Insertion Sort

```
for i = 1 to n - 1:
  insert A[i] in proper place amongst A[0..i]
```

```
INSERTIONSORT(A)
  n = A.length
  for i = 1 to n-1
    temp = A[i]
    j = i - 1
    while j > -1 && A[j] > temp
      A[j+1] = A[j]
      --j
    A[j+1] = temp
```

Loop Invariant

INV. At the start of iteration `i` of the outer (**`for`**) loop, subarray `A[0..i-1]` consists of the elements originally in `A[0..i-1]`, but in sorted order.

Loop Invariant

INV. At the start of iteration `i` of the outer (**`for`**) loop, subarray `A[0..i-1]` consists of the elements originally in `A[0..i-1]`, but in sorted order.

- (Initialization) INV is true at outset.

Loop Invariant

INV. At the start of iteration `i` of the outer (**`for`**) loop, subarray `A[0..i-1]` consists of the elements originally in `A[0..i-1]`, but in sorted order.

- (Initialization) INV is true at outset.

- (Maintenance) Each iteration of **`for`** loop maintains INV through shifting and insertion.

Loop Invariant

INV. At the start of iteration `i` of the outer (**`for`**) loop, subarray `A[0..i-1]` consists of the elements originally in `A[0..i-1]`, but in sorted order.

- (Initialization) INV is true at outset.

- (Maintenance) Each iteration of **`for`** loop maintains INV through shifting and insertion.

- (Termination) INV && (`i = n`)
  ⇒ `A[0..n-1]` — *the whole array* — is sorted.

Merge Sort

```
if n == 1  // there is nothing to sort!
   return

split A down the middle into two subarrays
   Aleft and Aright

recursively sort Aleft
recursively sort Aright

merge Aleft and Aright into a sorted array
```

Input 8 7 3 5 2 1

Input	8	7	3	5	2	1
1. Split array down the middle	8	7	3	5	2	1

Input	8	7	3	5	2	1
1. Split array down the middle	8	7	3	5	2	1
2. Recursively sort left half	3	7	8	5	2	1

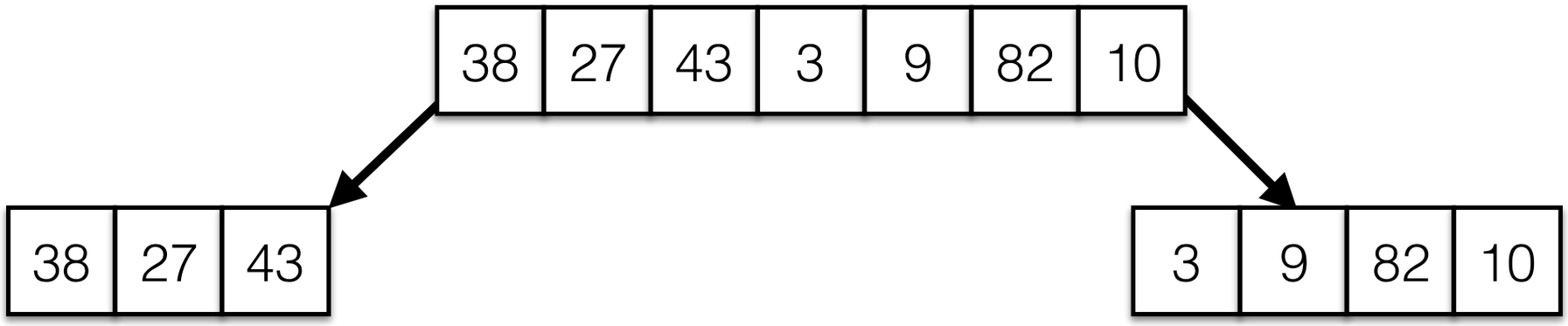
Input	8	7	3	5	2	1
1. Split array down the middle	8	7	3	5	2	1
2. Recursively sort left half	3	7	8	5	2	1
3. Recursively sort right half	3	7	8	1	2	5

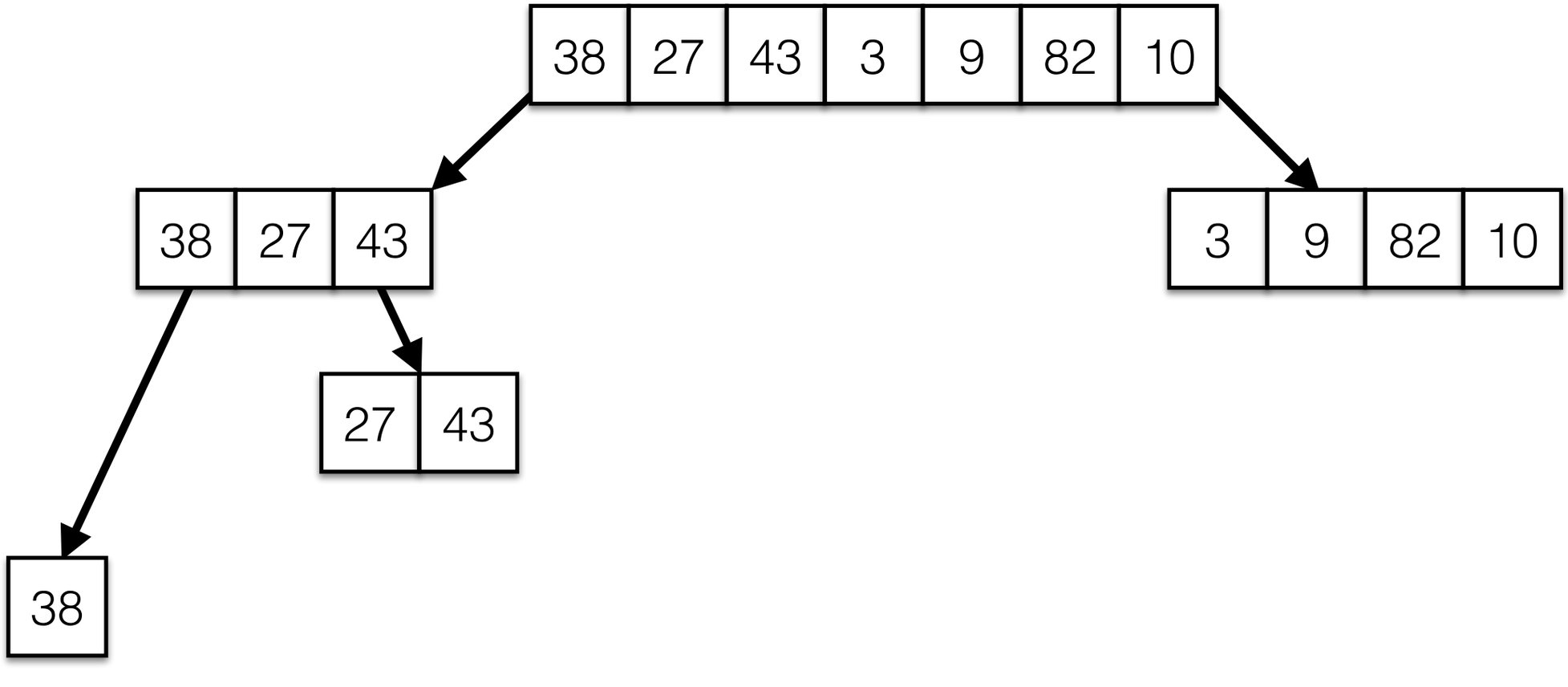
Input	8	7	3	5	2	1
1. Split array down the middle	8	7	3	5	2	1
2. Recursively sort left half	3	7	8	5	2	1
3. Recursively sort right half	3	7	8	1	2	5
4. Merge two halves to make a sorted whole	1	2	3	5	7	8

