COMP 250

Lecture 13

mergesort, quicksort

Oct. 6, 2017

Time complexity

$O(log_2 n)$	$O(n)$	$O(n^2)$
• convert to binary	• List operations: findMax, remove	• insertion/selection/ bubble sort
• binary search	• grade school addition or subtraction	• grade school multiplication
• ……	• …..	• ……

Computers perform $\sim 10^9$ operations per second.

$$2^{10} \approx 10^3$$

$$2^{20} \approx 10^6$$

$$2^{30} \approx 10^9$$

Computers perform $\sim 10^9$ operations per second.

$log_2 n$	n	n^2
10	$2^{10} \approx 10^3$	10^6
20	$2^{20} \approx 10^6$	10^{12}
30	$2^{30} \approx 10^9$	10^{18}

minutes…hours

second

centuries

Better sorting algorithms ?

$$O(n) \quad < \quad ? \quad < \quad O(\, n^2 \,)$$

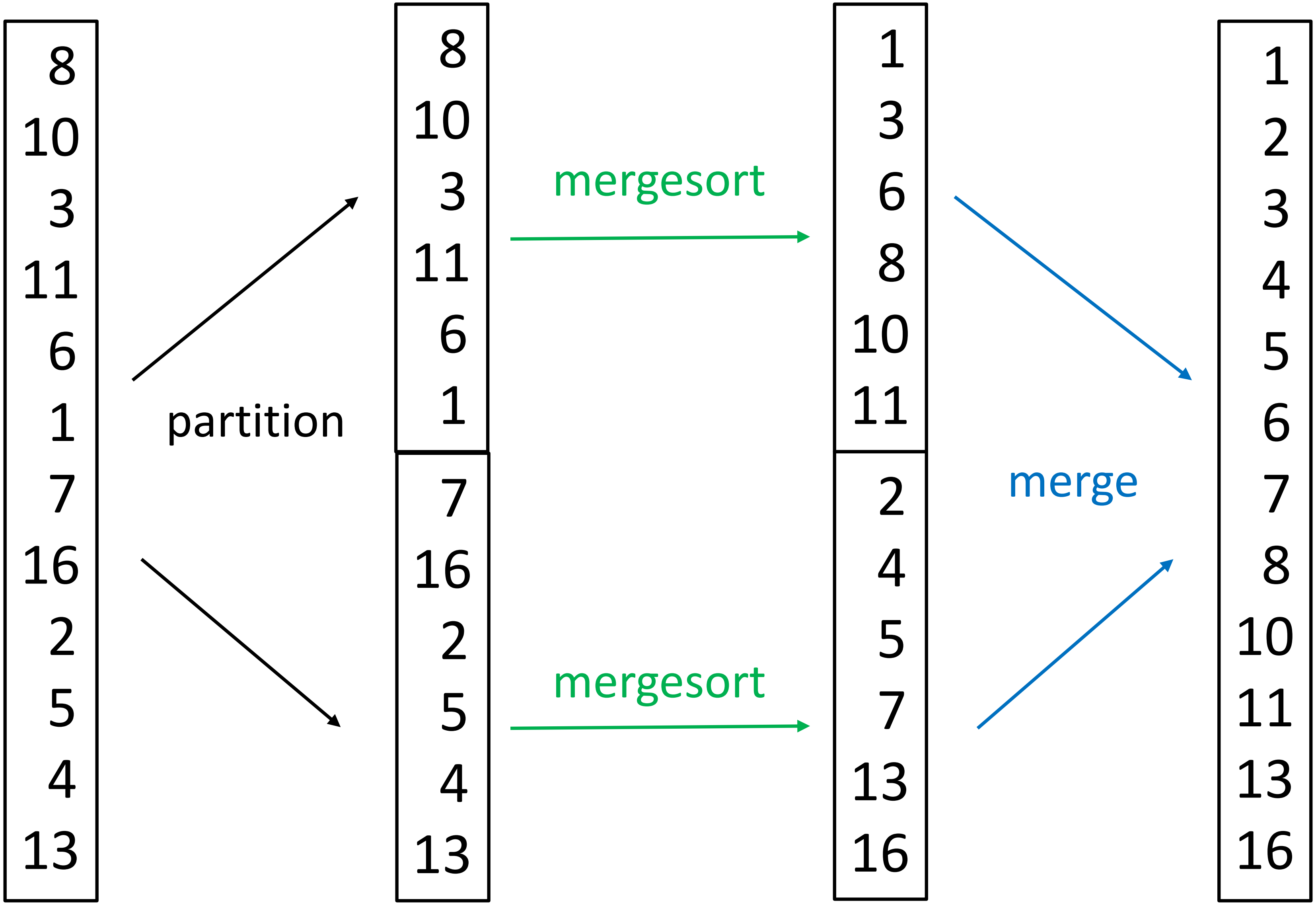
Mergesort

Given a list, partition it into two halves (1st & 2nd).

Sort each half (recursively).

Merge the two halves.

This turns out to be much faster than the other list sorting algorithms we have seen.



8 10 3 11 6 1 7 16 2 5 4 13

partition

8 10 3 11 6 1

7 16 2 5 4 13

mergesort

mergesort

1 3 6 8 10 11

2 4 5 7 13 16

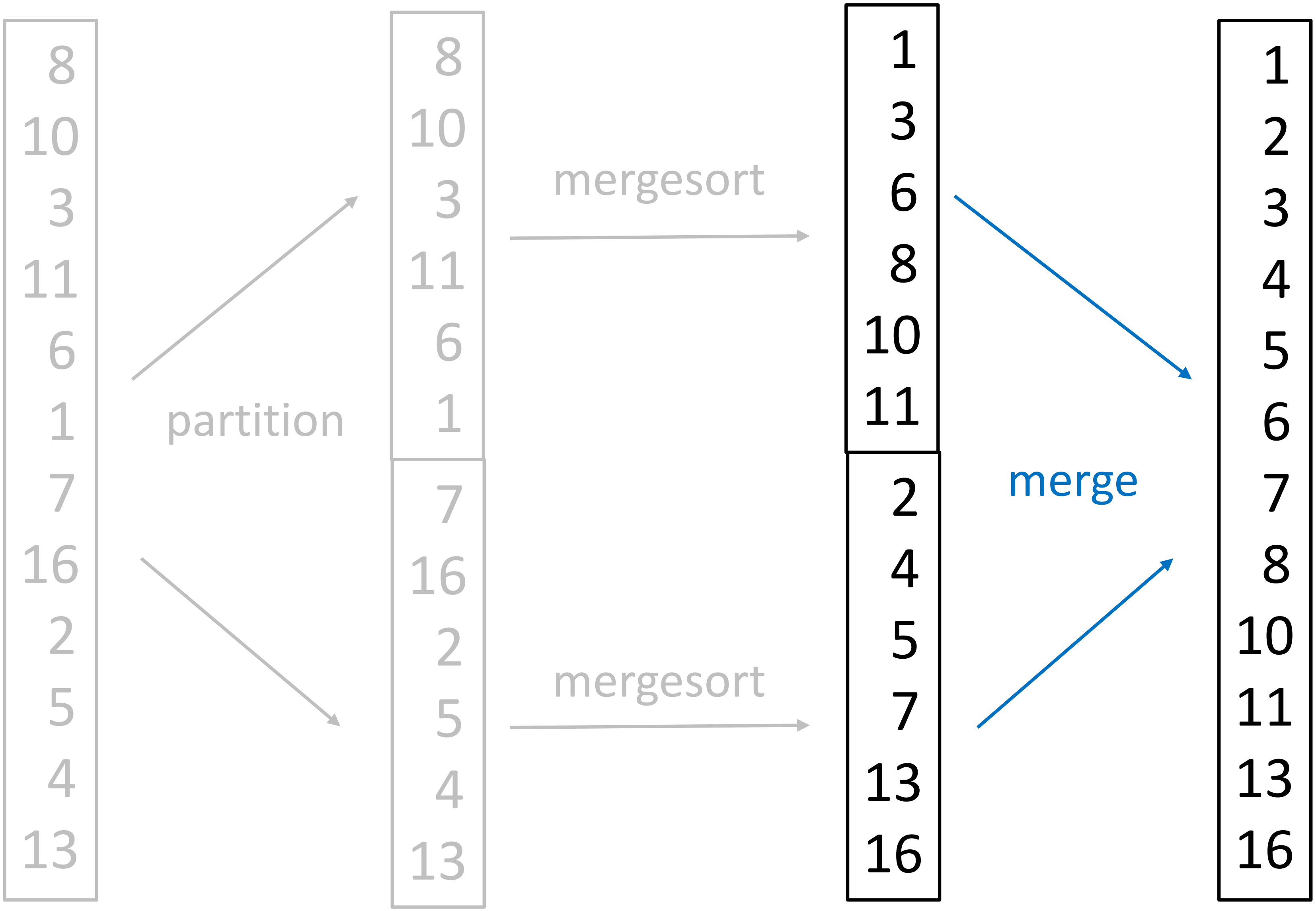
merge

1 2 3 4 5 6 7 8 10 11 13 16

```
mergesort(list){
    if  list.length == 1
        return list
    else{
        mid = (list.size - 1) / 2
        list1 =  list.getElements(0,mid)
        list2 =  list.getElements(mid+1, list.size-1)
        list1  =  mergesort(list1)
        list2  =  mergesort(list2)
        return   merge( list1, list2 )
    }
  }
```

```
mergesort(list){
    if  list.length == 1
        return list
    else{
        mid = (list.size - 1) / 2
        list1 =  list.getElements(0,mid)
        list2 =  list.getElements(mid+1, list.size-1)
        list1  =  mergesort(list1)
        list2  =  mergesort(list2)
        return   merge( list1, list2 )
    }
  }
```

```
mergesort(list){
    if  list.length == 1
        return list
    else{
        mid = (list.size - 1) / 2
        list1 =  list.getElements(0,mid)
        list2 =  list.getElements(mid+1, list.size-1)
        list1  =  mergesort(list1)
        list2  =  mergesort(list2)
        return   merge( list1, list2 )
    }
 }
```



