

COMP 250

Lecture 21

binary search, mergesort 1

## Recall: Converting to binary (iterative)

```
toBinary( n ){  
    while n > 0 {  
        print  n % 2  
        n     = n / 2  
    }  
}
```

This prints out bits  $b[0]$ ,  $b[1]$ , ..

Recall that  $n$  in binary needs approximately  $\log_2 n$  bits.

# Converting to binary (recursive)

```
toBinary( n ){  
    if n > 0 { // base case n==0  
        print n % 2  
        toBinary( n / 2 )  
    }  
}
```

This prints out bits  $b[0]$ ,  $b[1]$ , ...

There will be approximately  $\log_2 n$  recursive calls.

# Binary Search

-75

-31

-26

-4

1

6

25

26

28

39

72

141

290

300

Input:

- a *sorted list* of size  $n$
- a *value* that we are searching for

Output:

If the value is in the list, *return its index*.  
Otherwise, *return -1*.

# Binary Search

-75

-31

-26

-4

1

6

25

26

28

39

72

141

290

300

Example: Search for **17**.

(output will be -1)

What is an efficient way to do this ?

Think of how you search for a term in an index. Do you start at the beginning and then scan through to the end? (No.)

Exponential inequality 185  
 Exponential running time 186, 661, 911  
 Extended Church-Turing thesis 910  
 Extensible library 101  
 External path length 418, 832

**F**

Factor an integer 919  
 Factorial function 185  
 Fail-fast design 107  
 Fail-fast iterator 160, 171  
 Farthest pair 210  
 Fibonacci heap 628, 682  
 Fibonacci numbers 57  
 FIFO. *See* First-in first-out policy  
 FIFO queue.  
     *See* Queue data type  
 File system 493  
 Filter 60  
     blacklist 491  
     dedup 490  
     whitelist 8, 491  
 Final access modifier 105–106  
 Fingerprint search 774–778  
 Finite state automaton.  
     *See also* Maxflow problem  
 flow network 888  
 inflow and outflow 888  
 residual network 895  
 st-flow 888  
 st-flow network 888  
 value 888  
 Floyd, R. W. 326  
 Floyd's method 327  
 for loop 16  
 Ford-Fulkerson 891–893  
     analysis of 900  
     maximum-capacity path 901  
     shortest augmenting path 897  
 Ford, L. 683  
 Foreach loop 138  
     arrays 160  
     strings 160  
 Forest  
     graph 520  
     spanning 520  
 Forest-of-trees 225  
 Formatted output 37  
 Fortran language 217  
 Fragile base class problem 112  
 Frazer, W. 306  
 Fredman, M. L. 628  
 Function call stack 346, 415  
     symbol tables 363  
     type parameter 122, 134  
 Genomics 492, 498  
 Geometric data types 76–77  
 Geometric sum 185  
 getClass() method 101, 103  
 Girth of a graph 559  
 Global variable 113  
 Gosper, R. W. 759  
 Graph data type 522–527  
 Graph isomorphism 561, 919  
 Graph processing 514–693.  
     *See also* Directed graph;  
     *See also* Edge-weighted digraph;  
     *See also* Edge-weighted graph;  
     *See also* Undirected graph;  
     *See also* Directed acyclic graph  
     Bellman-Ford 668–681  
     breadth-first search 538–541  
     components 543–546  
     critical-path method 664–666  
     depth-first search 530–537  
     Dijkstra's algorithm 652  
     Kosaraju's algorithm 586–590  
     Kruskal's algorithm 624–627  
     largest paths 811, 812

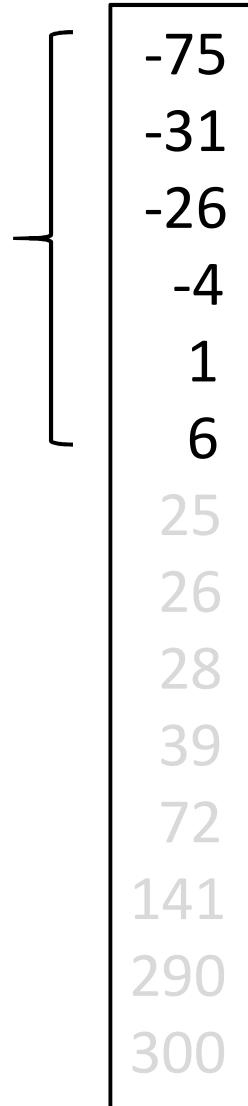
So what do we do?

Examine the item at the middle position of the list.  
If we find **17** there, then return the index (mid).  
Otherwise... what?

compare **17** to →

-75	low = 0
-31	
-26	
-4	
1	
6	
<b>25</b>	mid = (low + high) / 2
26	
28	
39	
72	
141	
290	
300	high = size - 1

Since 17 < 25,  
search for 17 here



-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

low = 0

mid = (low + high) / 2 = 6

high = size - 1 = 13



compare 17 to →

- 75
- 31
- 26
- 4
- 1
- 6
- 25
- 26
- 28
- 39
- 72
- 141
- 290
- 300

low = 0

mid =  $(\text{low} + \text{high}) / 2 = 2$

high = 5

search for 17 here

-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

$$\text{low} = 3$$

$$\text{mid} = (\text{low} + \text{high}) / 2 = 4$$

$$\text{high} = 5$$

compare 17 to →

-75  
-31  
-26  
-4  
1  
6  
25  
26  
28  
39  
72  
141  
290  
300

low = 3

mid = (low + high) / 2 = 4

high = 5

search for 17 here



-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

low = high = 5


compare 17 to →

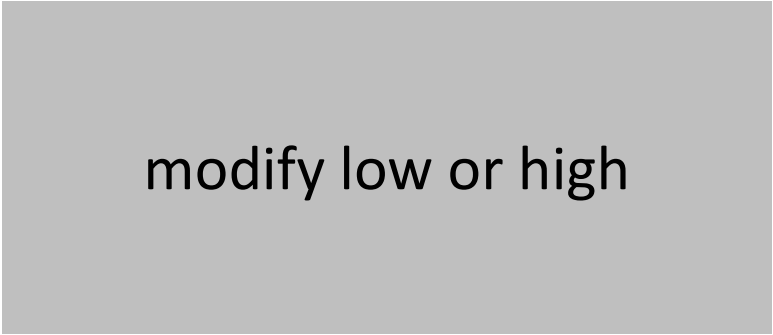
-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

low = high

It fails.

