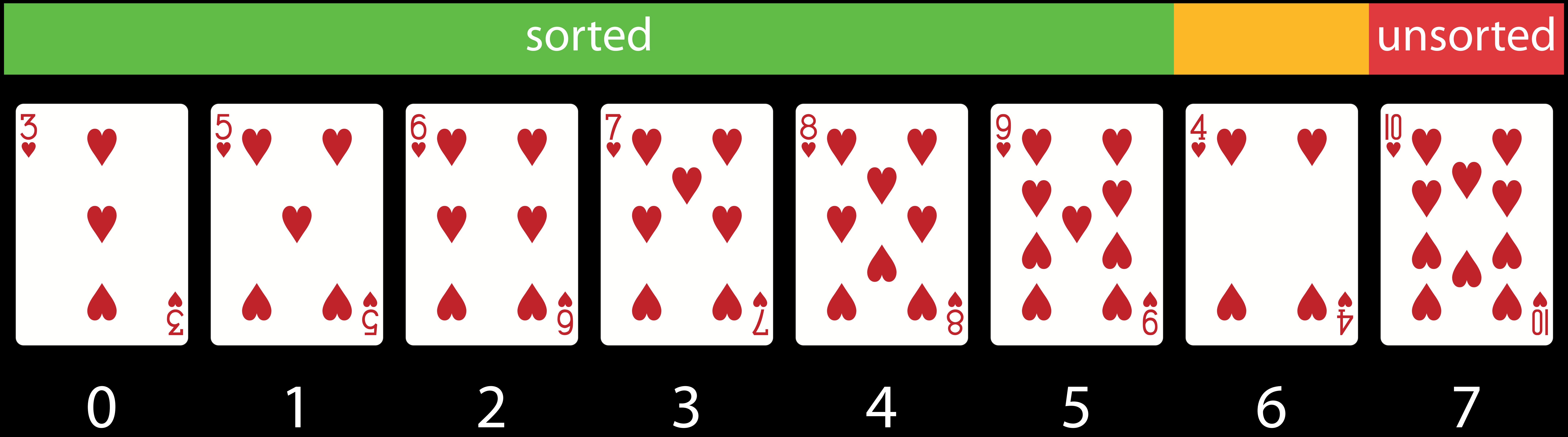
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



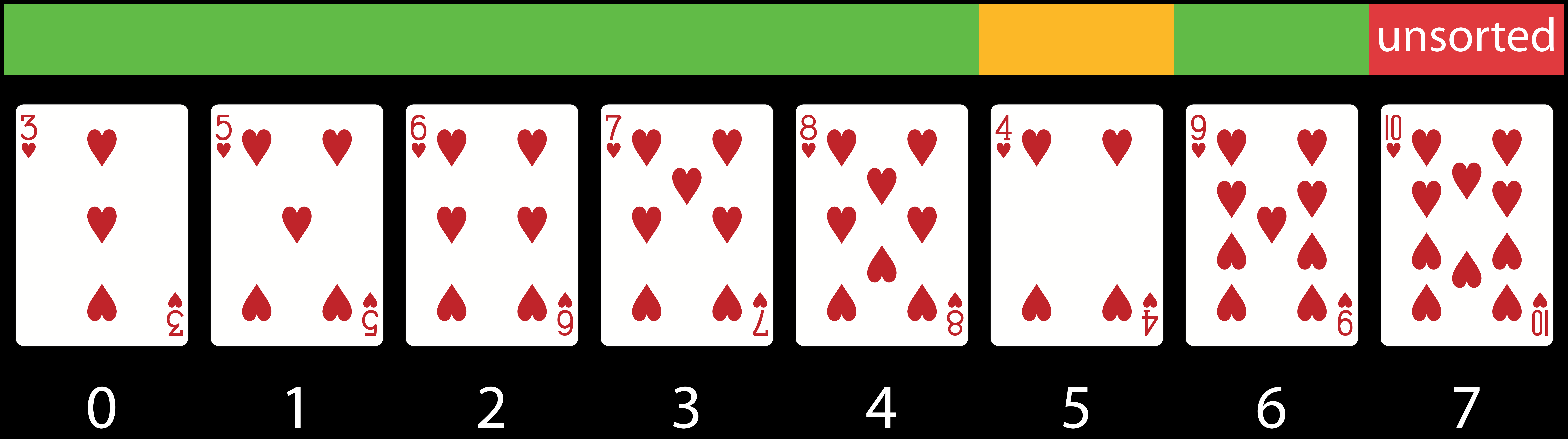
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



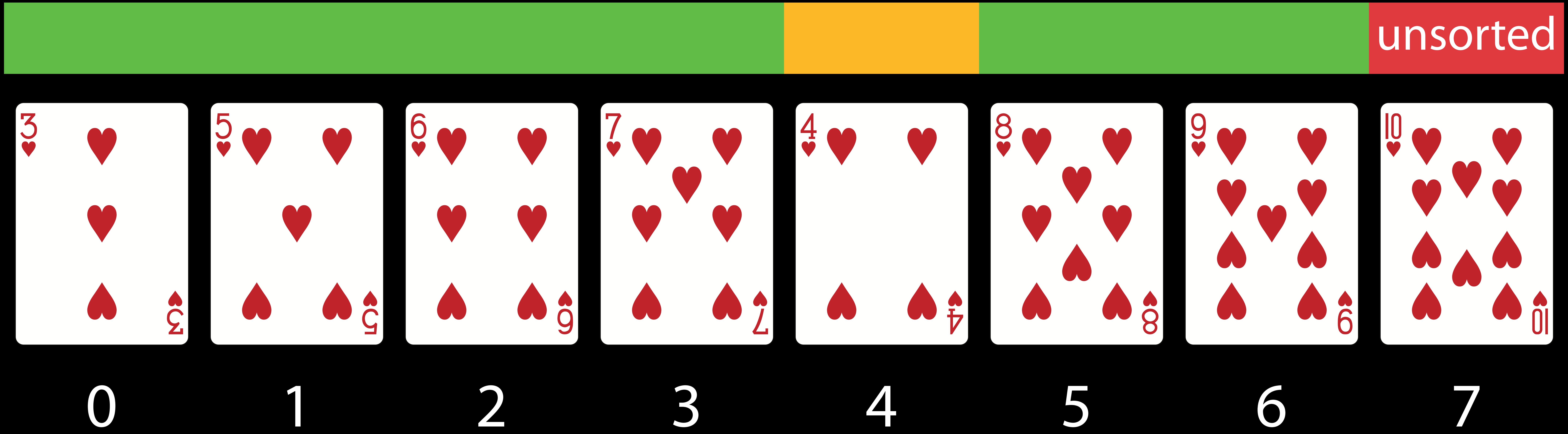
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



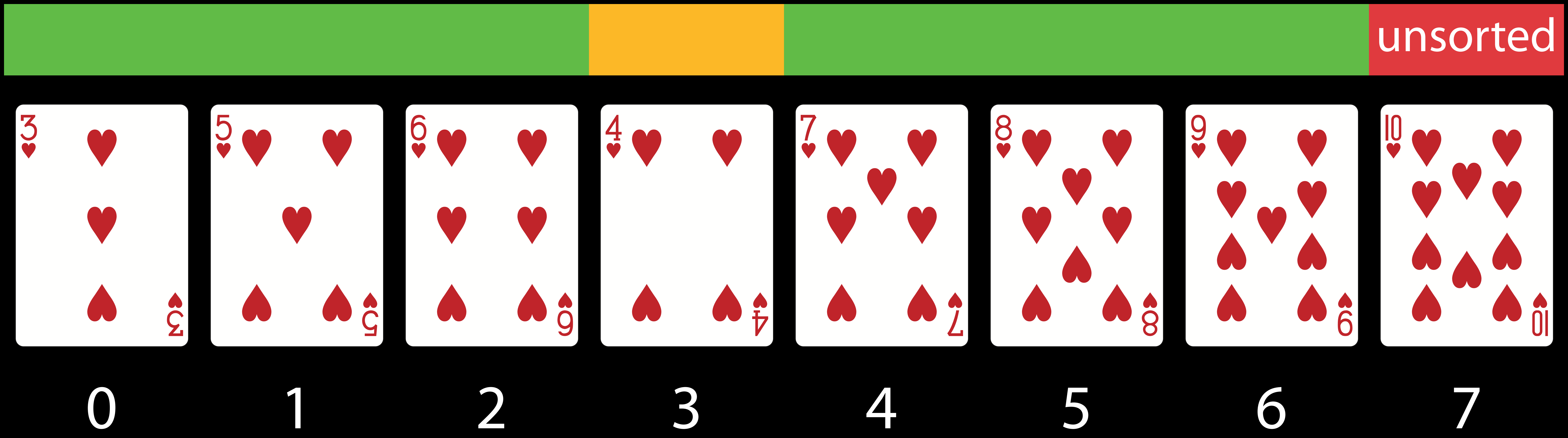
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



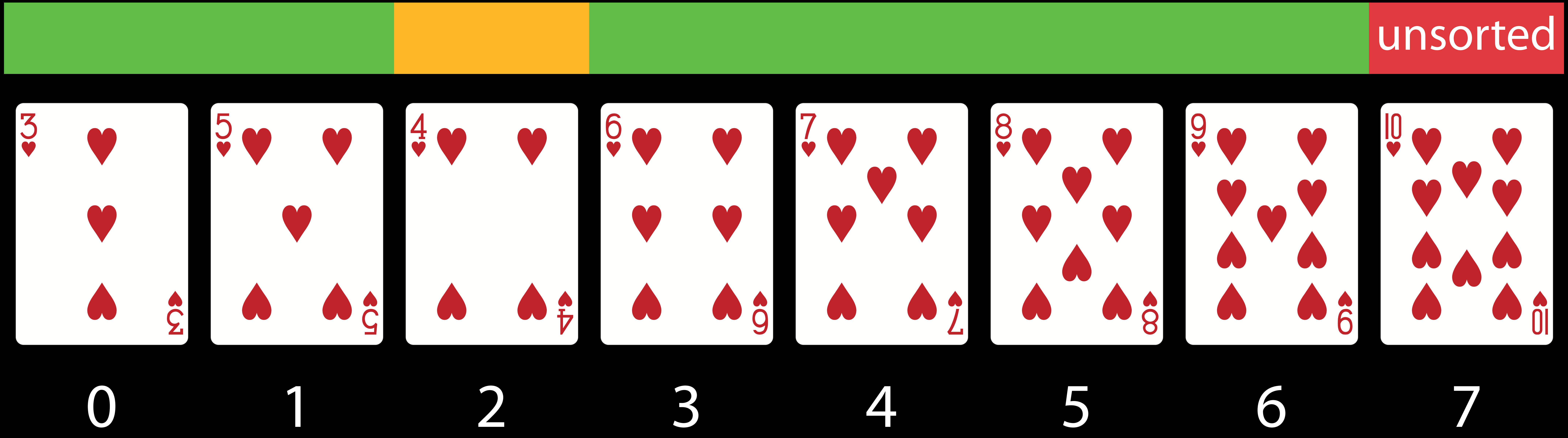
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



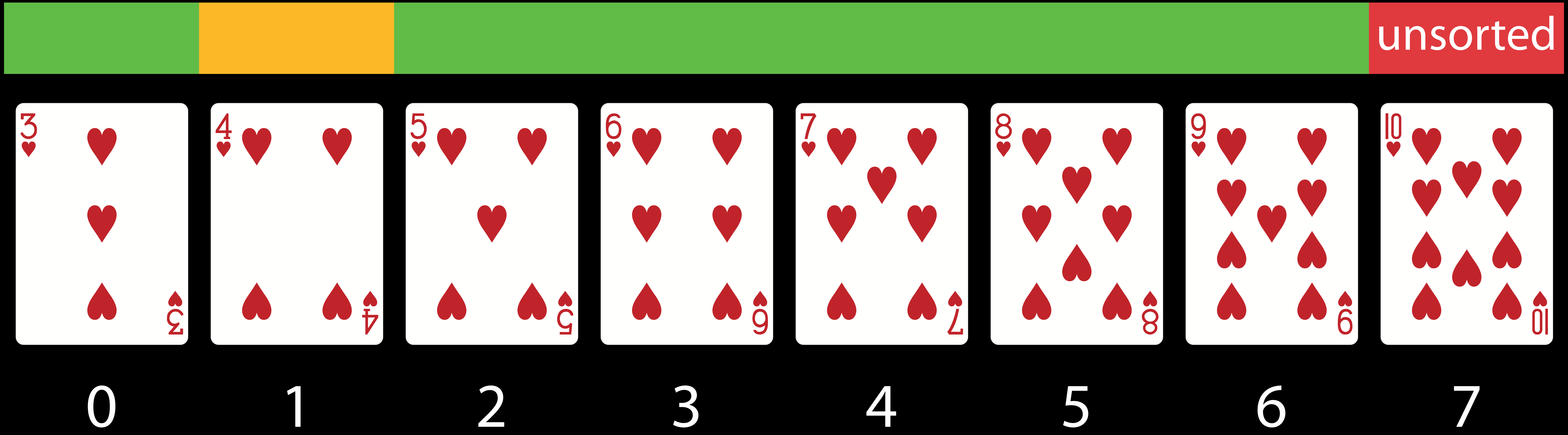
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



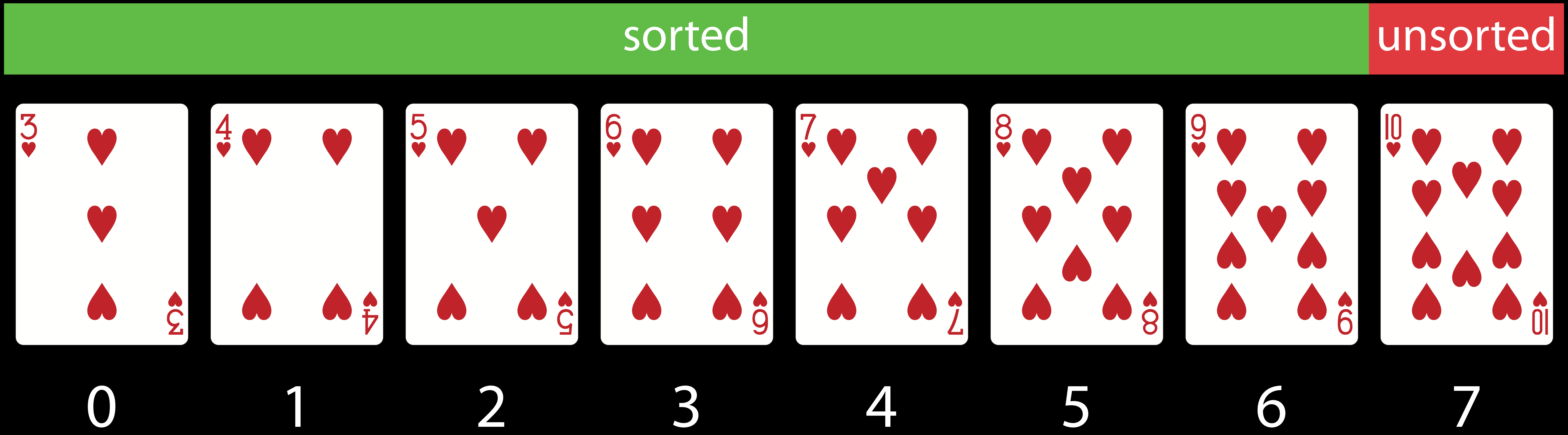
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



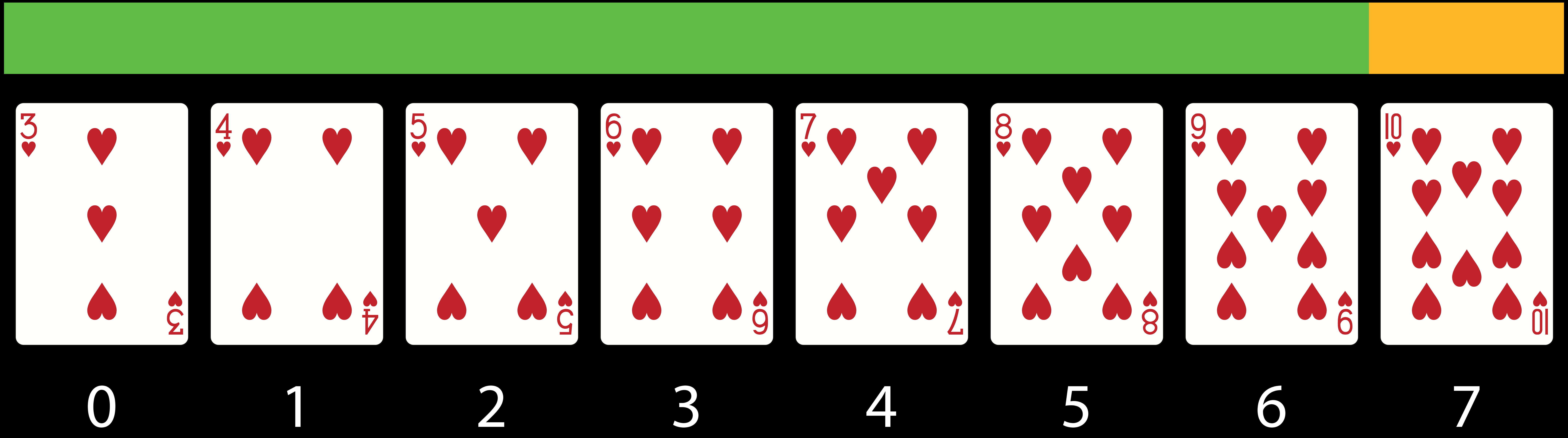
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



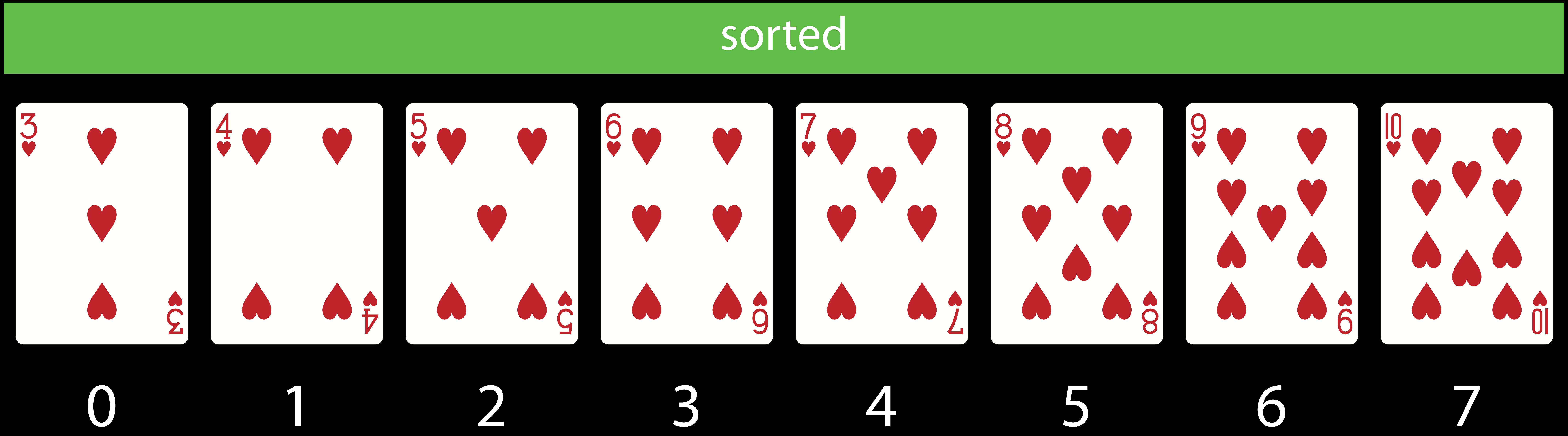
Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



Insertion Sort

Add each item from unsorted input, inserting into output at right position.
Do everything in place using swapping.



Heap sort

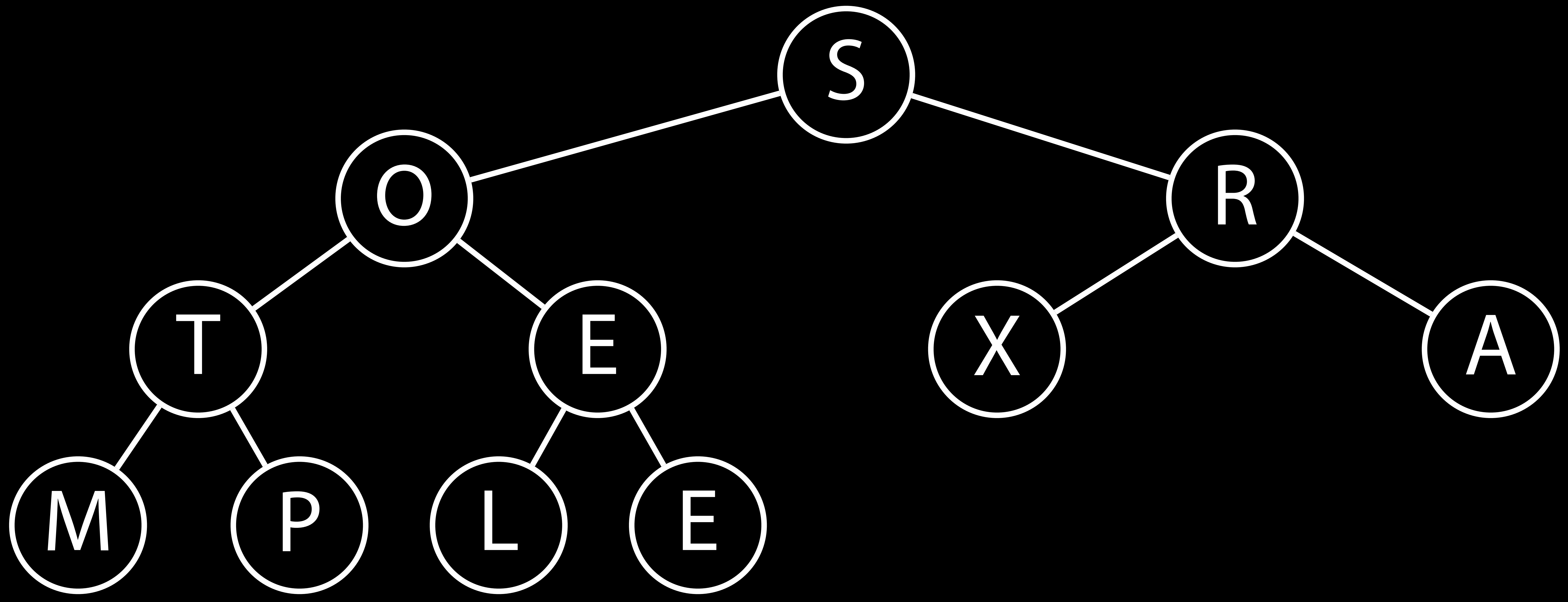
Build max-heap using bottom-up method.

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

Heap sort

Build max-heap using bottom-up method.

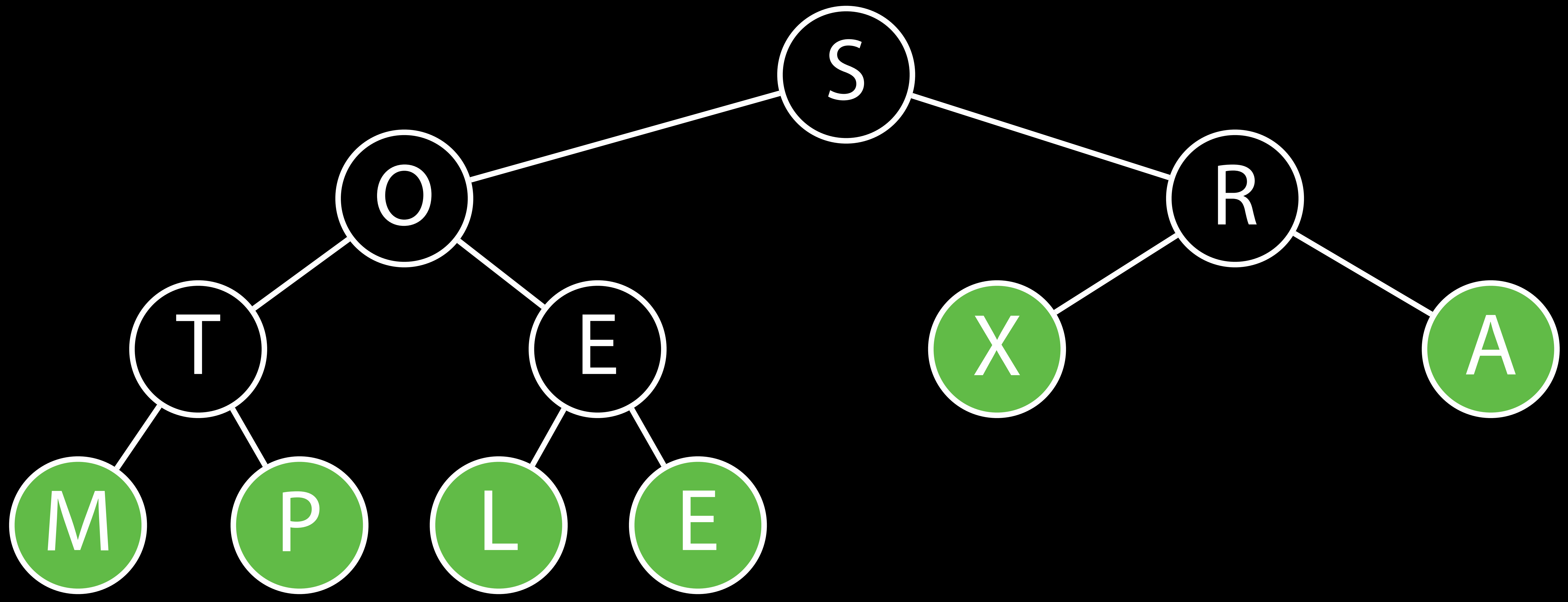
S	O	R	T	E	X	A	M	P	L	E
0	1	2	3	4	5	6	7	8	9	10



Heap sort

Build max-heap using bottom-up method.

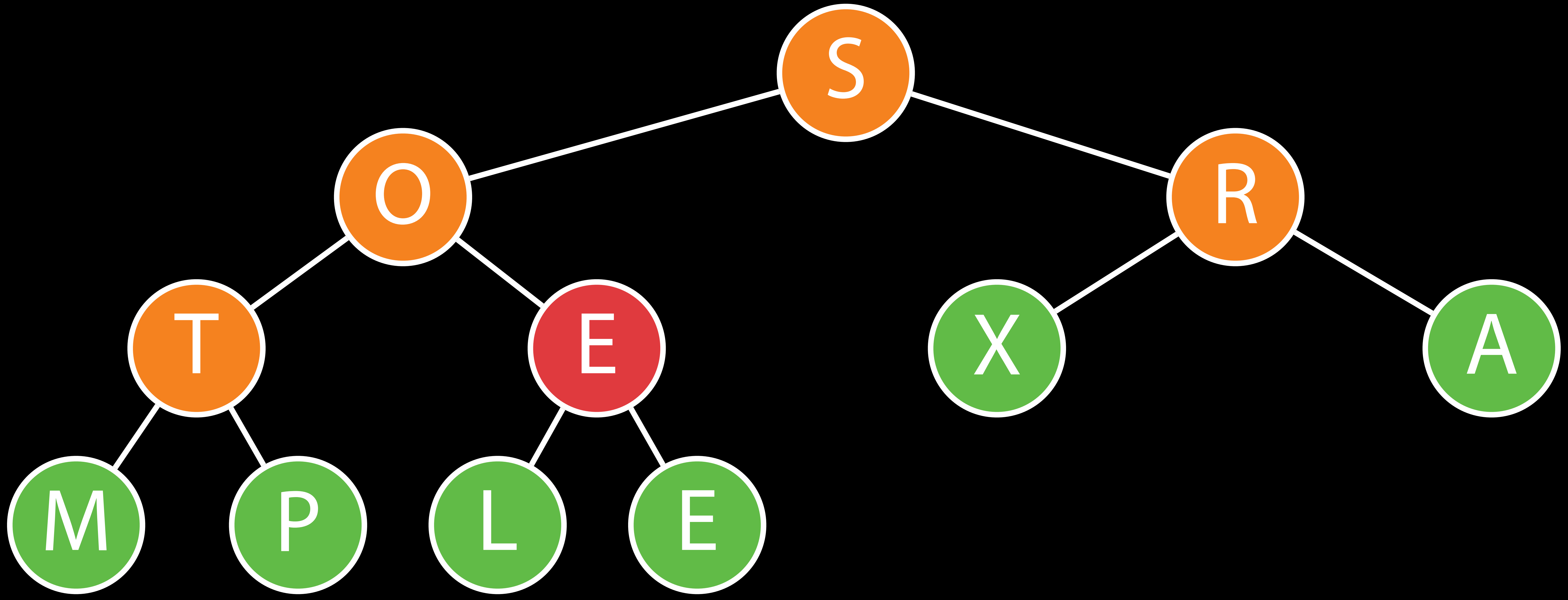
S	O	R	T	E	X	A	M	P	L	E
0	1	2	3	4	5	6	7	8	9	10



Heap sort

Build max-heap using bottom-up method.