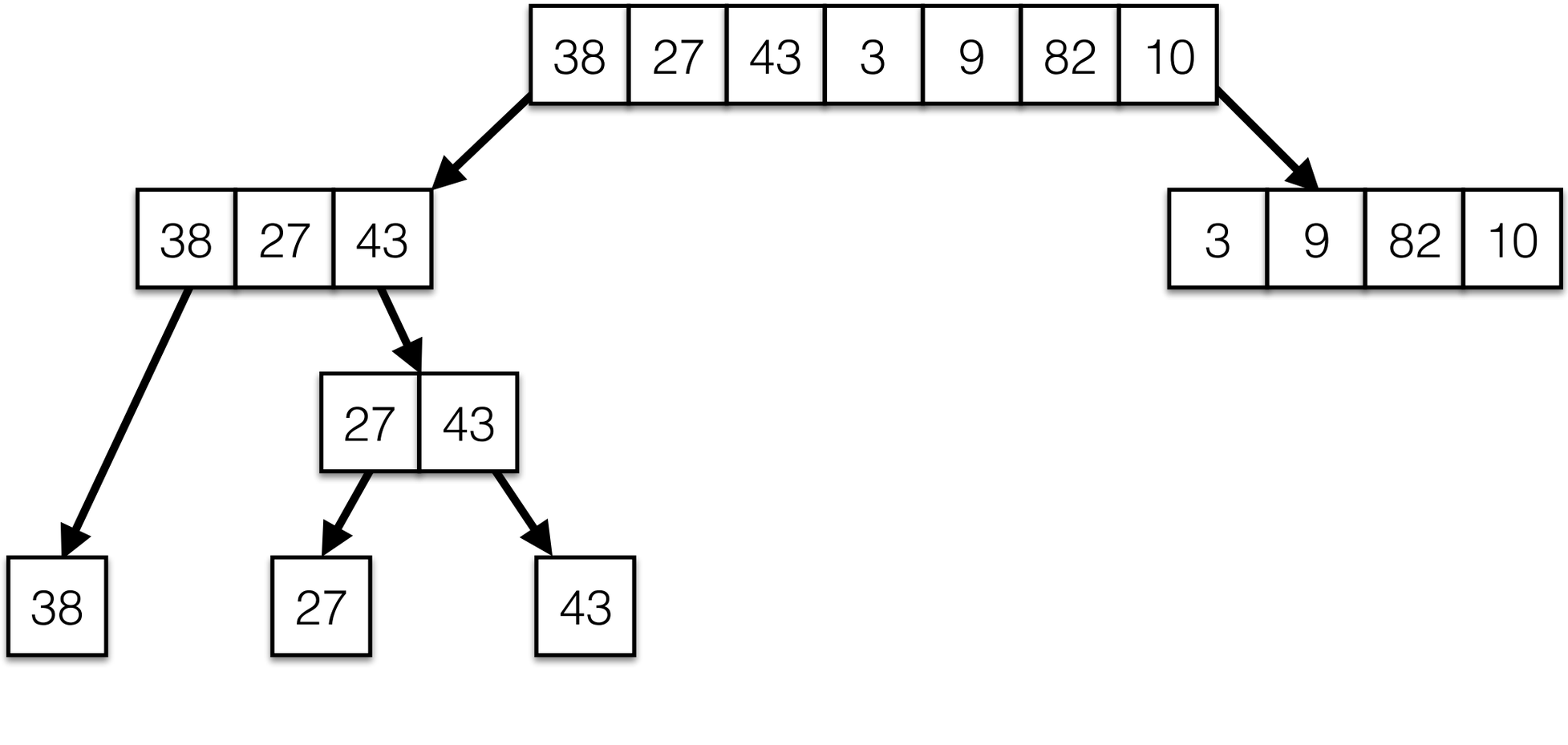
```
MergeSort(A)
  n = A.length
  if n ≤ 1 return
  m = n/2
  A_left = (A[0], . . . , A[m−1])
  A_right = (A[m], . . . , A[n−1])
  MergeSort(A_left)
  MergeSort(A_right)
  A = Merge(A_left, A_right)
```