So return index -1  
(value 17 not found)

```
binarySearch( list, value){ // Iterative solution
  low = 0
  high = list.size - 1
  while low <= high {
    
  }
  return -1 // value not in list
}
```

```
binarySearch( list, value ){ // Iterative solution
  low = 0
  high = list.size - 1
  while low <= high {
    mid = (low + high)/ 2 // if high == low + 1, then mid == low
    if list[mid] == value
      return mid
    else{
      
      modify low or high
    }
  }
  return -1 // value not in list
}
```

```

binarySearch( list, value ){                                     // Iterative solution
    low = 0
    high = list.size - 1
    while low <= high {
        mid = (low + high)/ 2    // if high == low + 1, then mid == low
        if list[mid] == value
            return mid
        else{ if value < list[mid]
                high = mid - 1    // high can become less than low
            else
                low = mid + 1    // namely, if mid == low
        }
    }
    return -1    // value not found, namely if high < low
}

```



```
binarySearch( list, value ){
```

how to make this recursive ?

```
}
```

```
binarySearch( list, value, low, high ){ // pass as parameters
```

```
    if low <= high { // if instead of while
```

```
        mid = (low + high)/ 2
```

```
        if value == list[mid]
```

```
            return mid
```

what if value != list[mid] ?

```
    else
```

```
        return -1
```

```
}
```

```

binarySearch( list, value, low, high ){ // pass as parameters

    if low <= high { // if instead of while
        mid = (low + high)/ 2
        if value == list[mid] // base case 1
            return mid
        else if value < list[mid]
            return binarySearch( list, value, low, mid - 1 )
            // mid-1 can be less than low
        else
            return binarySearch( list, value, mid+1, high)
    }
    else return -1 // base case 2: high < low
}

```

# Observations about binary search

Q: How many times through the while loop ? (iterative)  
How many recursive calls? (recursive)

A: Time to search is worst case  $O(\log_2 n)$  where  $n$  is size of the list. Why? Because each time we are approximately halving the size of the list.

The best case is that we find the value right away, i.e.  $O(1)$ .

# Time complexity (for worst case)

$O(\log_2 n)$	$O(n)$	$O(n^2)$
<ul style="list-style-type: none"><li>• convert to binary</li><li>• binary search</li><li>• .....</li></ul>	<ul style="list-style-type: none"><li>• List operations: findMax, remove</li><li>• grade school addition or subtraction (<math>n</math> is number of bits or digits)</li><li>• .....</li></ul>	<ul style="list-style-type: none"><li>• insertion/selection/ bubble sort</li><li>• grade school multiplication (<math>n</math> is number of bits or digits)</li><li>• .....</li></ul>

More concretely....

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

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




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


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

More concretely....

Suppose a computer performs  $\sim 10^9$  operations per second.

How long does it take for  $n$  vs.  $\log_2 n$  vs.  $n^2$  operations?

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

  $\sim 15$  minutes...

$\sim$  one second

centuries

# Faster sorting algorithms ?

$$O(n) < ? < O(n^2)$$



mergesort  
quicksort  
heapsort  
.....



COMP 250

Lecture 20

recursion 2:

binary search  
mergesort (part 1)