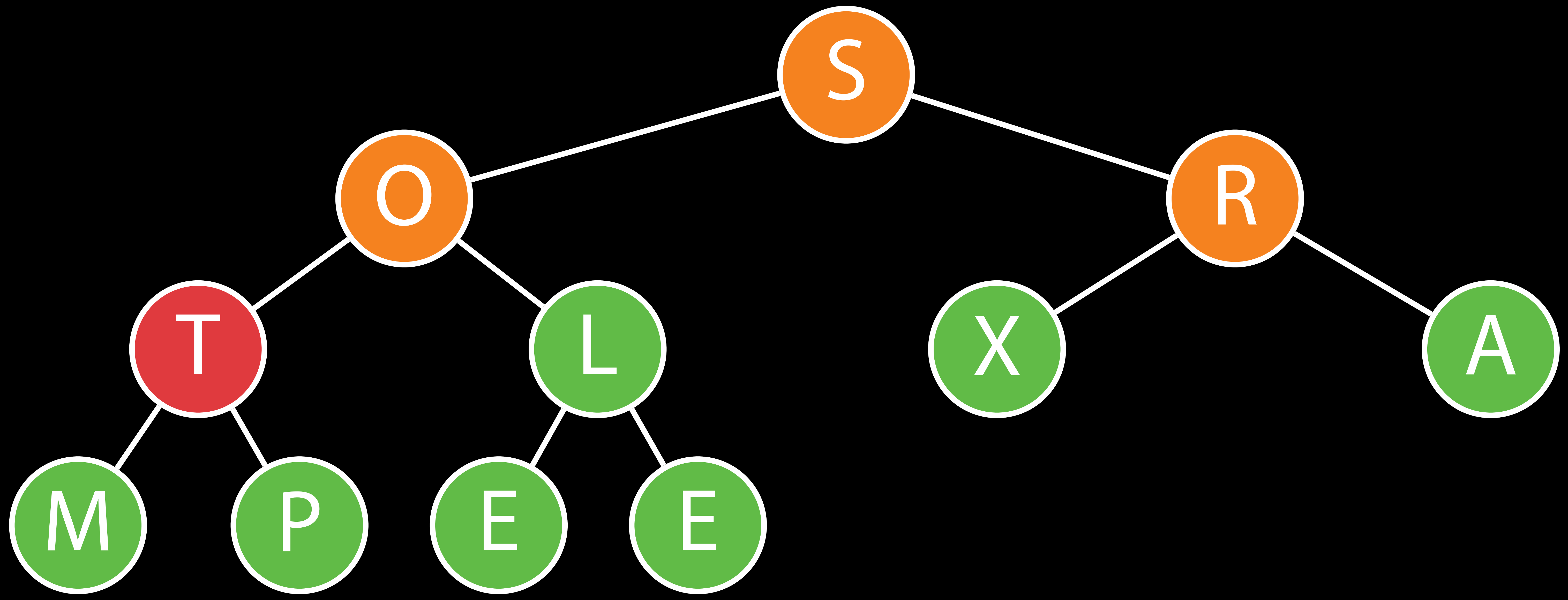
S	O	R	T	E	X	A	M	P	L	E
0	1	2	3	4	5	6	7	8	9	10



Heap sort

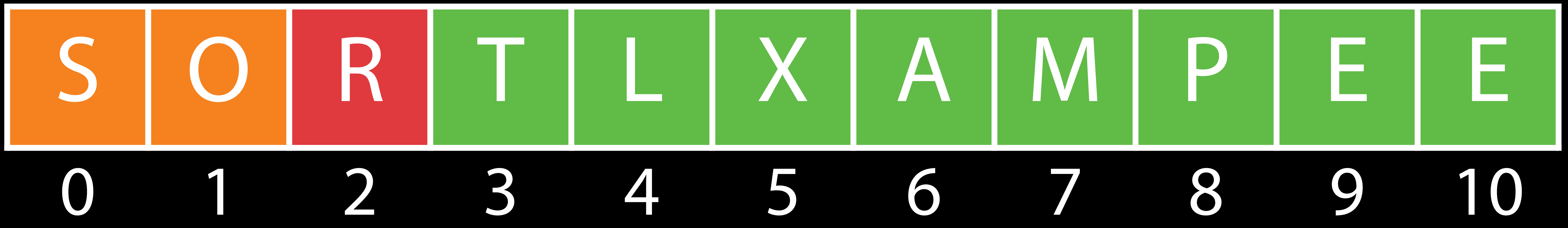
Build max-heap using bottom-up method.

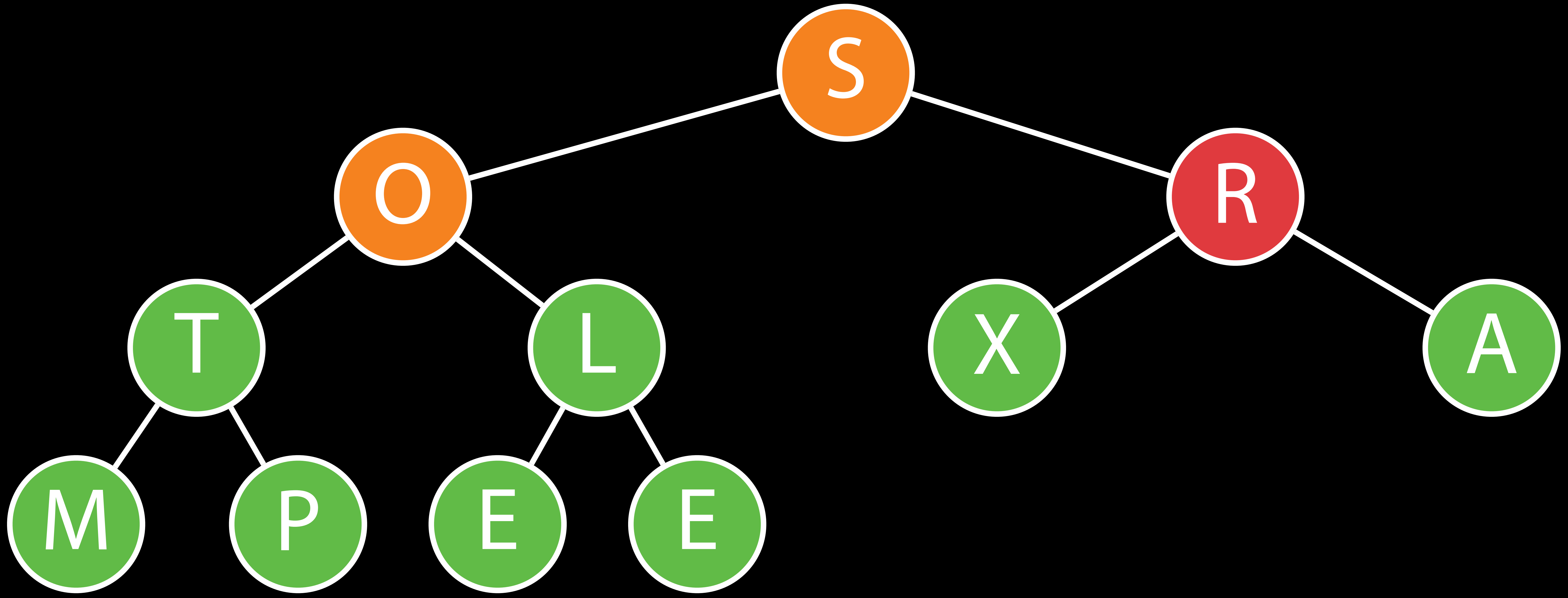
S	O	R	T	L	X	A	M	P	E	E
0	1	2	3	4	5	6	7	8	9	10



Heap sort

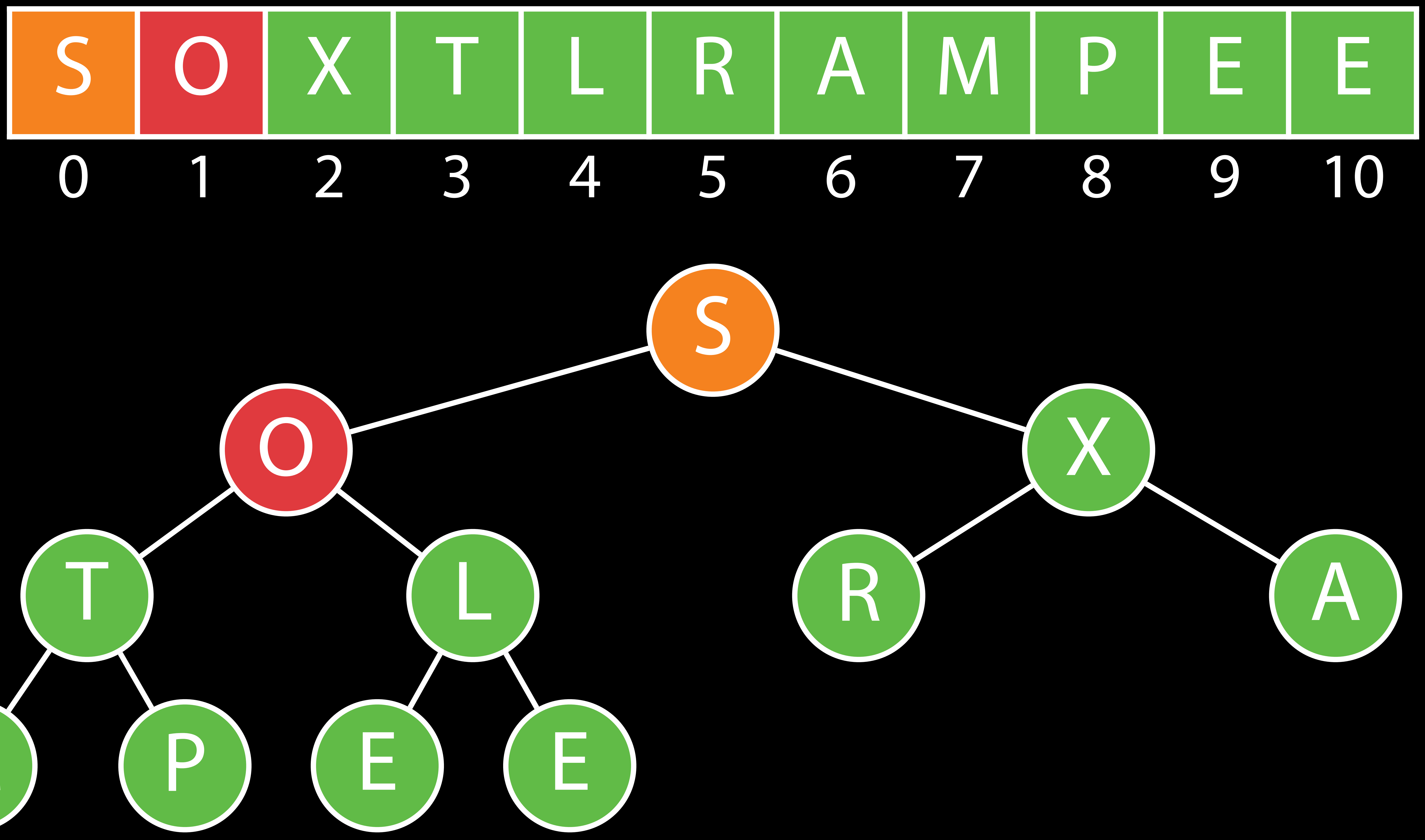
Build max-heap using bottom-up method.





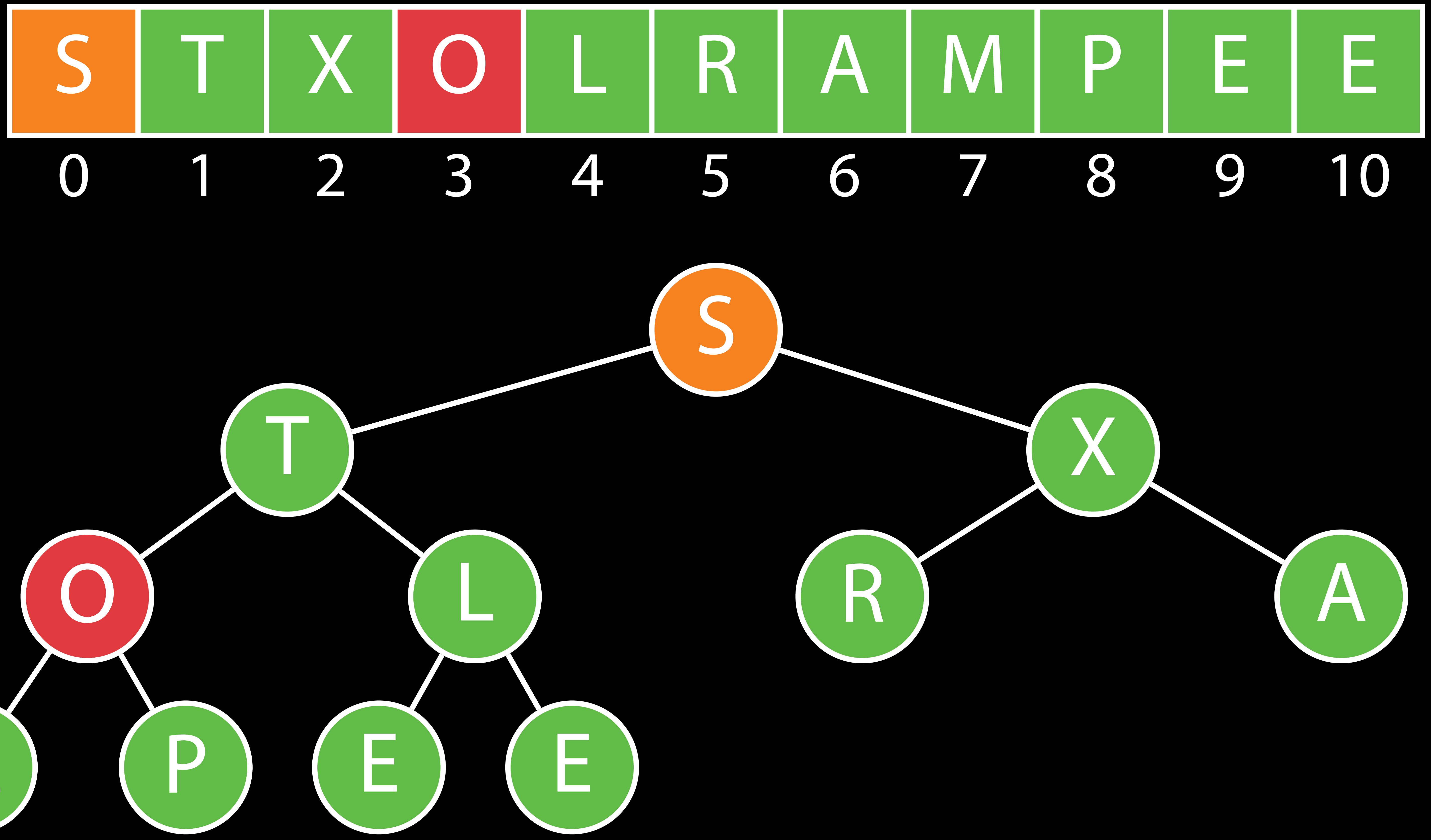
Heap sort

Build max-heap using bottom-up method.



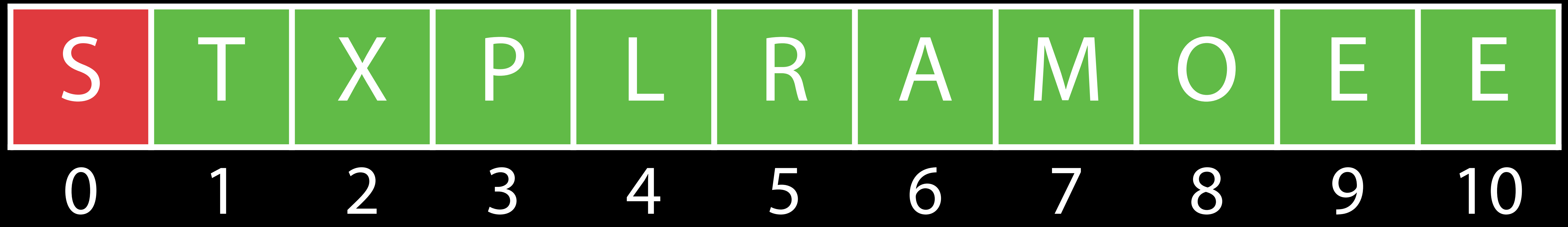
Heap sort

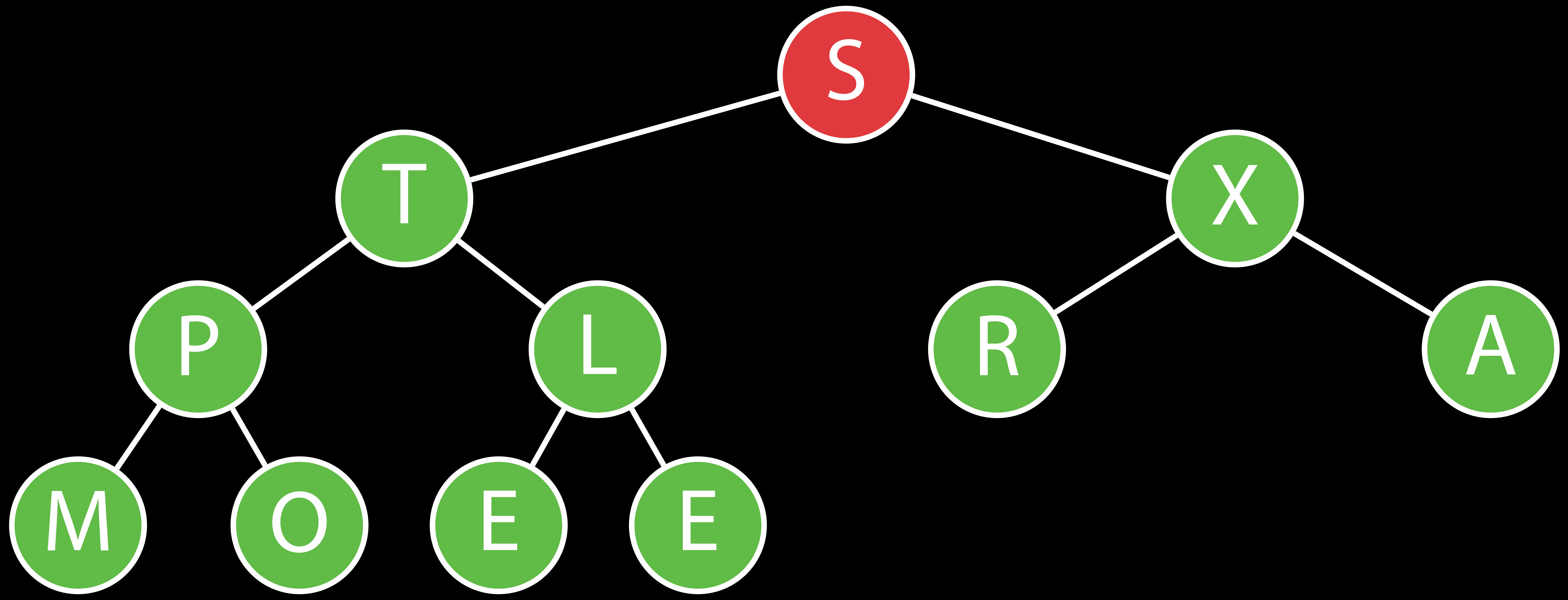
Build max-heap using bottom-up method.



Heap sort

Build max-heap using bottom-up method.

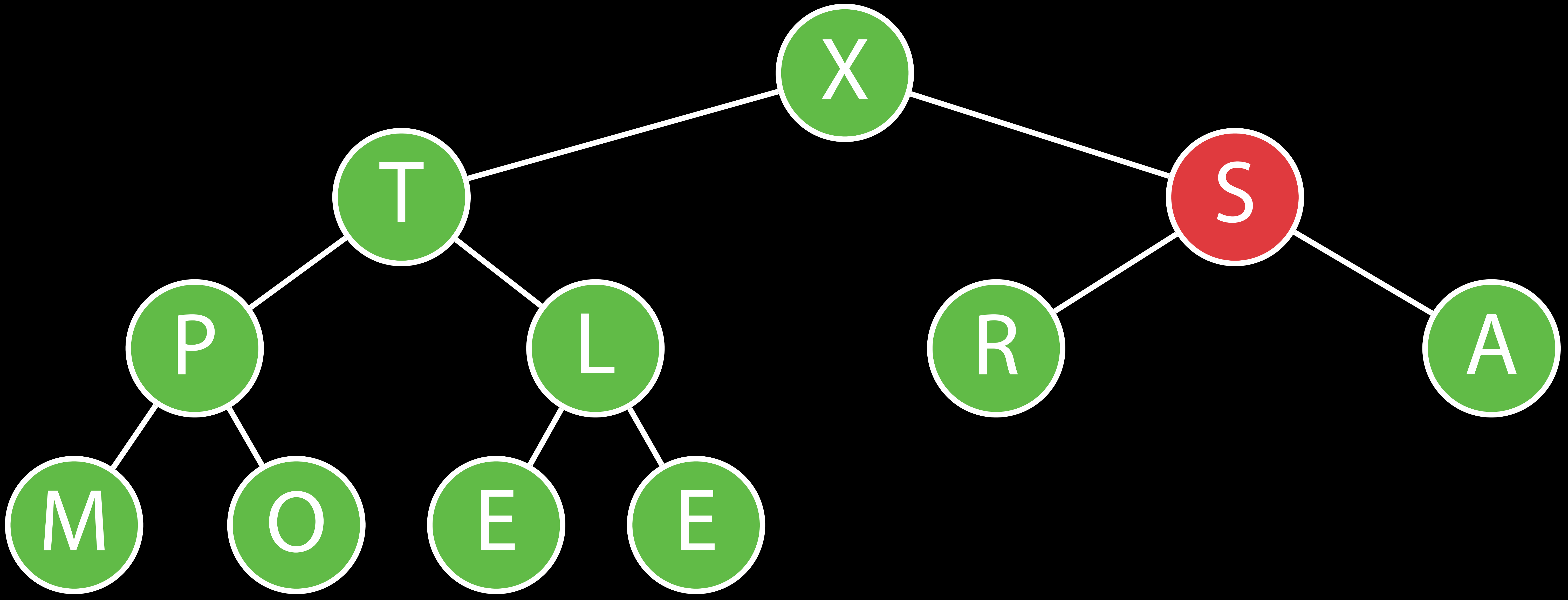




Heap sort

Build max-heap using bottom-up method.

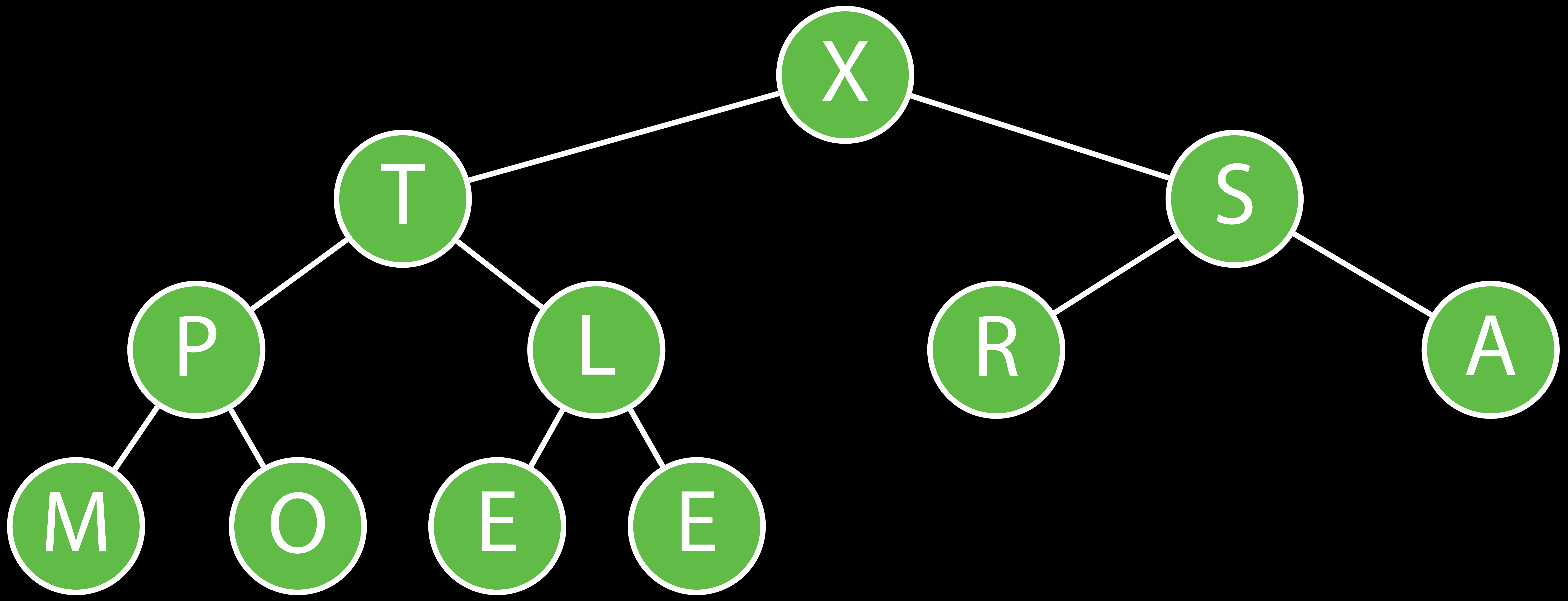
0	1	2	3	4	5	6	7	8	9	10
X	T	S	P	L	R	A	M	O	E	E



Heap sort

Build max-heap using bottom-up method.