8 10 3 11 6 1 7 16 2 5 4 13

partition

8 10 3 11 6 1

7 16 2 5 4 13

mergesort

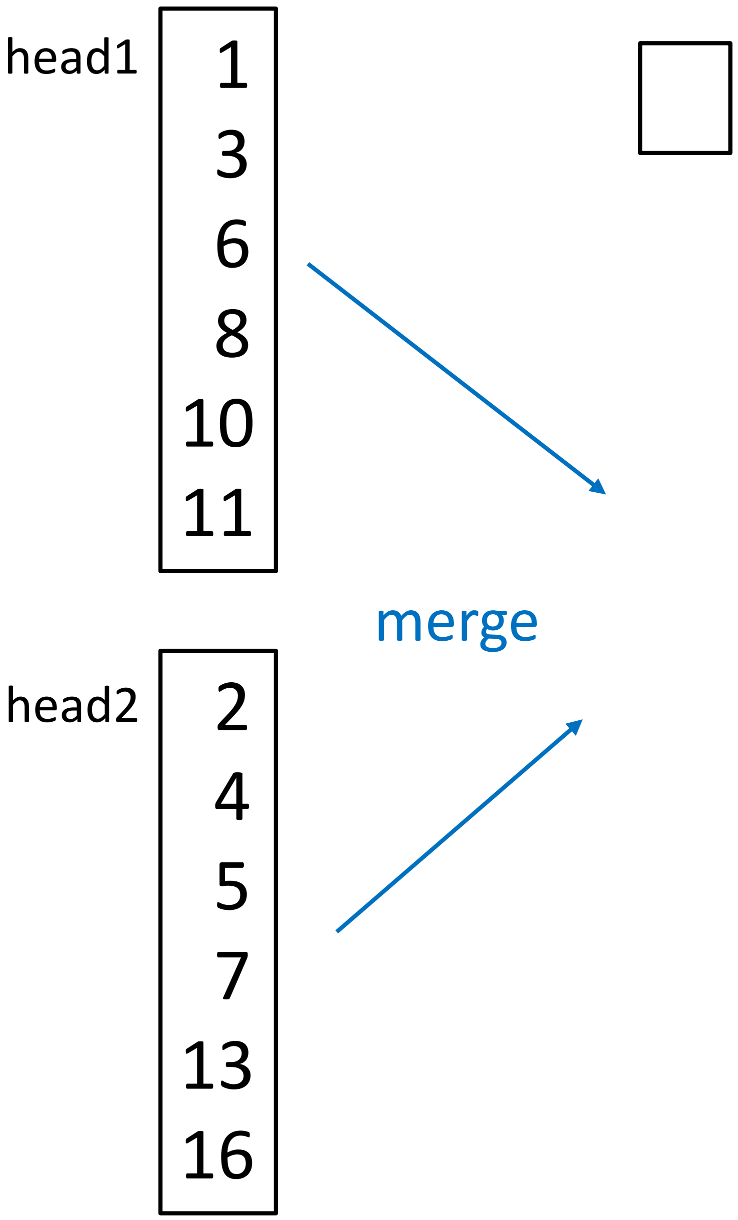
mergesort

1 3 6 8 10 11

2 4 5 7 13 16

merge

1 2 3 4 5 6 7 8 10 11 13 16



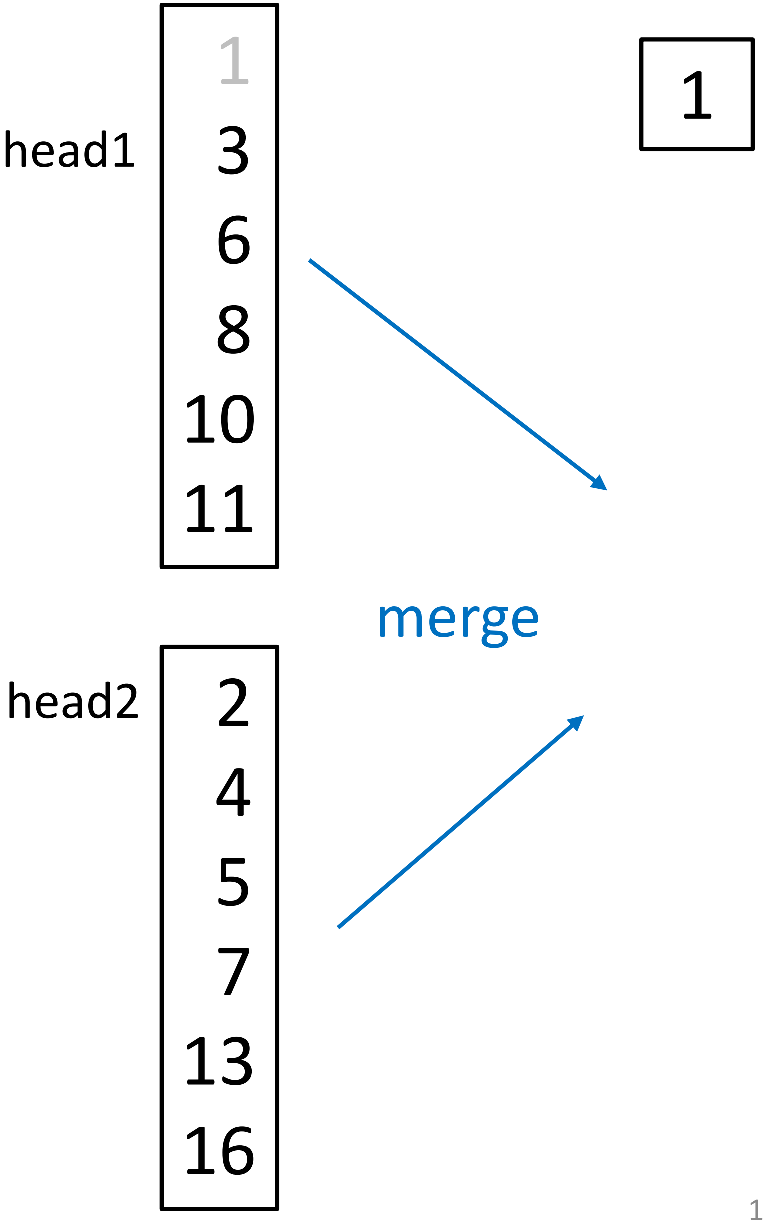
head1

1
3
6
8
10
11

merge

head2

2
4
5
7
13
16



head1

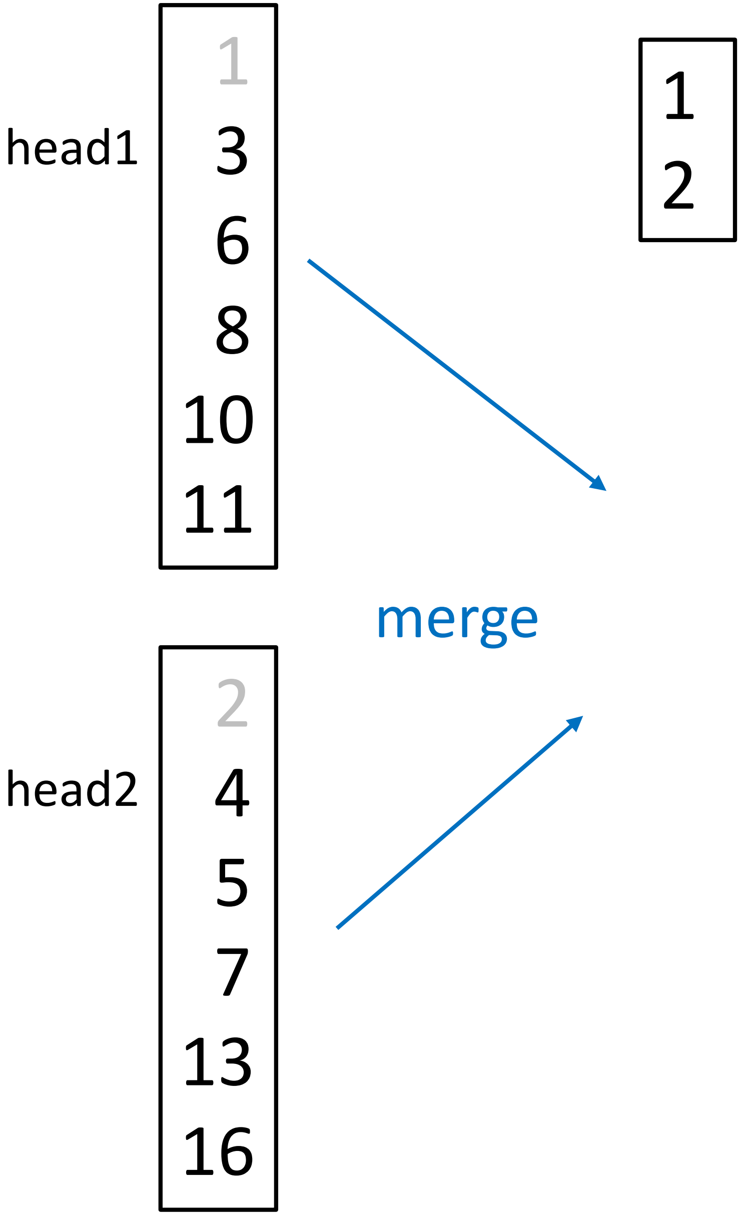
1
3
6
8
10
11

head2

2
4
5
7
13
16

merge

1



head1

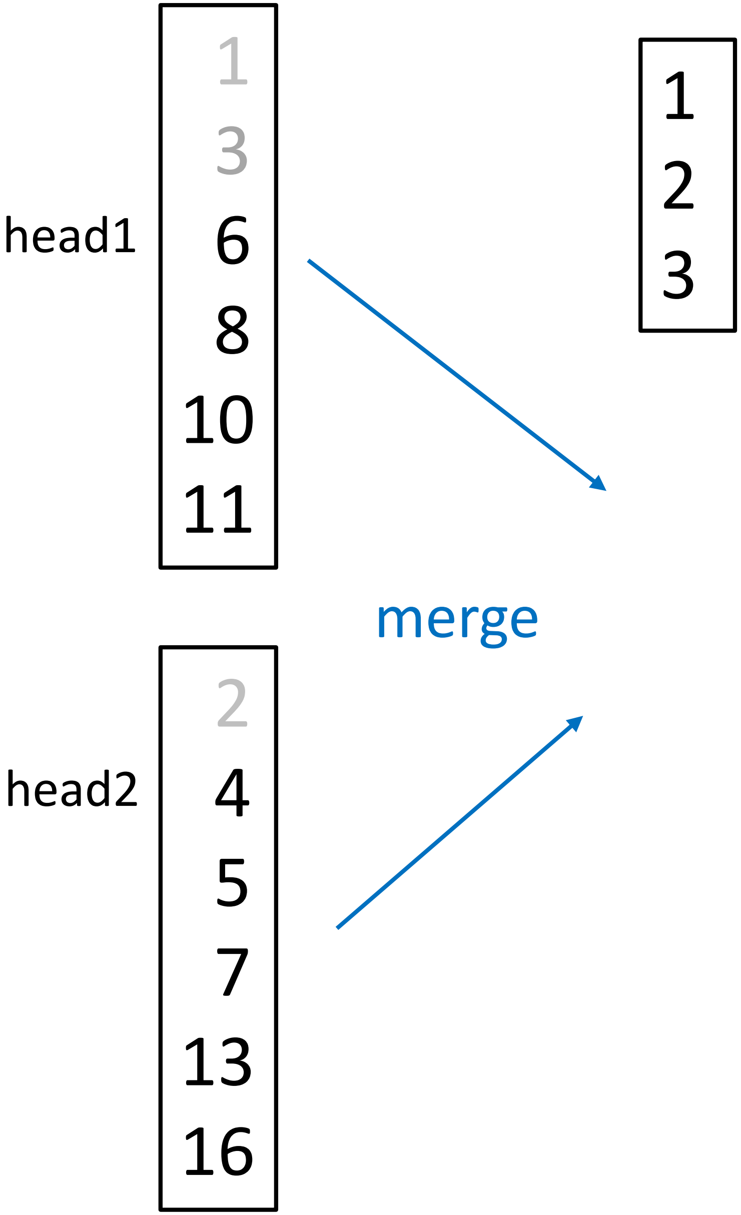
1
3
6
8
10
11

head2

2
4
5
7
13
16

merge

1
2



head1

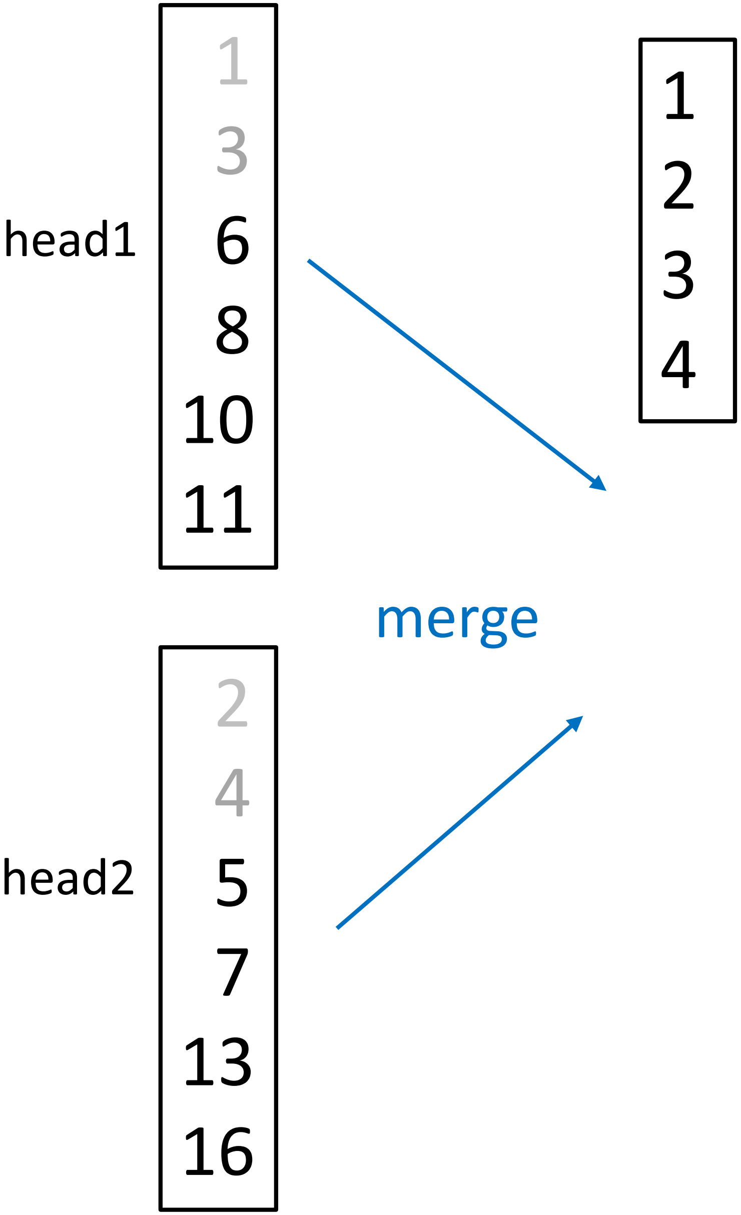
1
3
6
8
10
11

head2

2
4
5
7
13
16

merge

1
2
3



head1

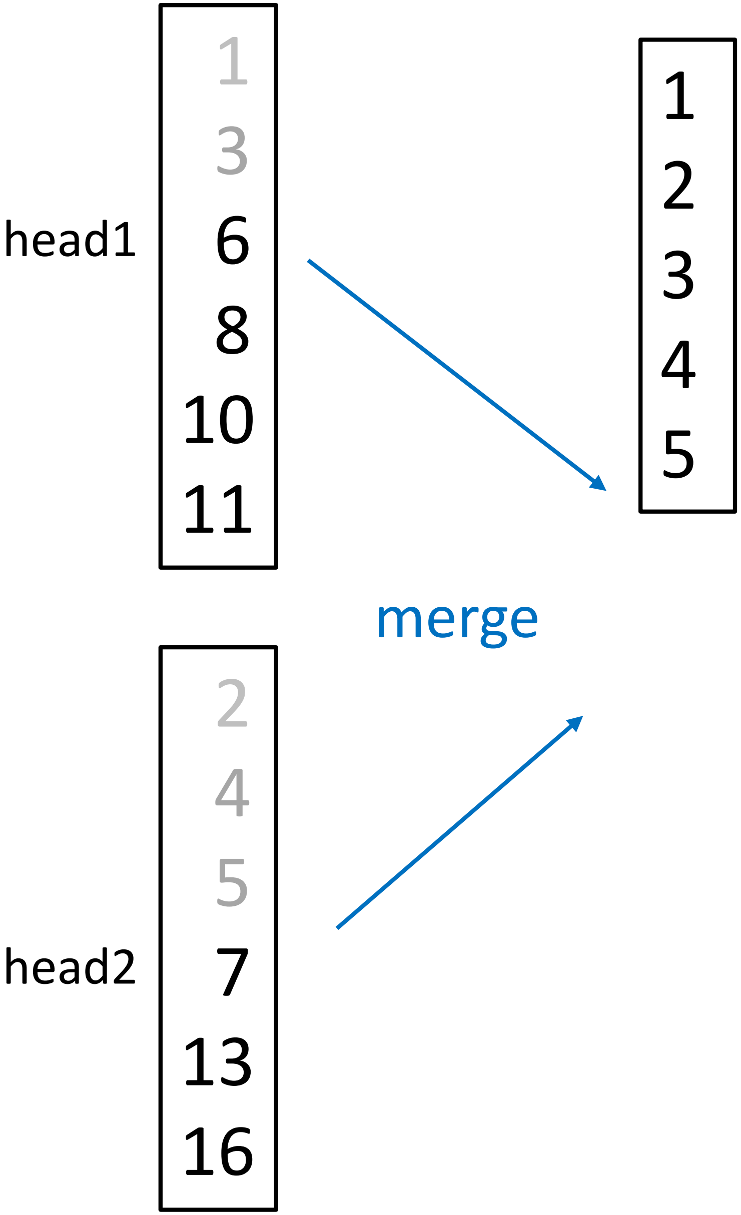
1
3
6
8
10
11

head2

2
4
5
7
13
16

merge

1
2
3
4



head1

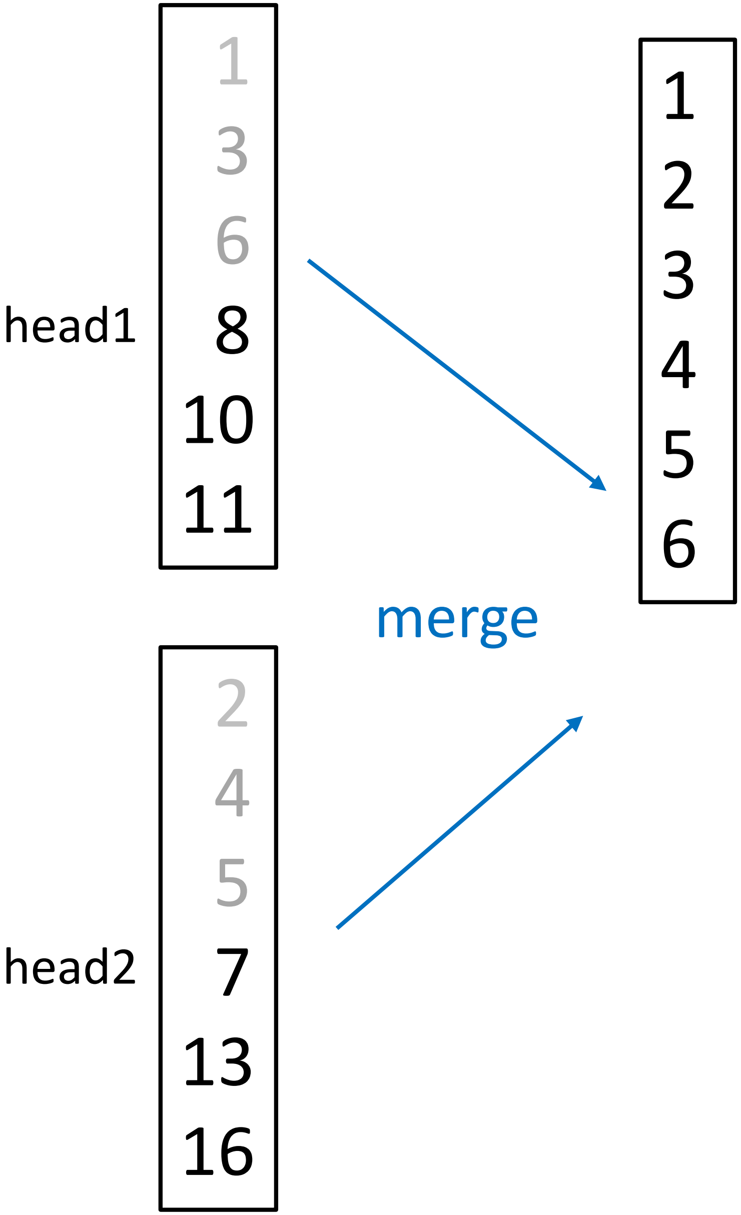
1
3
6
8
10
11

head2

2
4
5
7
13
16

merge

1
2
3
4
5

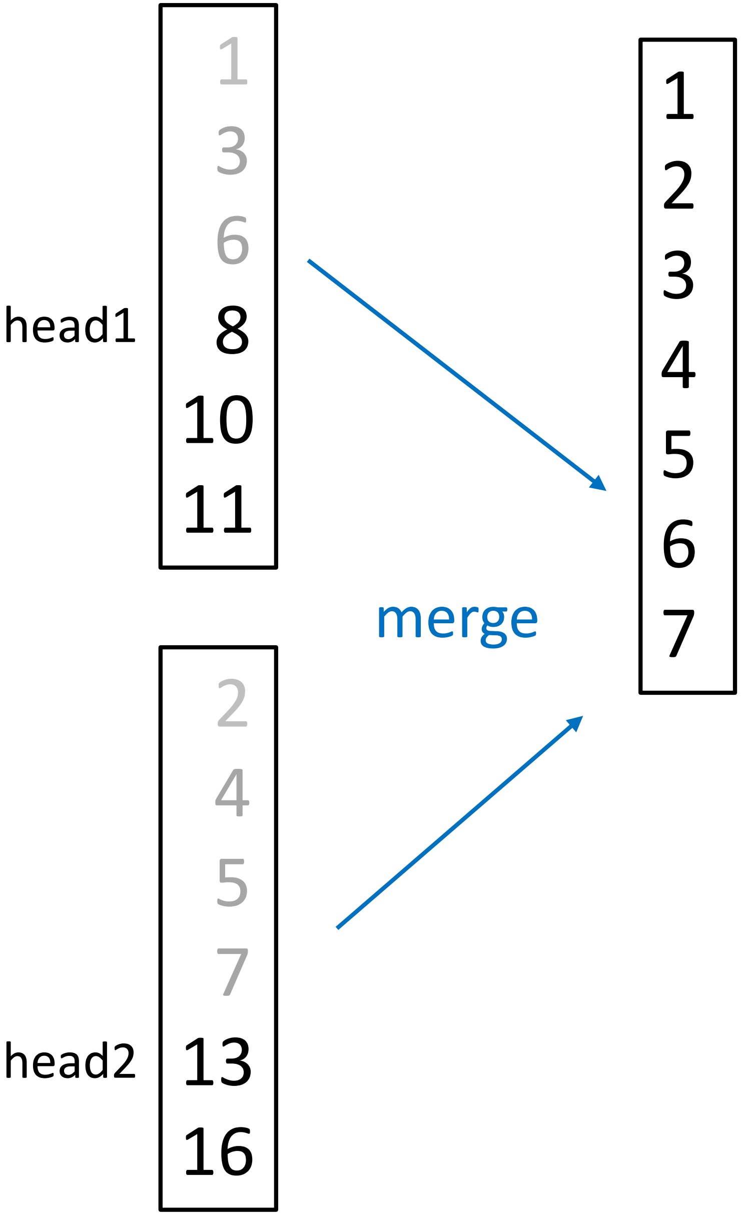


