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.)

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?

	-75	low = 0
	-31	
	-26	
	-4	
	1	
	6	
compare 17 to →	25	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

	-75	low = 0
	-31	
compare 17 to →	-26	mid = (low + high) / 2 = 2
	-4	
	1	
	6	high = 5
	25	
	26	
	28	
	39	
	72	
	141	
	290	
	300	

search for 17 here

-75	
-31	
-26	
-4	low = 3
1	mid = (low + high) / 2 = 4
6	high = 5
25	
26	
28	
39	
72	
141	
290	
300	

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

compare 17 to → 1

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

-75
-31
-26
-4
1
compare 17 to → 6 low = high
25
26 It fails.
28 So return index -1
39 (value 17 not found)
72
141
290
300

```
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                     //  namely,   if mid == low
                    low  = mid + 1
              }
    }
    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)

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

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} ~15 minutes…
$2^{30} \approx 10^9$ ~ one second	30	10^{18} centuries

Faster sorting algorithms ?

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

mergesort
quicksort
heapsort
......

COMP 250

Lecture 20

recursion 2:

binary search
mergesort (part 1)

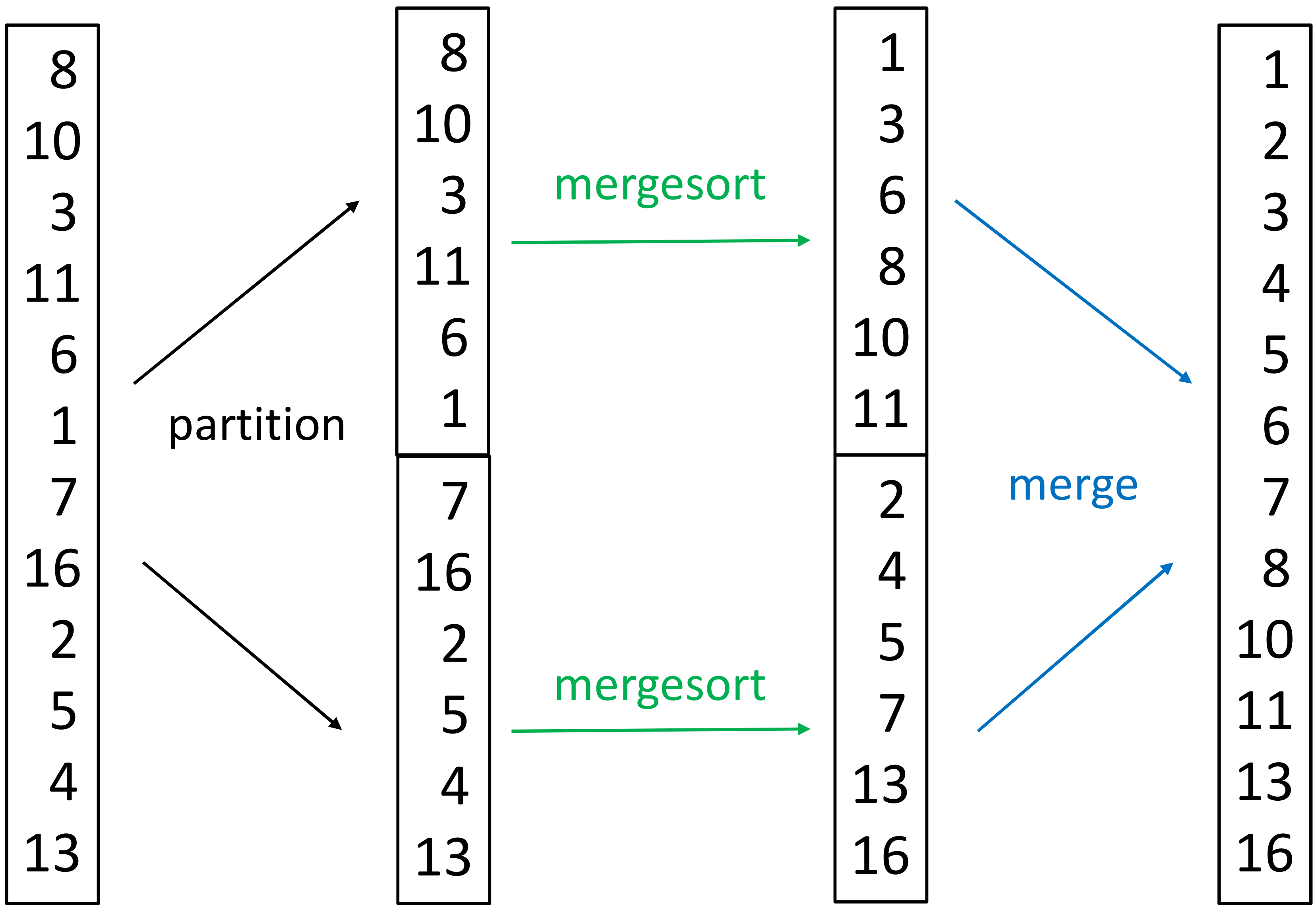
Oct. 25, 2021

Mergesort

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

Sort each half (recursively).

Merge the two halves.



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            //  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 )
    }
}
```