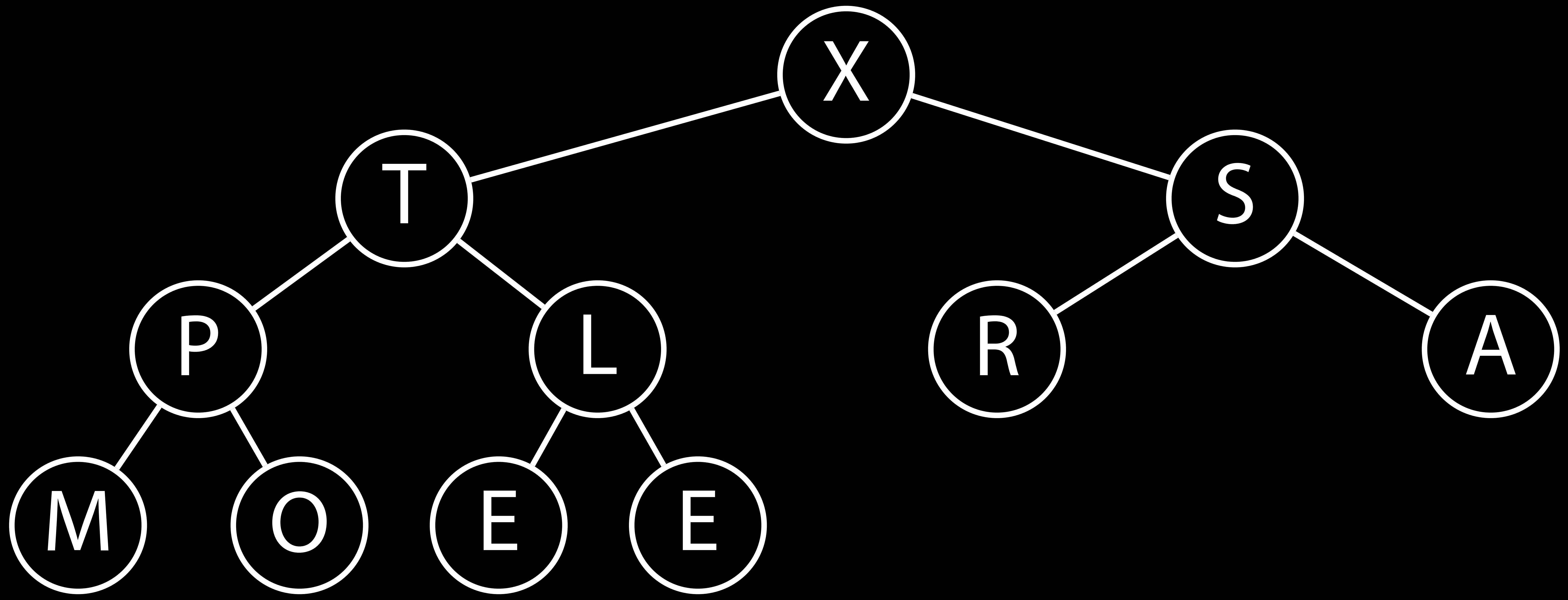
X	T	S	P	L	R	A	M	O	E	E
0	1	2	3	4	5	6	7	8	9	10



Heap sort

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

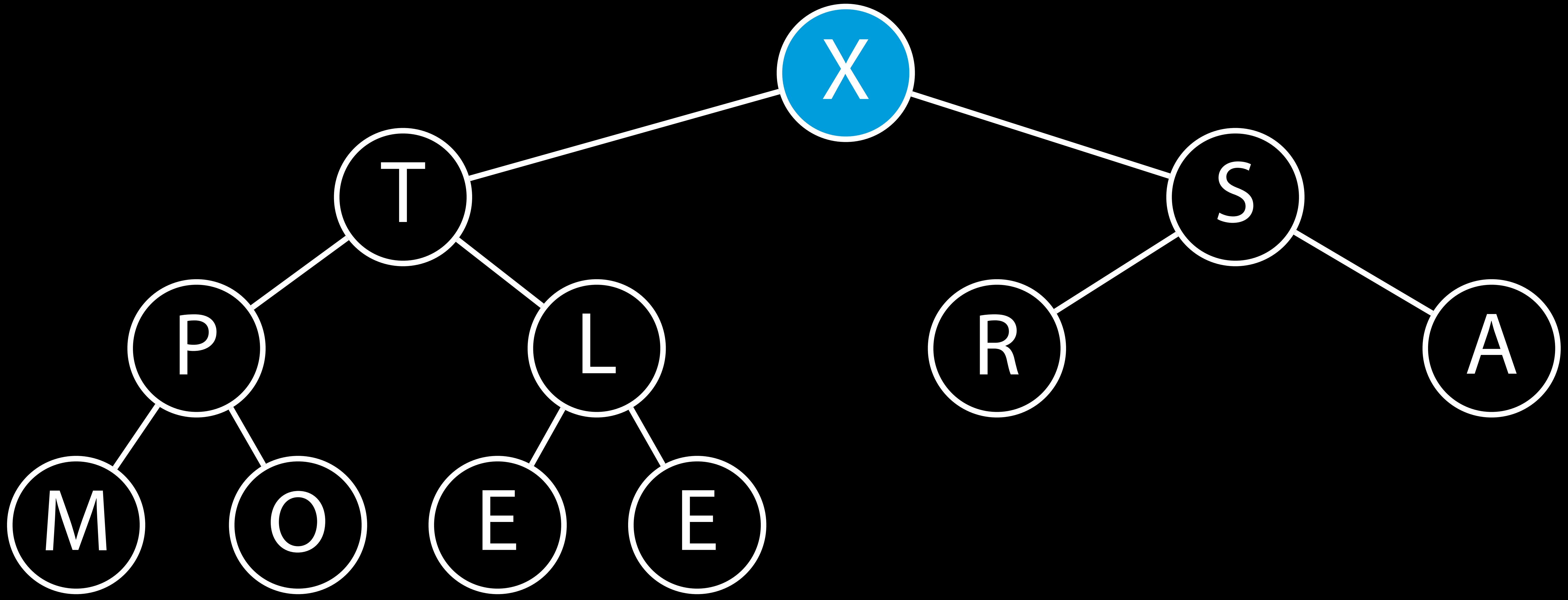
X	T	S	P	L	R	A	M	O	E	E
0	1	2	3	4	5	6	7	8	9	10



Heap sort

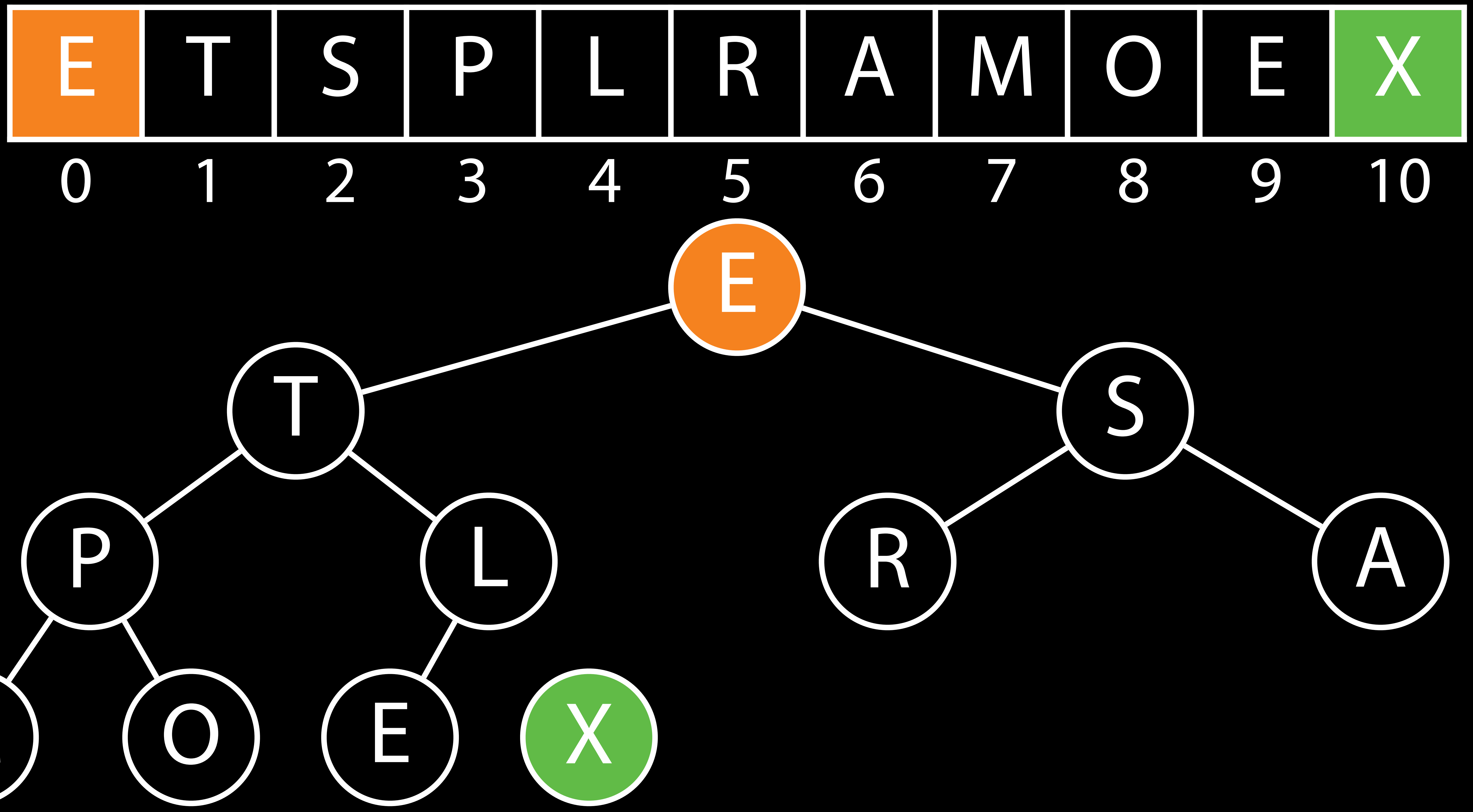
Repeatedly delete the largest remaining item by swapping with the last item.

