# Insertion and merge sort

*Tom Smedsaas*

Two sorting algorithms: insertion sort and merge sort.
A comparison.

# Simple sorting algorithms

Common algorithms to cover are

- Insertion sort
- Selection sort
- Bubble sort

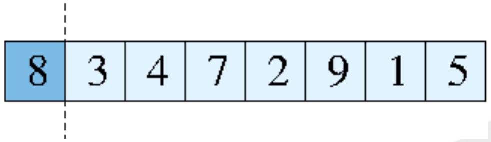
Properties:

- Simple to understand
- Simple to program
- All $\Theta(n^2)$ on the average

# Insertion sort

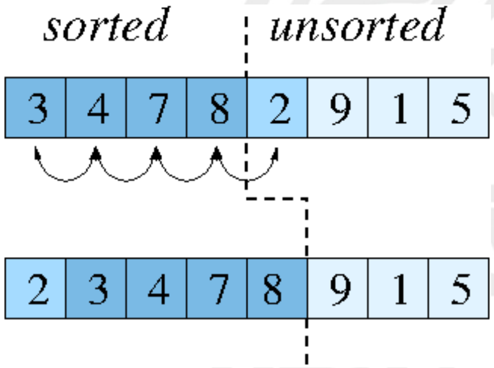
A common way to describe insertion sort is to say that the list consists of two parts – one sorted and one unsorted.

Initially, the sorted part contains only the first element:




For each step, the sorted part is expanded with what is first in the unsorted part

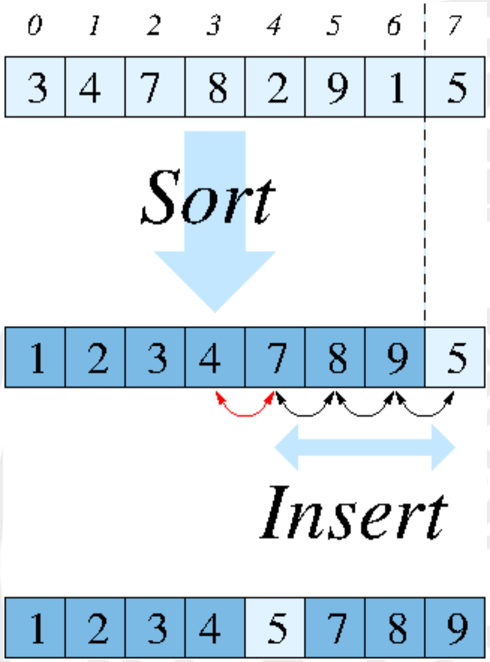
Here's what it might look like at the fourth expansion:



# Insertion sort recursively

To sort a list of $n$ elements, we first sort the first $n-1$ elements and then insert the last element to preserve the order.

```
def ins_sort(lst, n):
    if n <= 1:
        return
    ins_sort(lst, n-1)
    i = n-1
    while i>0 and lst[i] < lst[i-1]:
        lst[i-1], lst[i] = \
            lst[i], lst[i-1]
        i -= 1
```



# Insertion sort analysis – best case

```
def ins_sort(lst, n):
    if n <= 1:
        return
    ins_sort(lst, n-1)
    i = n-1
    while i>0 and lst[i] < lst[i-1]:
        lst[i-1], lst[i] = lst[i], lst[i-1]
        i -= 1
```

Let $t(n)$ be the number of times the while condition is evaluated.

In *best* case (if the values already are sorted):

$$t(n) = t(n-1) + 1 = (t(n-2) + 1) + 1 = \cdots = n$$

# Insertion sort analysis – worst case

```
def ins_sort(lst, n):
    if n <= 1:
        return
    ins_sort(lst, n-1)
    i = n-1
    while i>0 and lst[i] < lst[i-1]:
        lst[i-1], lst[i] = lst[i], lst[i-1]
        i -= 1
```

In *worst* case (if sorted in reverse order):

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

$$= 1 + 2 + \cdots + n = \frac{n \cdot (n+1)}{2}$$

# Insertion sort analysis – average case

```
def ins_sort(lst, n):
    if n <= 1:
        return
    ins_sort(lst, n-1)
    i = n-1
    while i>0 and lst[i] < lst[i-1]:
        lst[i-1], lst[i] = lst[i], lst[i-1]
        i -= 1
```

On the *average* we assume that the elements travel half the way:

$$t(n) = t(n-1) + n/2 = (t(n-2) + (n-1)/2) + n/2 = \cdots = \frac{n \cdot (n+1)}{4}$$

Thus also $\Theta(n^2)$ .