1
3
6
head1 8
10
11

2
4
5
head2 7
13
16

merge

1
2
3
4
5
6



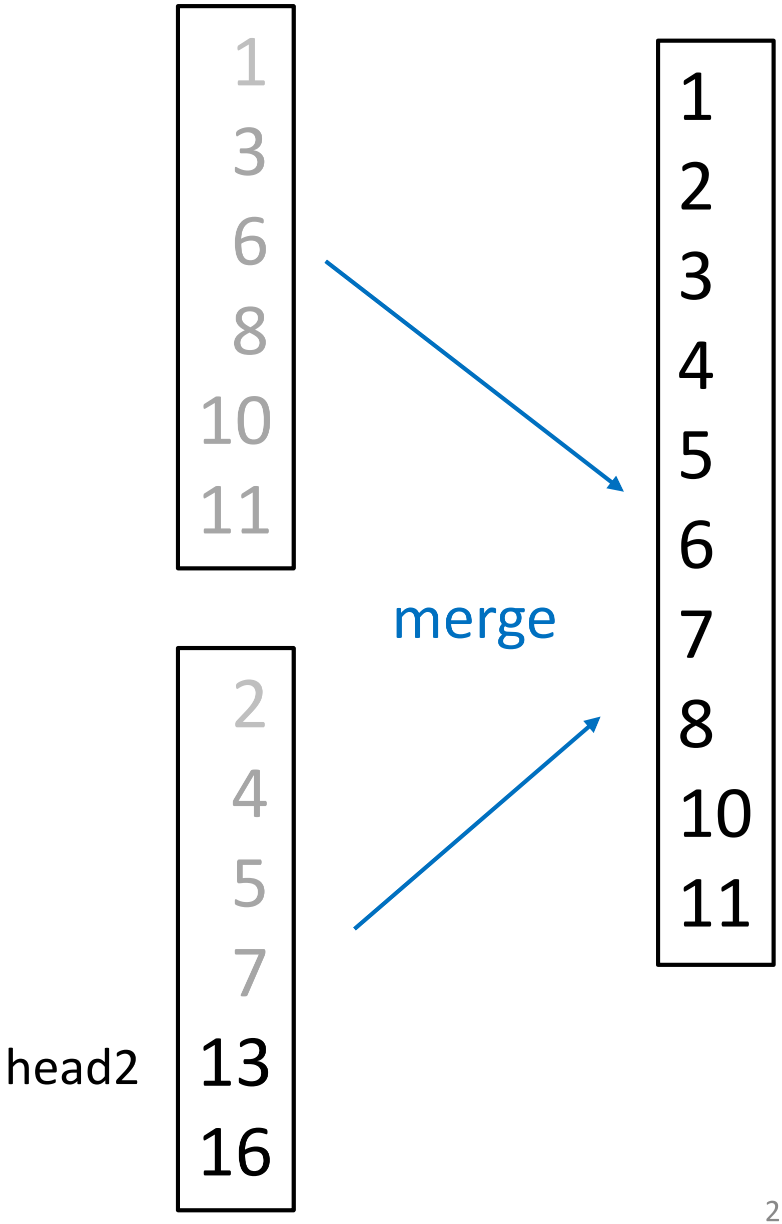
1
3
6
head1 8
10
11

merge

2
4
5
7
head2 13
16

1
2
3
4
5
6
7

....and so on until one list is empty.



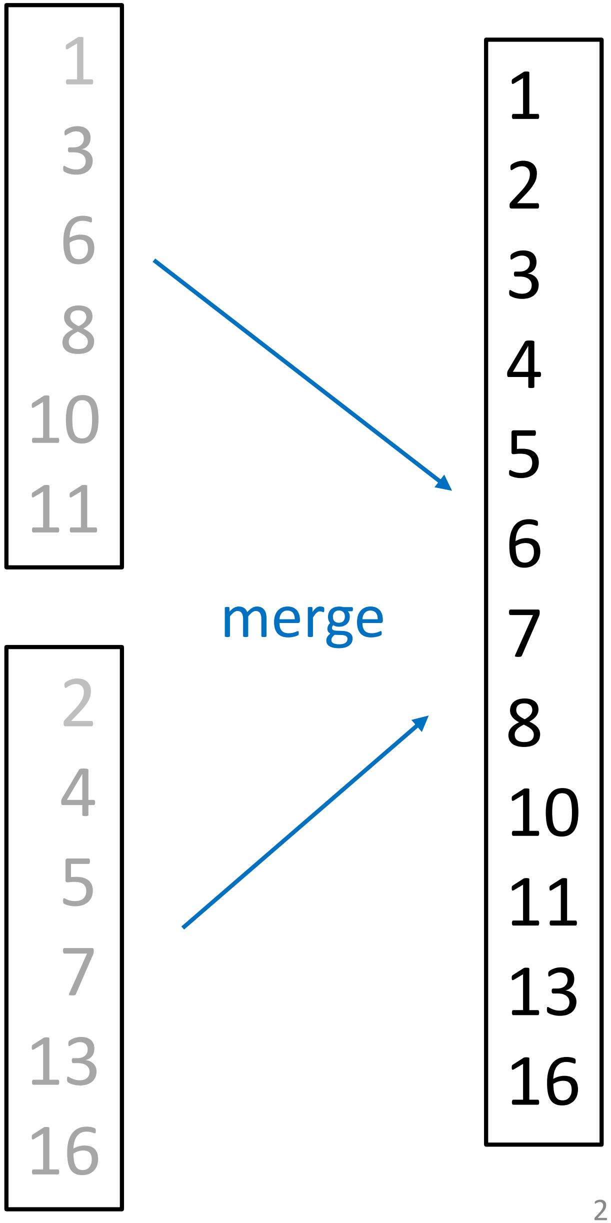
1
3
6
8
10
11

2
4
5
7
head2 13
16

merge

1
2
3
4
5
6
7
8
10
11

Then, copy the remaining elements.



1 3 6 8 10 11

2 4 5 7 13 16

merge

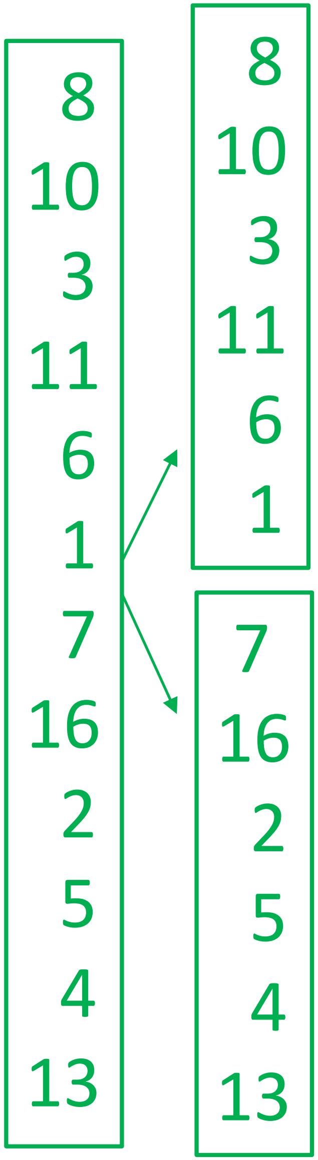
1 2 3 4 5 6 7 8 10 11 13 16

```
merge( list1,  list2){
  initialize list to be empty
  while  (list1 is not empty) & (list2 is not empty){
    if  (list1.first < list2.first)
        list.addlast( list1.removeFirst(list1) )
    else
        list.addlast( list2.removeFirst(list2) )
  }
  while list1 is not empty
    list.addlast( list1.removeFirst(list1) )
  while list2 is not empty
    list.addlast( list2.removeFirst(list2) )

  return  list
}
```

```
merge( list1,  list2){
  initialize list to be empty
  while  (list1 is not empty) & (list2 is not empty){
    if  (list1.first < list2.first)
          list.addlast( list1.removeFirst(list1) )
    else
          list.addlast( list2.removeFirst(list2) )
  }
  while list1 is not empty
    list.addlast( list1.removeFirst(list1) )
  while list2 is not empty
    list.addlast( list2.removeFirst(list2) )

  return  list
}
```

8
10
3
11
6
1
7
16
2
5
4
13

```
mergesort(list){
   if  list.length == 1
      return list
   else{
      mid = (list.size - 1) / 2
      list1 =  list.getElements(0,mid)
      list2 =  list.getElements(mid+1, list.size-1)
      list1  =  mergesort(list1)
      list2  =  mergesort(list2)
      return   merge( list1, list2 )
   }
 }
```