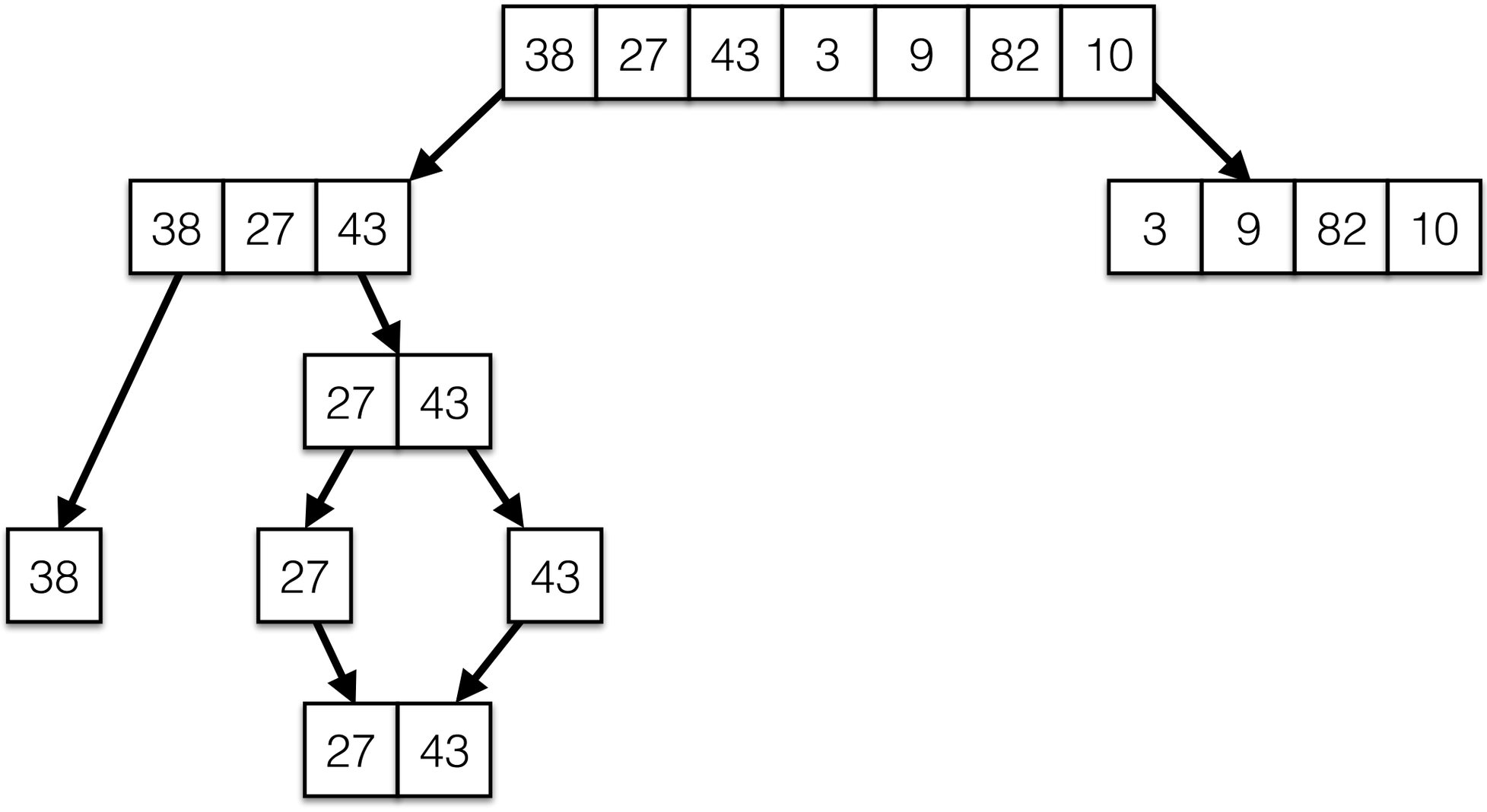
```
MERGE(B,C)
  p = B.length, q = C.length
  create an empty array D of length p+q
  i=0, j=0
  while i < p && j < q
    if B[i] ≤ C[j]
      append B[i] to D
      i++
    else
      append C[j] to D
      j++
if i ≥ p
  append C[j],...,C[q-1] to D
else
  append B[i],...,B[p-1] to D
return D
```