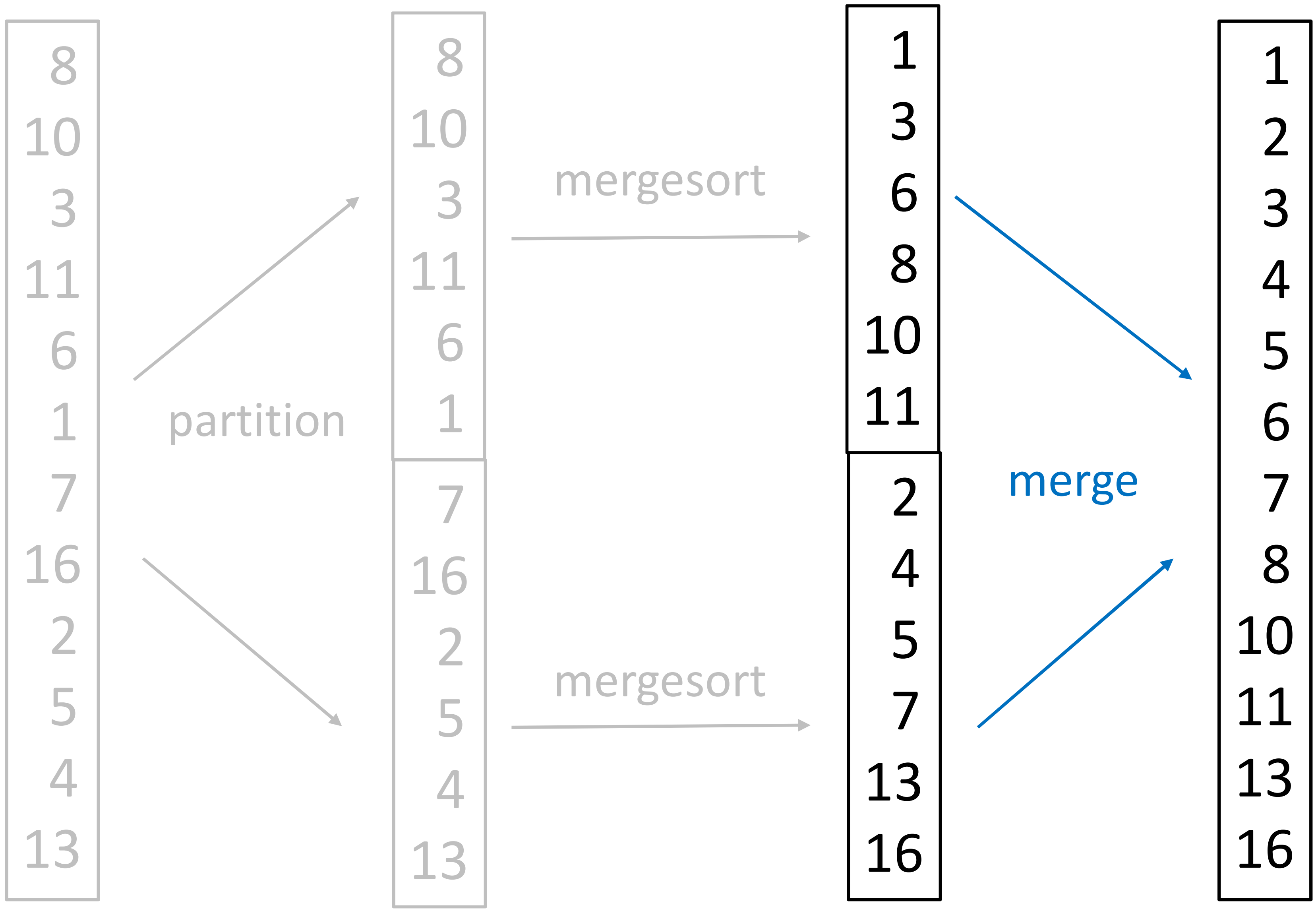
Winter 2022 note

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

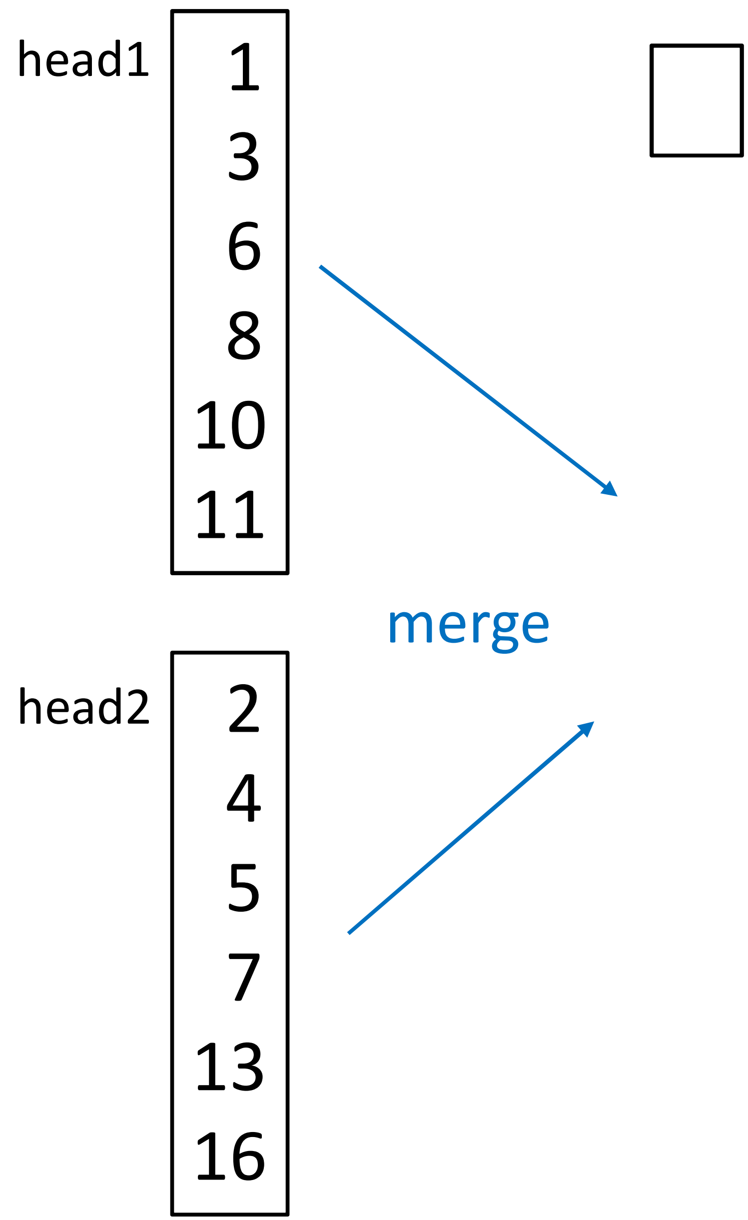
So in practice we would test that the length of the list is greater than 1 before called recursively.   (But note that doing such a test still does take some work, i.e. not free.)