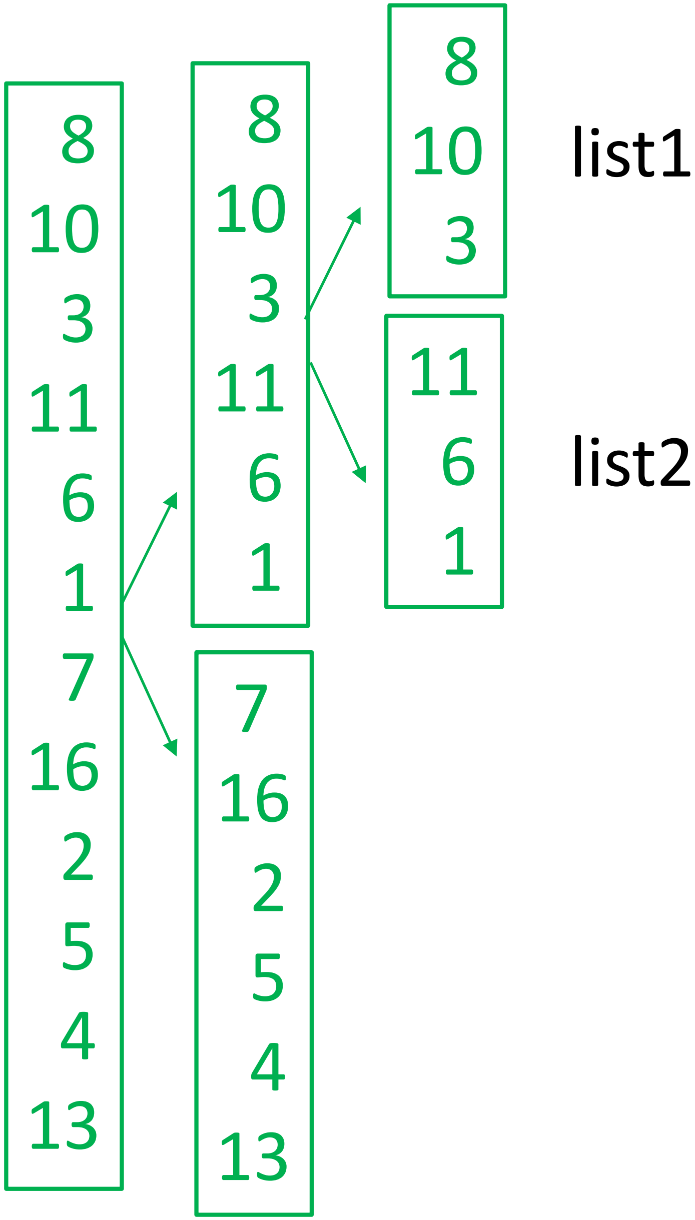
8
10
3
11
6
1
7
16
2
5
4
13

8
10
3
11
6
1

list1

7
16
2
5
4
13

list2



8
10
3
11
6
1
7
16
2
5
4
13

8
10
3
11
6
1

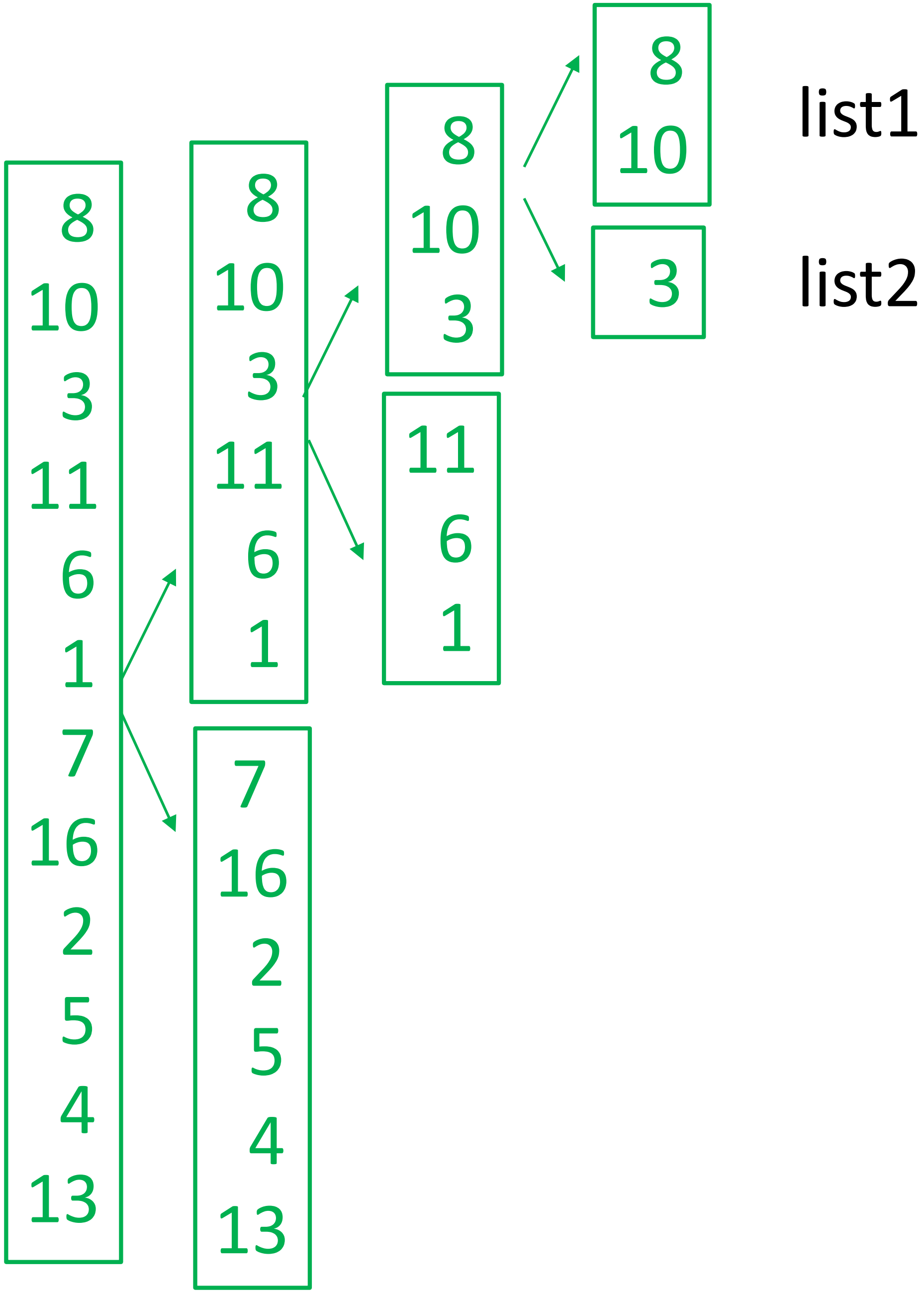
7
16
2
5
4
13

8
10
3

list1

11
6
1

list2



8
10
3
11
6
1
7
16
2
5
4
13

8
10
3
11
6
1

7
16
2
5
4
13

8
10
3

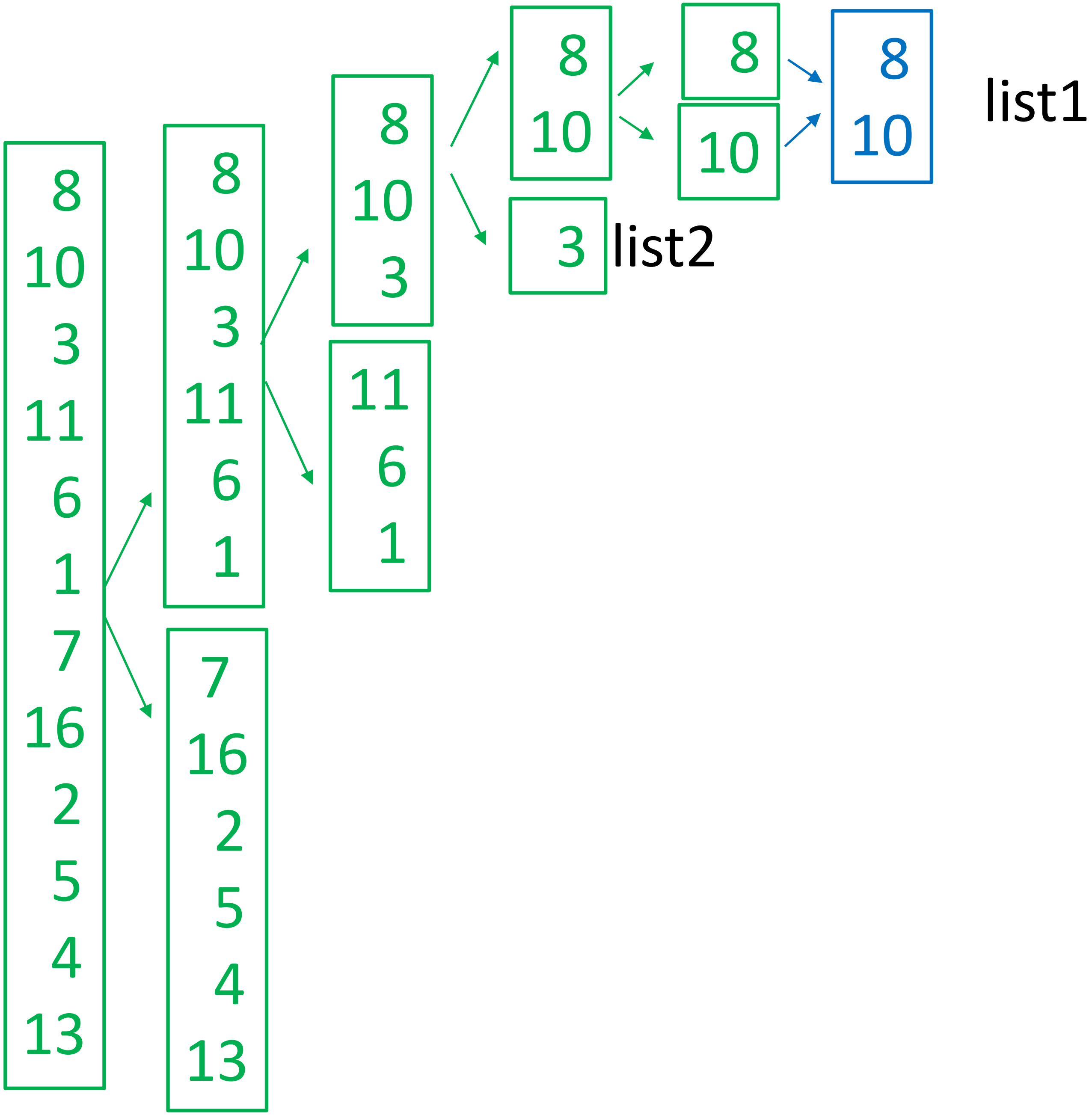
11
6
1

8
10

3

list1

list2



8 10 3 11 6 1 7 16 2 5 4 13

8 10 3 11 6 1