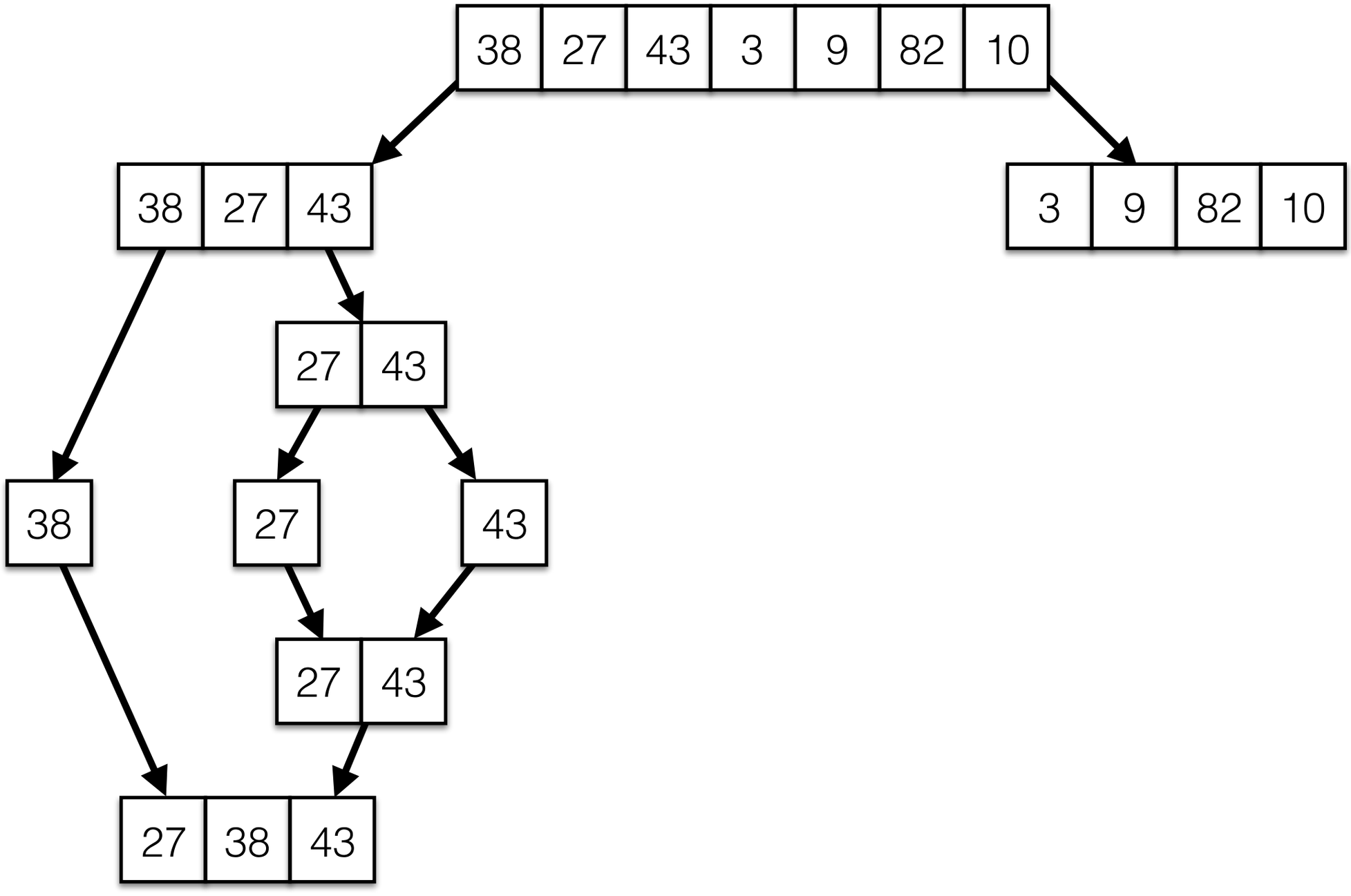
38	27	43	3	9	82	10

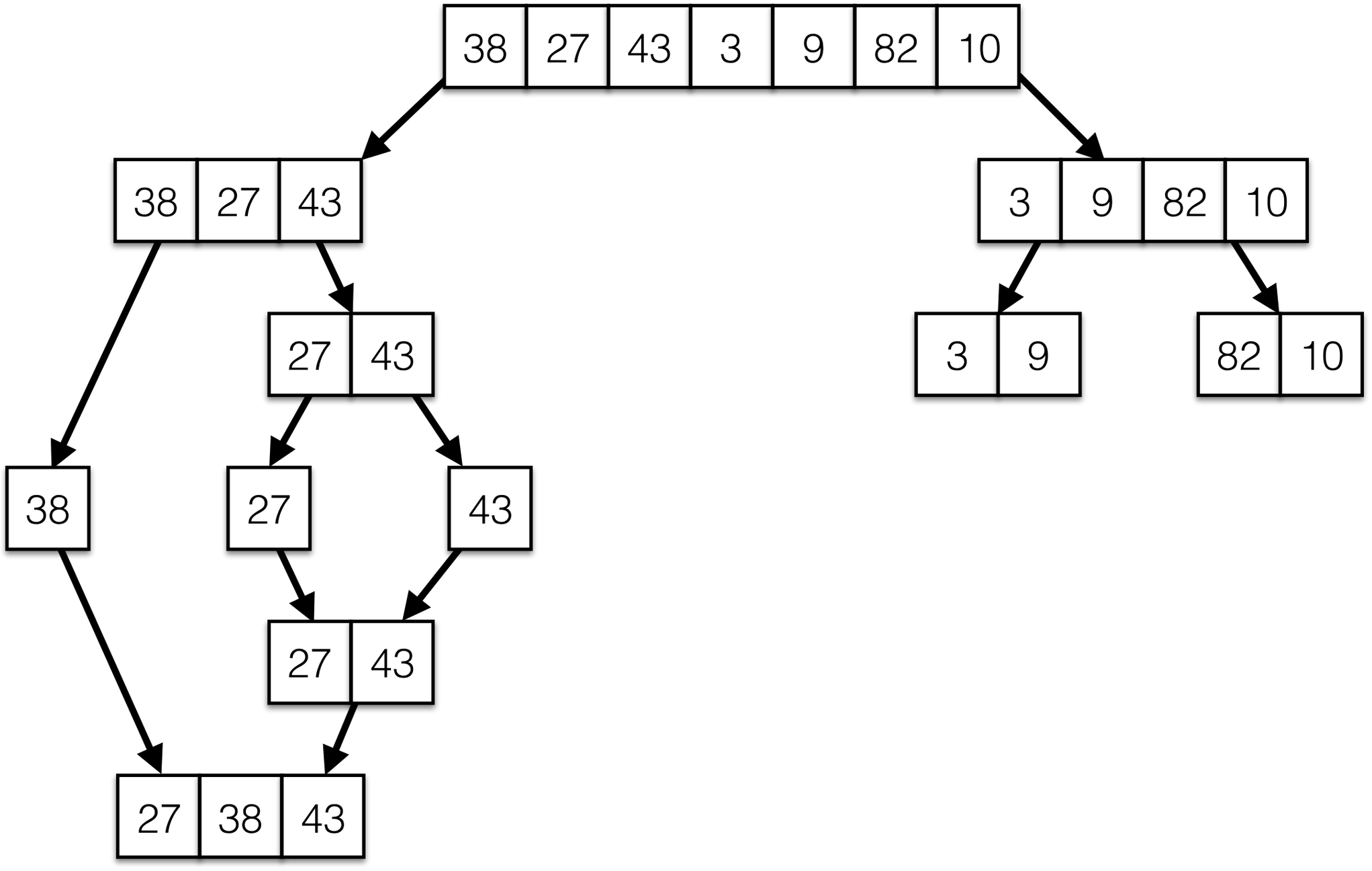

38
27
43
3
9
82
10
38
27
43
3
9
82
10

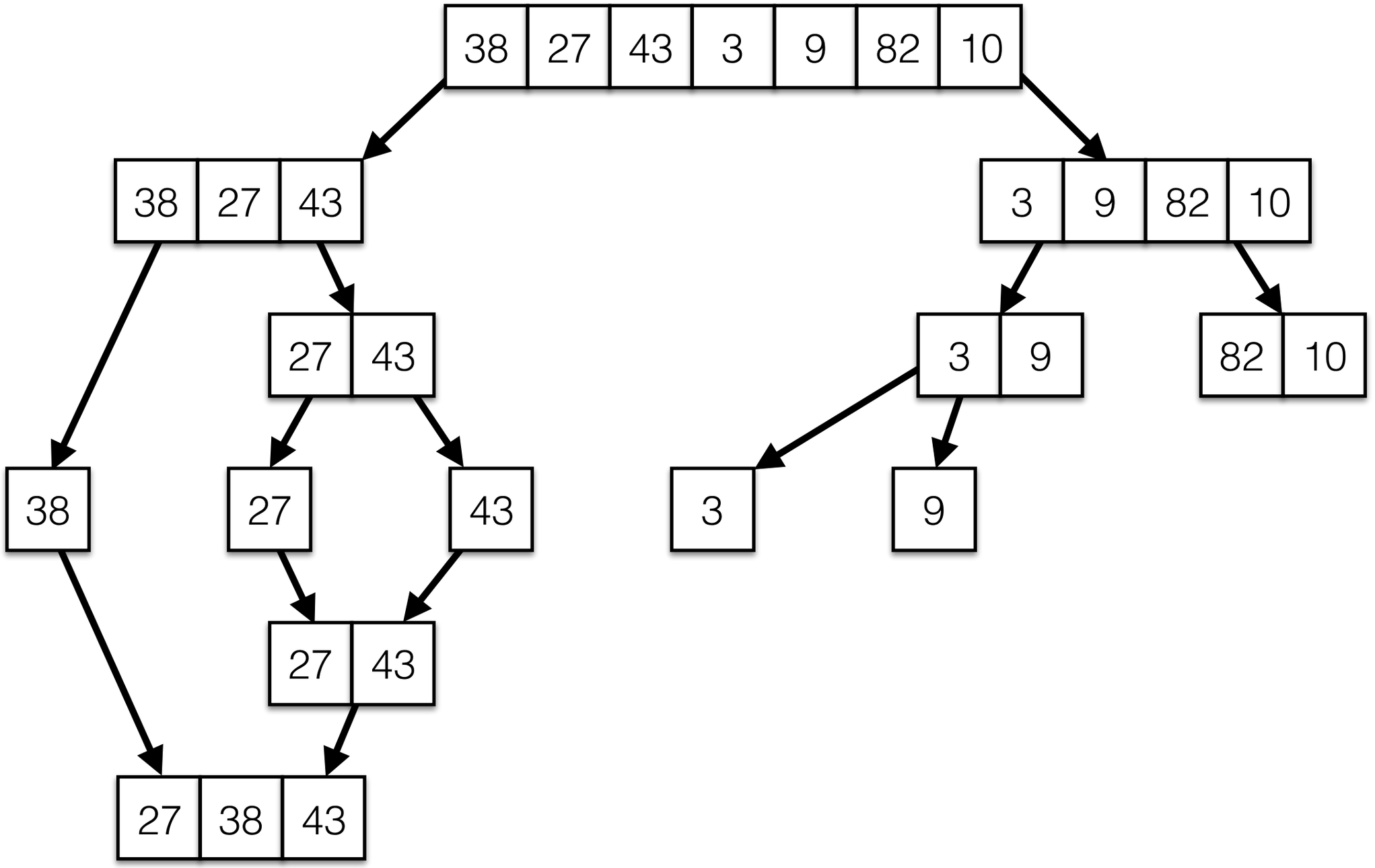

38
27
43
3
9
82
10
38
27
43
3
9
82
10
27
43
38

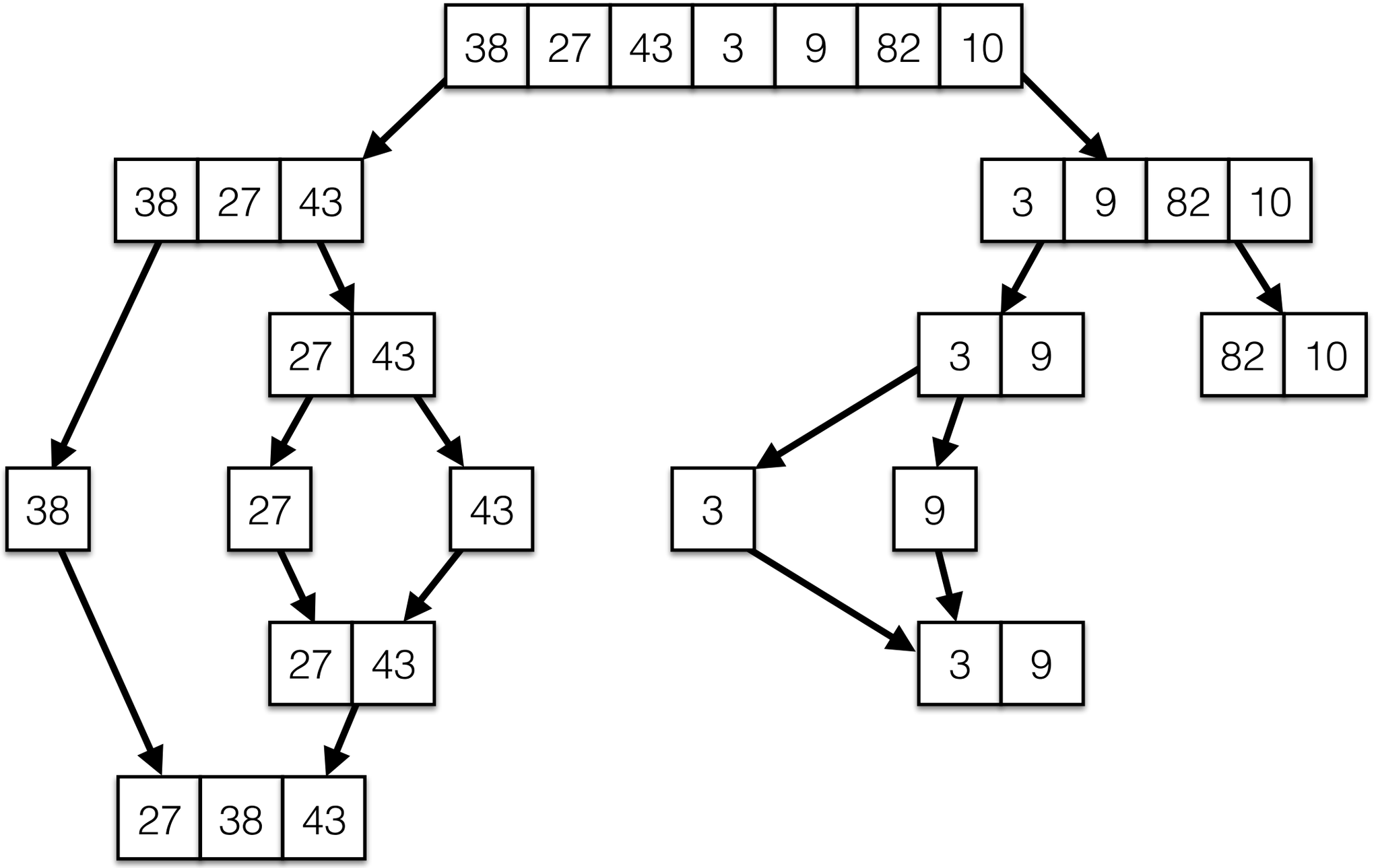

38
27