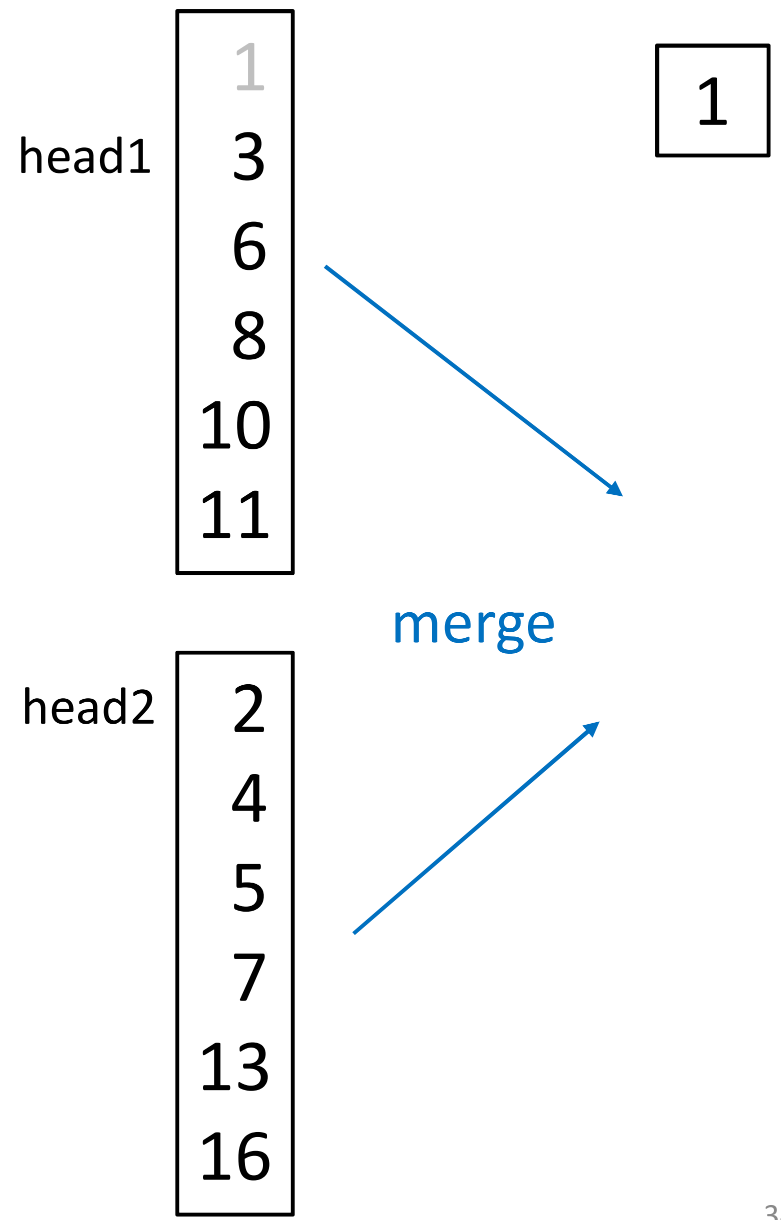
```
mergesort(list){
    if  list.length == 1               //  base case
        return list
    else{
        mid = (list.size - 1) / 2
        list1 =  list.getElements(0,mid)
        list2 =  list.getElements(mid+1, list.size-1)
        list1  =  mergesort(list1)
        list2  =  mergesort(list2)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

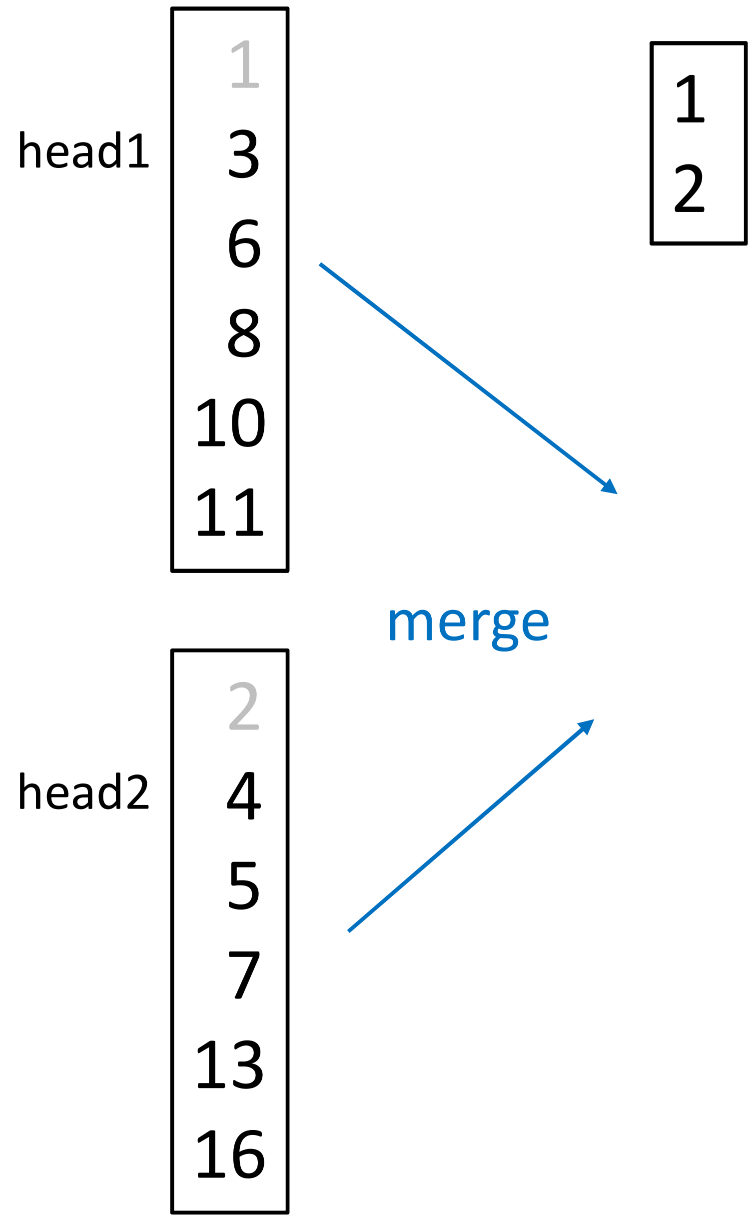
```
mergesort(list){
    if  list.length == 1              //  base case
        return list
    else{
        mid = (list.size - 1) / 2
        list1 =  list.getElements(0,mid)
        list2 =  list.getElements(mid+1, list.size-1)
        list1  =  mergesort(list1)
        list2  =  mergesort(list2)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