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



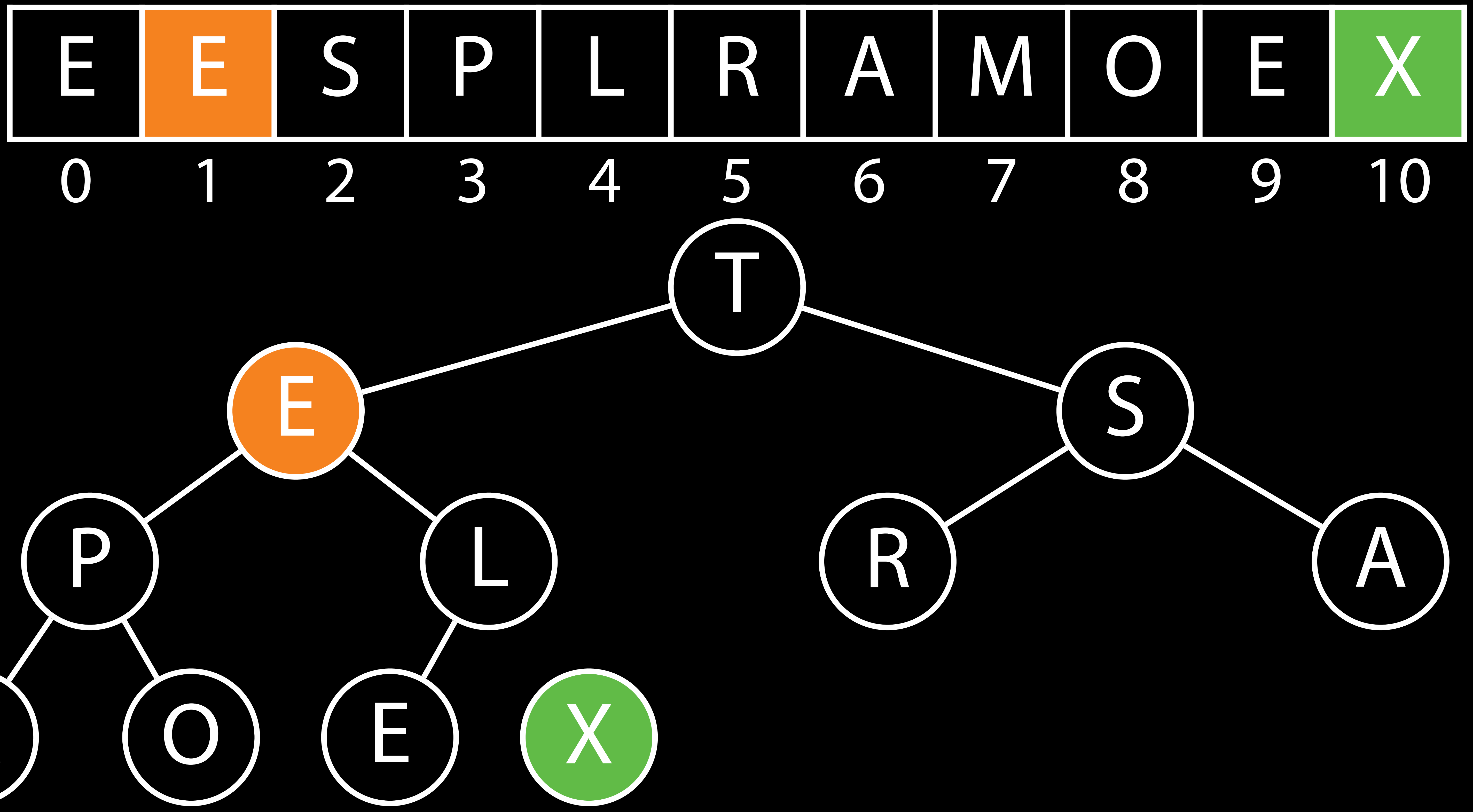
Heap sort

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



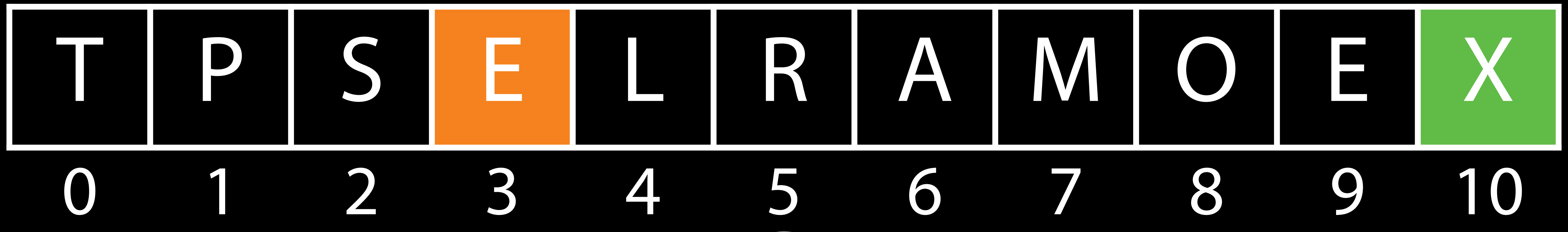
Heap sort

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

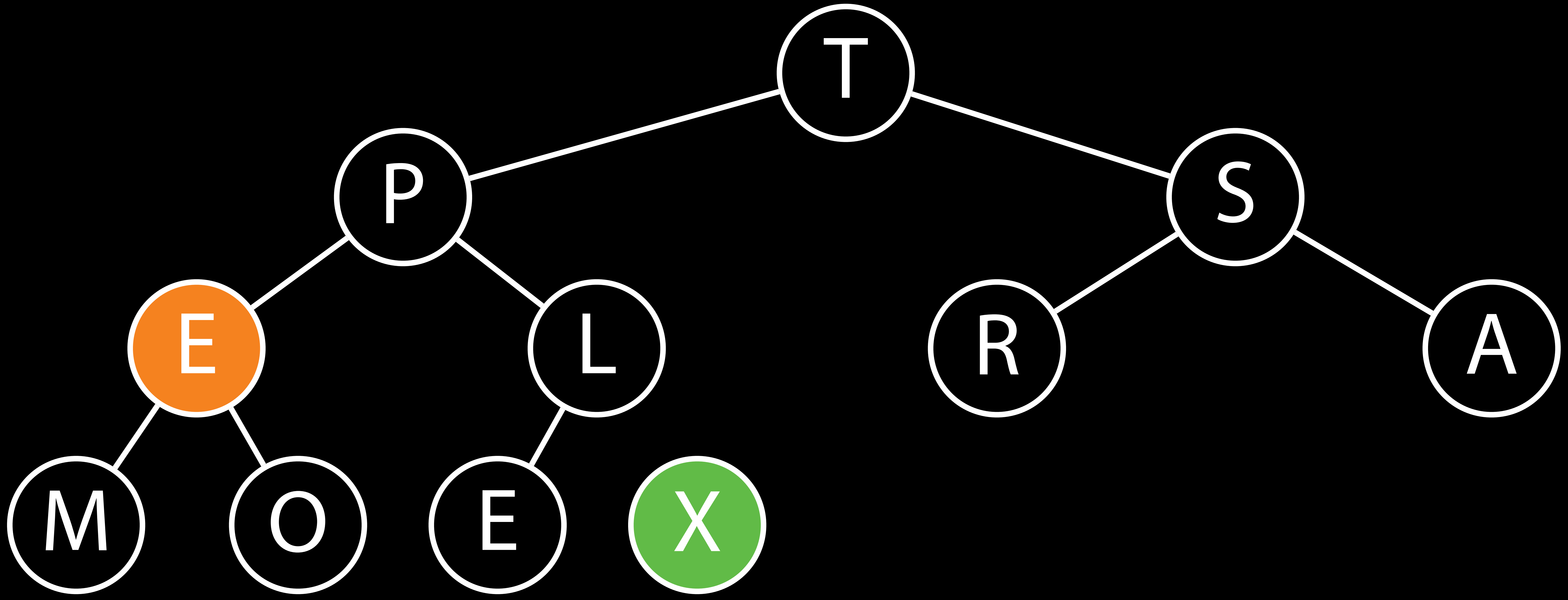


Heap sort

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

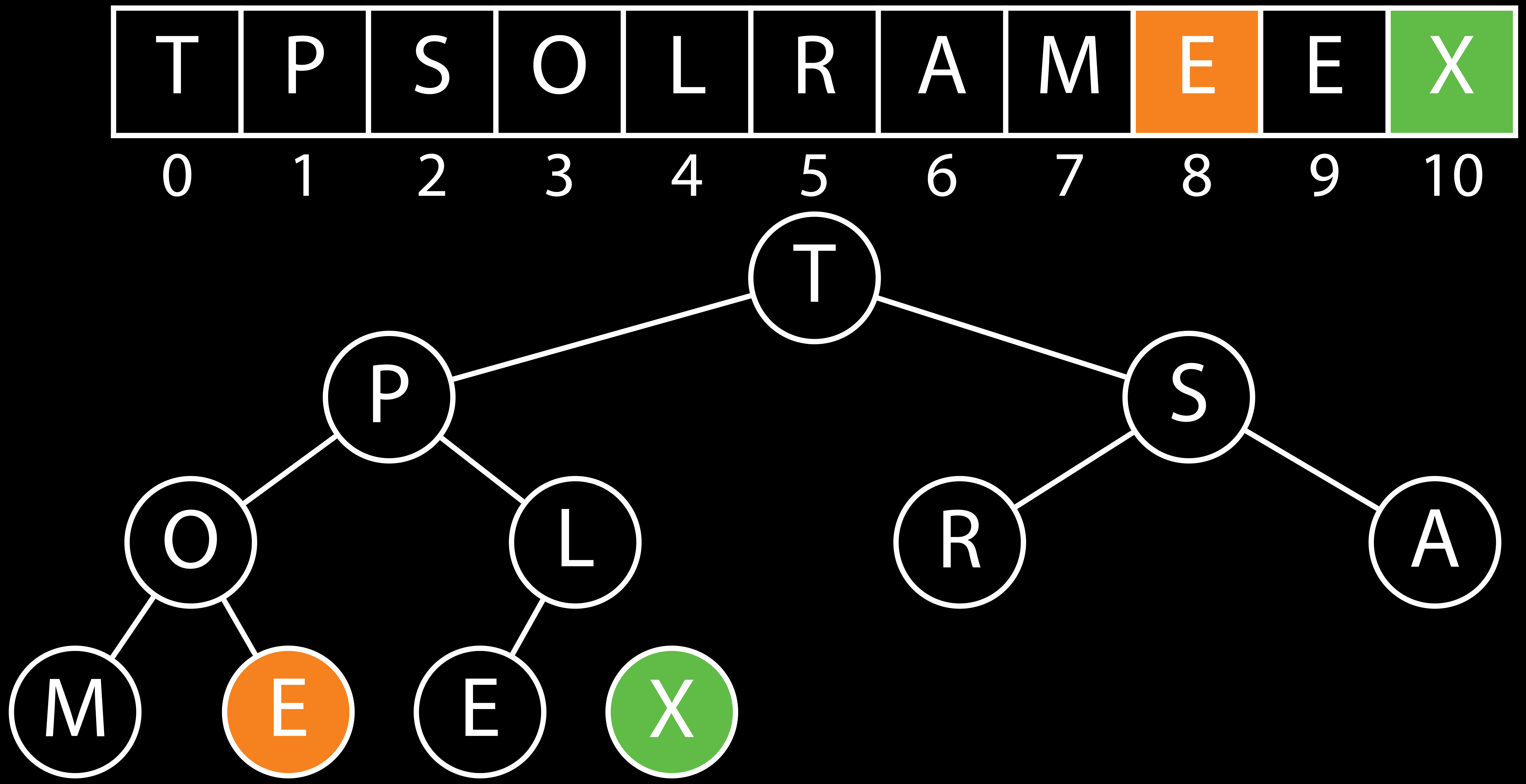


0	1	2	3	4	5	6	7	8	9	10
T	P	S	E	L	R	A	M	O	E	X



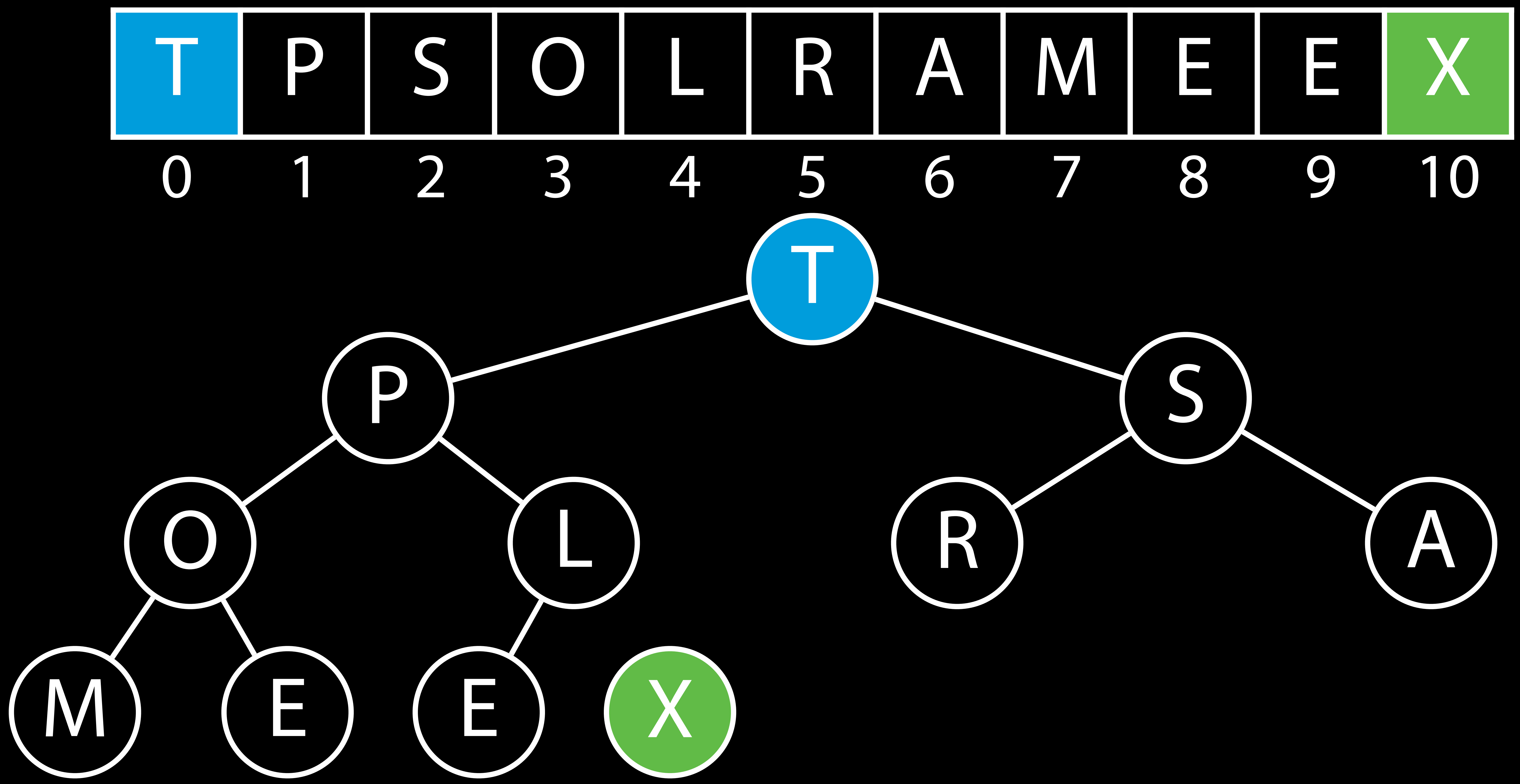
Heap sort

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



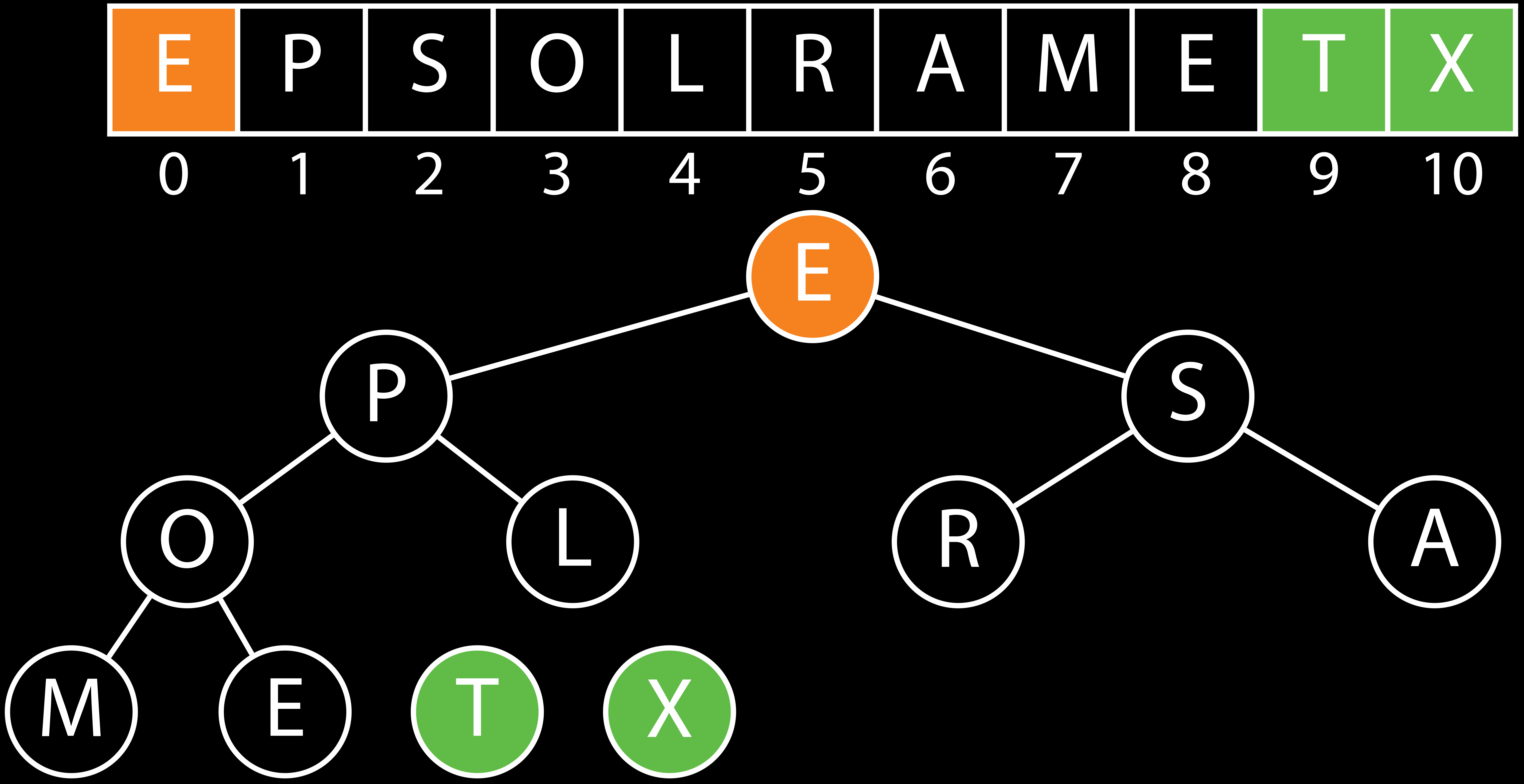
Heap sort

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



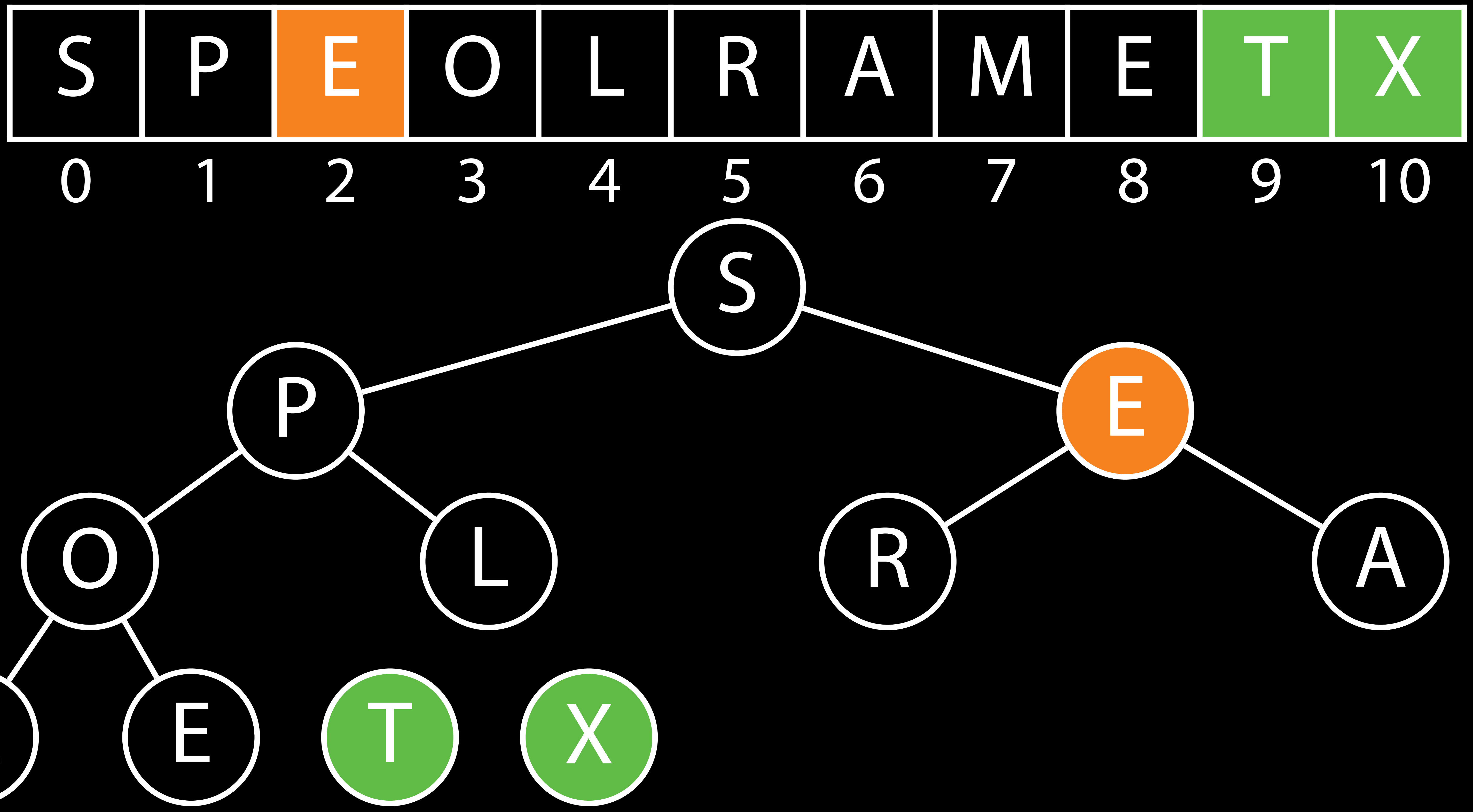
Heap sort

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



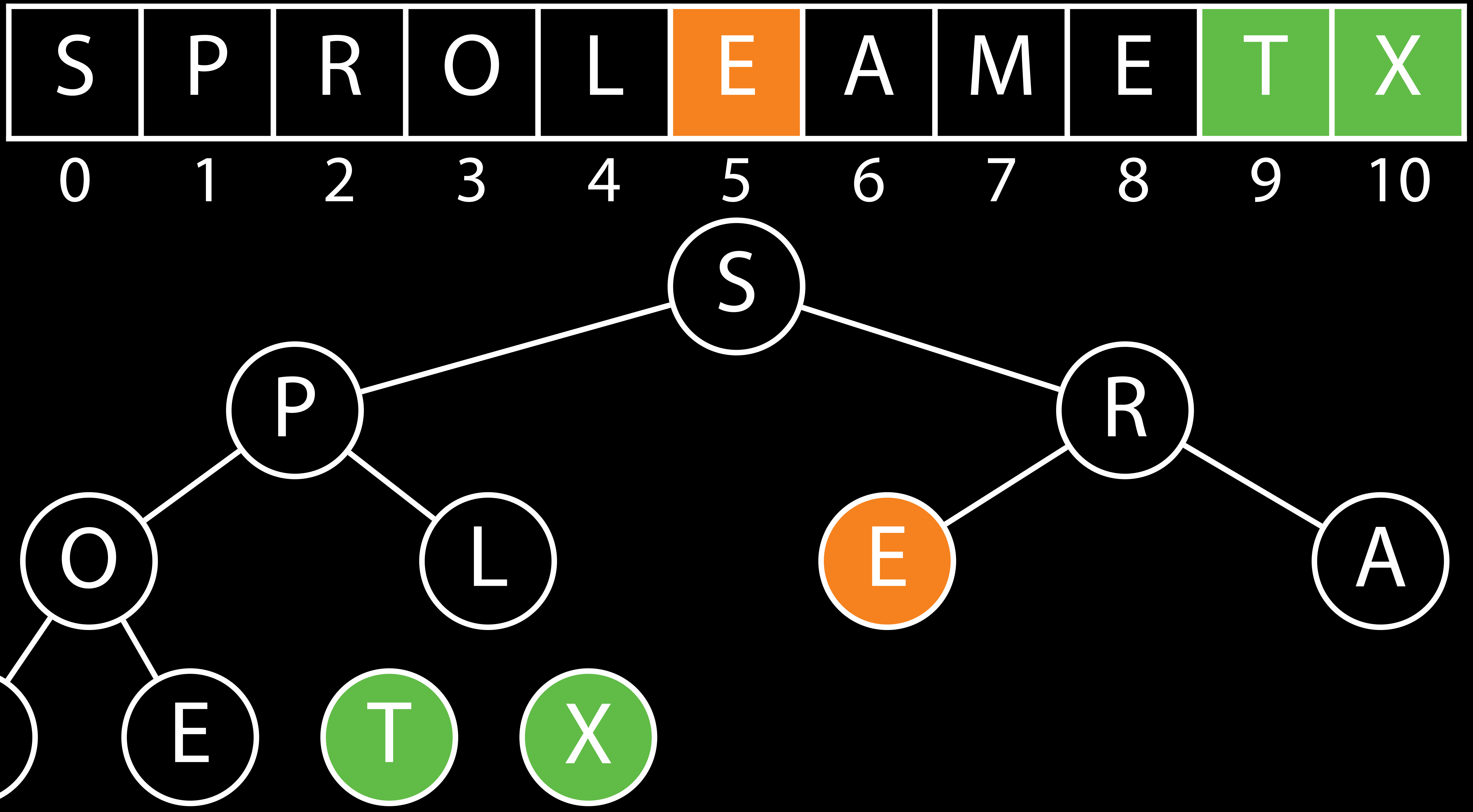
Heap sort

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



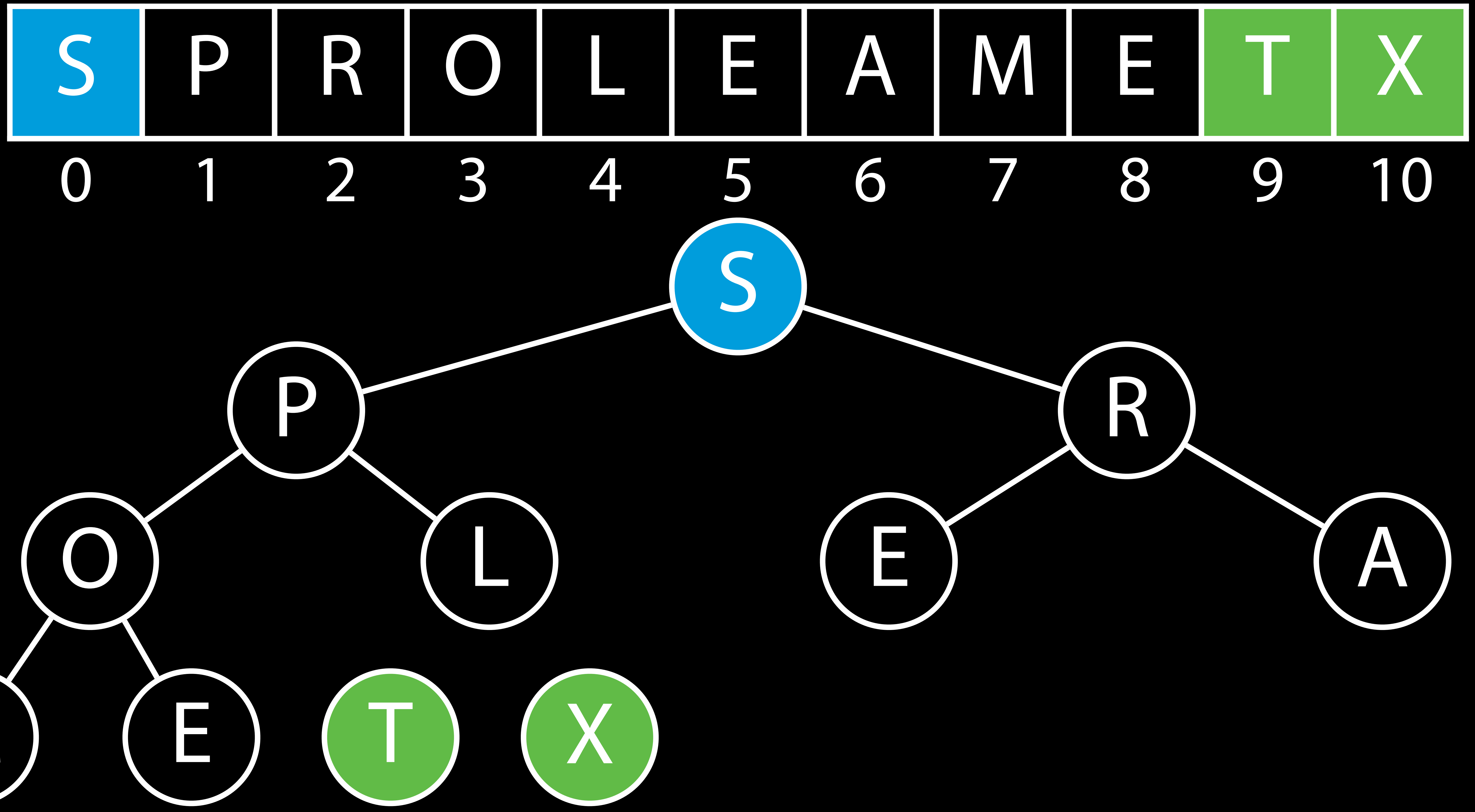
Heap sort

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



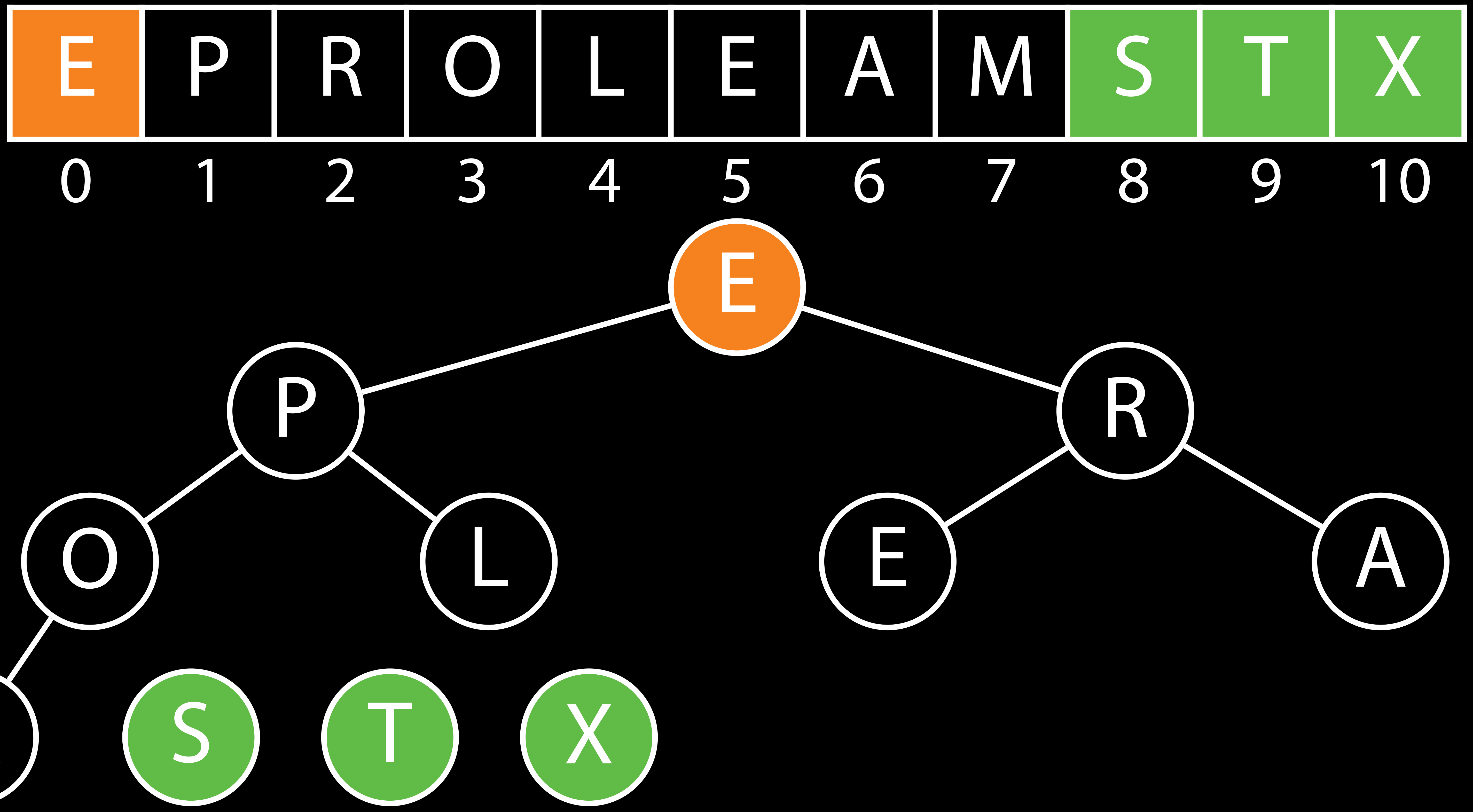