7 16 2 5 4 13

8 10 3

11 6 1

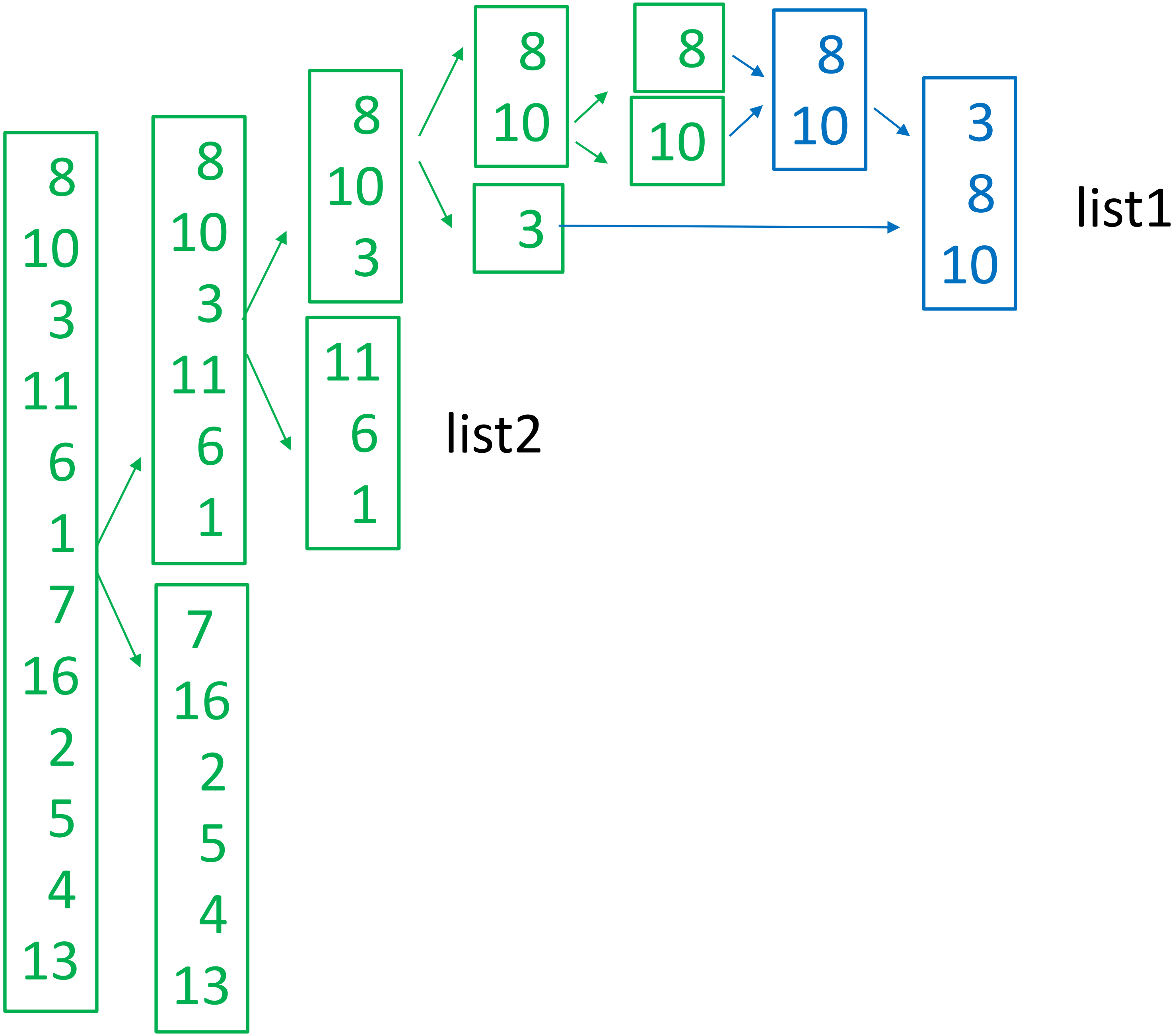
8 10

3 list2

8

10

8 10 list1



8
10
3
11
6
1
7
16
2
5
4
13

8
10
3
11
6
1

7
16
2
5
4
13

8
10
3

11
6
1

list2

8
10

3

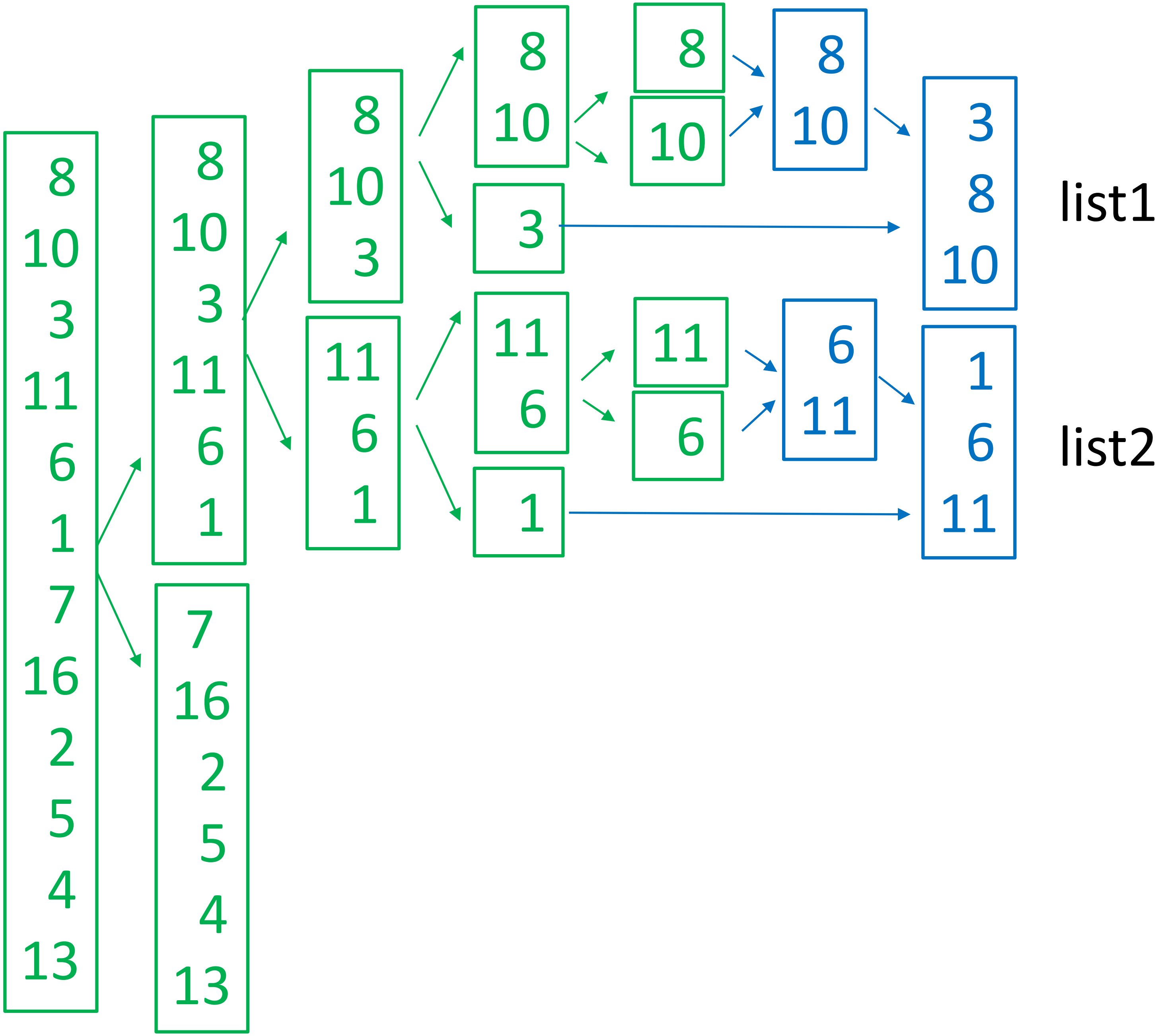
8

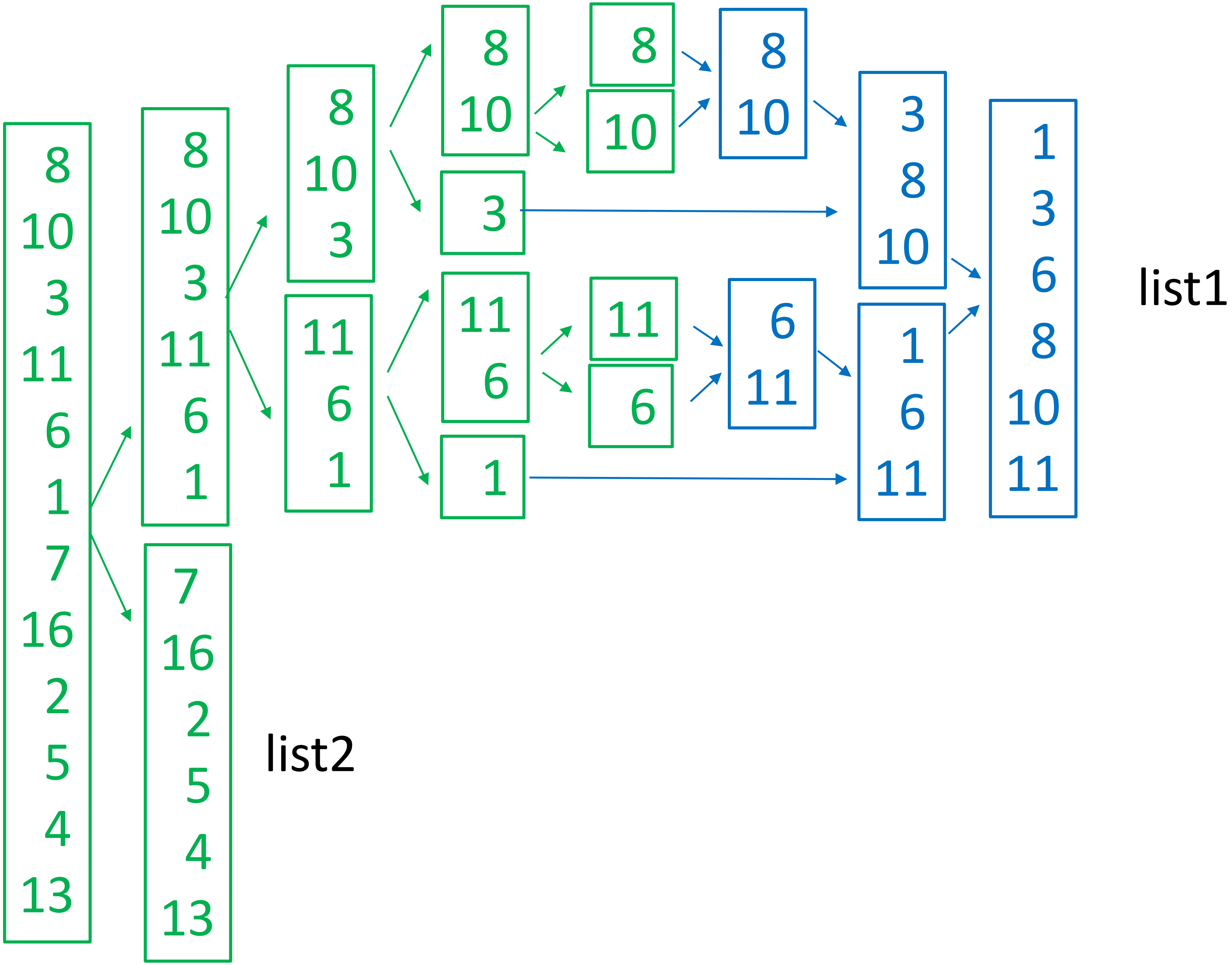
10

8
10

3
8
10

list1


8 10 3 11 6 1 7 16 2 5 4 13
8 10 3 11 6 1
7 16 2 5 4 13
8 10 3
11 6 1
8 10
3
11 6
1
8
10
11
6
8 10
6 11
3 8 10
1 6 11
list1
list2