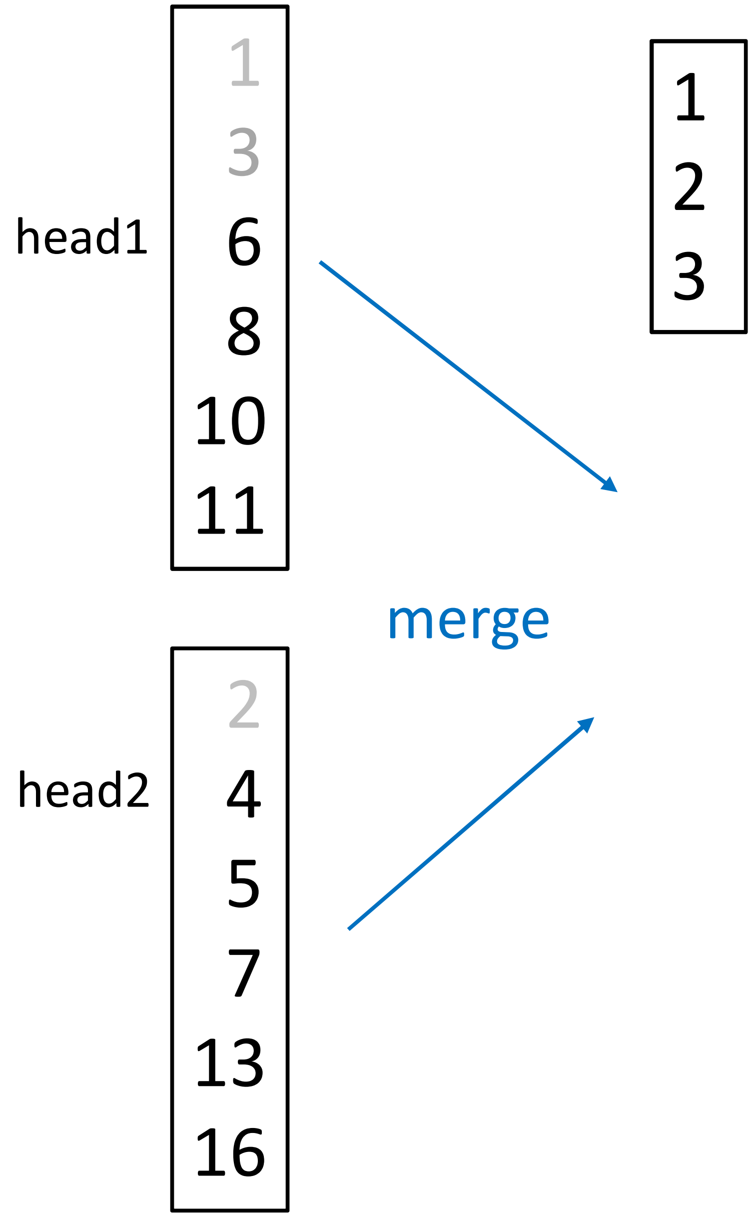


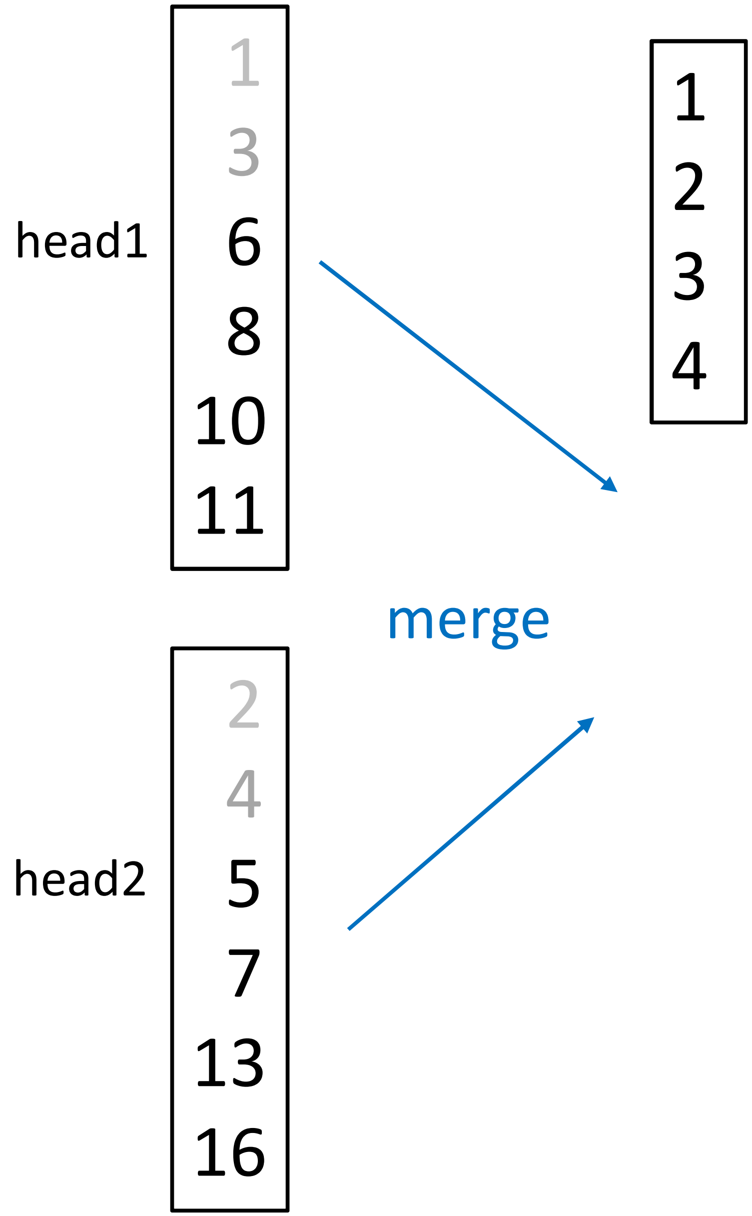
8
10
3
11
6
1
7
16
2
5
4
13
partition
8
10
3
11
6
1
7
16
2
5
4
13
mergesort
mergesort
1
3
6
8
10
11
2
4
5
7
13
16
merge
1
2
3
4
5
6
7
8
10
11
13
16