43
3
9
82
10
38
27
43
3
9
82
10
27
43
38
27
43

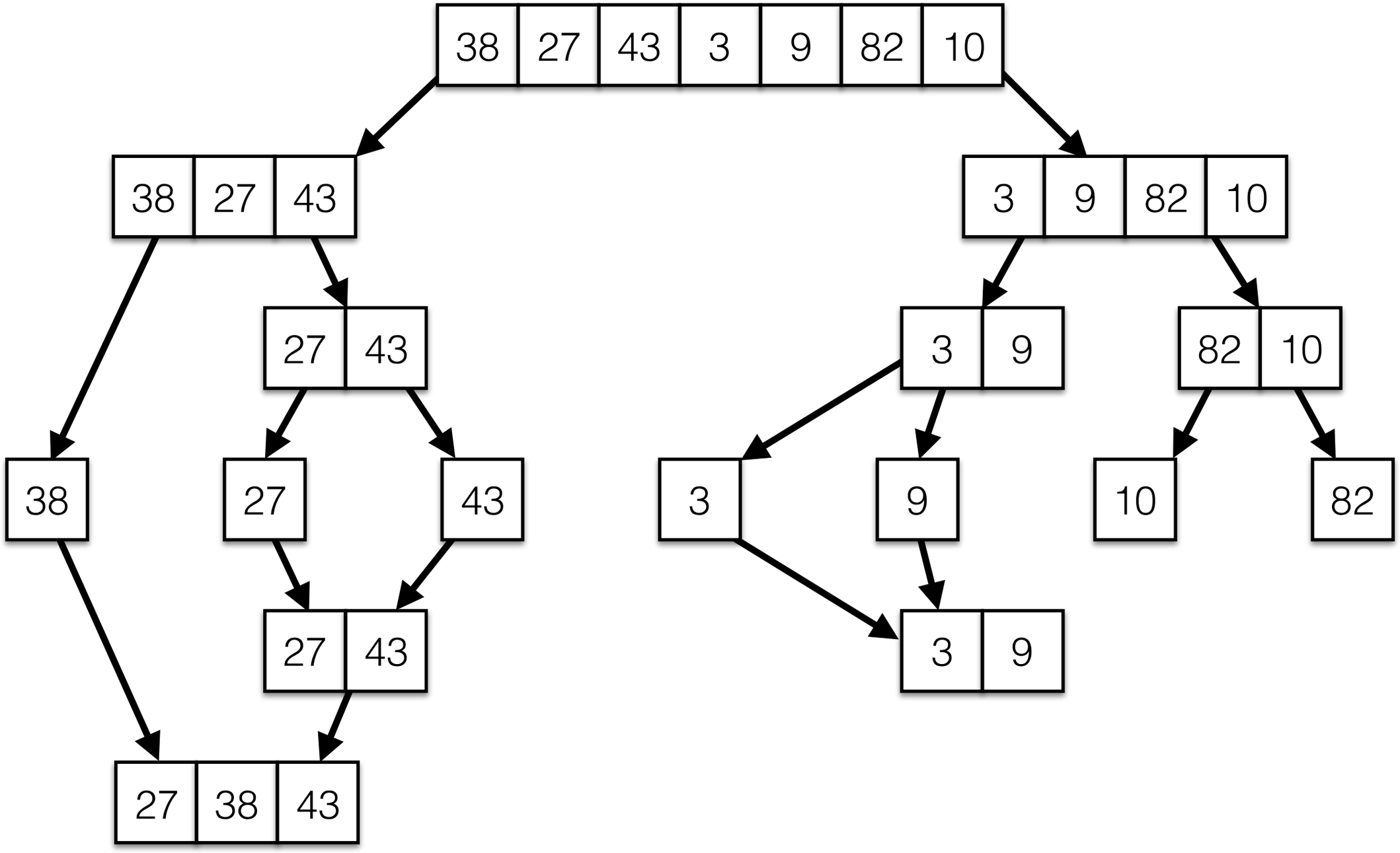

38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
38
27
43
27 43

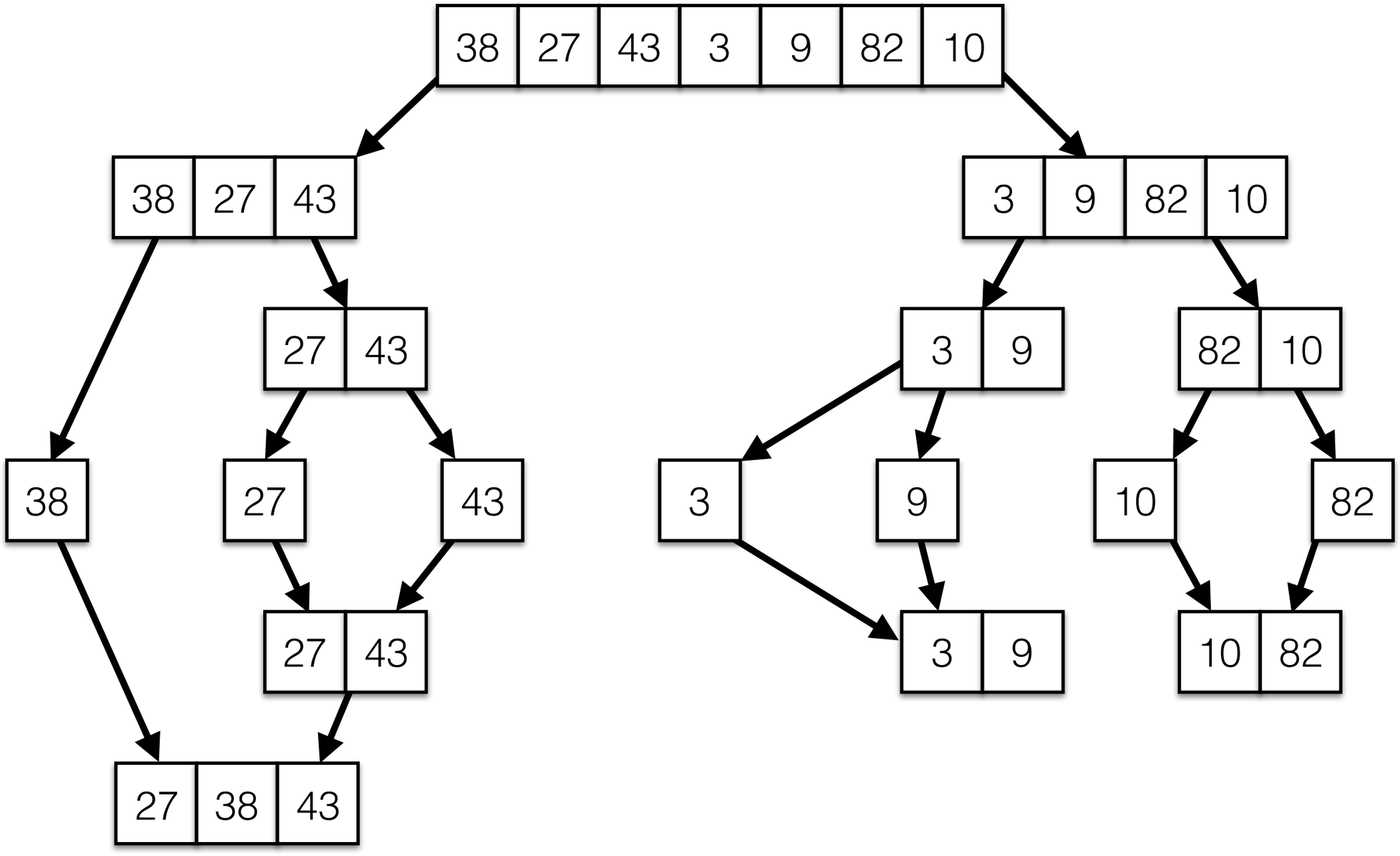

38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
38
27
43
27 43
27 38 43

