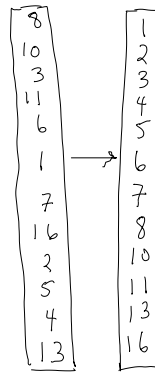


lecture 12

mergesort

Recall Insertion Sort (lecture 3)



for $k = 1$ to $n-1$ {
 // assume elements $0, \dots, k-1$
 // are already sorted.
 insert element k into the
 list of elements $0, \dots, k-1$
 }

Example

after k loops \rightarrow

	initial	1	2	3	4	5
$a[0]$	3	3	-5	-5	-5	-5
$a[1]$	7	17	3	-2	-2	-2
$a[2]$	-5	-5	17	3	3	3
$a[3]$	-2	-2	-2	17	17	4
$a[4]$	23	23	23	23	23	17
$a[5]$	4	4	4	4	4	23

Insertion sort

- worst case - elements are sorted in wrong order. This requires $1 + 2 + 3 + \dots + n-1 = \frac{n(n-1)}{2}$ comparisons & swaps
- best case - elements are already sorted: $n-1$ comparisons, 0 swaps

Typical machines today can perform $\sim 10^9$ operation / sec. (GHz)

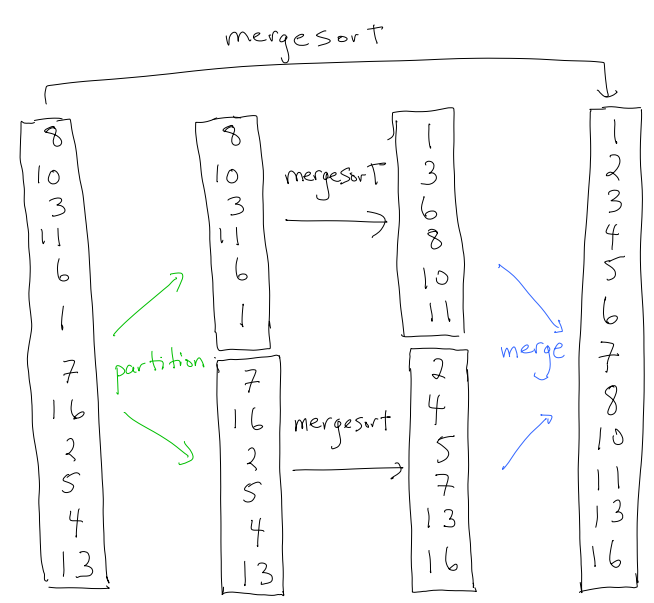
n	n^2	
10^3	10^6	(instant)
10^6	10^{12}	(minutes)
10^9	10^{18}	(centuries)

eg. Computer graphics

3D models can easily have 10^6 elements (point, lines, surface pieces)

that needed to be sorted in depth from eye (to figure out which are visible)

- Insertion sorts takes cn^2 operations in the worst case?
- Are there sorting algorithms that take cn operations in the worst case? **No.**
- What about something in between?
 $cn < ? < cn^2$



```

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