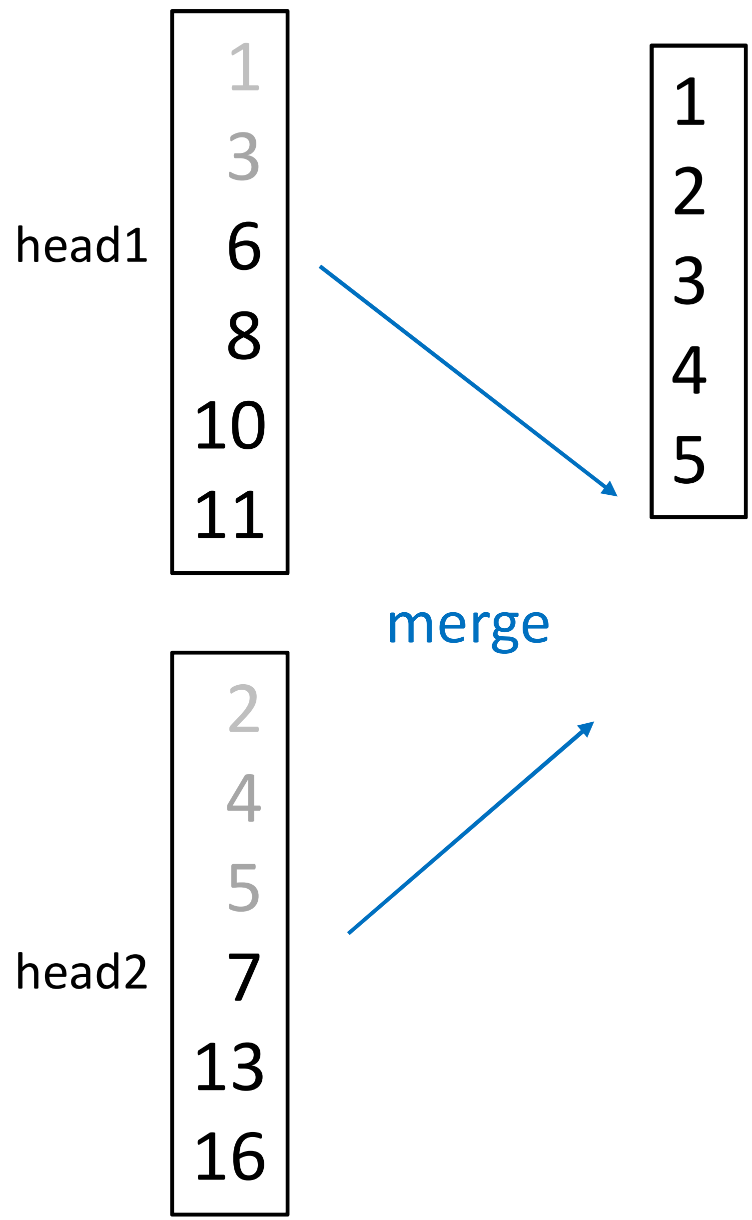

head1
1
3
6
8
10
11
merge
head2
2
4
5
7
13
16

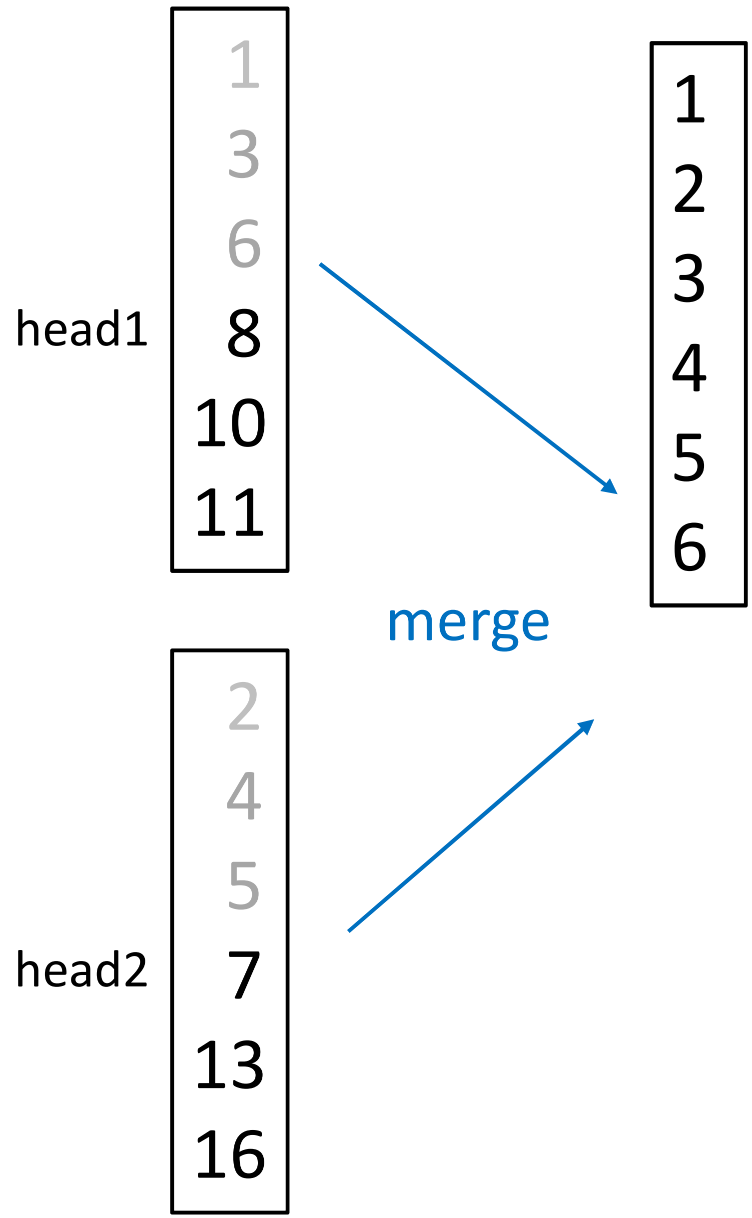

1
head1
3
6
8
10
11
merge
1
head2
2
4
5
7
13
16

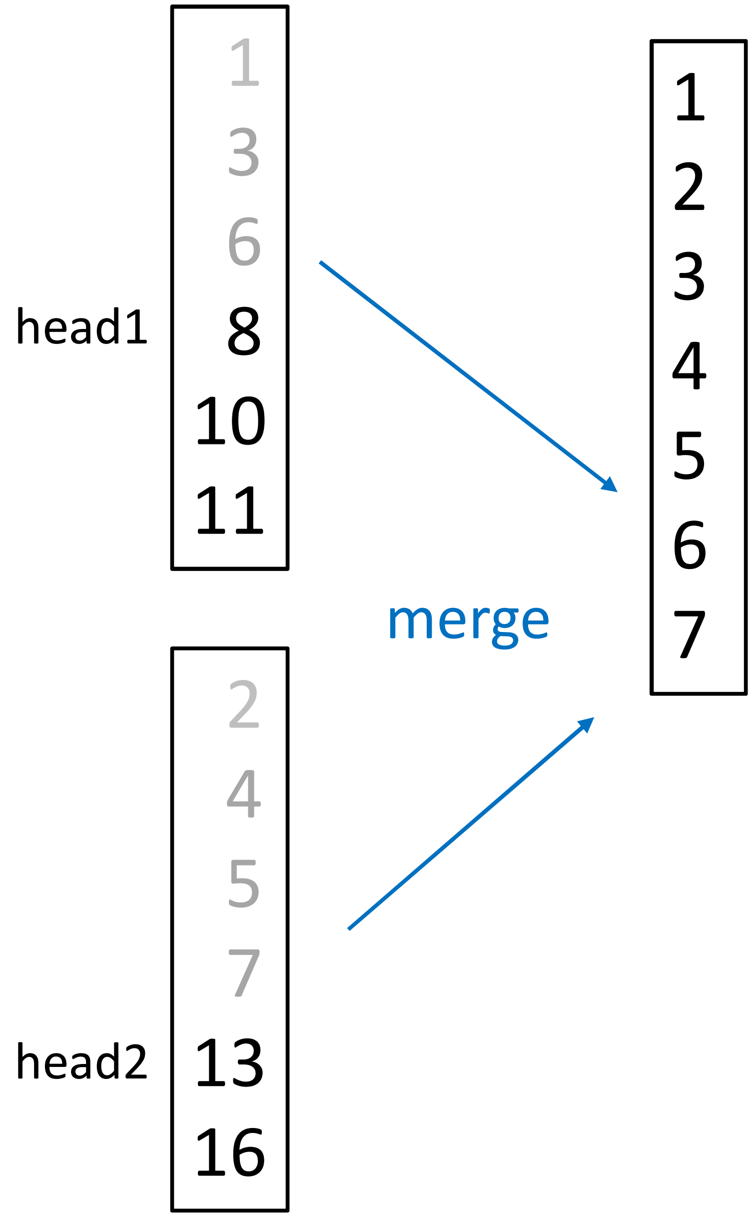

head1
1
3
6
8