# Balancing the algorithm

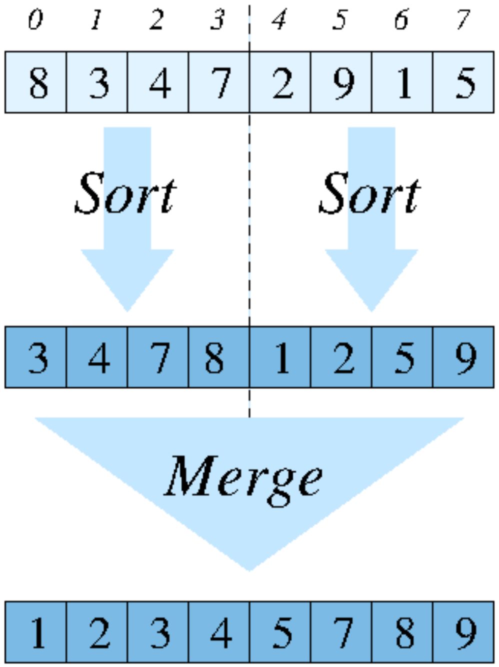
Instead of inserting the elements one by one, you can take several at a time.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 3 | 4 | 7 | 2 | 9 | 1 | 5 |

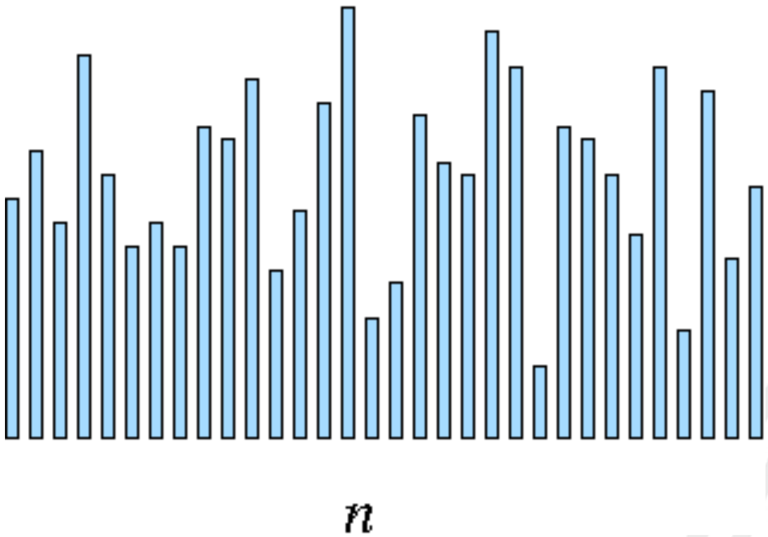
We get the best performance if we divide divide in the middle