```

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

```

mid ← (list.size - 1) / 2
l1 ← list.getElements(0, mid)
l2 ← list.getElements(mid + 1, size - 1)

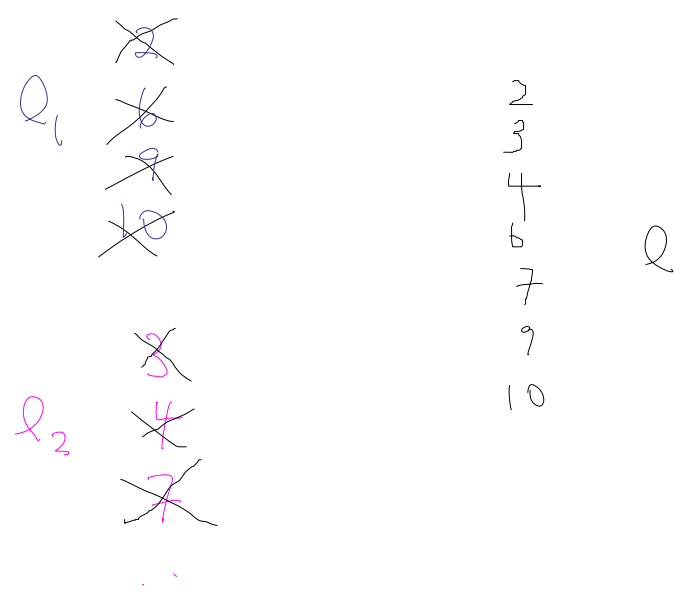
// getElements(low, high) returns a
// sublist

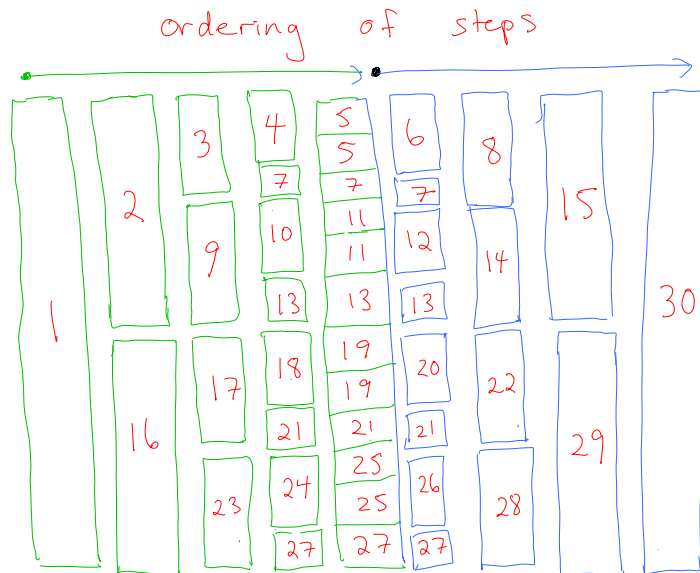
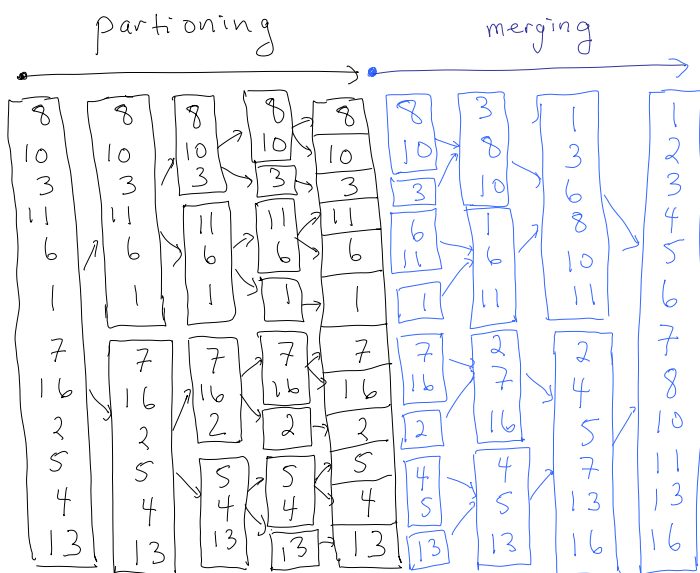
```

```

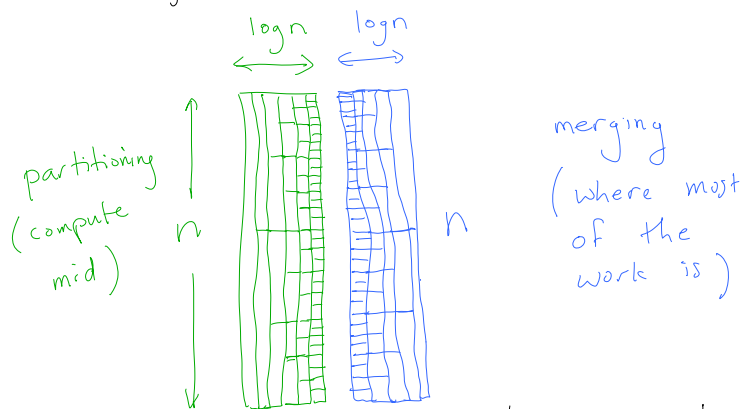
merge (l1, l2) { // two sorted lists
  l ← empty list
  while l1 and l2 are both not empty {
    if l1.get(0) < l2.get(0)
      l.addLast(l1.remove(0))
    else
      l.addLast(l2.remove(0))
  }
  while l1 is not empty
    l.addLast(l1.remove(0))
  while l2 is not empty
    l.addLast(l2.remove(0))
 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

```





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



Idea: we need $\sim c \cdot n \cdot \log_2 n$ operations.

Comparison

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

Upcoming

Friday Feb 4 - Big O

Monday Feb 7 - Not Big O

Wed Feb 9 - recurrences

Fri Feb 11 - midterm.

covers up to today's lecture(12)

Exercises coming soon