10
11
head2
2
4
5
7
13
16
merge
1
2


head1
1
3
6
8
10
11
merge
head2
2
4
5
7
13
16
1
2
3


head1
1
3
6
8
10
11
head2
2
4
5
7
13
16
merge
1
2
3
4

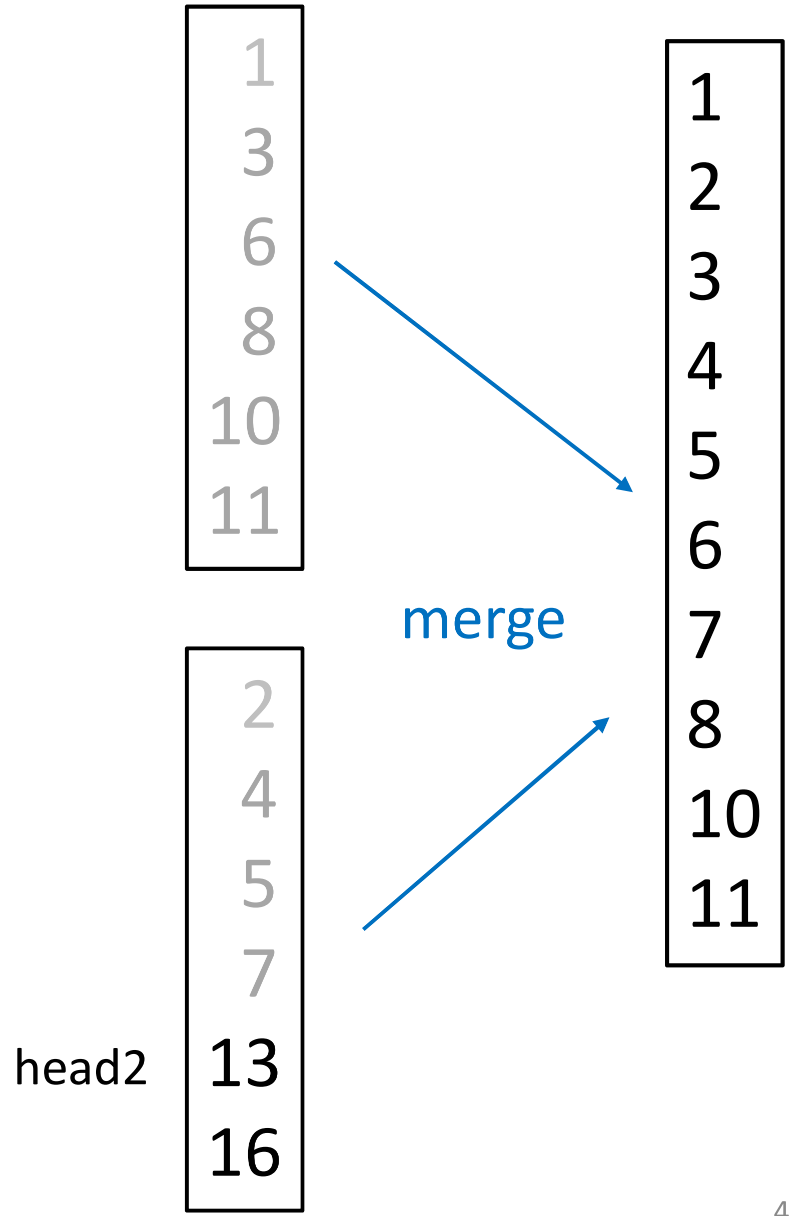

head1
1
3
6
8
10
11
head2
2
4
5
7
13
16
merge