Heap sort

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



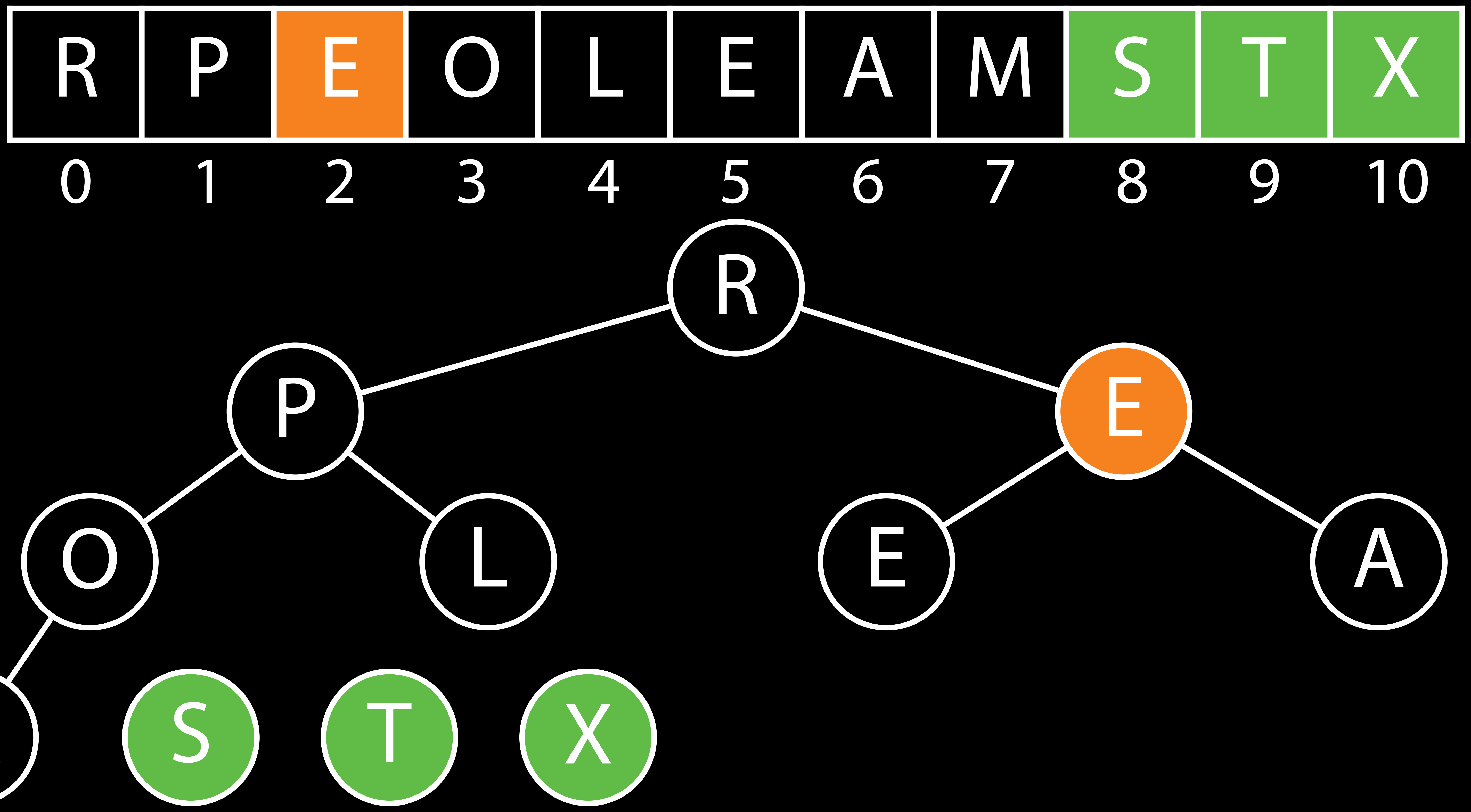
Heap sort

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



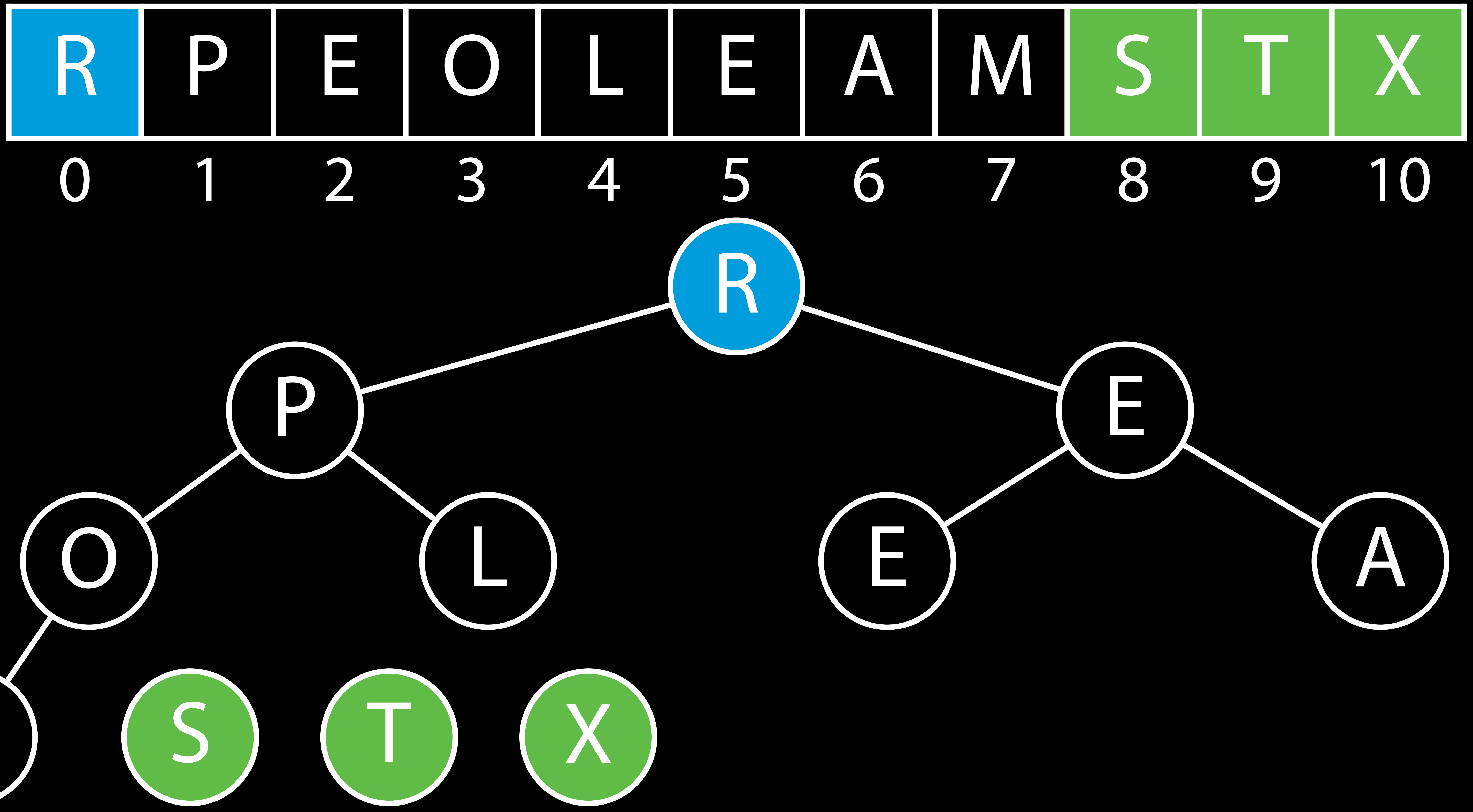
Heap sort

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



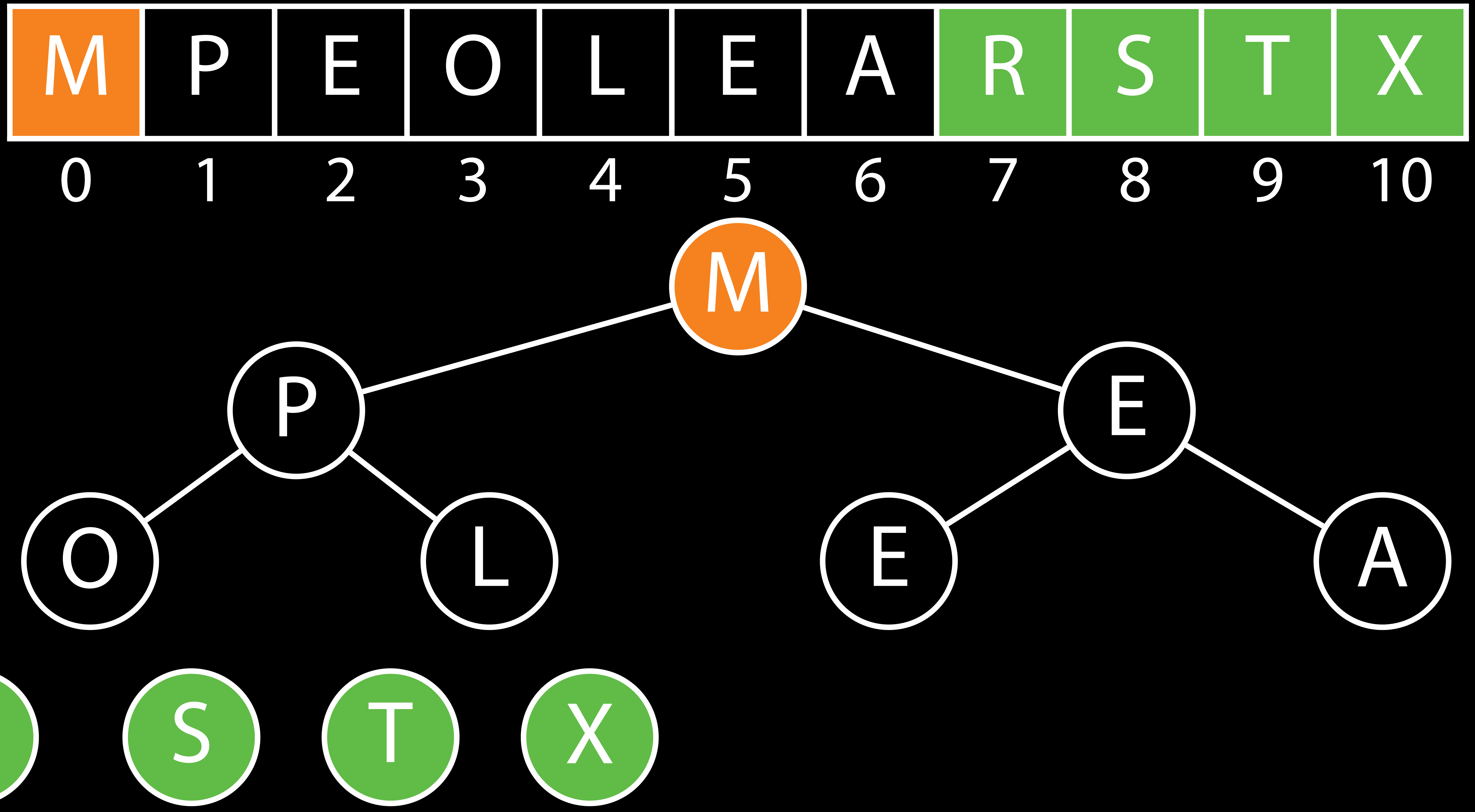
Heap sort

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



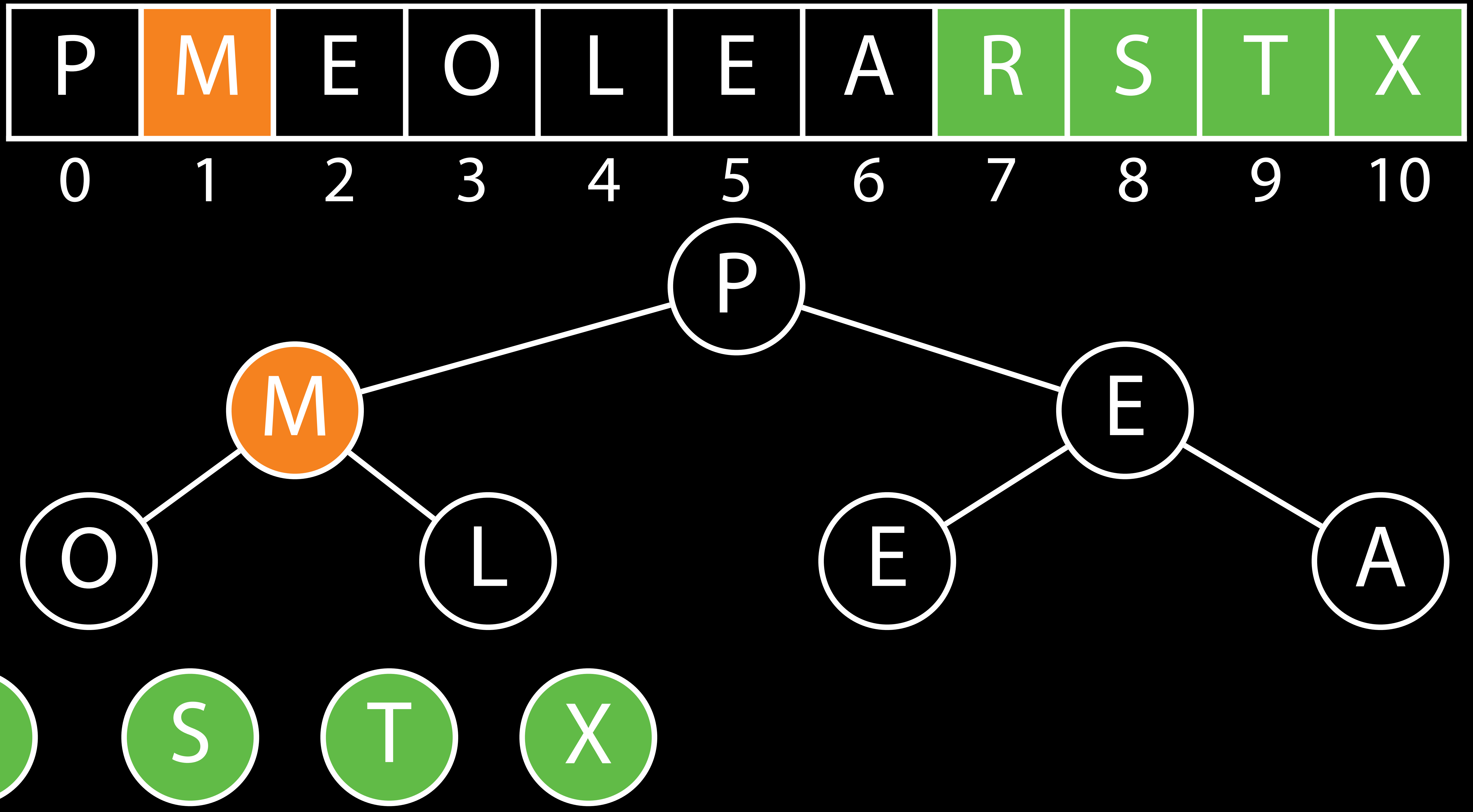
Heap sort

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



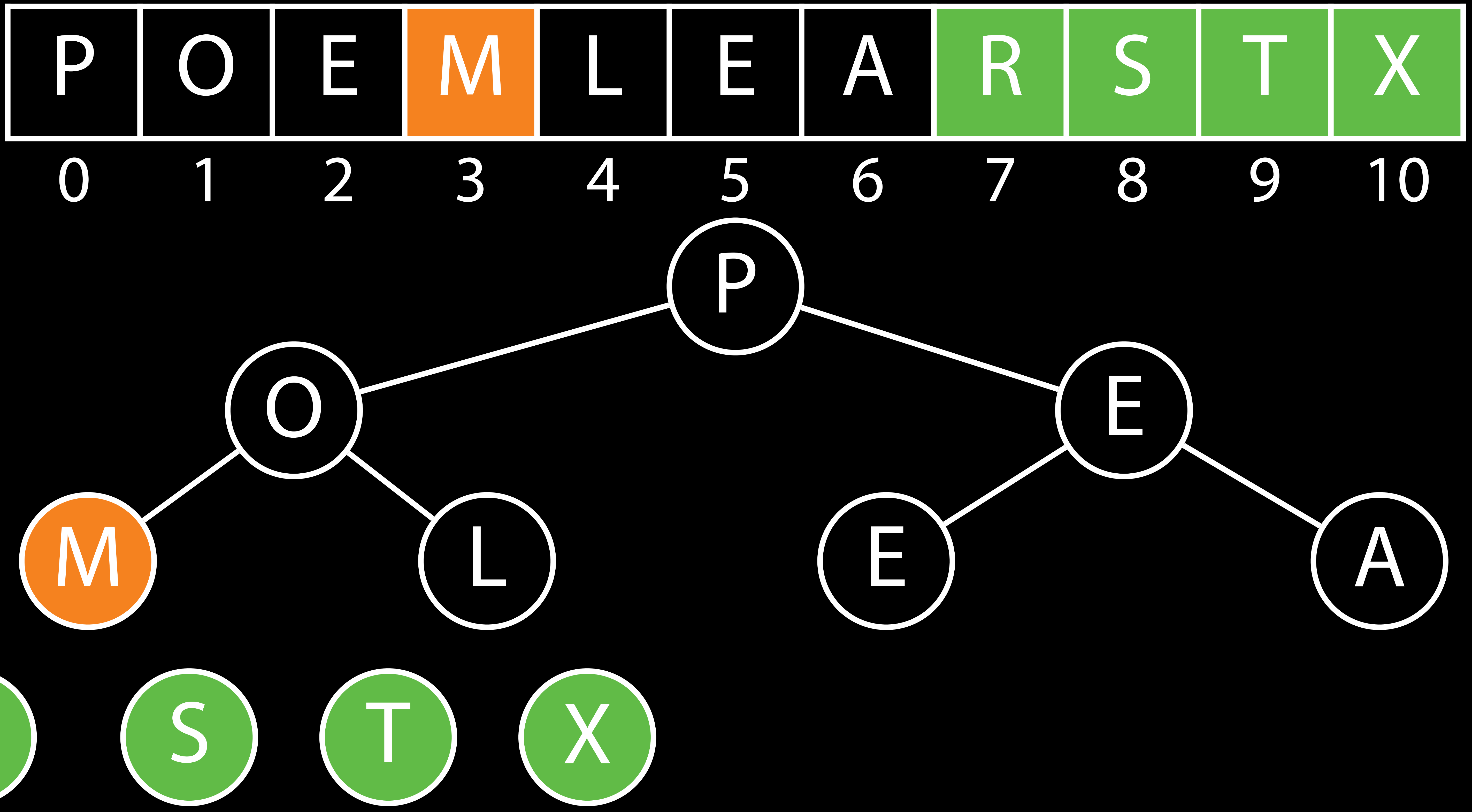
Heap sort

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



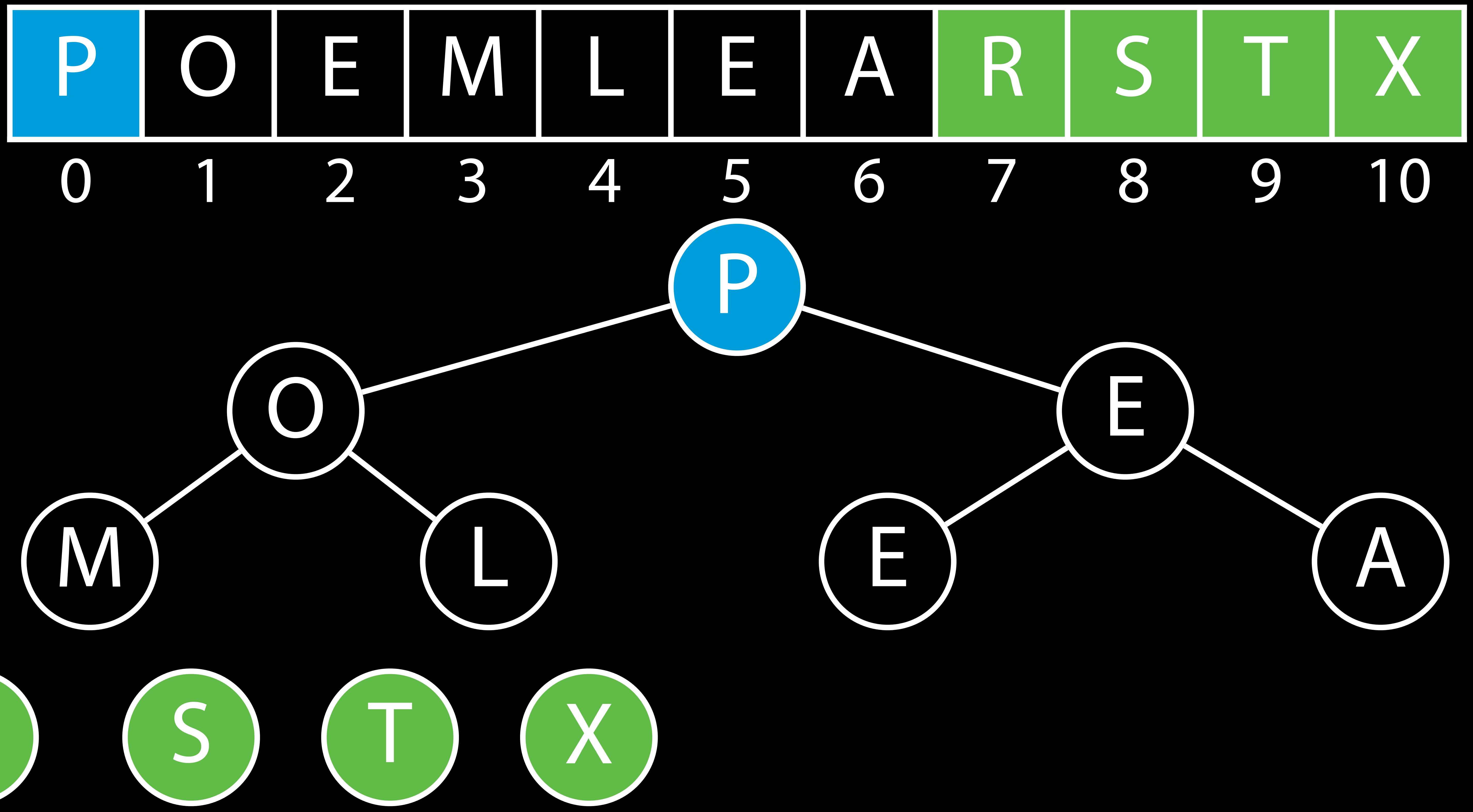
Heap sort

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



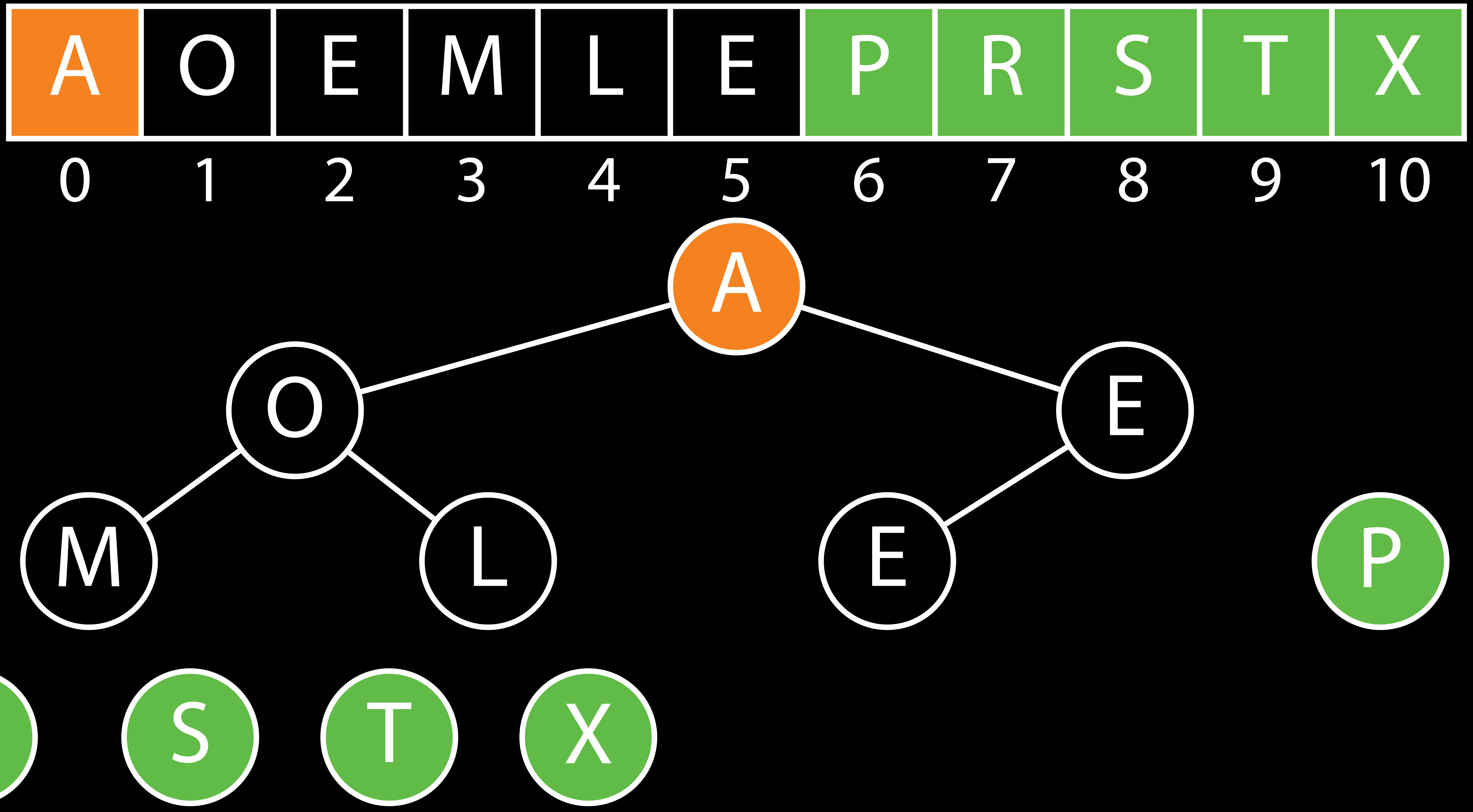
Heap sort

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



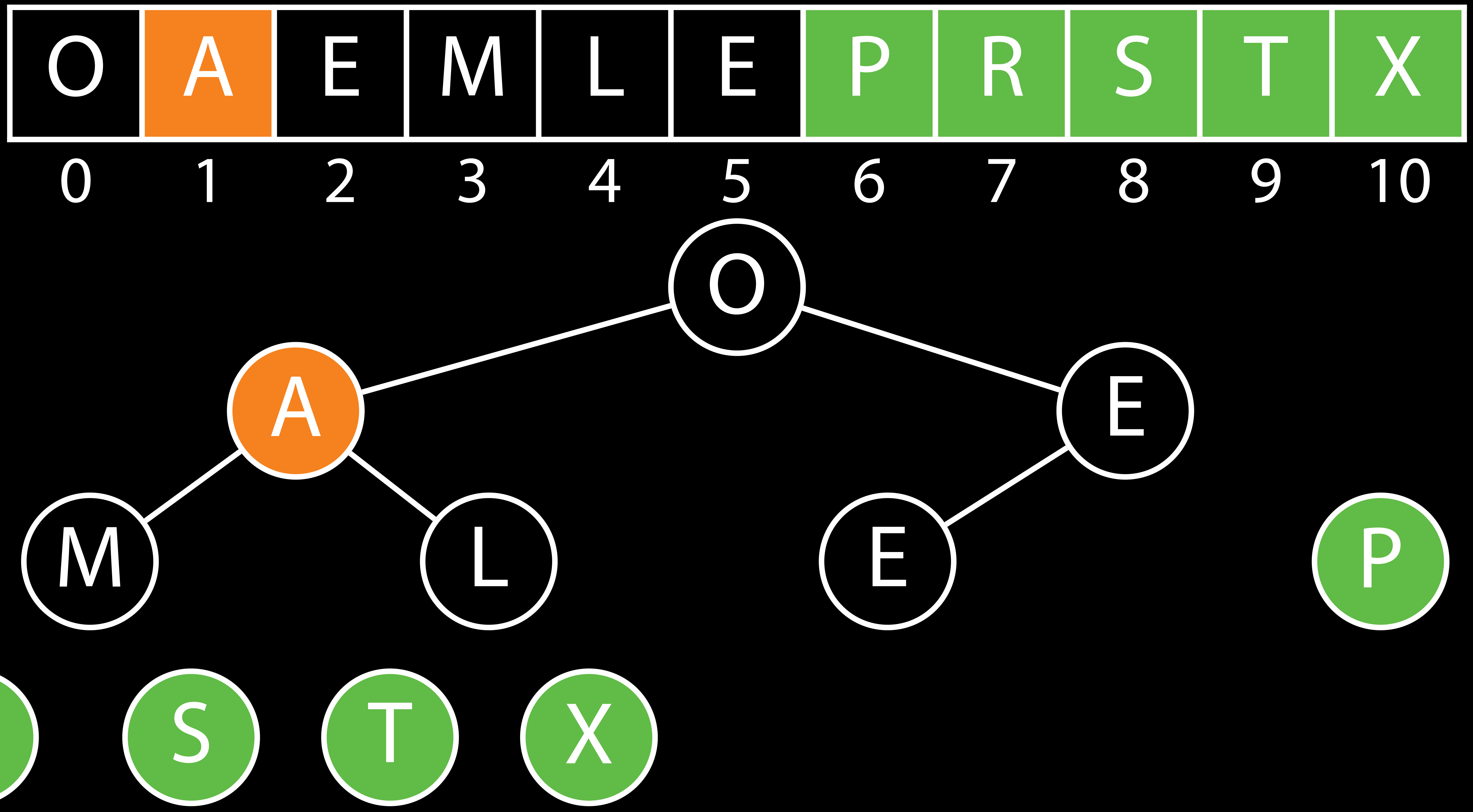
Heap sort

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



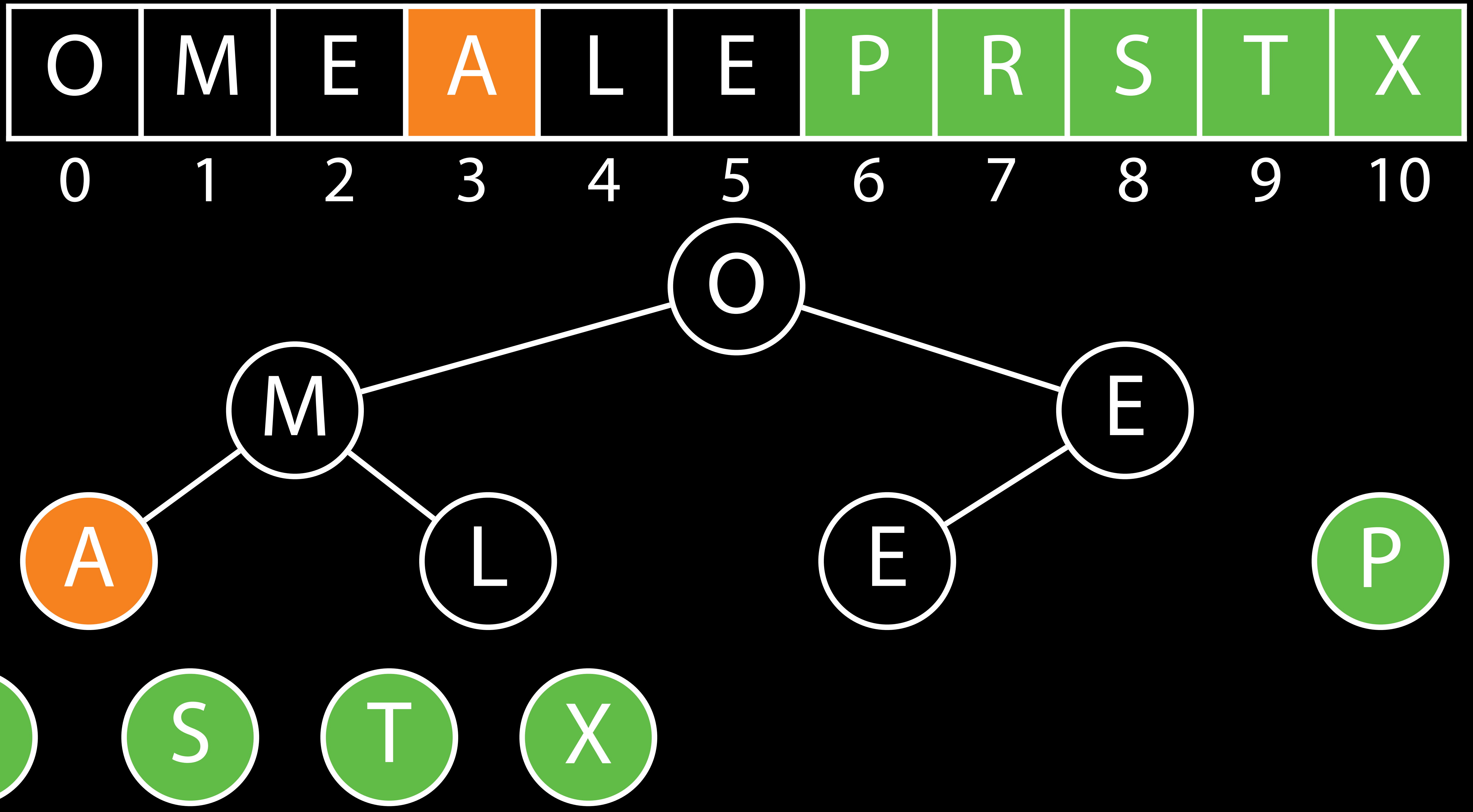
Heap sort

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