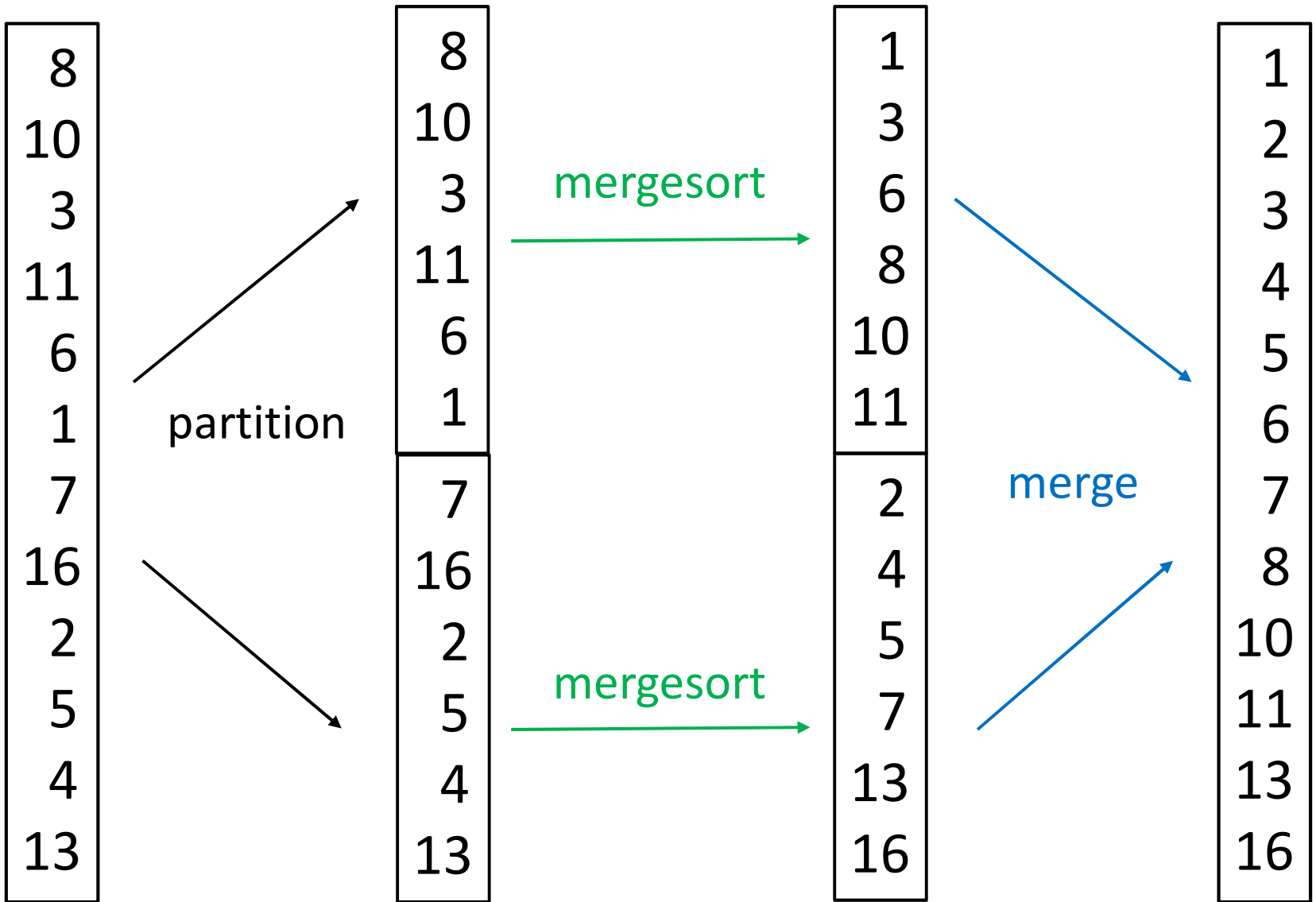
Oct. 25, 2021

# Mergesort

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

Sort each half (recursively).

Merge the two halves.



```
mergesort(list){
  if list.length == 1      // base case
    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 )
  }
}
```

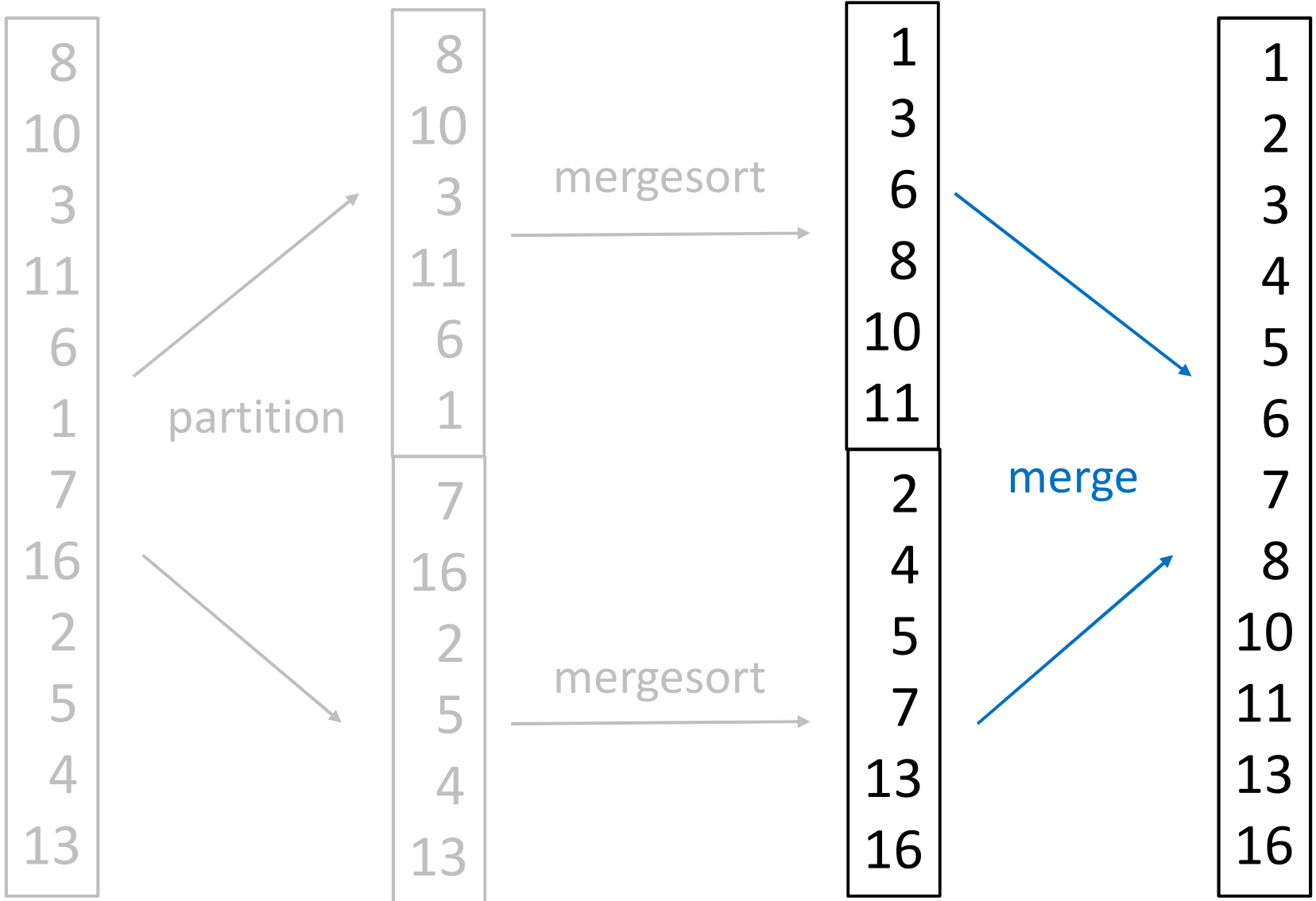
## Winter 2022 note

It is wasteful to call mergesort when list has just one element.