1
2
3
4
5


head1
1
3
6
8
10
11
head2
2
4
5
7
13
16
merge
1
2
3
4
5
6

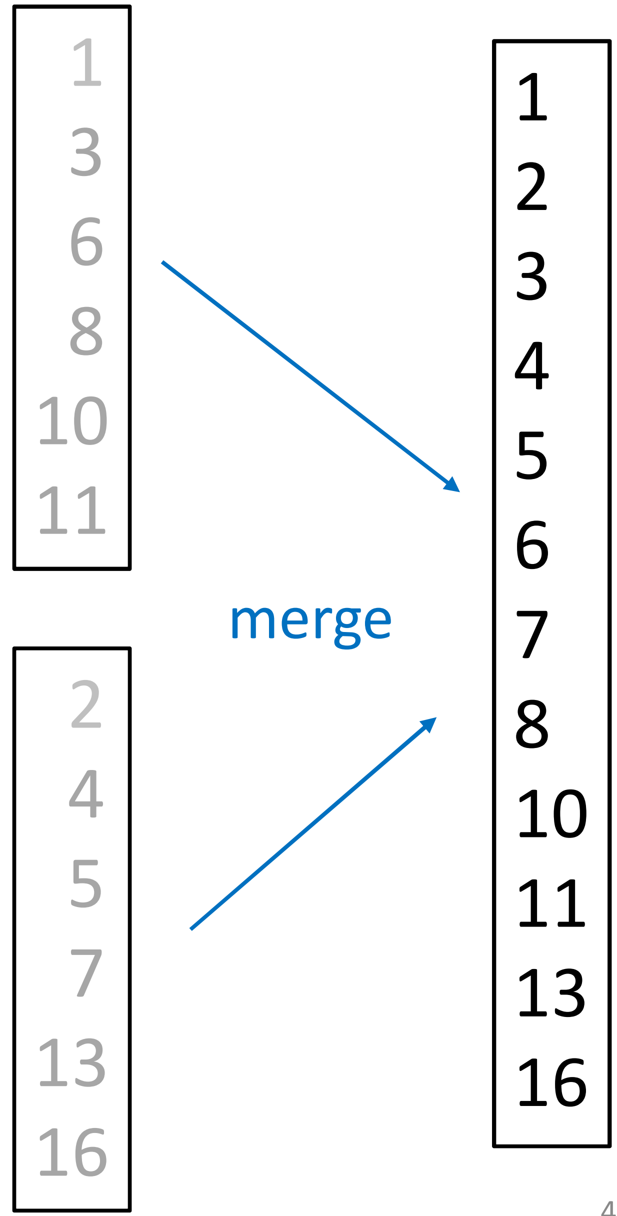


head1
1
3
6
8
10
11
head2
2
4
5
7
13
16
merge
1
2
3
4
5
6
7

....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...