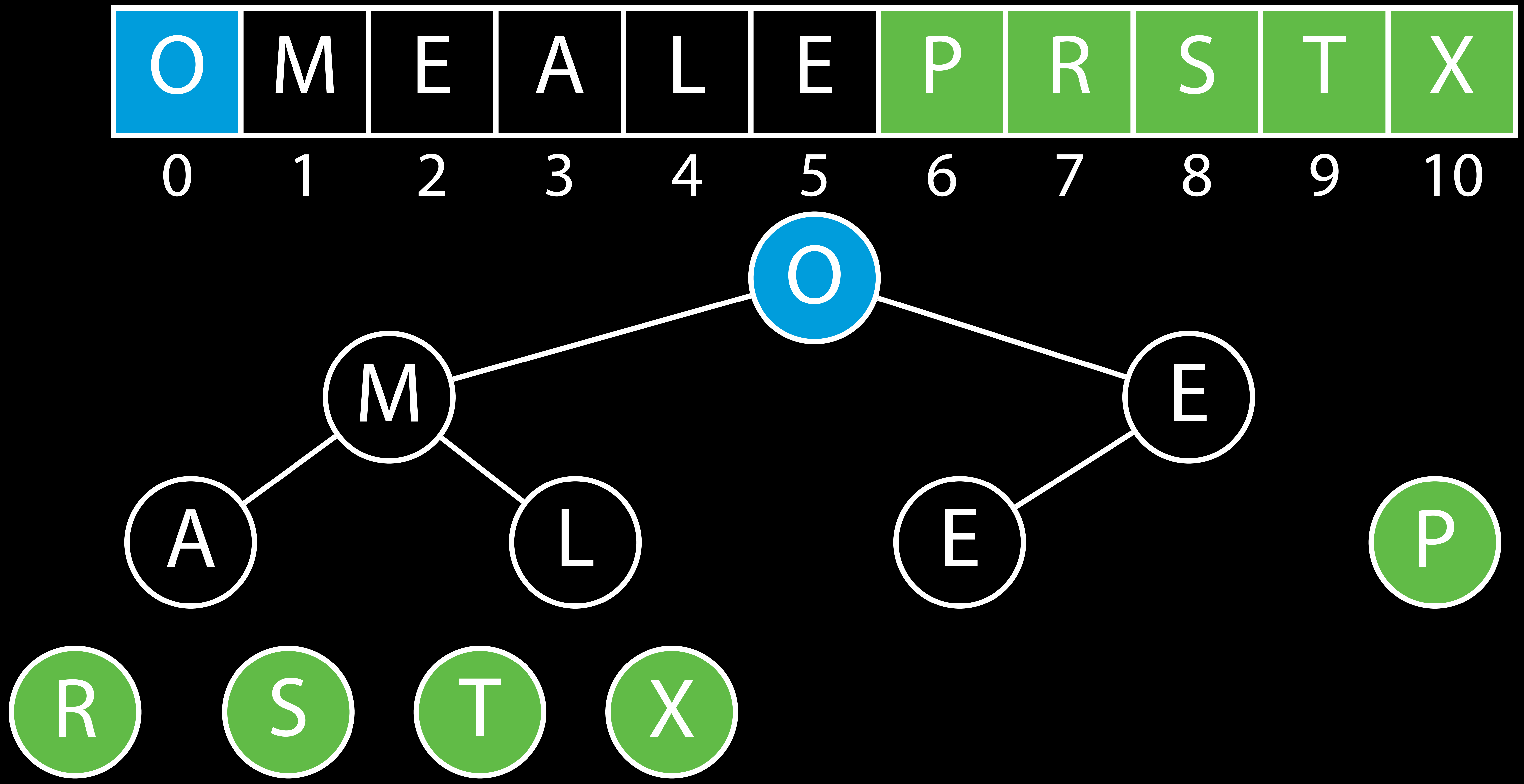
Heap sort

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



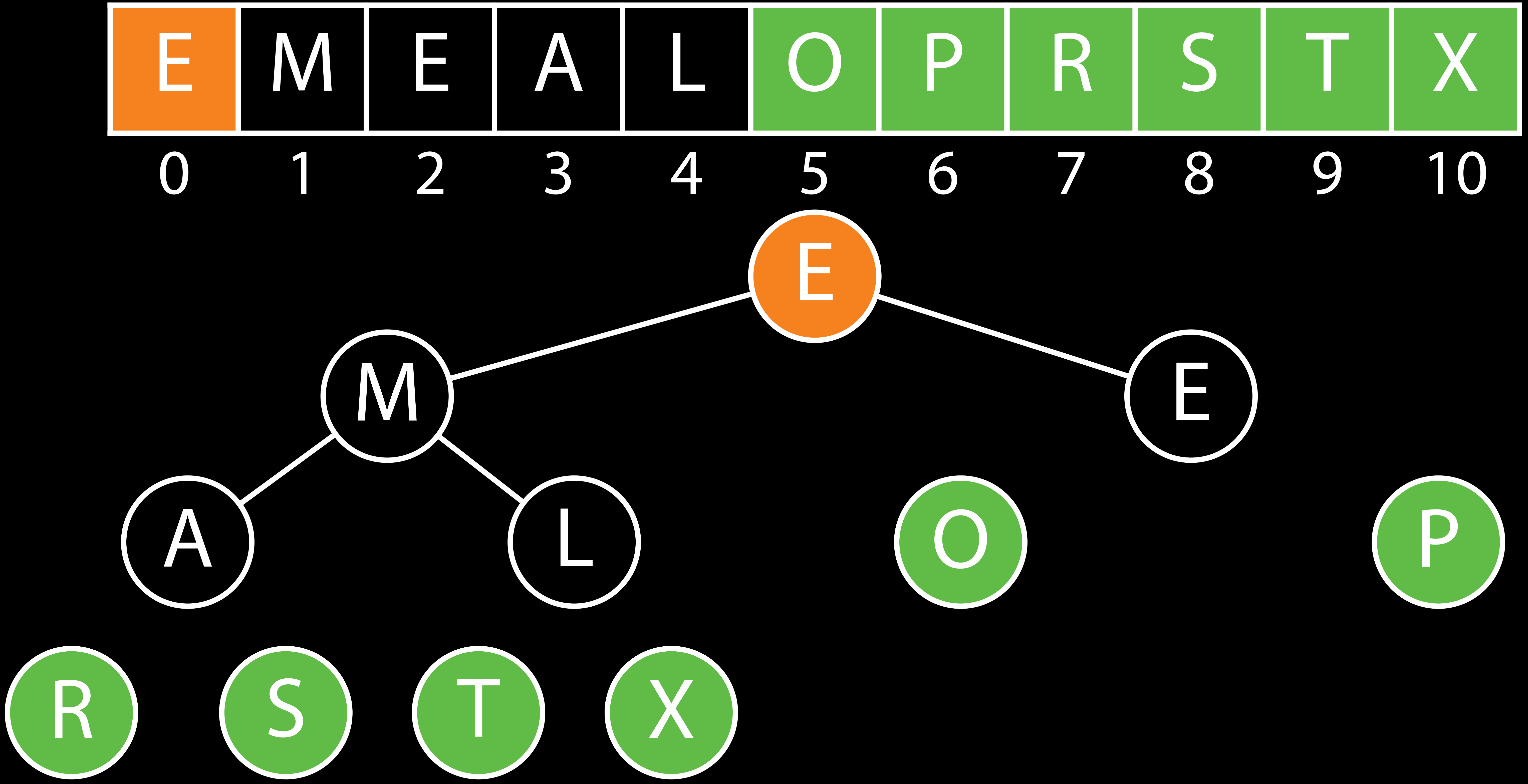
Heap sort

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



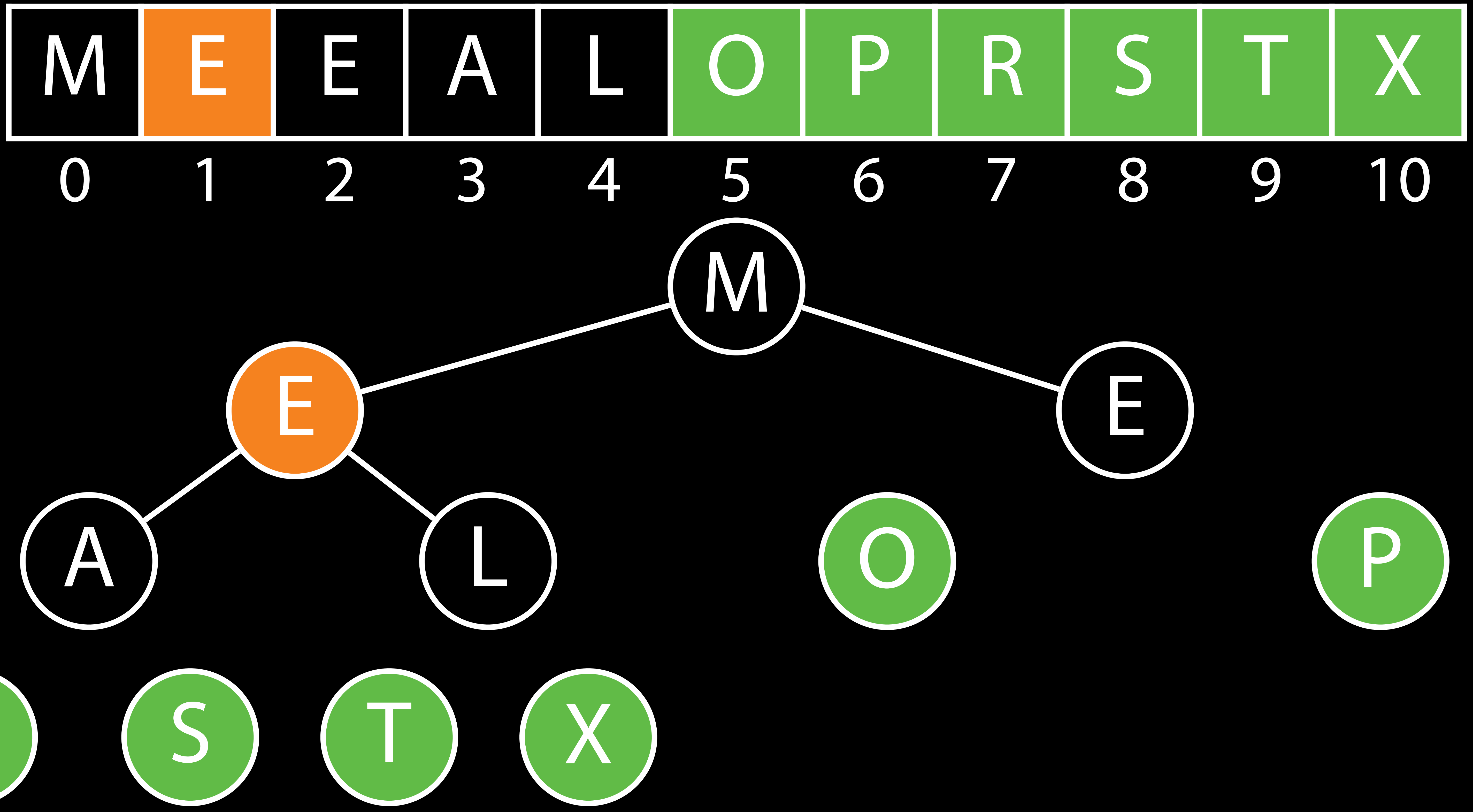
Heap sort

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



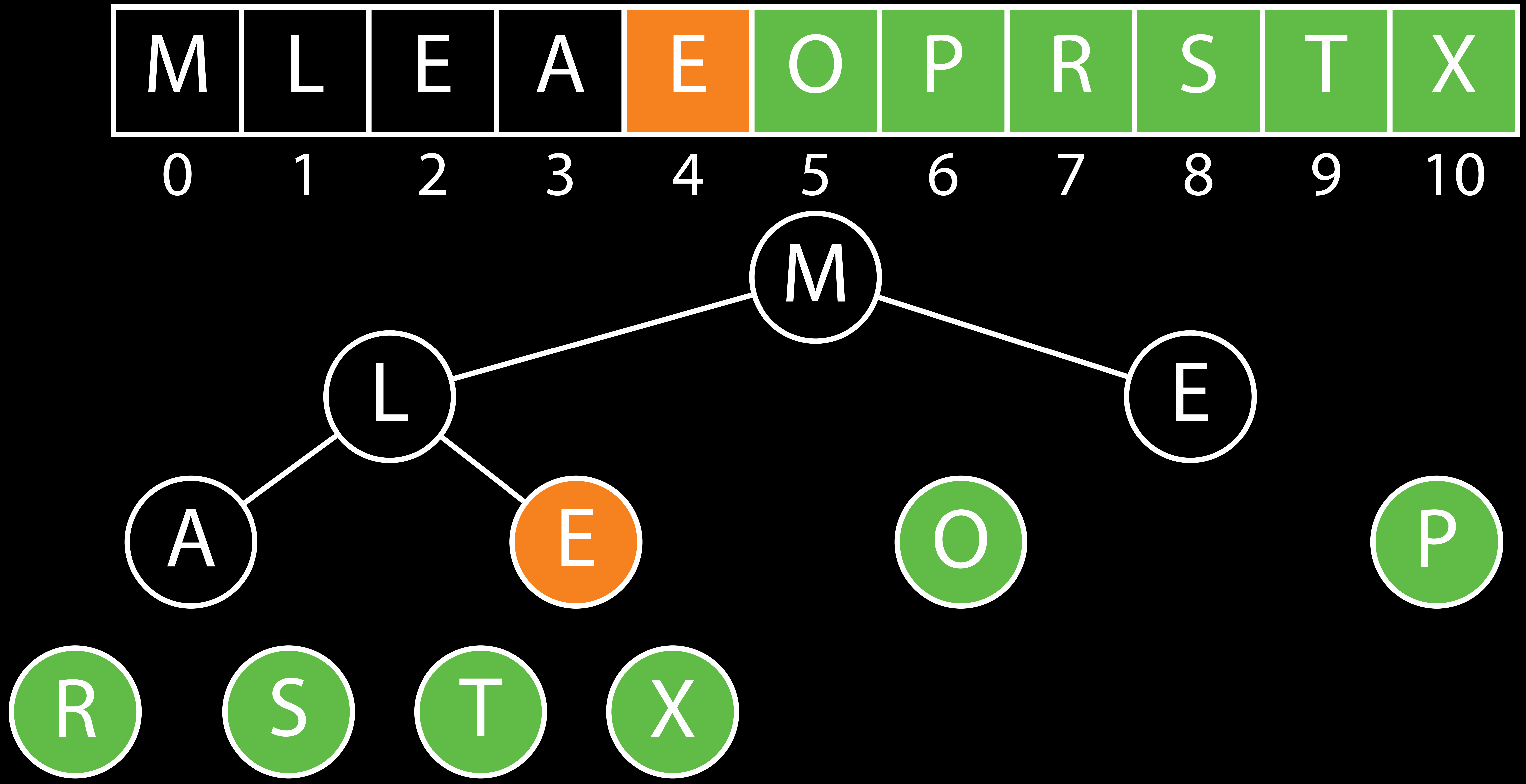
Heap sort

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



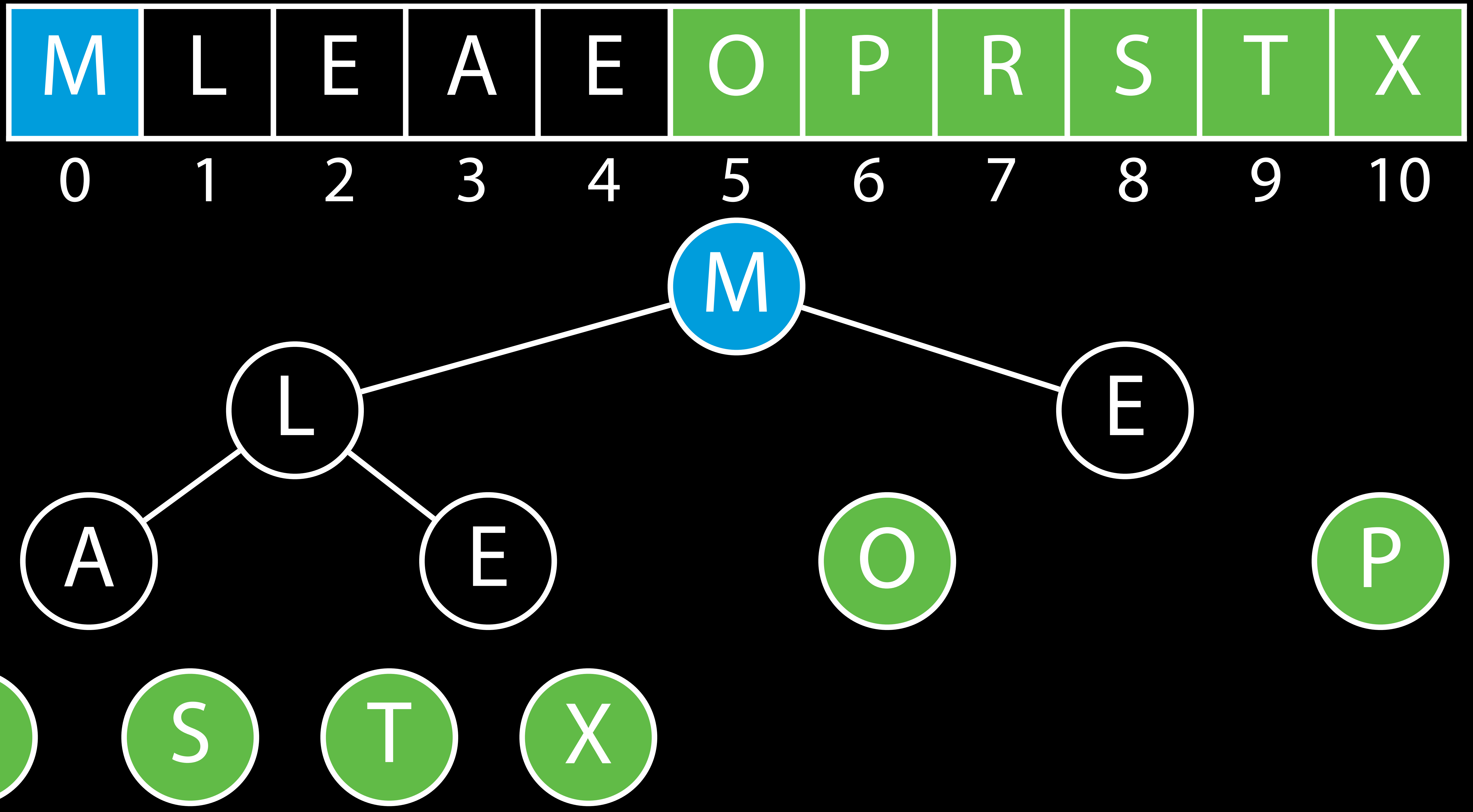
Heap sort

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



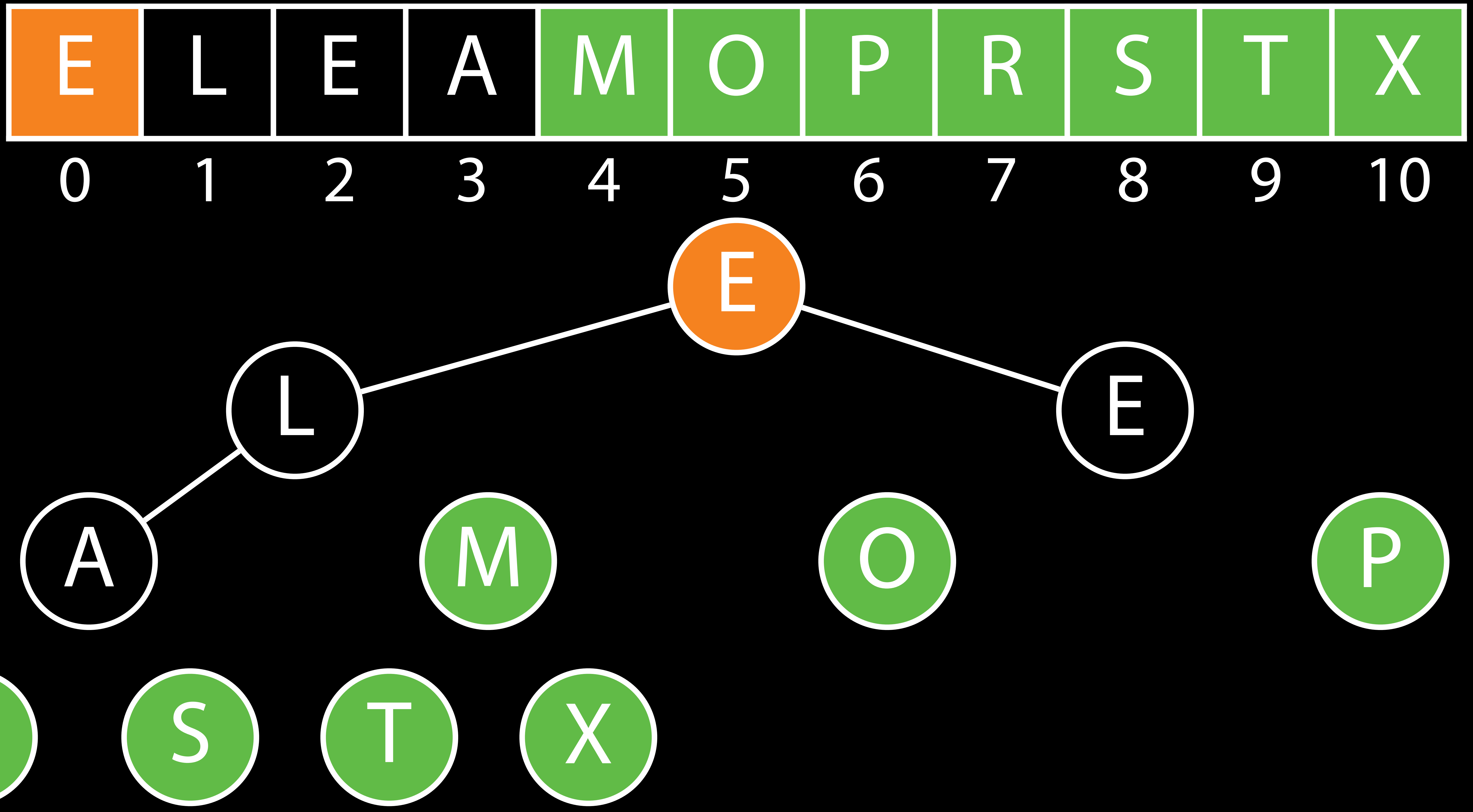
Heap sort

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



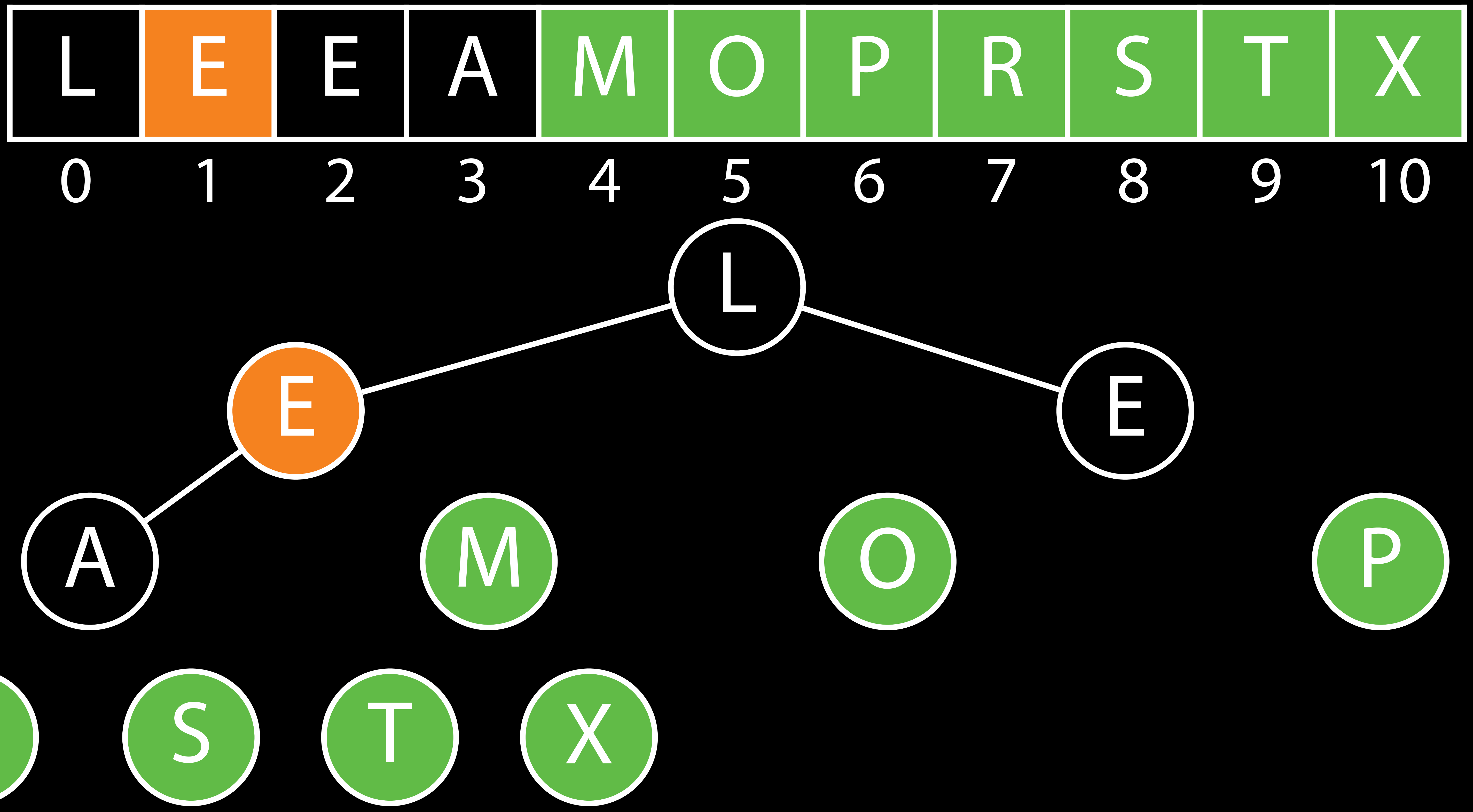
Heap sort

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



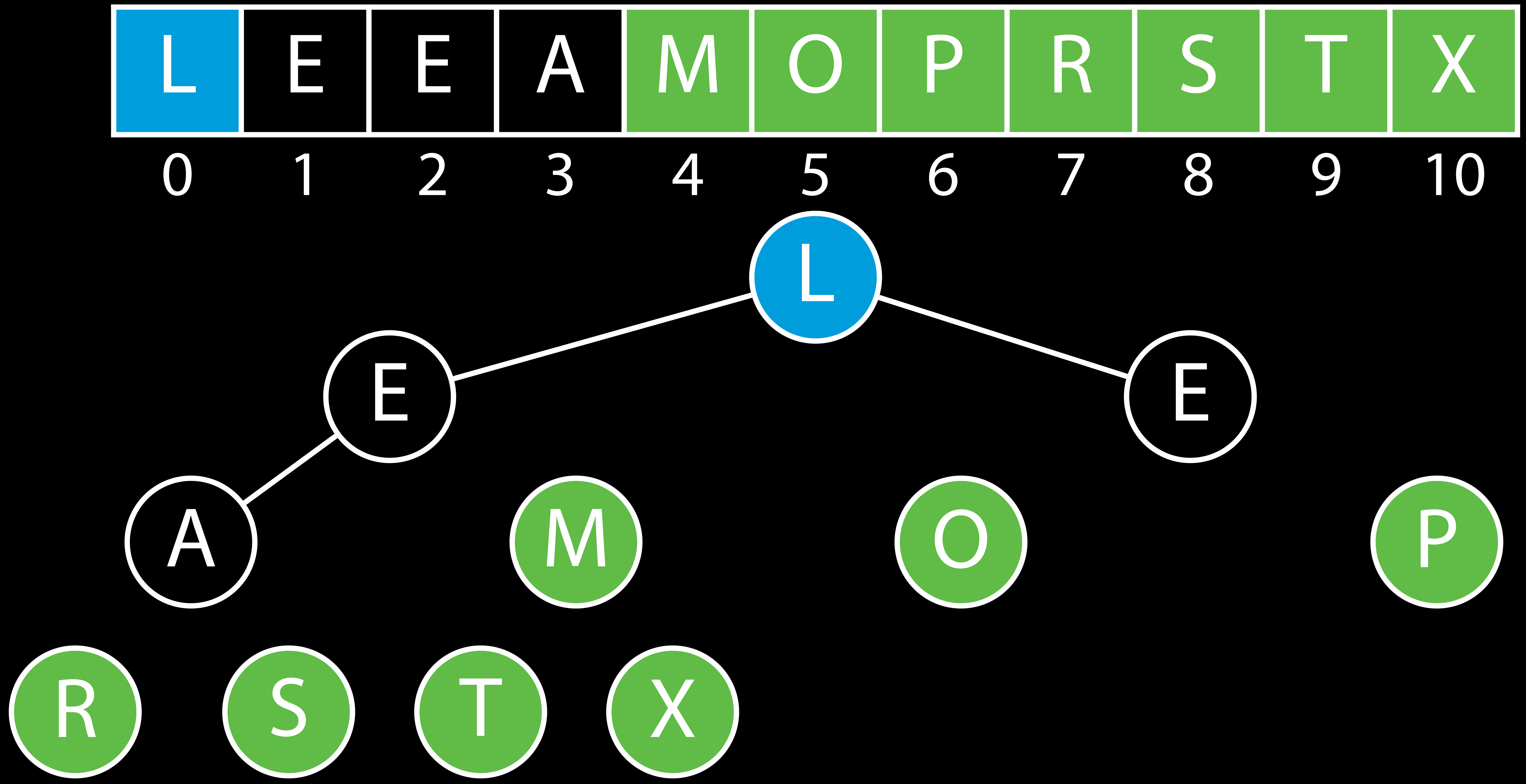
Heap sort

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



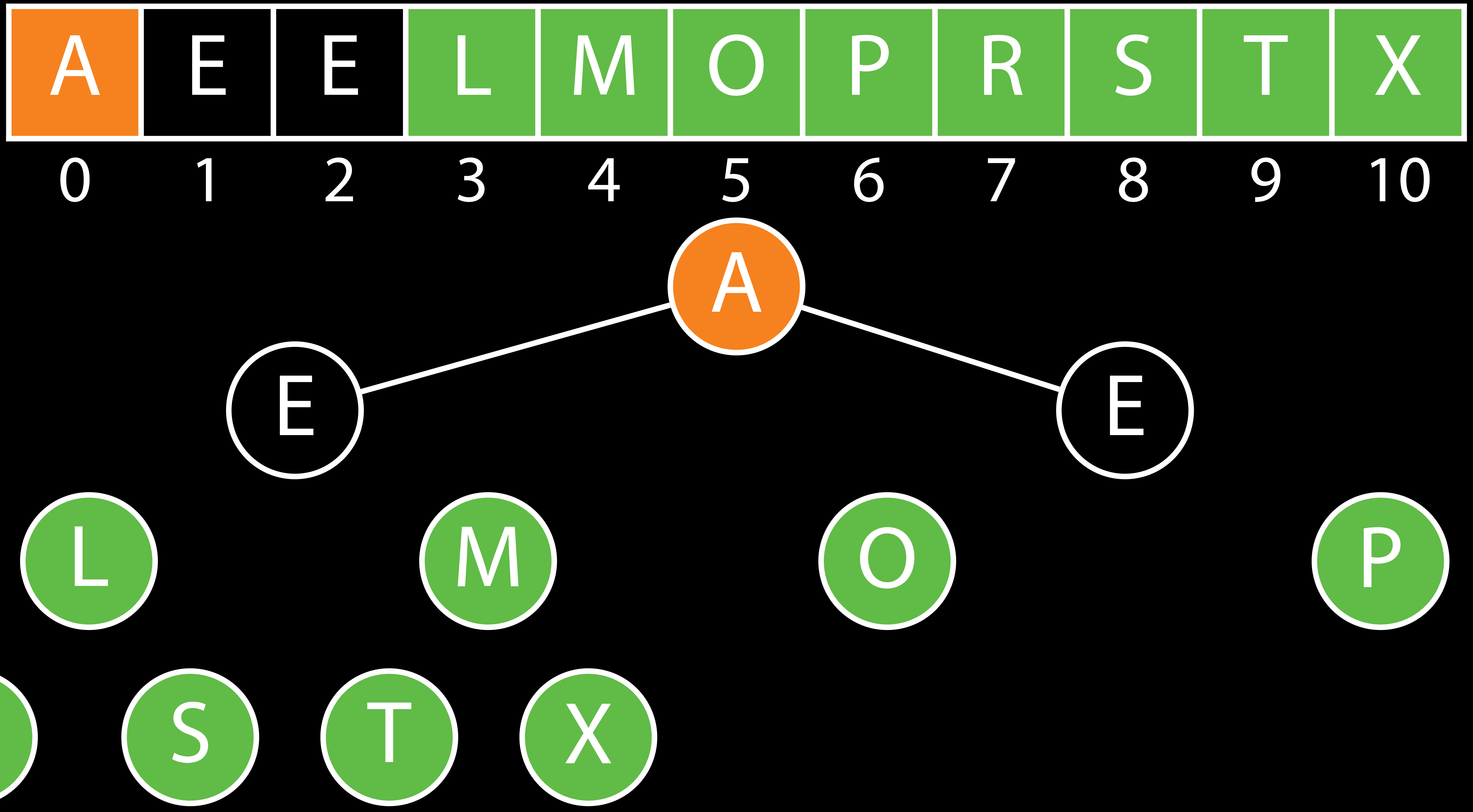
Heap sort

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



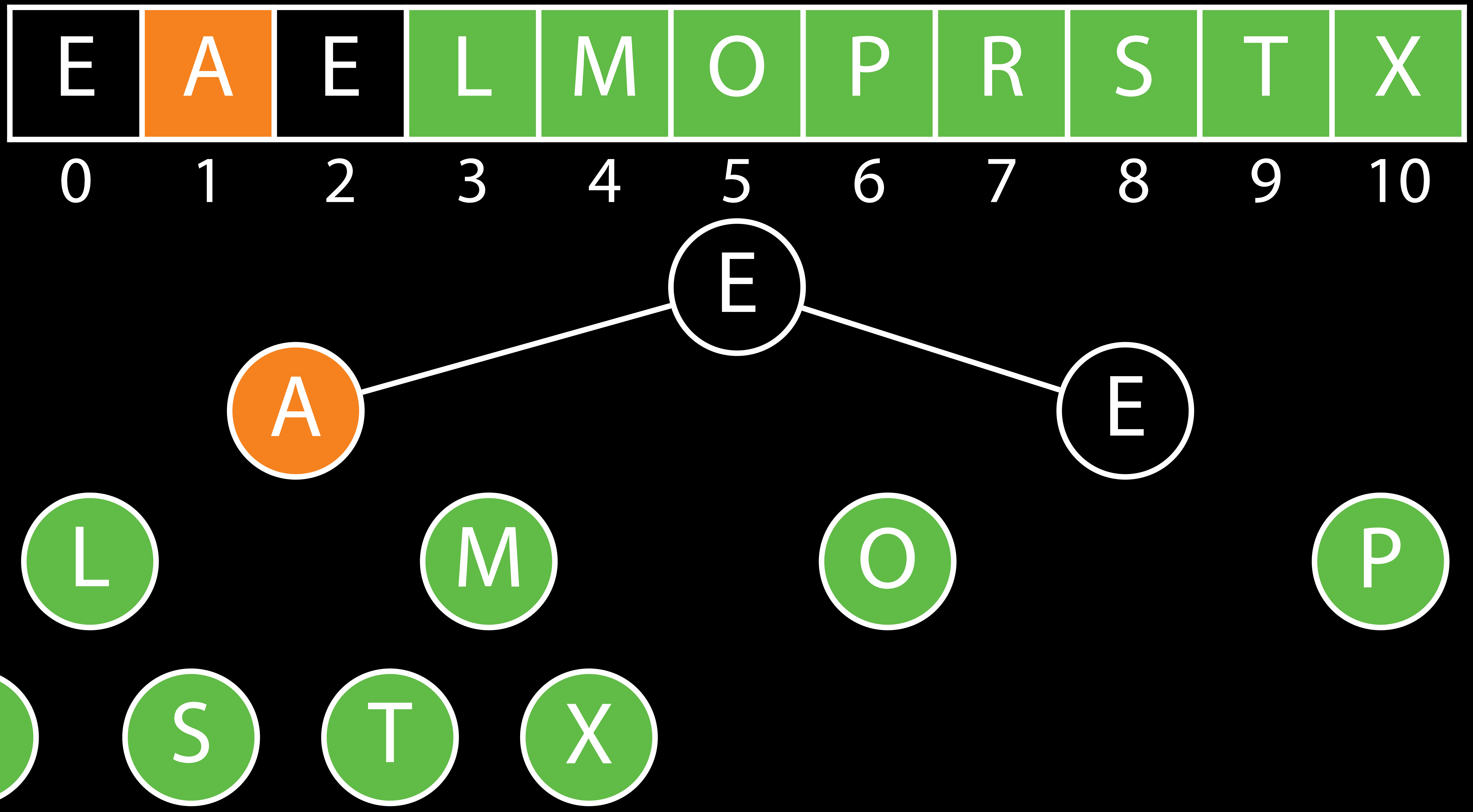
Heap sort