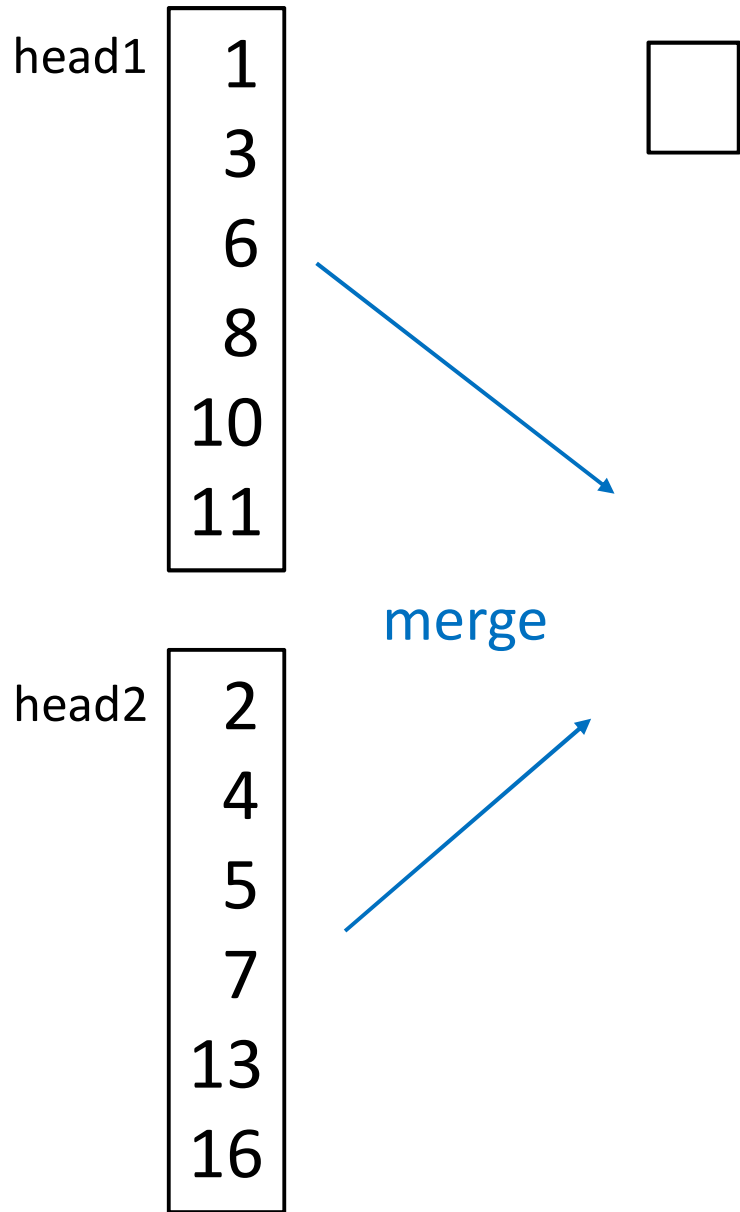
So in practice we would test that the length of the list is greater than 1 before called recursively. (But note that doing such a test still does take some work, i.e. not free.)

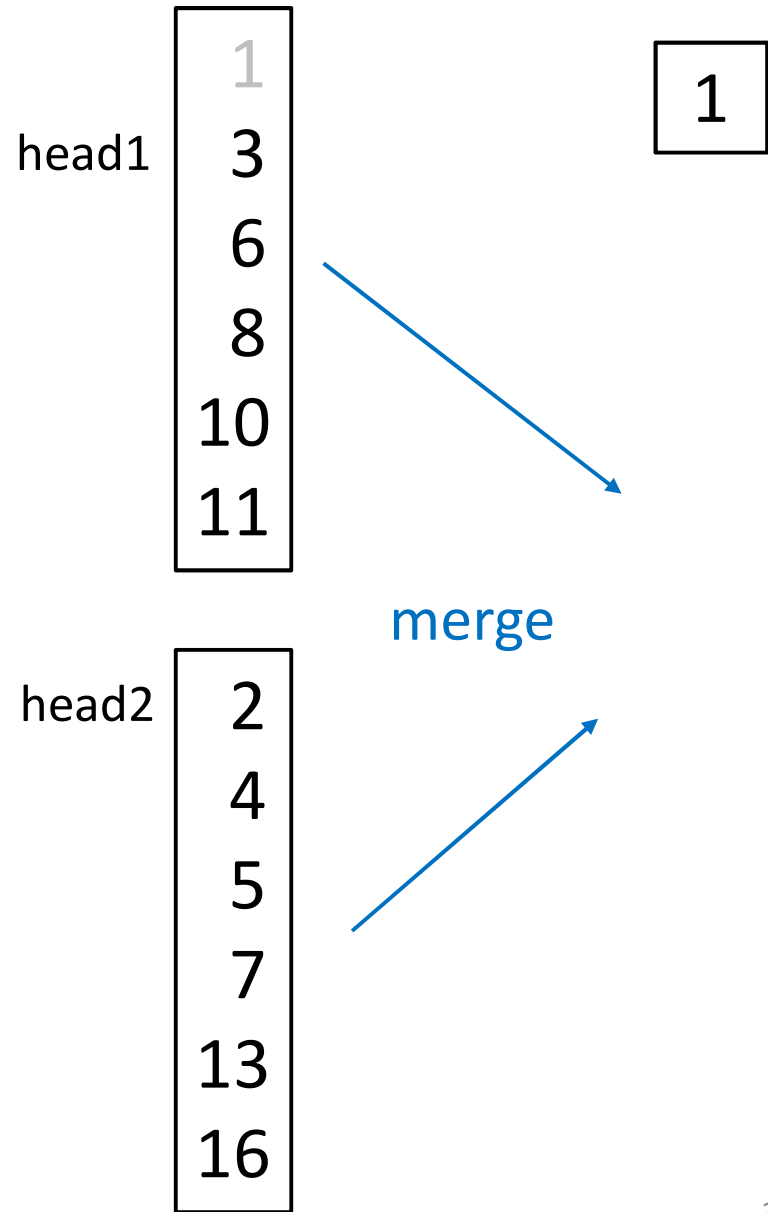
```
mergesort(list){
  if list.length == 1           // base case
    return list
  else{
    mid = (list.size - 1) / 2
    list1 = list.getElements(0,mid)
    list2 = list.getElements(mid+1, list.size-1)
    list1 = mergesort(list1)