8 10 3 11 6 1 7 16 2 5 4 13

8 10 3 11 6 1

7 16 2 5 4 13

list2

8 10 3

11 6 1

8 10

3

11 6

1

8

10

11

6

8 10

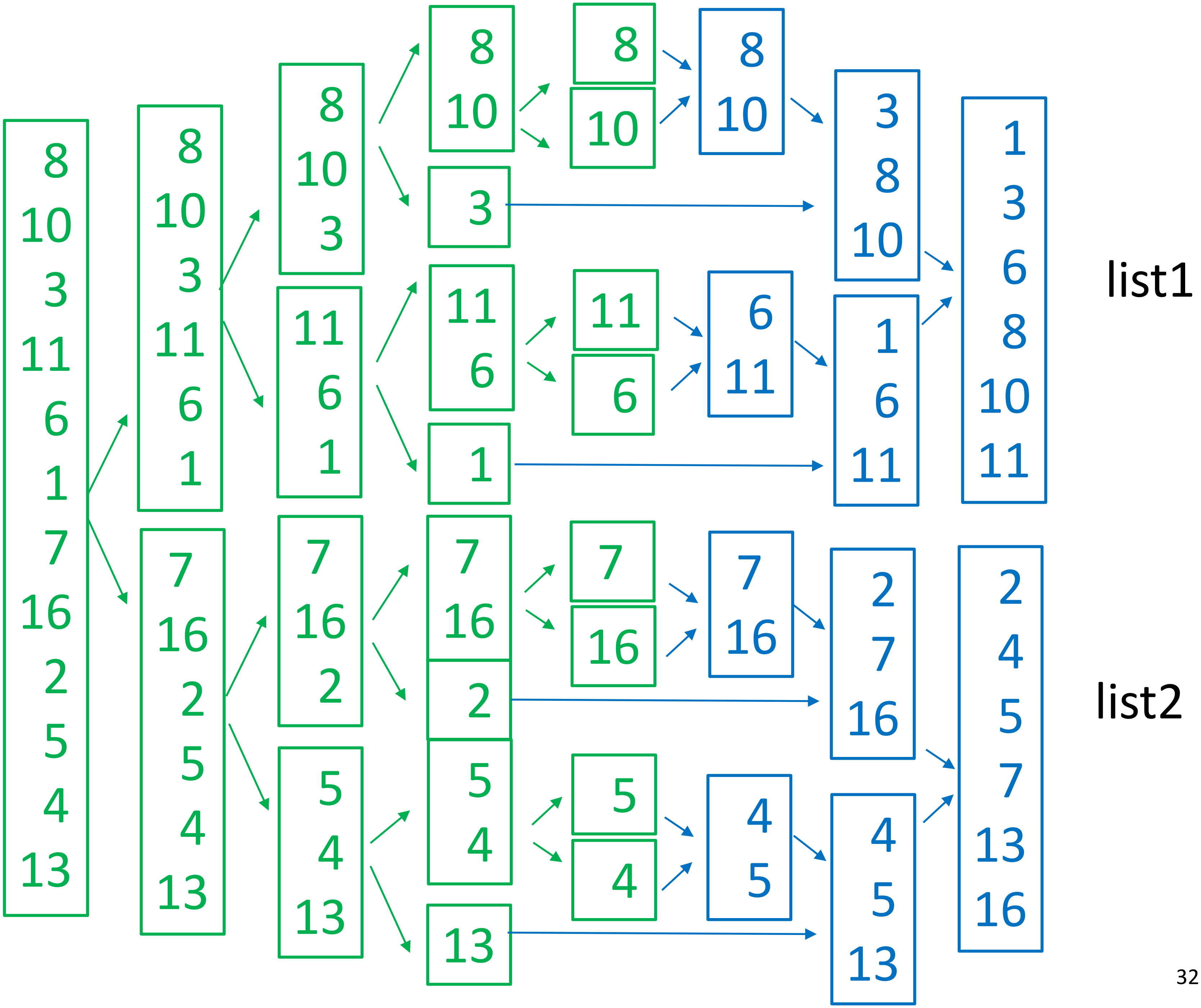
6 11

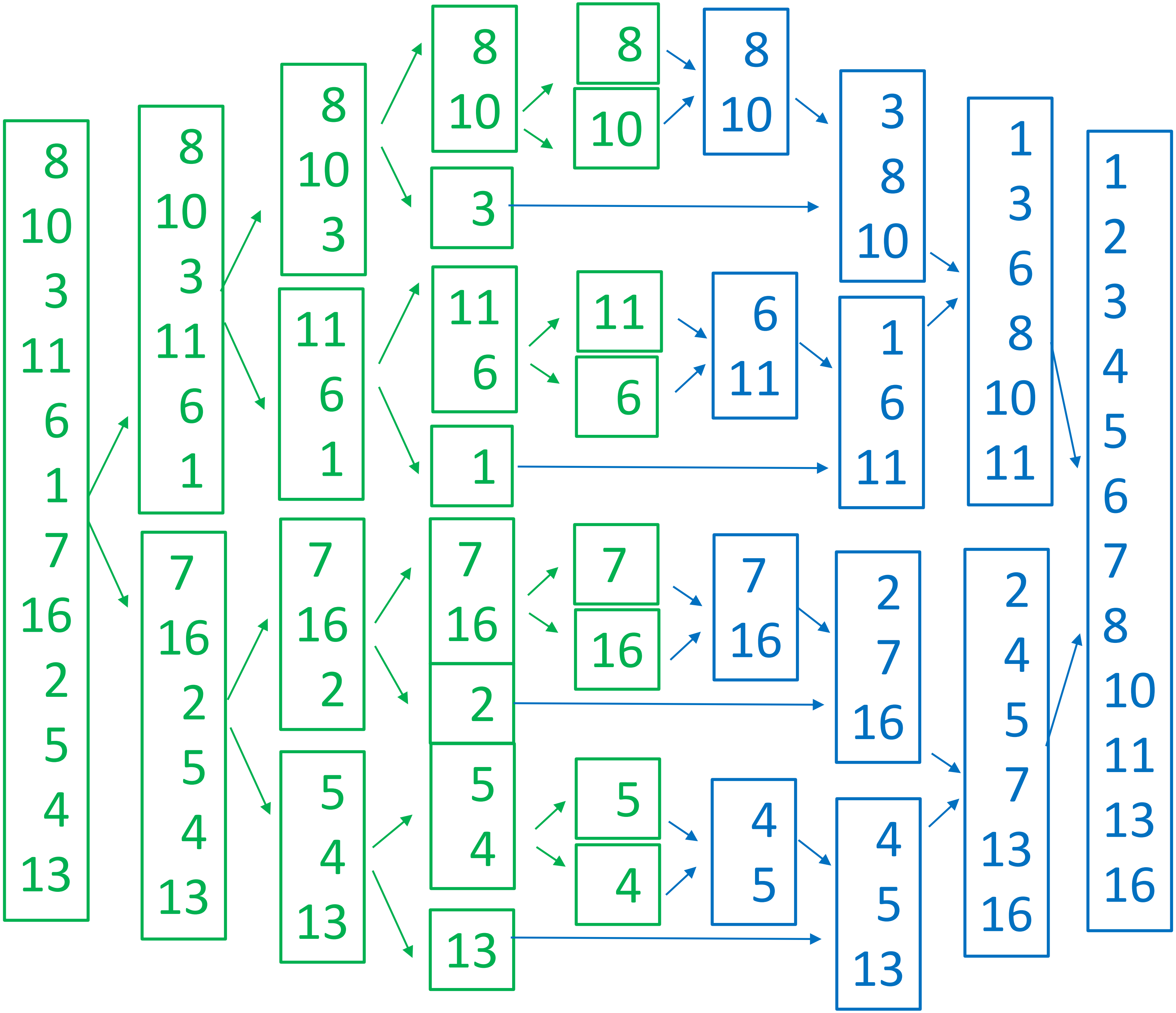
3 8 10

1 6 11

1 3 6 8 10 11

list1


8 10 3 11 6 1 7 16 2 5 4 13
8 10 3 11 6 1
7 16 2 5 4 13
8 10 3
11 6 1
7 16 2
5 4 13
8 10
3
11 6
1
7 16
2
5 4
13
8
10
11
6
7
16
5
4
8 10
6 11
7 16
4 5
3 8 10
1 6 11
2 7 16
4 5 13
1 3 6 8 10 11
2 4 5 7 13 16
list1
list2



8 10 3 11 6 1 7 16 2 5 4 13

8 10 3 11 6 1

7 16 2 5 4 13

8 10 3

11 6 1

7 16 2

5 4 13

8 10

3

11 6

1

7 16

2

5 4

13

8

10

11

6

7

16

5

4

8 10

6 11

7 16

4 5

3 8 10

1 6 11

2 7 16

4 5 13

1 3 6 8 10 11

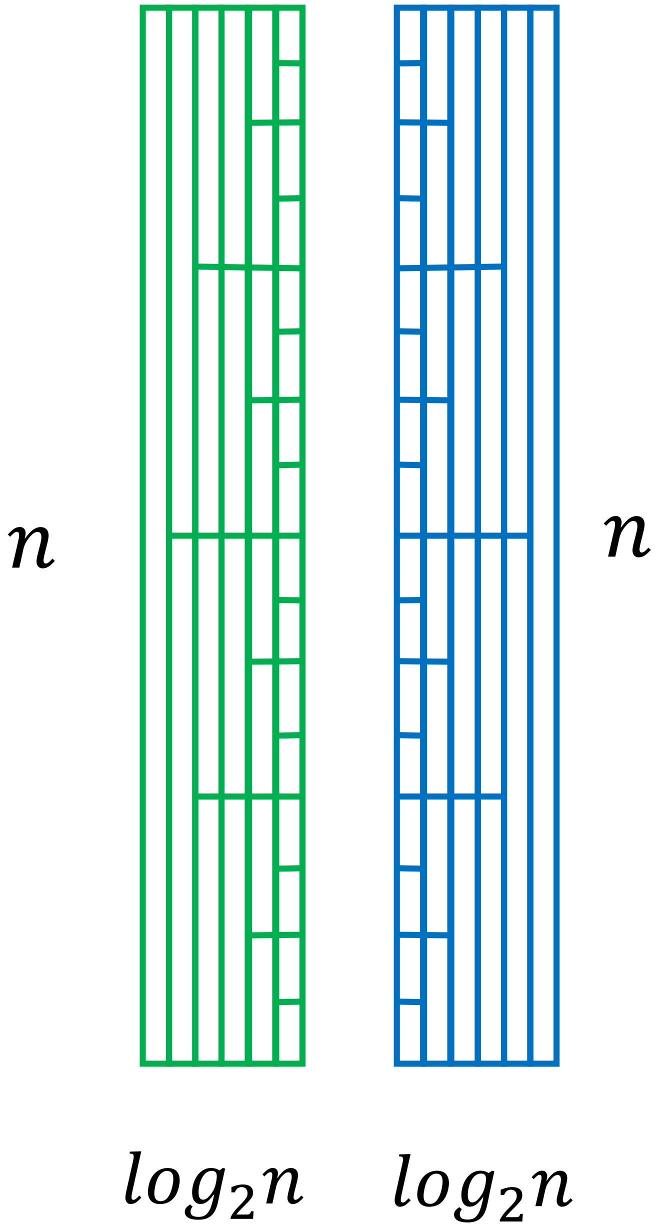
2 4 5 7 13 16

1 2 3 4 5 6 7 8 10 11 13 16

Q: How many operations are required to mergesort a list of size n ?