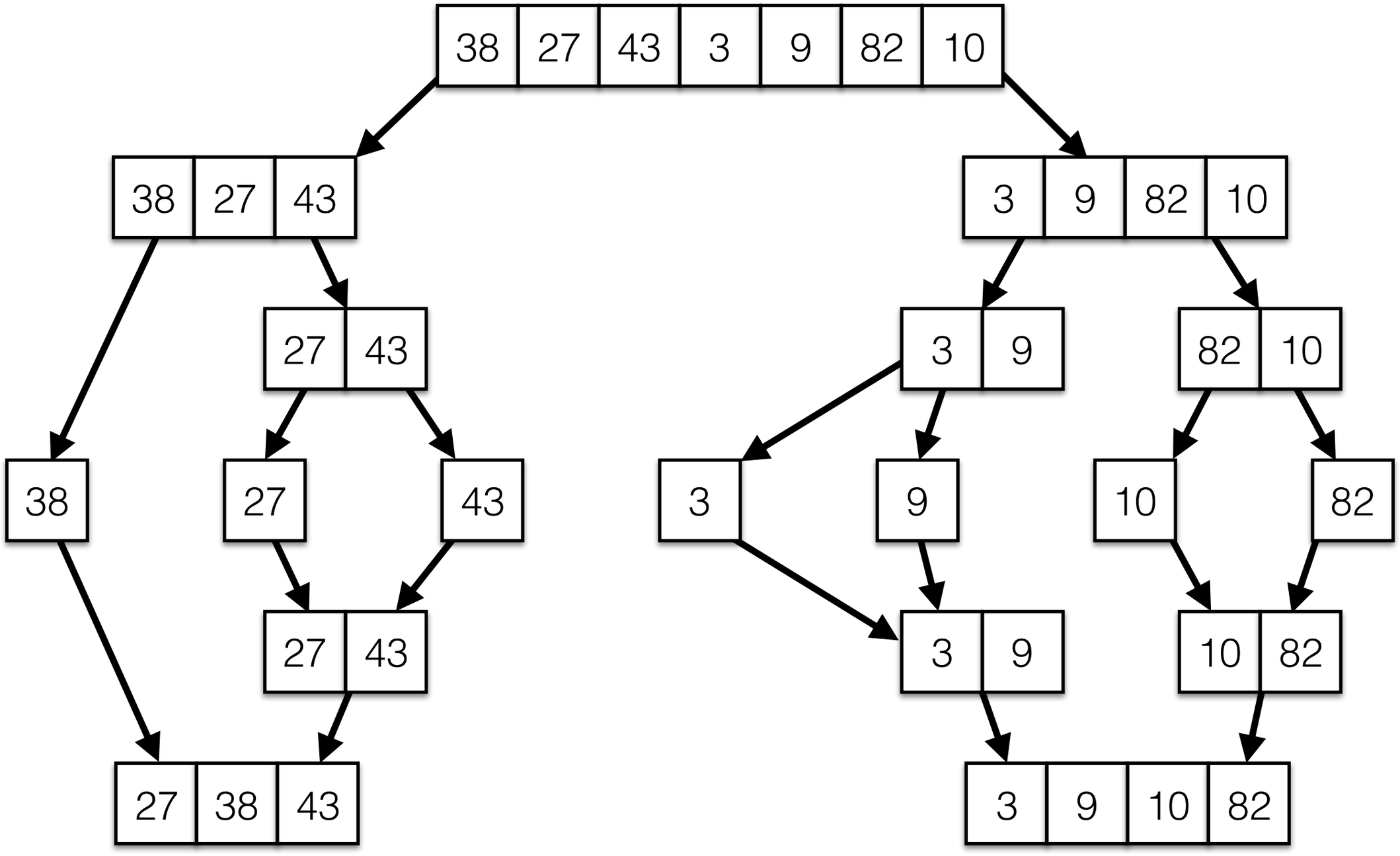

38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
3 9
82 10
38
27
43
27 43
27 38 43

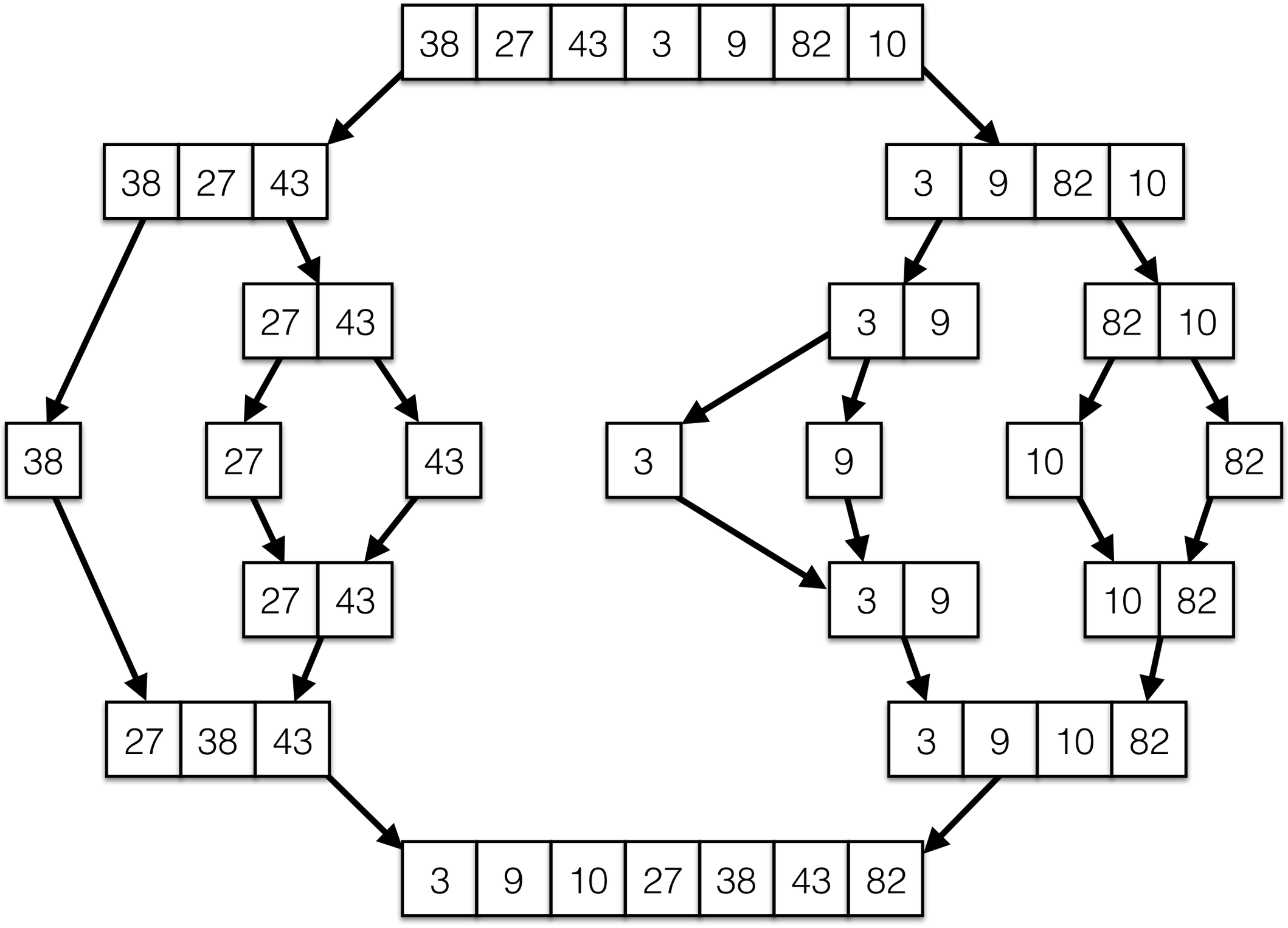

38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
3 9
82 10
38
27
43
3
9
27 43
27 38 43


38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
3 9
82 10
38
27
43
3
9
27 43
3 9
27 38 43


38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
3 9
82 10
38
27
43
3
9
10
82
27 43
3 9
27 38 43


38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
3 9
82 10
38
27
43
3
9
10
82
27 43
3 9
10 82
27 38 43


38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
3 9
82 10
38
27
43
3
9
10
82
27 43
3 9
10 82
27 38 43
3 9 10 82


38 27 43 3 9 82 10
38 27 43
3 9 82 10
27 43
3 9
82 10
38
27
43
3
9
10
82
27 43
3 9
10 82
27 38 43
3 9 10 82
3 9 10 27 38 43 82