    list2 = mergesort(list2)
    return merge( list1, list2 )
  }
}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

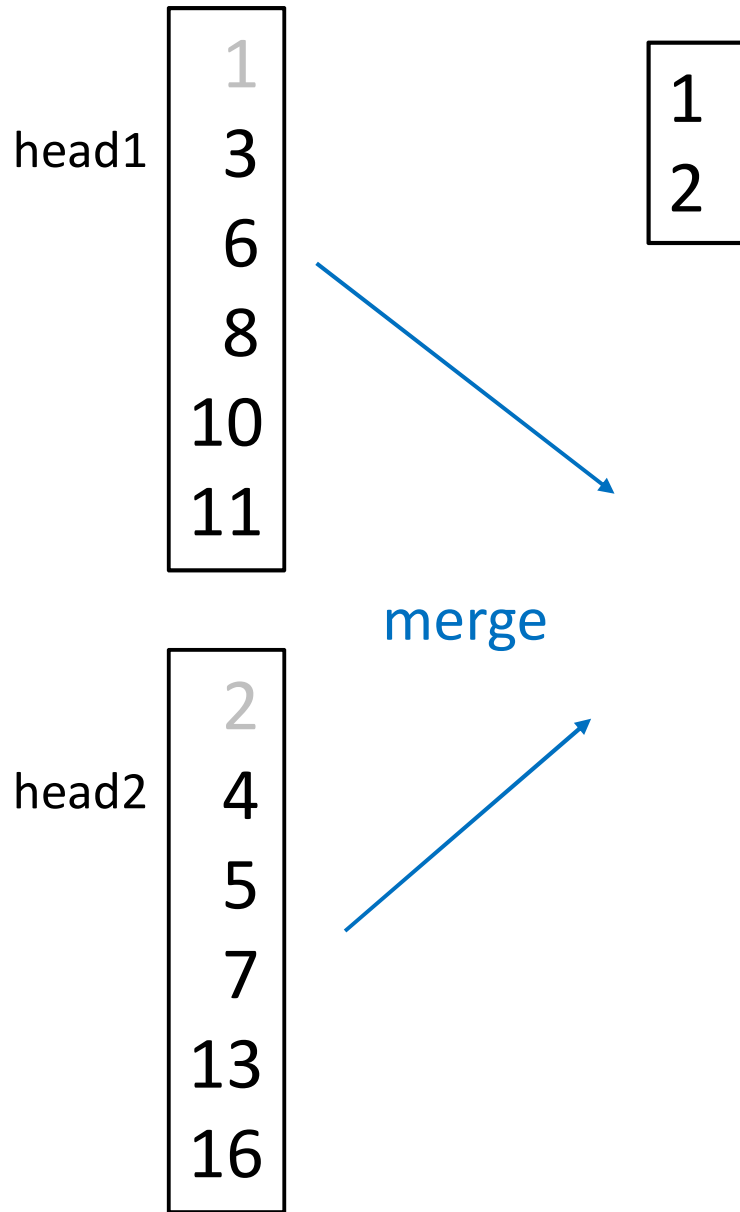
```
mergesort(list){
  if list.length == 1           // base case
    return list
  else{
    mid = (list.size - 1) / 2
    list1 = list.getElements(0,mid)
    list2 = list.getElements(mid+1, list.size-1)
    list1 = mergesort(list1)
    list2 = mergesort(list2)
    return merge( list1, list2 )
  }
}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