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



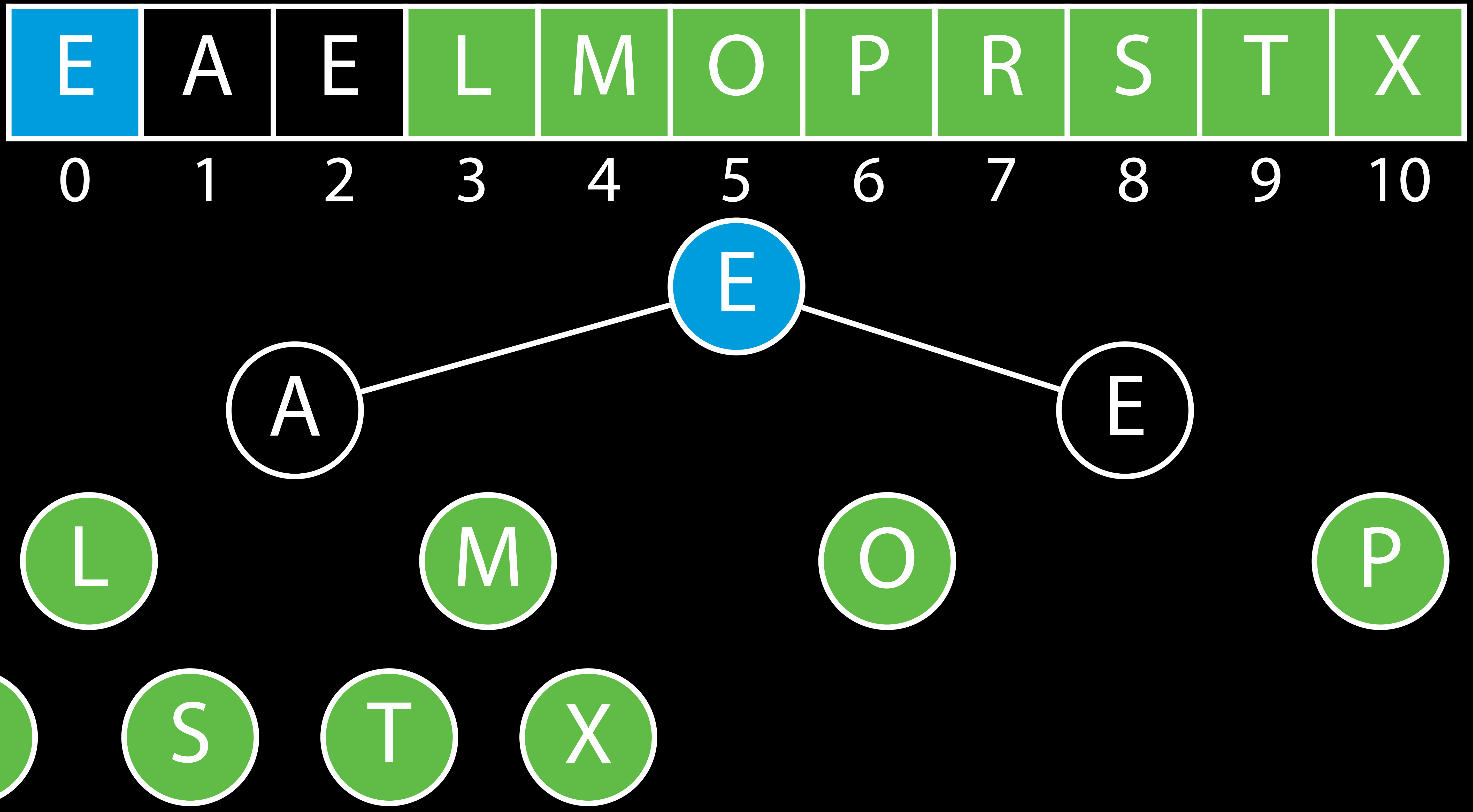
Heap sort

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



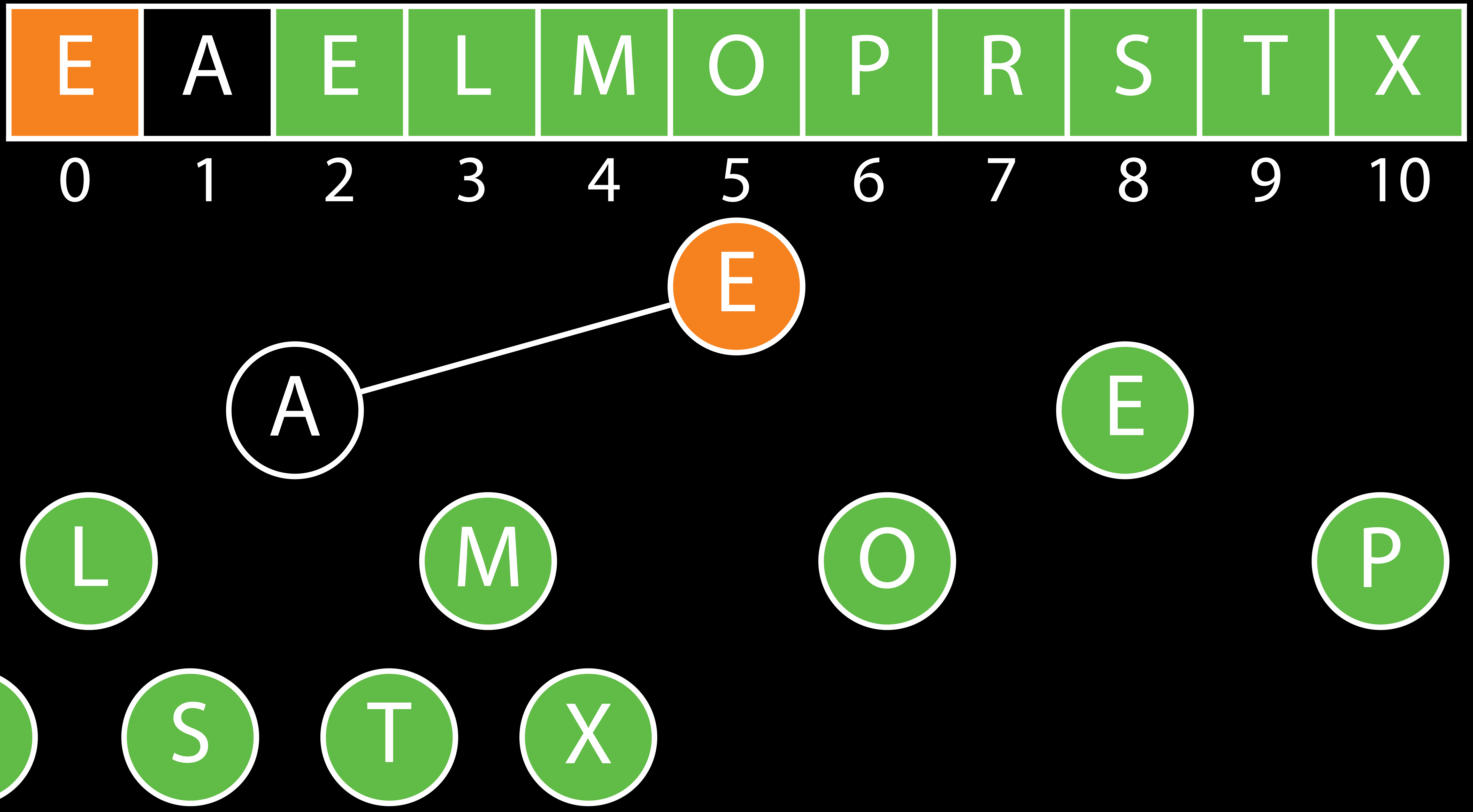
Heap sort

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



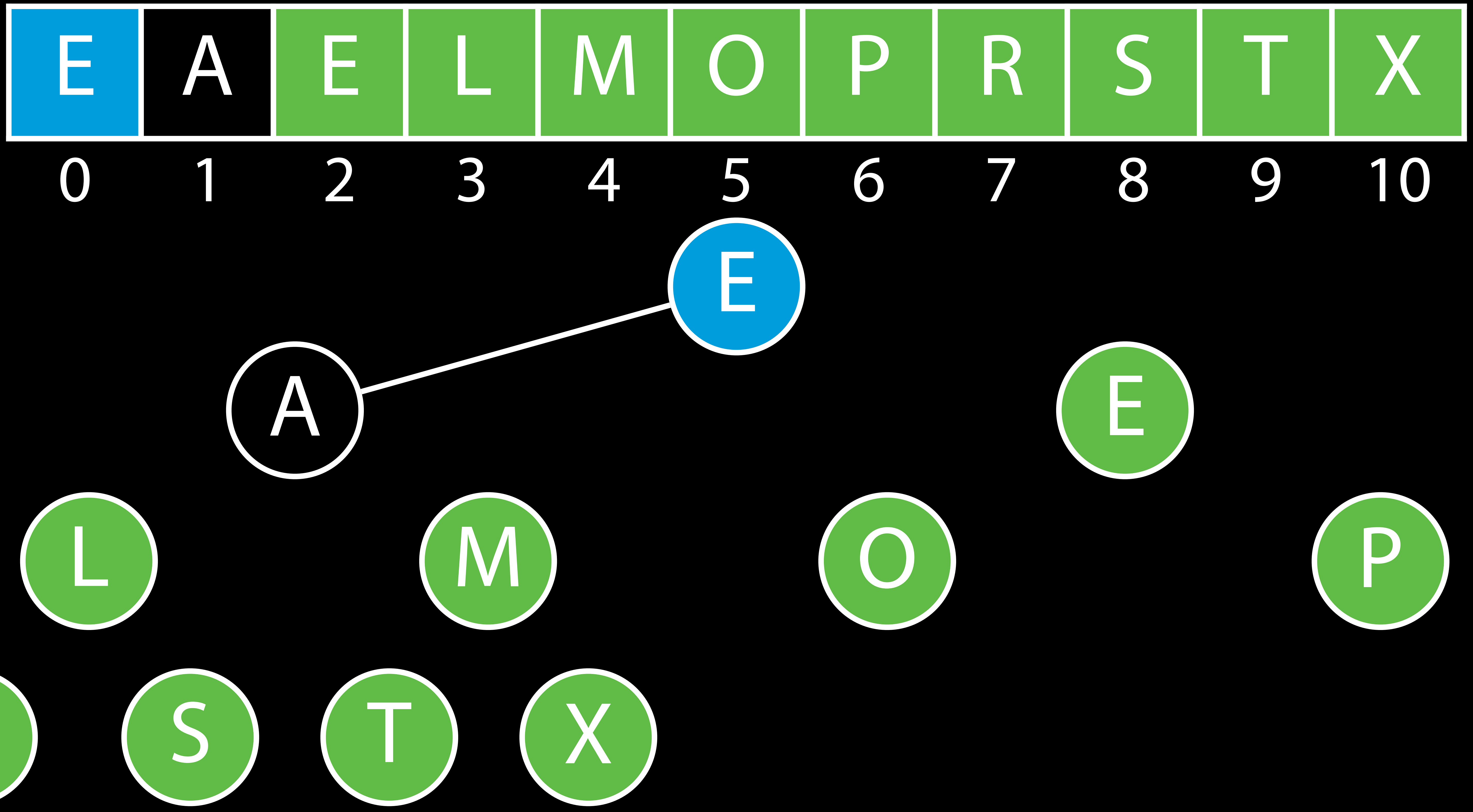
Heap sort

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



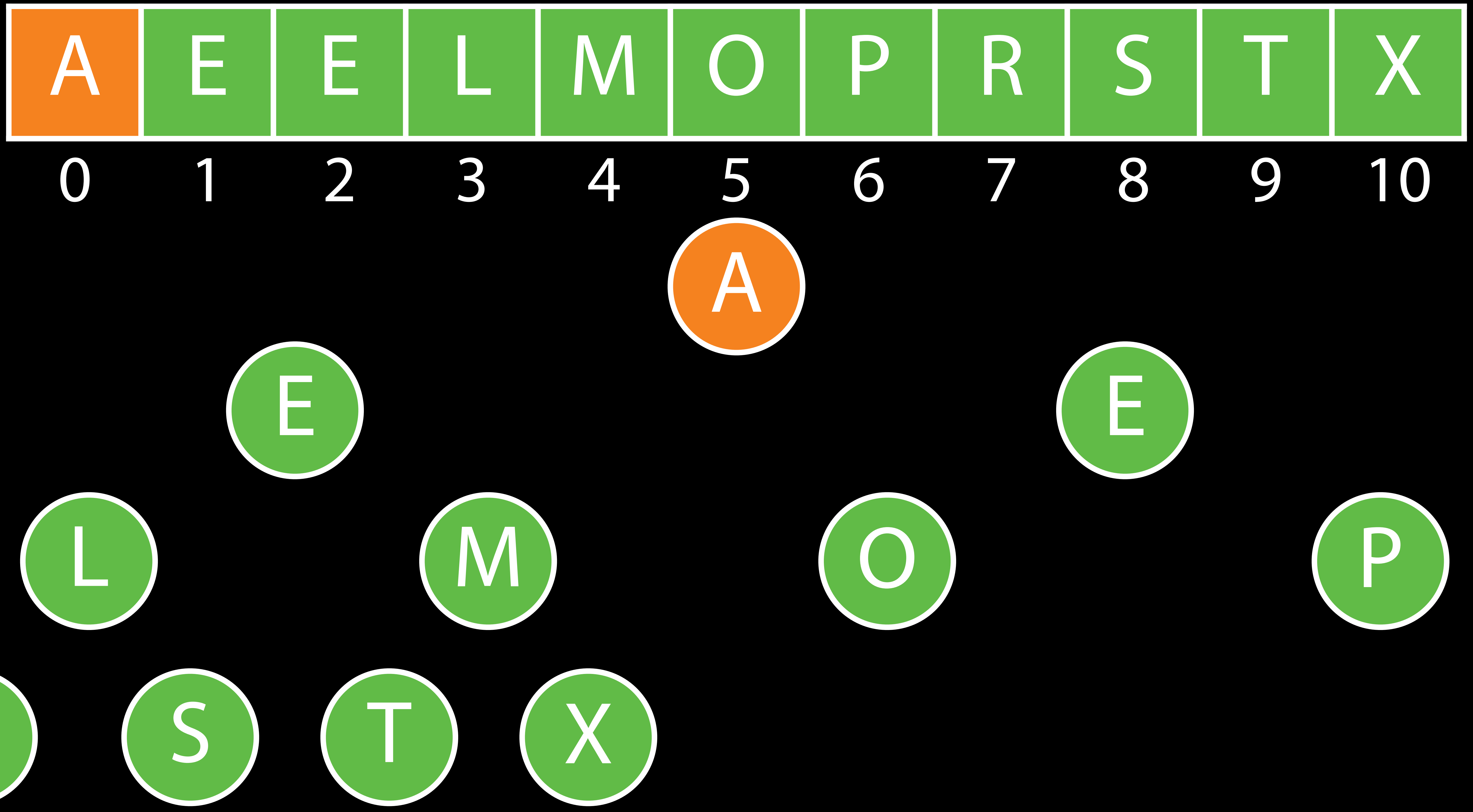
Heap sort

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



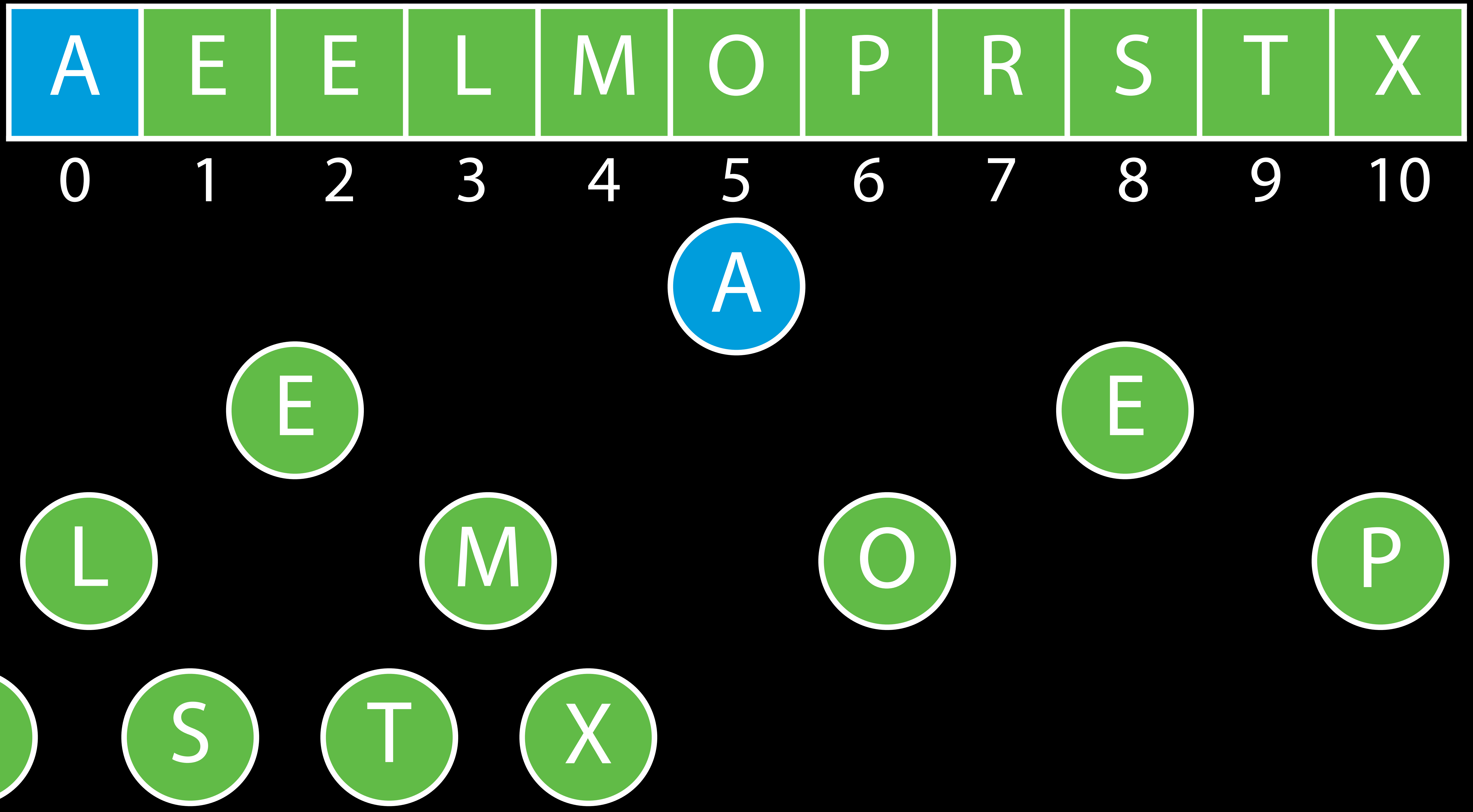
Heap sort

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



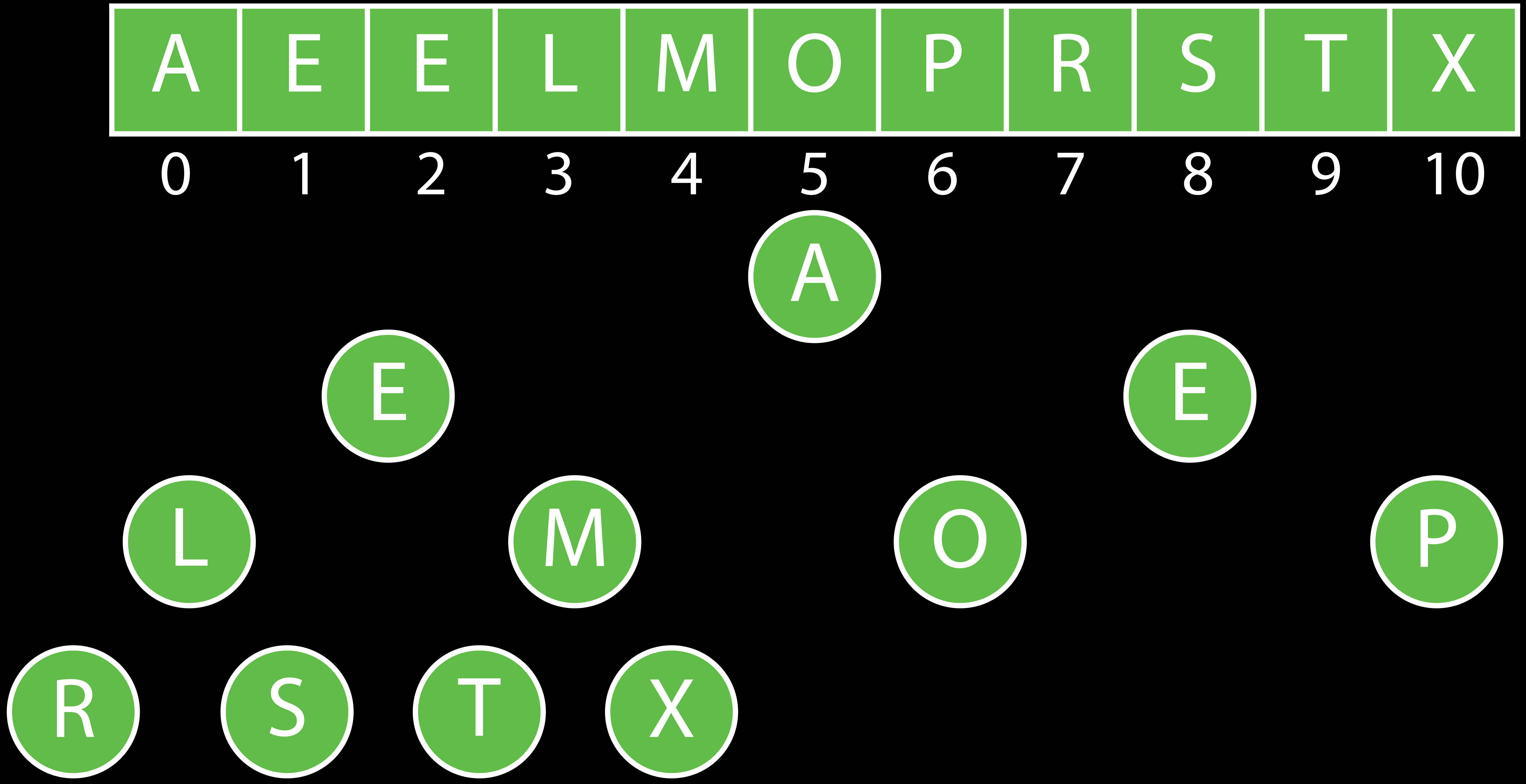
Heap sort

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



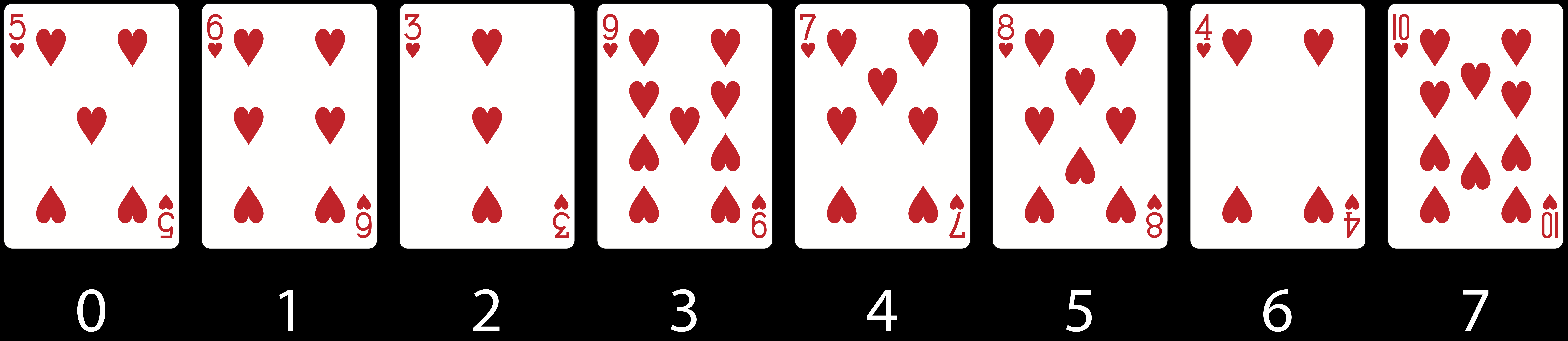
Heap sort

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



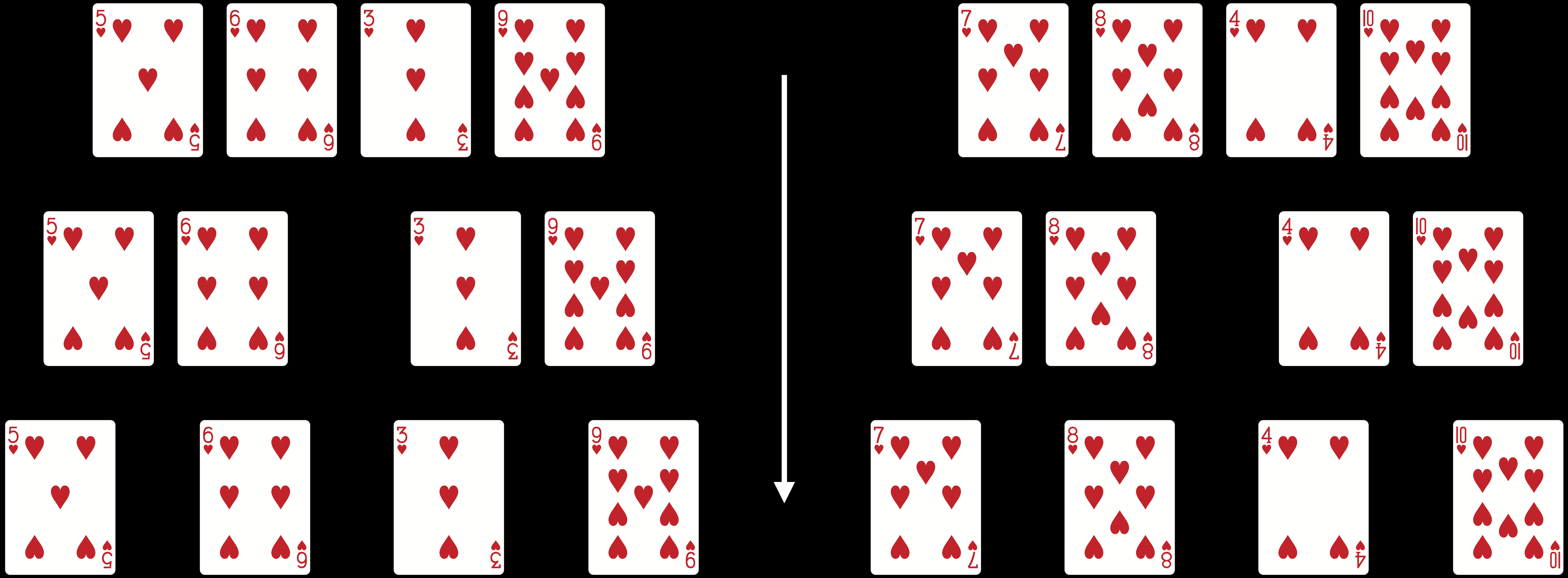
Merge Sort

Sort left half.
Sort right half.
Merge sorted halves.



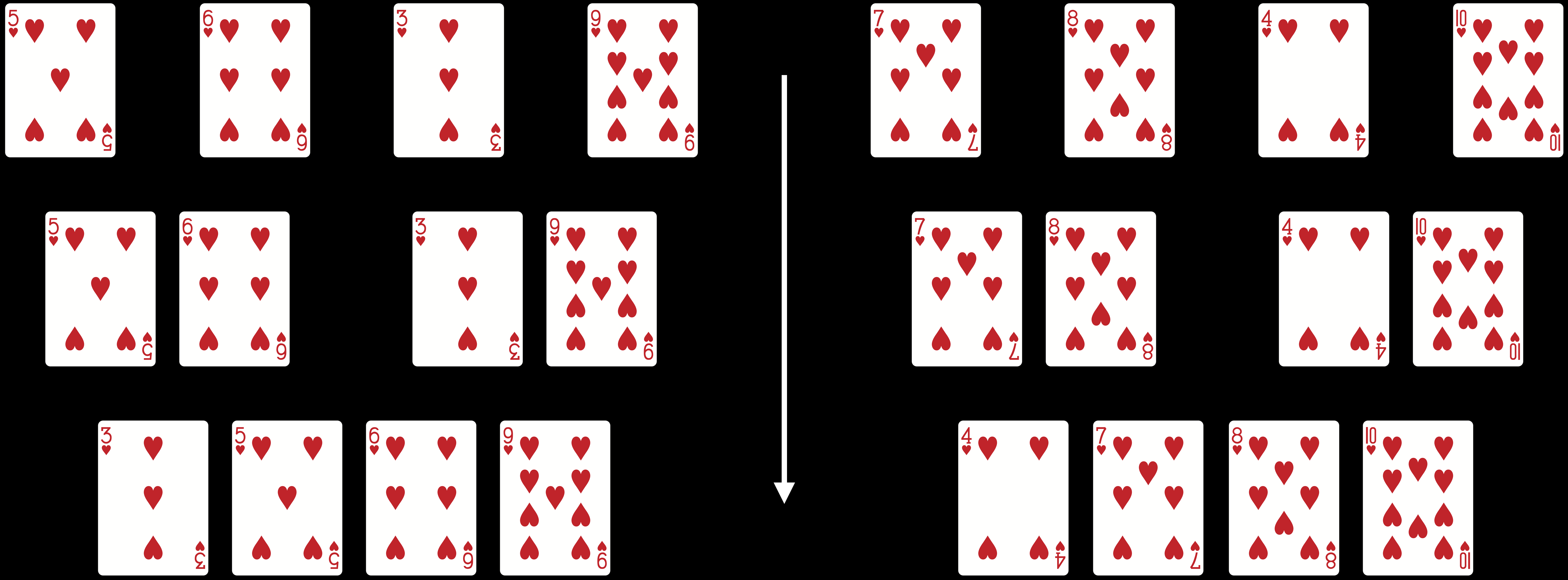
Merge Sort

Splitting into two halves until base case (a single item is sorted).



