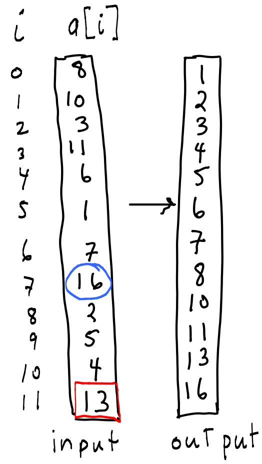
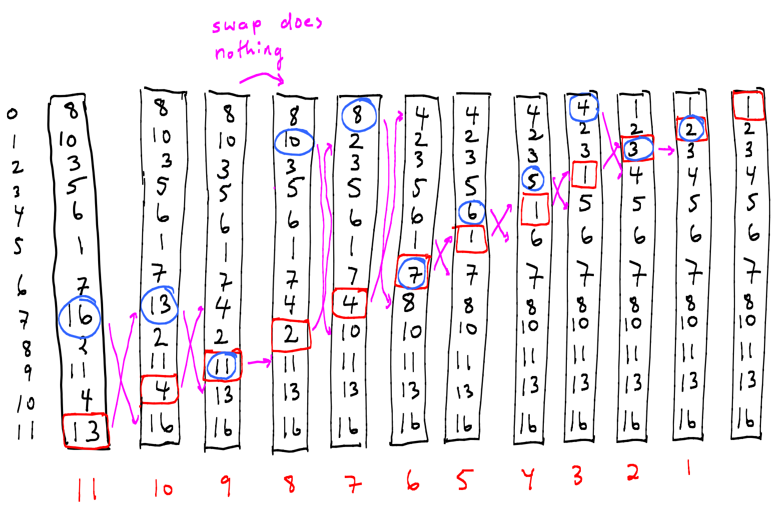
Mergesort

(guest lecture by Michael Langer)

Recall "Selection Sort"




```
i = n-1
while i > 0 {
    indexOfMax = findMax(a, i)
    // finds the location
    // of max in
    //    a[0], .... a[i]
    swap(a, i, indexOfMax)
    i = i-1
}
```



The number of comparisons need to find the max

$$= (n-1) + (n-2) + (n-3) + \cdots \quad 1$$

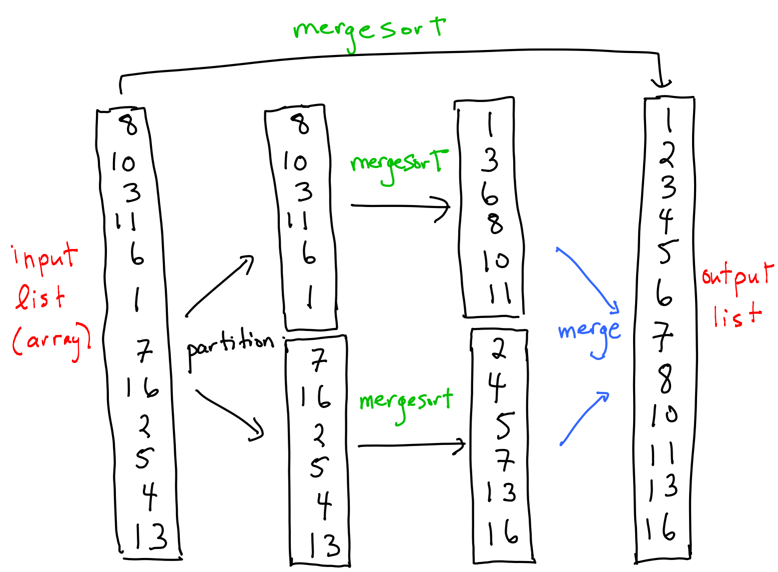
$$= \frac{n(n-1)}{2}$$

Typical computers today can perform $\sim 10^9$ comparisons per second (GHz)

n (size of problem)	n^2 (≈ number of comparisons)	time in seconds
10^3 (~KB)	10^6	10^{-3}
10^6 (~MB)	10^{12}	10^3 (several minutes)
10^9 (~GB)	10^{18}	10^9 (centuries)

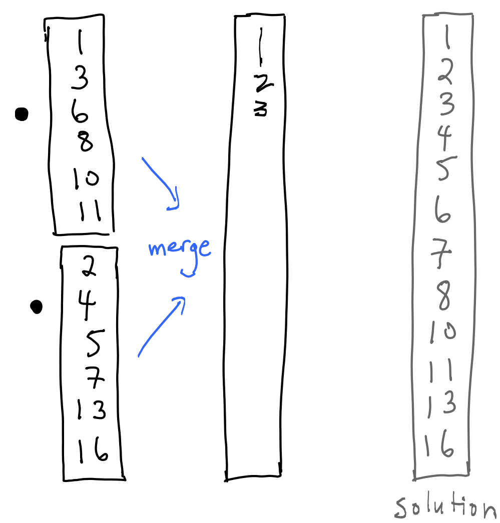
Today we will look at a sorting algorithm that is significantly faster: "merge sort."