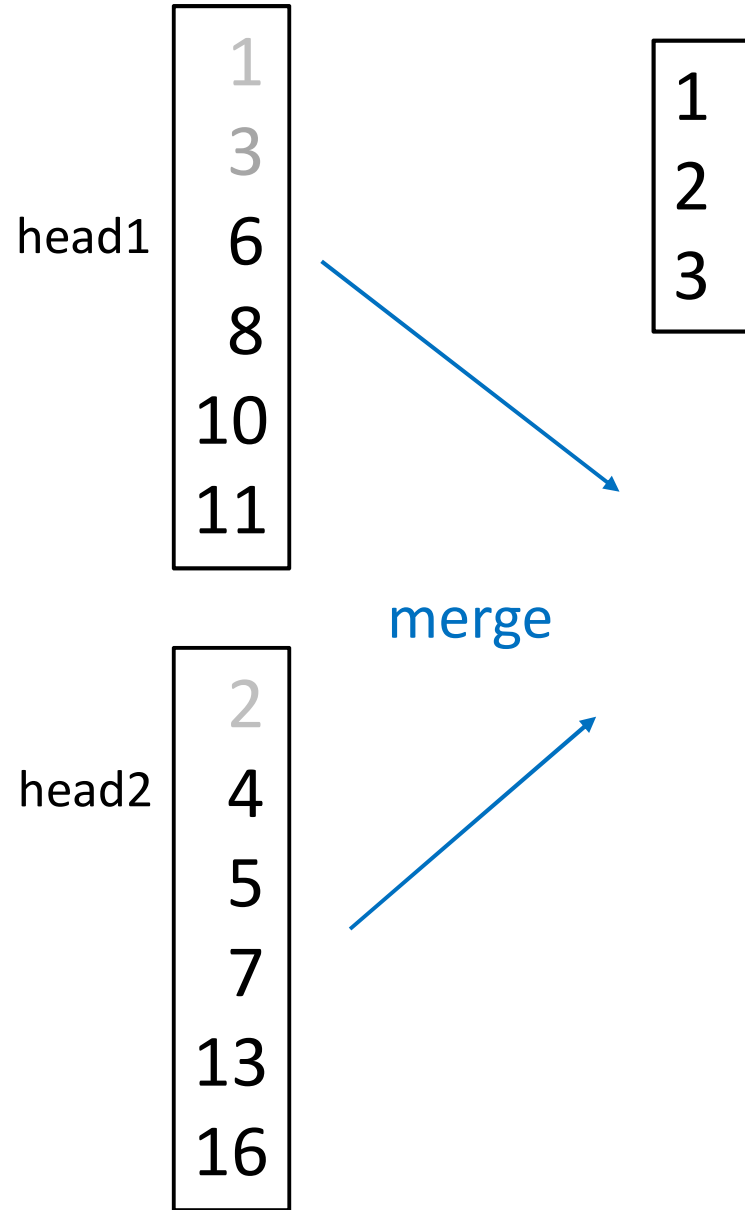


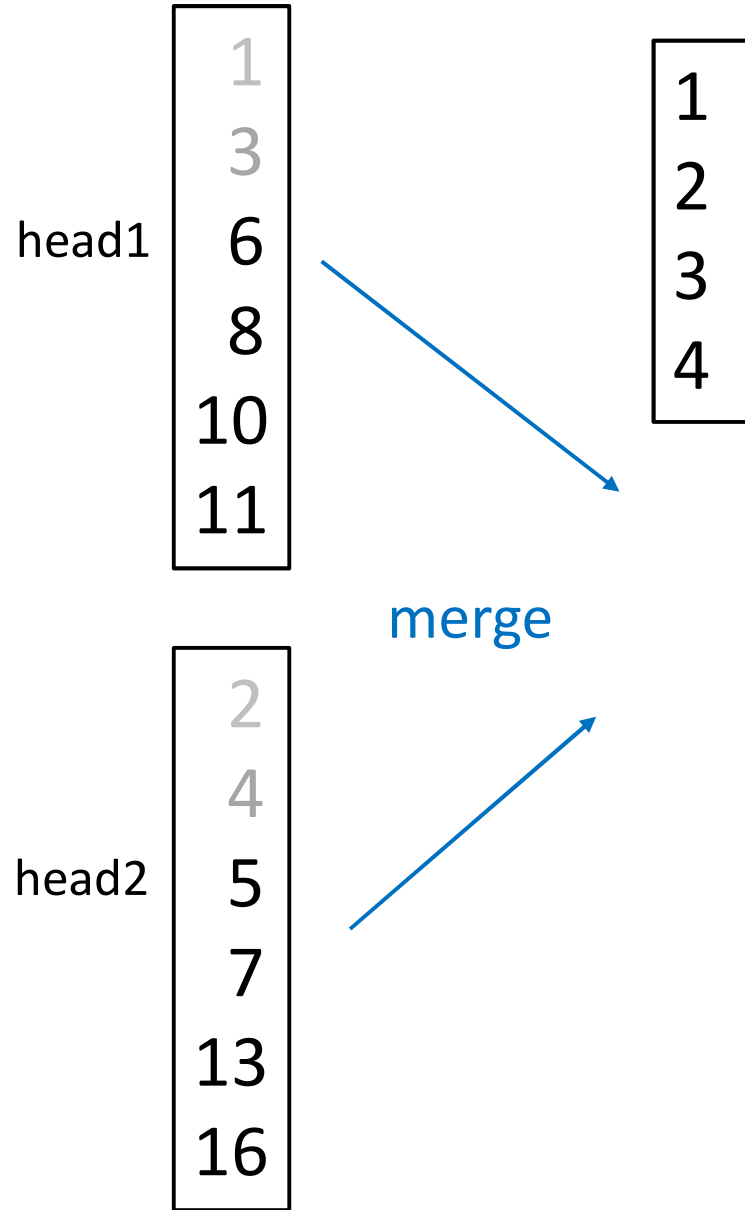


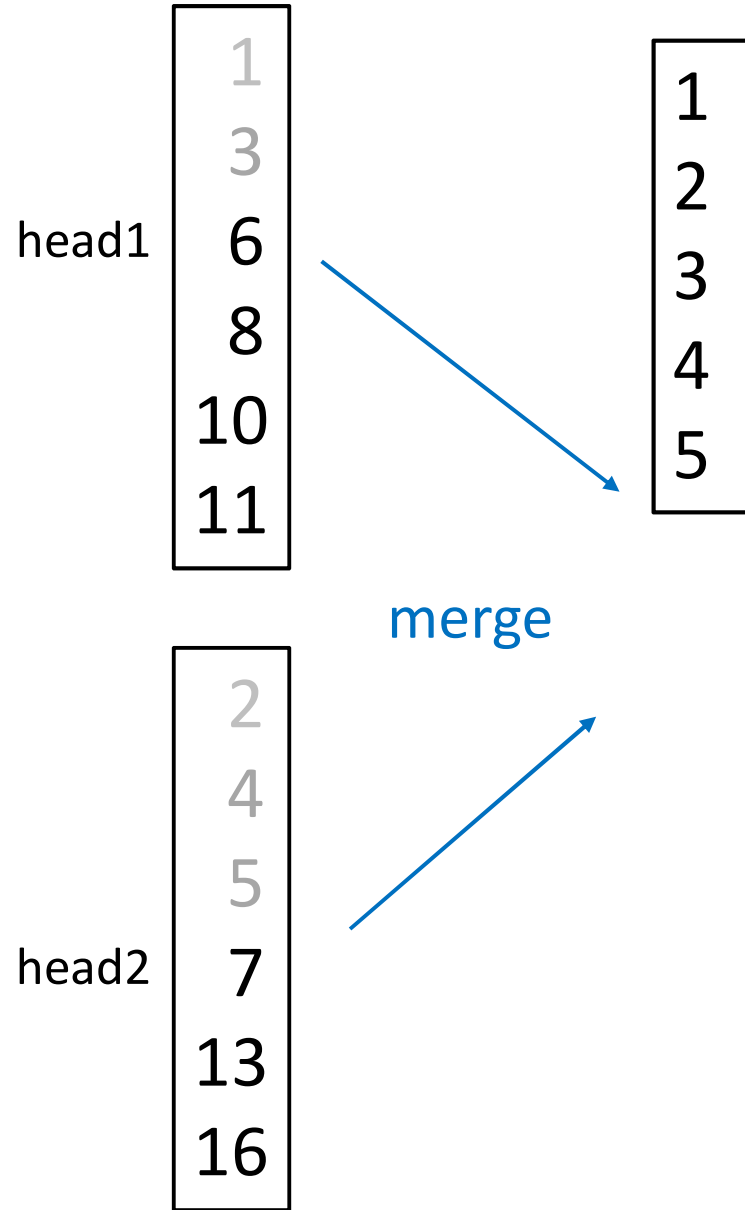


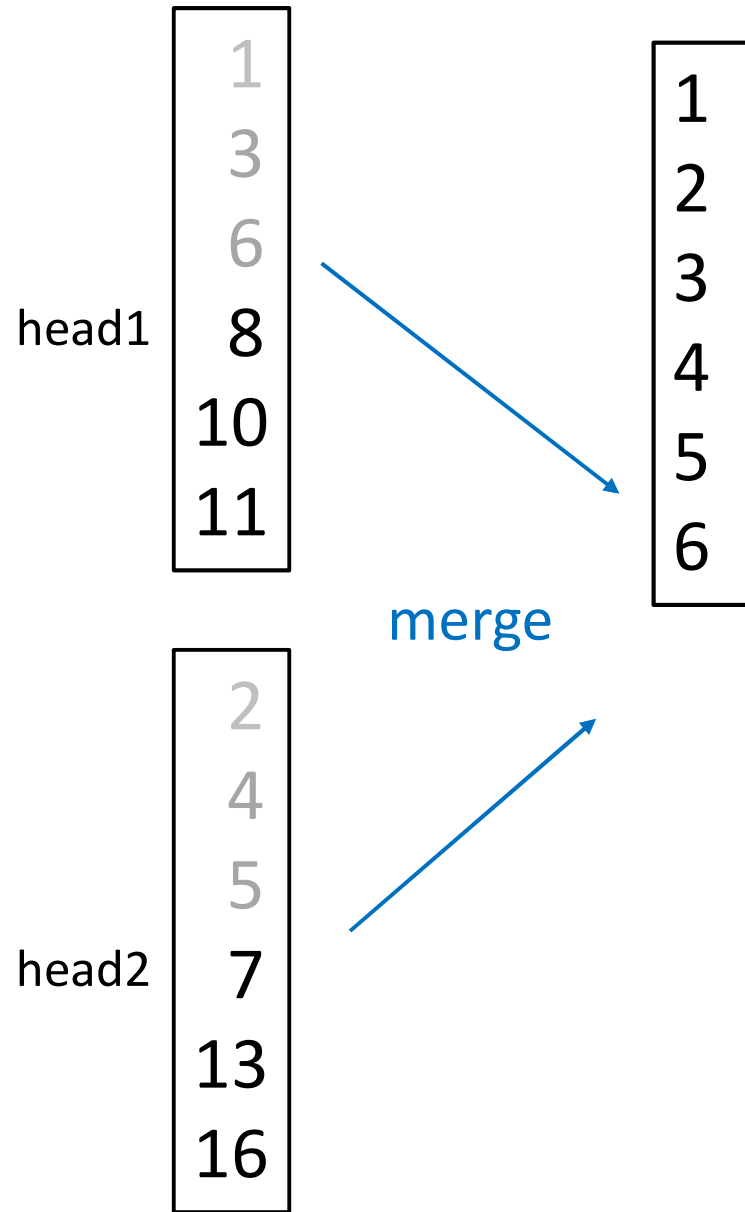


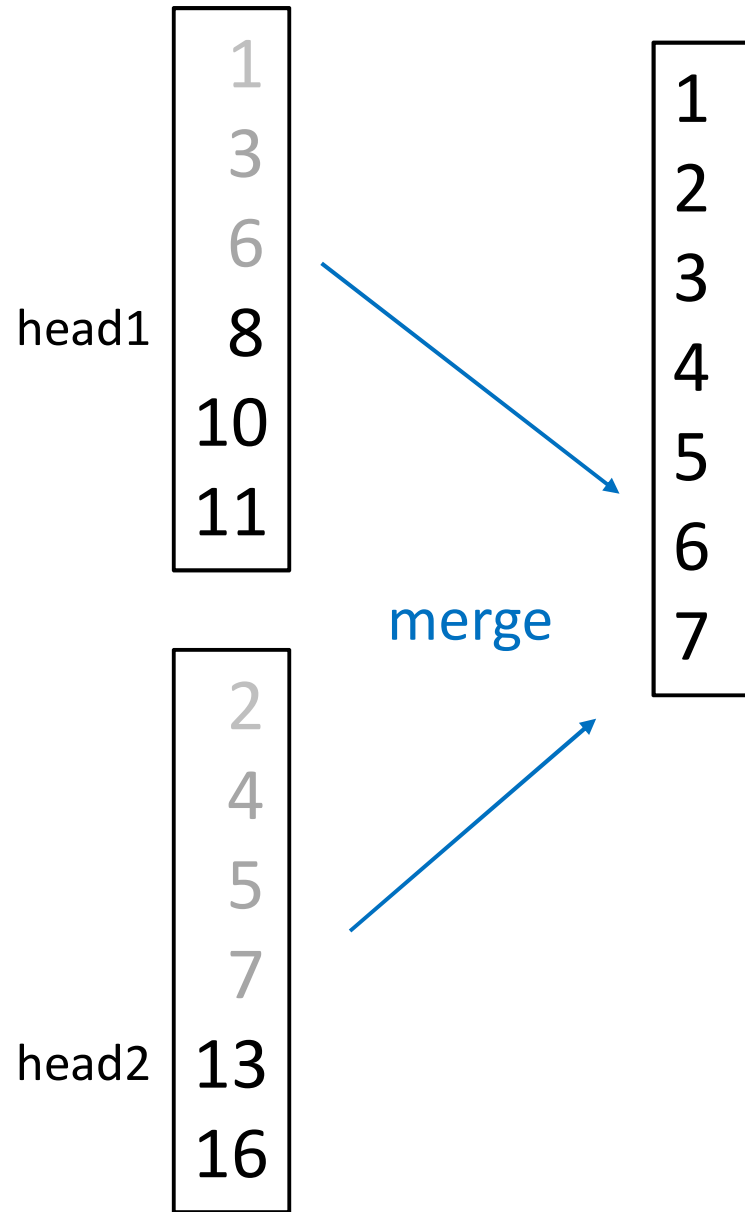






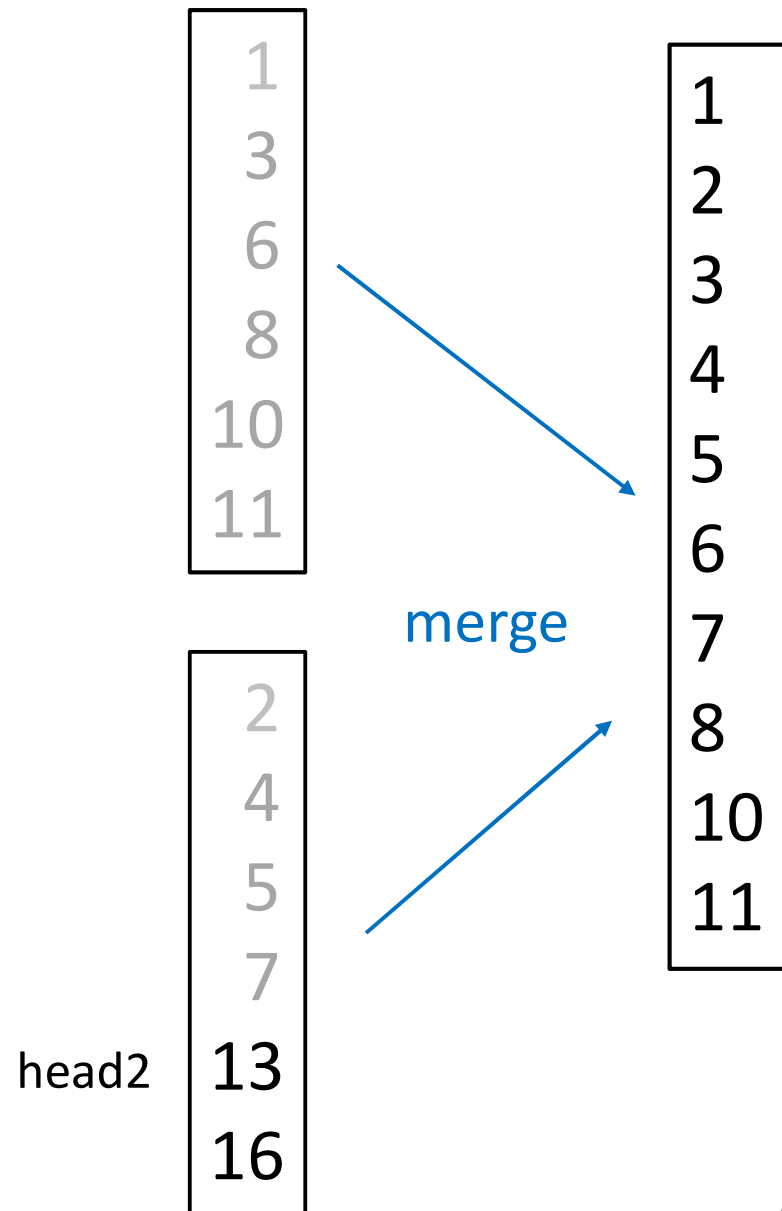