A: $O(\ n\ \ log_2 n\ )$

This will become more clear a few lectures from now when we discuss recurrences.



$n\ log_2\, n$ is
much closer to n
than to n^2

$log_2\, n$	n	$\boldsymbol{n\ \ log_2\, n}$	n^2
10	$2^{10} \approx 10^3$	$\mathbf{10^4}$	10^6
20	$2^{20} \approx 10^6$	$\boldsymbol{\sim 10^7}$	10^{12}
30	$2^{30} \approx 10^9$	$\boldsymbol{\sim 10^{10}}$	10^{18}

Computers perform $\sim 10^9$ operations per second.

$log_2\, n$	n	$\boldsymbol{n\ log_2\, n}$	n^2
10	$2^{10} \approx 10^3$	$\mathbf{10^4}$	10^6
20	$2^{20} \approx 10^6$	$\mathbf{\sim 10^7}$	10^{12}
30	$2^{30} \approx 10^9$	$\mathbf{\sim 10^{10}}$	10^{18}

milliseconds minutes hours centuries

$$O(n) \quad < \quad O(\,n\ \ log_2 n) \quad \ll \quad O(\,n^2\ )$$

mergesort
quicksort

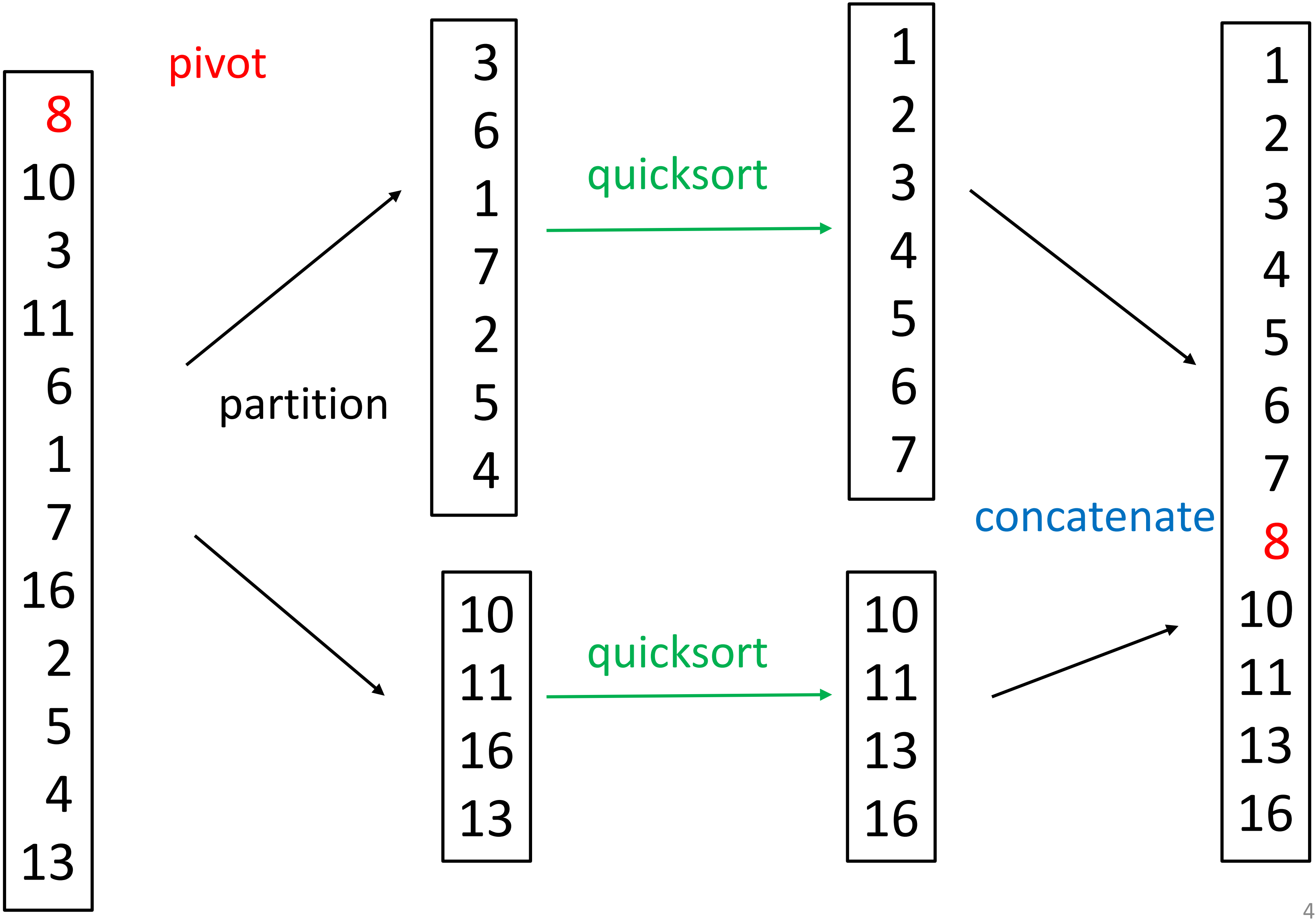
bubble sort
selection sort
insertion sort

Quicksort

```
quicksort(list){
    if  list.length <= 1
        return list
    else{
        pivot =  list.removeFirst()   // or some other element
        list1  =   list.getElementsLessThan(pivot)
        list2  =   list.getElementsGreaterOrEqual(pivot)
        list1 = quicksort(list1)
        list2 = quicksort(list2)
    }
    return  concatenate( list1, e, list2 )
}
```

Quicksort

```
quicksort(list){
   if  list.length <= 1
      return list
   else{
      pivot =  list.removeFirst()   // or some other element
      list1  =   list.getElementsLessThan(pivot)
      list2  =   list.getElementsGreaterOrEqual(pivot)
      list1 = quicksort(list1)
      list2 = quicksort(list2)
   }
   return  concatenate( list1, e, list2 )
}
```



pivot

8
10
3
11
6
1
7
16
2
5
4
13

partition

3
6
1
7
2
5
4

10
11
16
13

quicksort

1
2
3
4
5
6
7

quicksort

10
11
13
16

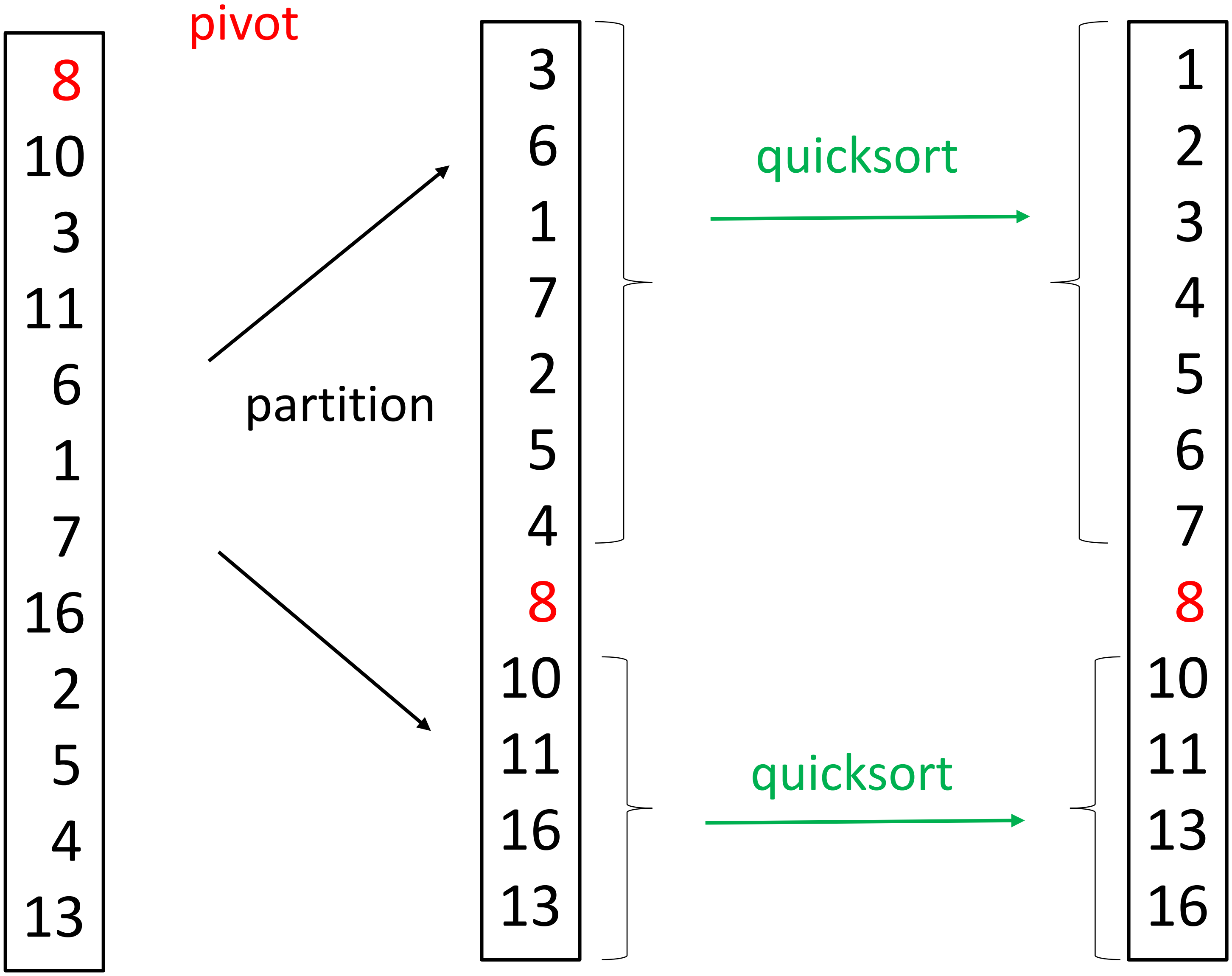
concatenate

1
2
3
4
5
6
7
8
10
11
13
16

Quicksort can be done “in place”

```
quicksort( low, high ){
    if  low > high
        return
    else{
        pivot =  ___;   // select index in {low, ..., high}
        partitionIndex  =   makePartition (low, high, pivot)
        quicksort(low,  partitionIndex - 1)
        quicksort(partionIndex + 1,  high)
     }
}
```

Quicksort partitioning can be done ‘in place’ using a clever swapping and scanning technique. (See web for details, if interested.)



pivot

8
10
3
11
6
1
7
16
2
5
4
13

partition

3
6
1
7
2
5
4
8
10
11
16
13

quicksort

quicksort

1
2
3
4
5
6
7
8
10
11
13
16

Mergesort vs. Quicksort

- Mergesort typically uses an extra list. More space can hurt performance for big lists.
- We will discuss worst case performance of quicksort later in the course.
- See stackoverflow if you want opinions on which is better.