It is a recursive algorithm.

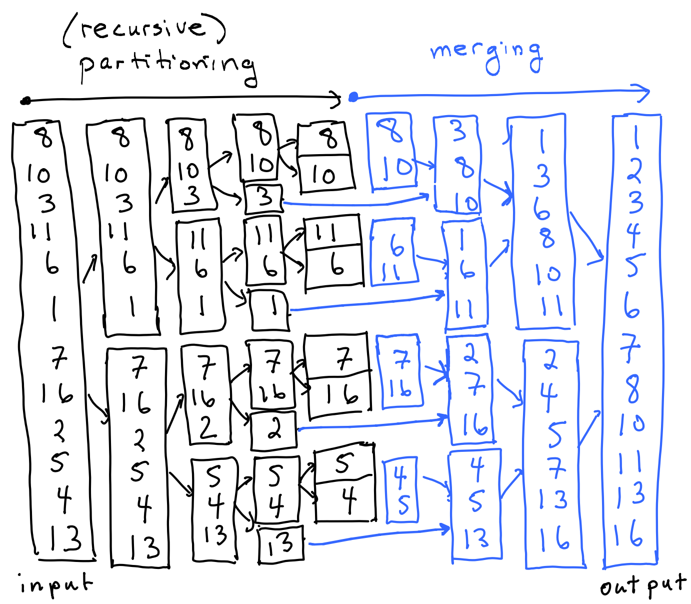



```
mergesort (list) {
    if (list.size == 1)
        return list
    else {
        partition list into two approximately
          equal size lists ℓ1, ℓ2
        ℓ1 ← mergesort (ℓ1)
        ℓ2 ← mergesort (ℓ2)

        return merge (ℓ1, ℓ2)
    }
}
```

```
// Partition list into two approximately
//   equal size lists ℓ1, ℓ2.

mid ←  (list.size - 1) / 2

// getElements(low, high) returns a sublist

ℓ1 ← list.getElements(0, mid)
ℓ2 ← list.getElements(mid+1, size-1)
```

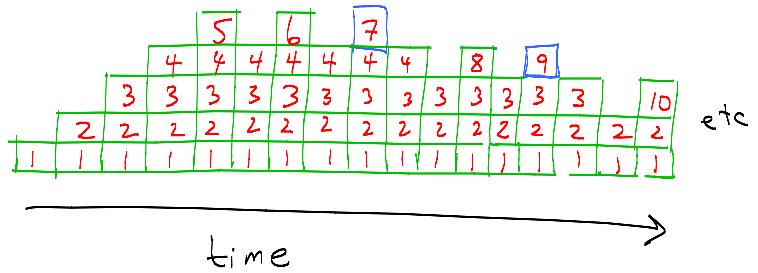



```
merge (ℓ1, ℓ2) {    // two sorted lists
    ℓ ← empty list
    while ℓ1 and ℓ2 are not empty {
        if ℓ1.get(0) < ℓ2.get(0)
            ℓ.addLast (ℓ1.remove(0))
        else
            ℓ.addLast (ℓ2.remove(0))
    }
    while ℓ1 is not empty
        ℓ.addLast (ℓ1.remove(0))
    while ℓ2 is not empty
        ℓ.addLast (ℓ2.remove(0))
    return ℓ
}
```



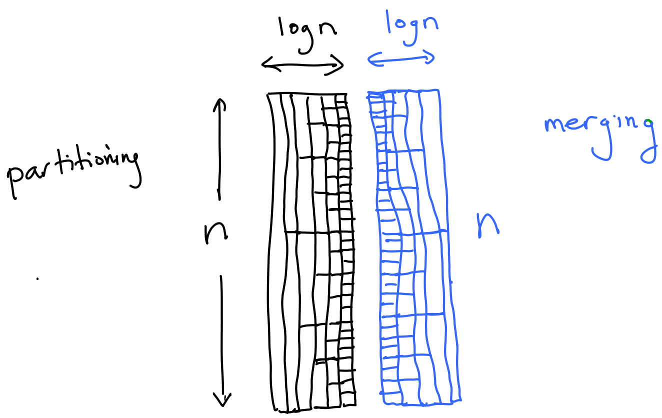
Ordering of method calls

Call Stack (see numbers on previous slide)



How many operations are required to mergesort a list of n elements?

(SKETCH OF IDEA)



n	$\log n$	$n \cdot \log n$	n^2
10^3	≈ 10	10^4	10^6
10^6	≈ 20	2×10^7	10^{12}
10^9	≈ 30	3×10^{10}	10^{18}

wow! minutes centuries