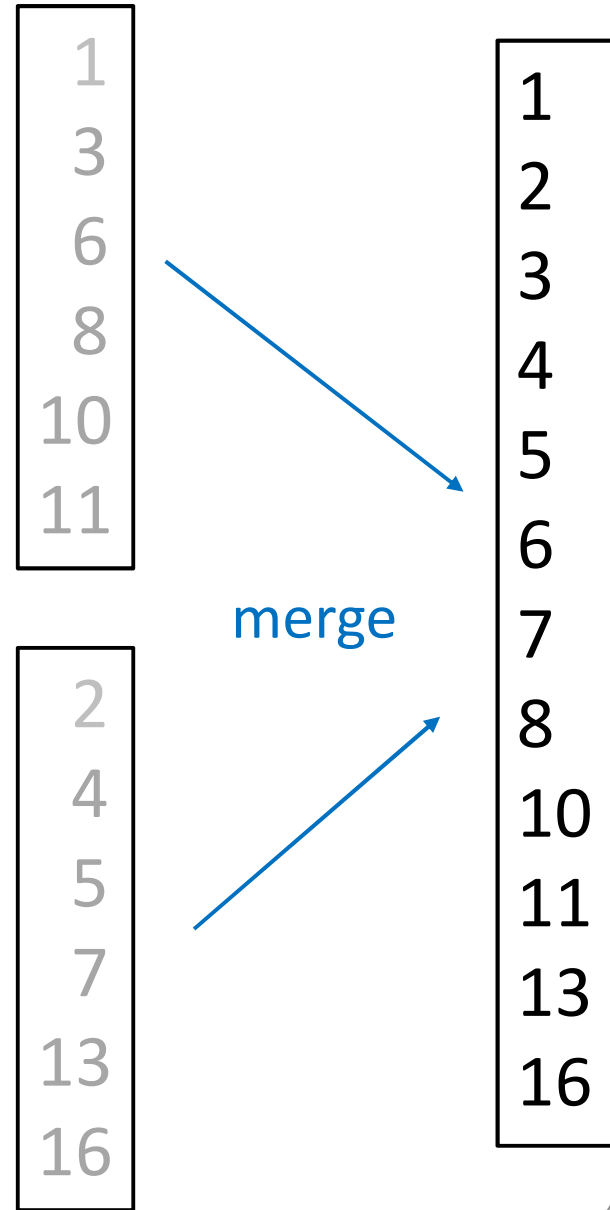




...and so on until one list is empty.



Then, copy the remaining elements.



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

  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() )
    else
      list.addlast( list2.removeFirst() )
  }
  while list1 is not empty
    list.addlast( list1.removeFirst() )
  while list2 is not empty
    list.addlast( list2.removeFirst() )

  return list
}
```

We will continue  
mergesort next lecture...