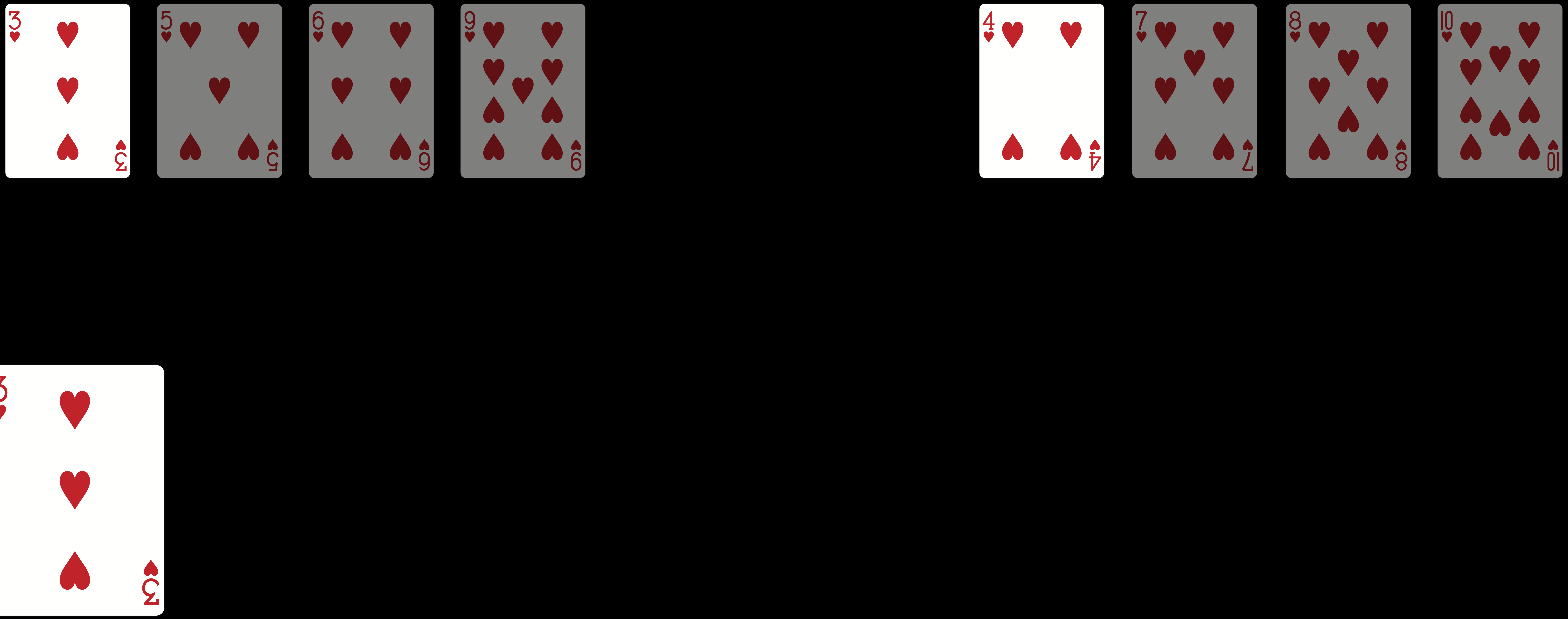
Merge Sort

Merging each half



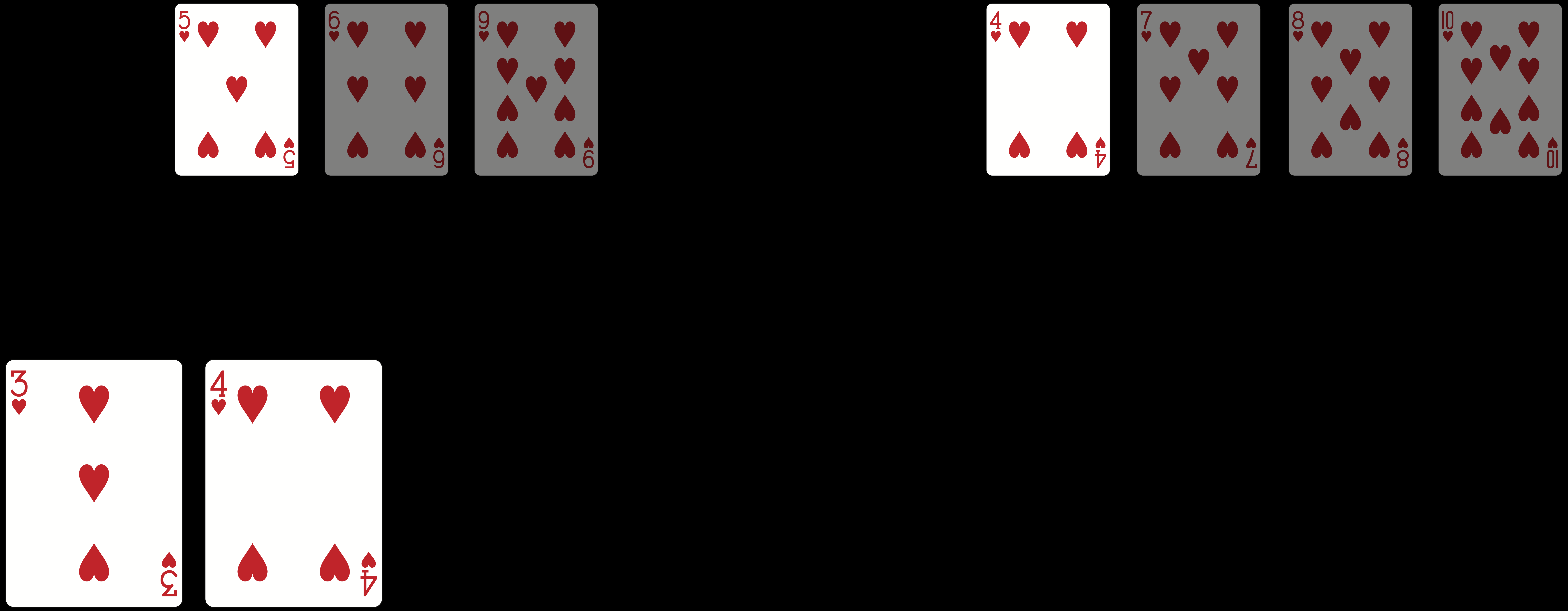
Merge Sort

Final merging (takes O(*n*) time)



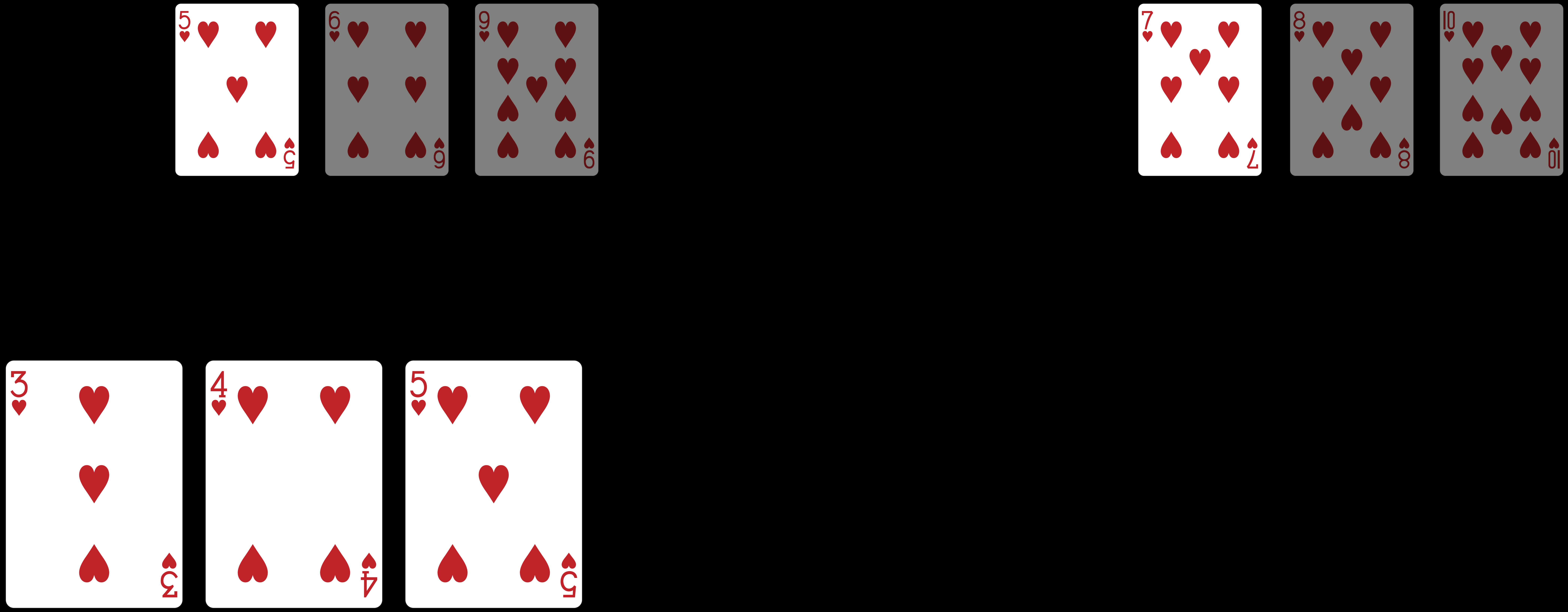
Merge Sort

Final merging (takes O(*n*) time)



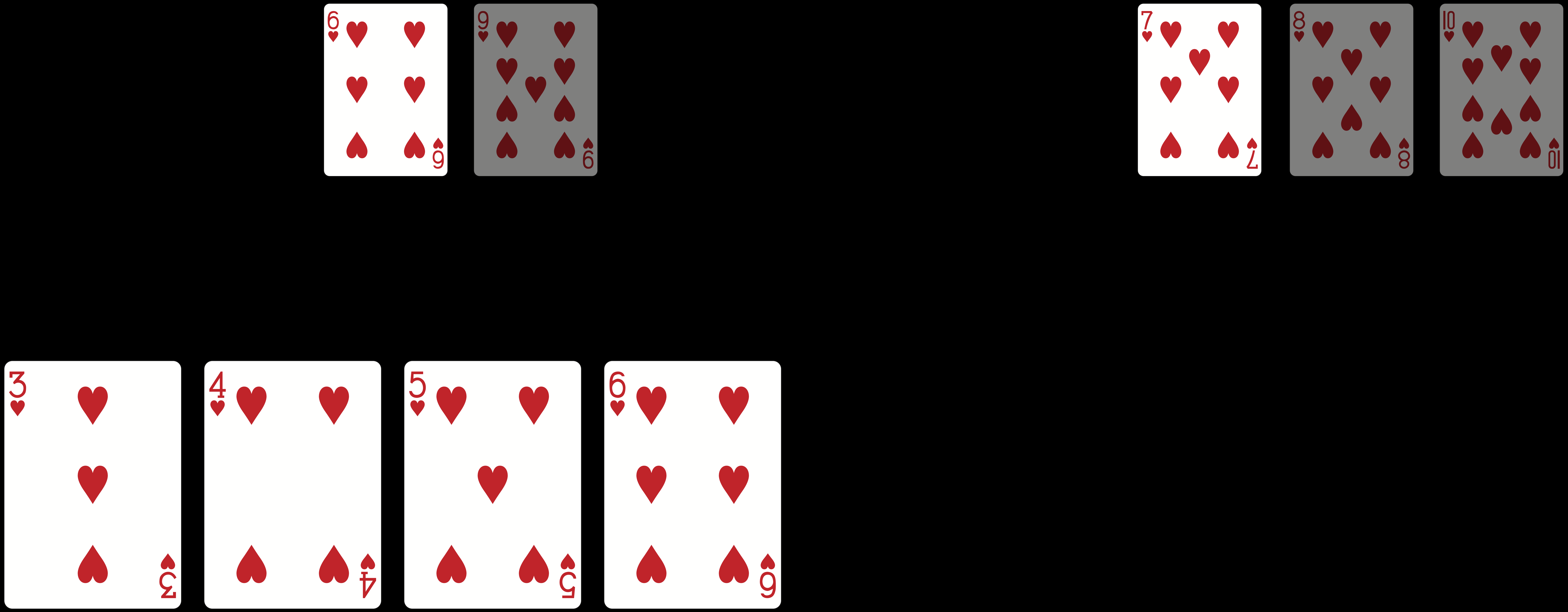
Merge Sort

Final merging (takes O(*n*) time)



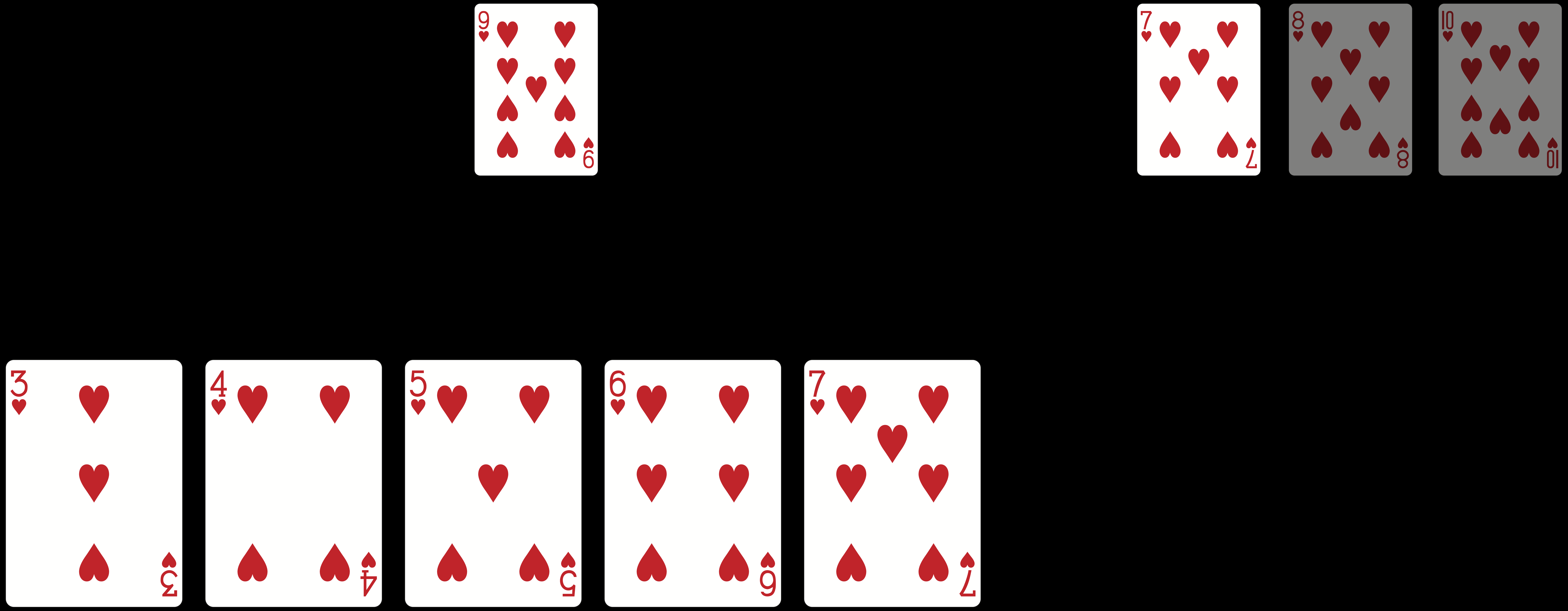
Merge Sort

Final merging (takes O(*n*) time)



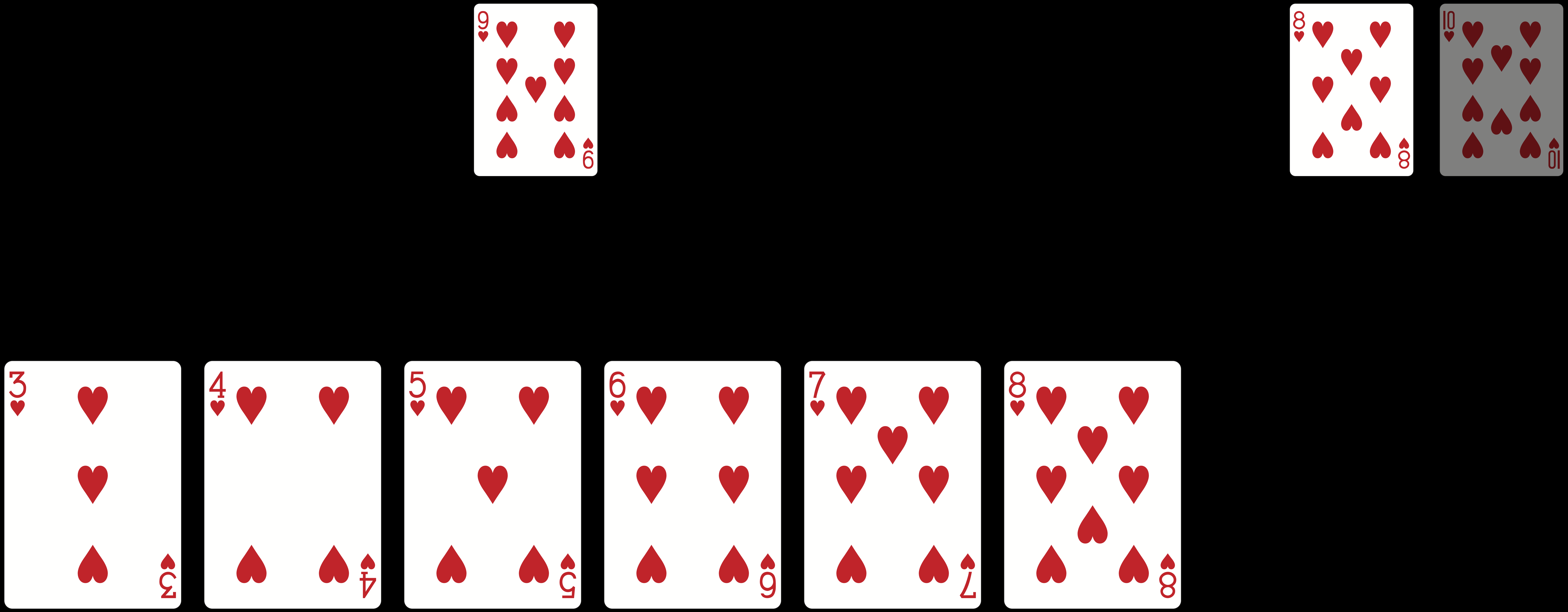
Merge Sort

Final merging (takes O(*n*) time)



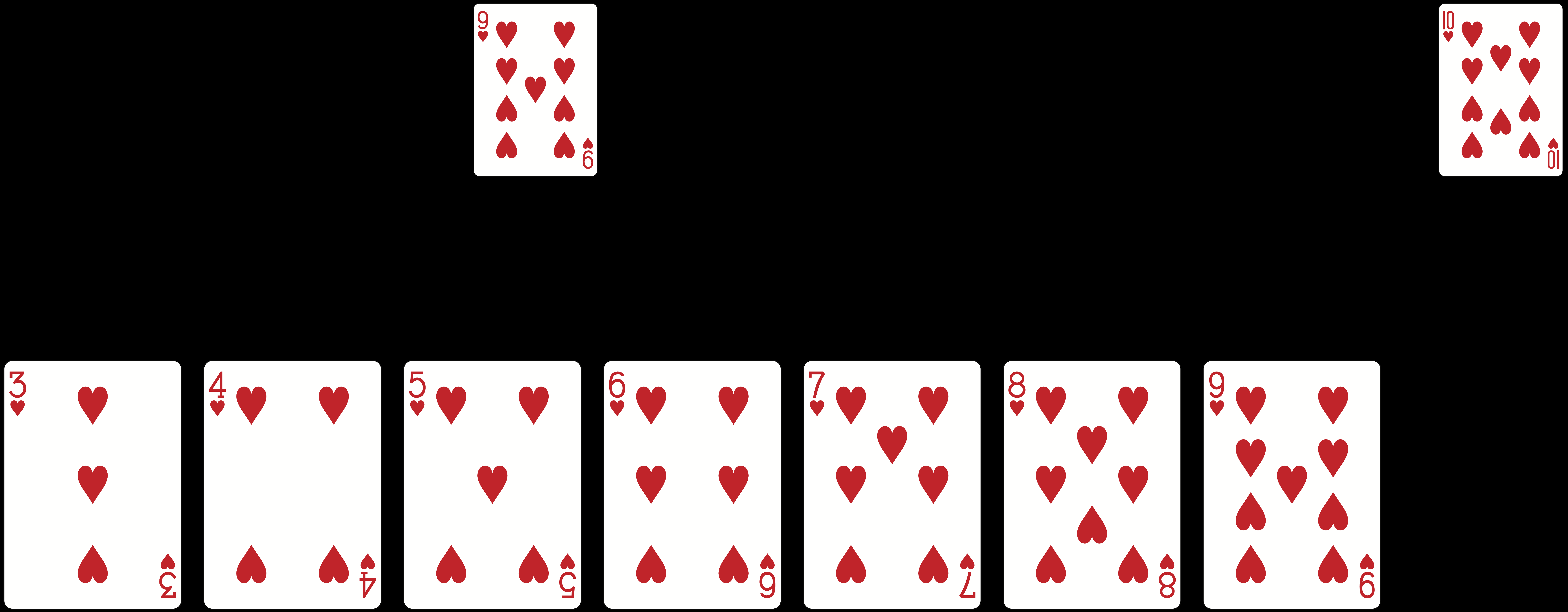
Merge Sort

Final merging (takes O(*n*) time)



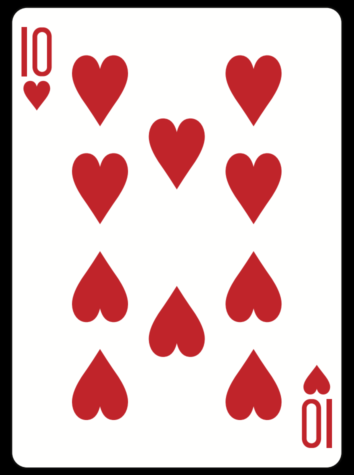
Merge Sort

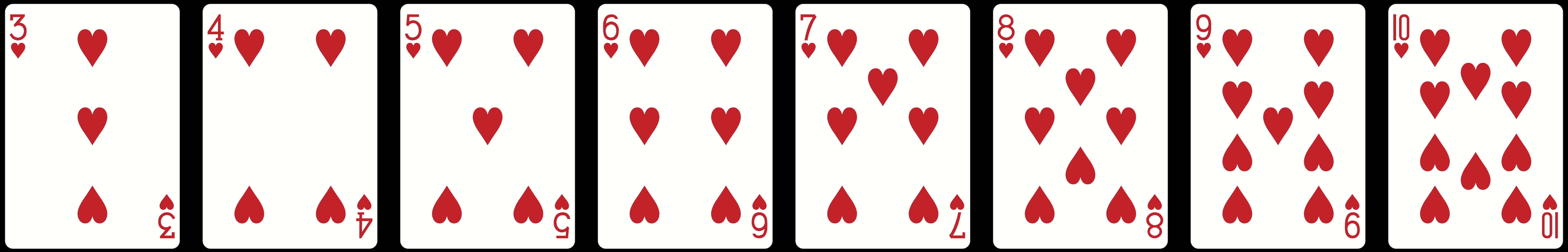
Final merging (takes O(*n*) time)



Merge Sort

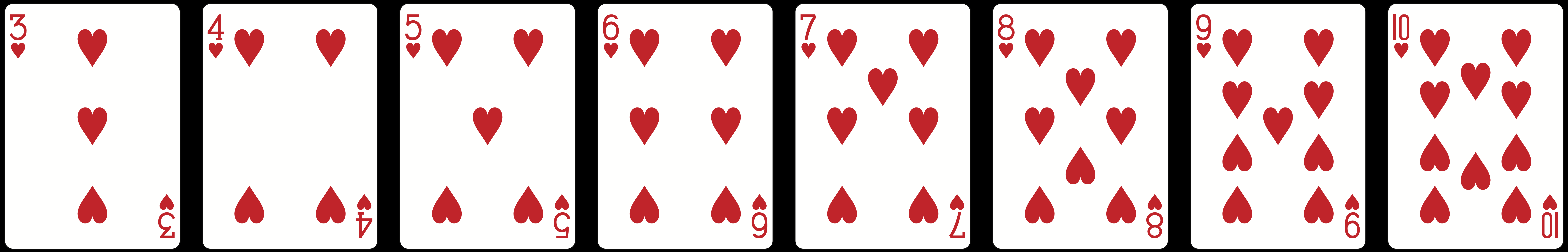
Final merging (takes O(*n*) time)





Merge Sort

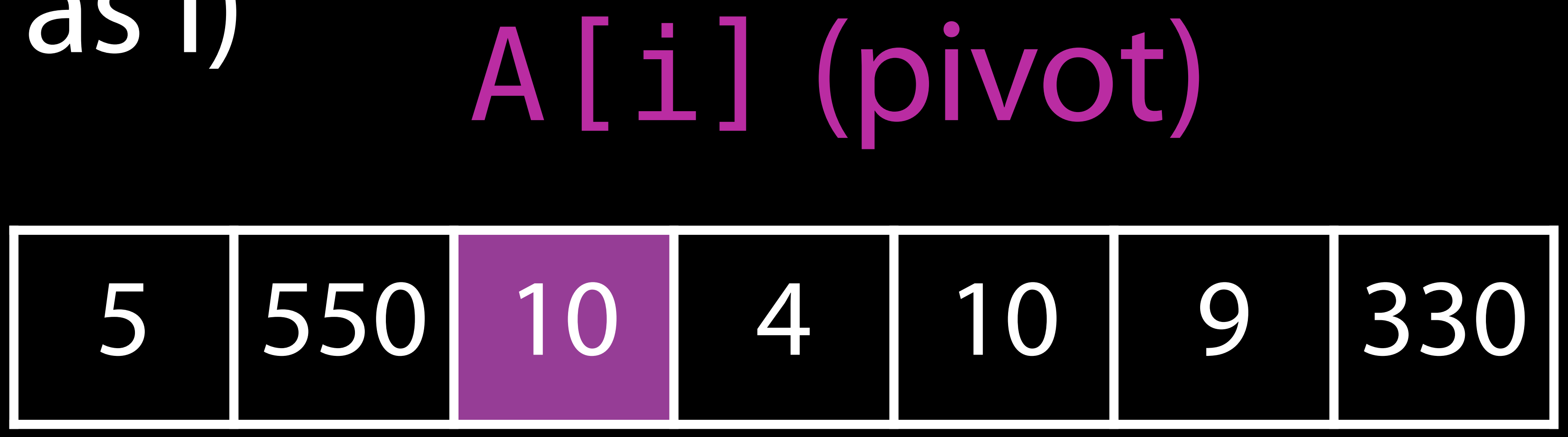
Total O(*n* log *n*) time.



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

Quick sort

Partitioning puts pivot in the correct position.

Keep partitioning until all items become pivots.

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

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.

```
A B D E E C S Q X Y Z
A B D E E C S Q X Y Z
A B D E E C S Q X Y Z
A B D E E C S Q X Y Z
A B D E E C S Q X Y Z
A B D E E C S Q X Y Z
A B C D E E S Q X Y Z
A B C D E E S Q X Y Z
A B C D E E S Q X Y Z
A B C D E E S Q X Y Z
A B C D E E S Q X Y Z
A B C D E E S Q X Y Z
```