1. Partition the set in two equally sized parts.
2. Sort the parts separately
3. Merge the parts

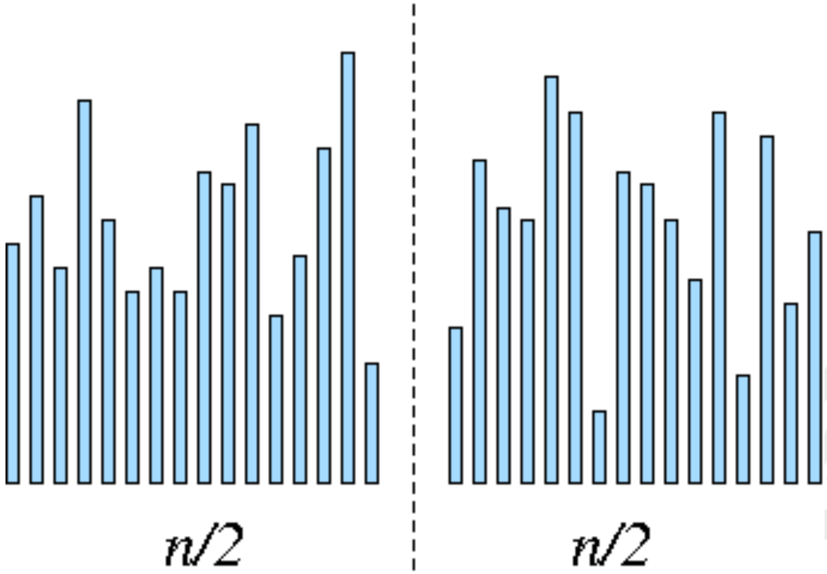
# Merge sort



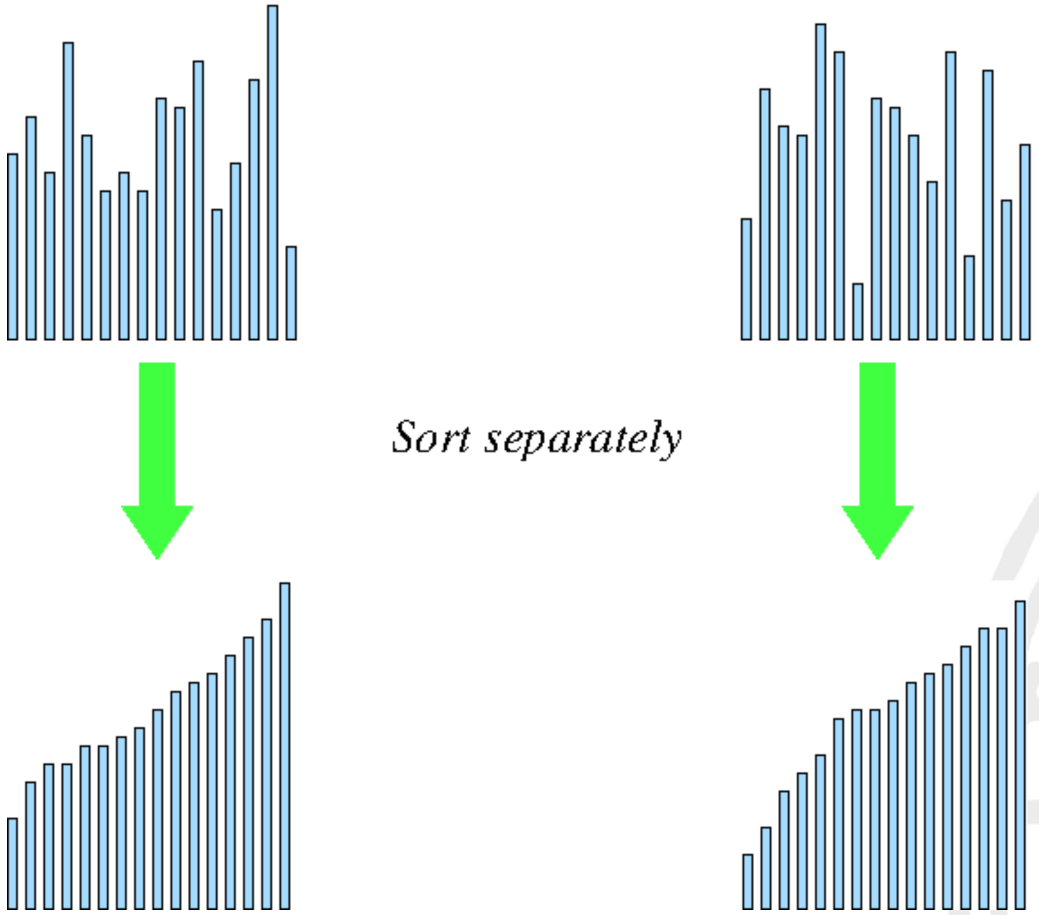
# Mergesort - illustration



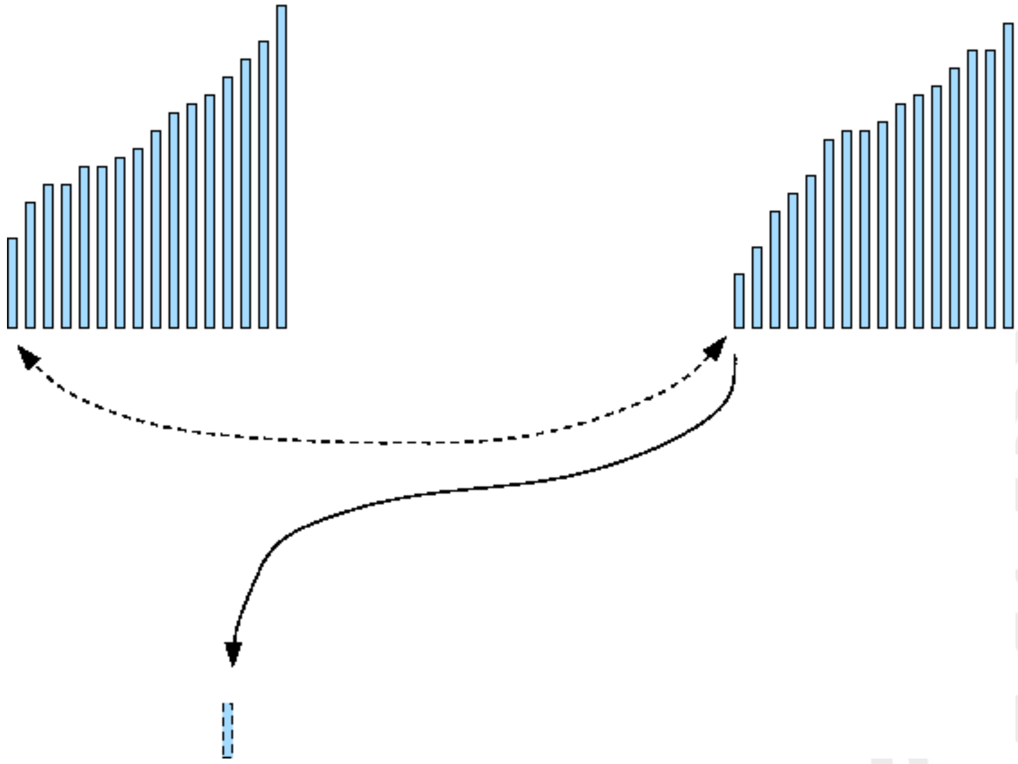
# Merge sort illustration



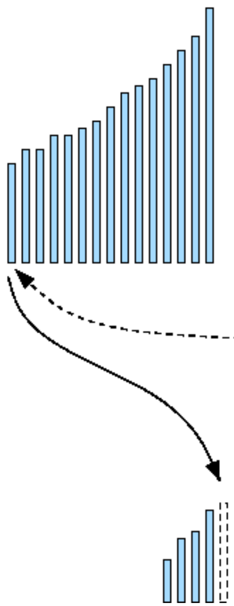
# Merge sort illustration



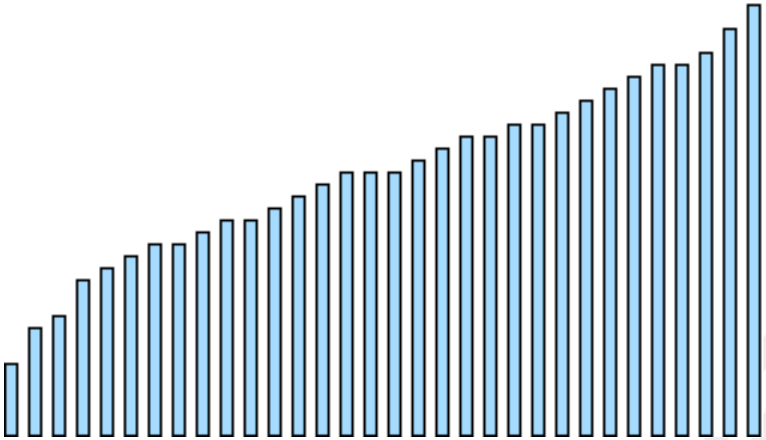
# Merge sort illustration



# Merge sort illustration



# Merge sort illustration



# Analysis of merge sort

We solve a problem of size $n$ by solving two problems of size $n/2$ and merging the two solutions.

The time to merge the two solutions is proportional to $n$.

Let $t(n)$ be the time it takes to sort $n$ elements. Then

$$t(n) = \begin{cases} c & \text{if } n = 0 \\ 2t(n/2) + d \cdot n & \text{if } n > 0 \end{cases}$$

where $c$ and $d$ are unknown constants.

# Analysis of merge sort

If $n$ is an even power of 2, $n = 2^k$, then

$$t(n) = 2t(n/2) + dn =$$

$$= 2\left(2t(n/4) + \frac{dn}{2}\right) + dn =$$

$$= 4t(n/4) + dn + dn =$$

$$= 2^k t(n/2^k) + kdn = nt(1) + dn\log_2 n$$

Thus the complexity of the algorithm is $\Theta(n \log n)$.

This is true even if $n$ is not an even power of two.

# Two questions

- Suppose it takes 1 second to sort 10^3 numbers the simple insertion sort. Then how long will it take to sort 10^6 numbers?

- The same question for merge sort.

# *The end*