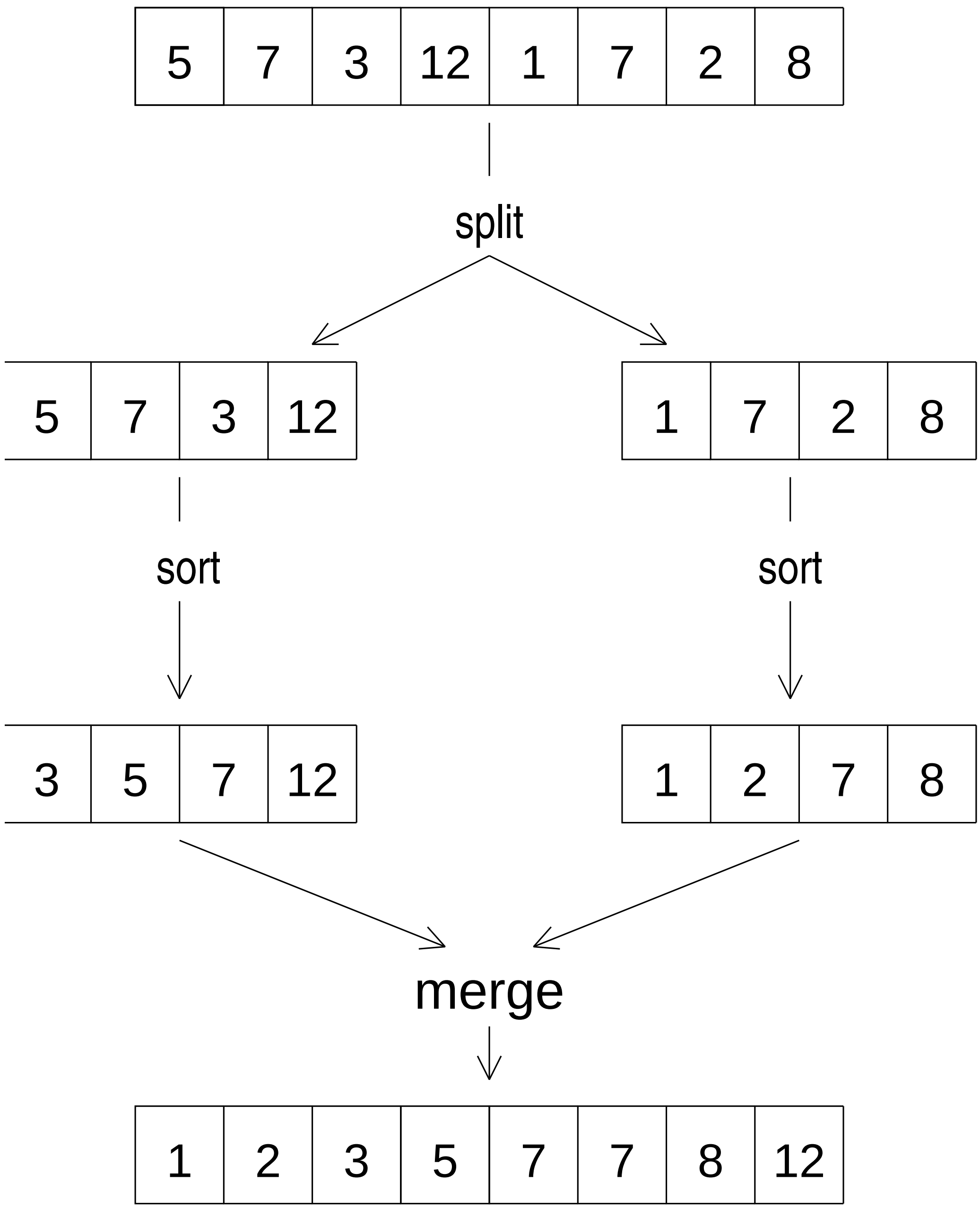
Sorting

(Version of 16 November 2005)

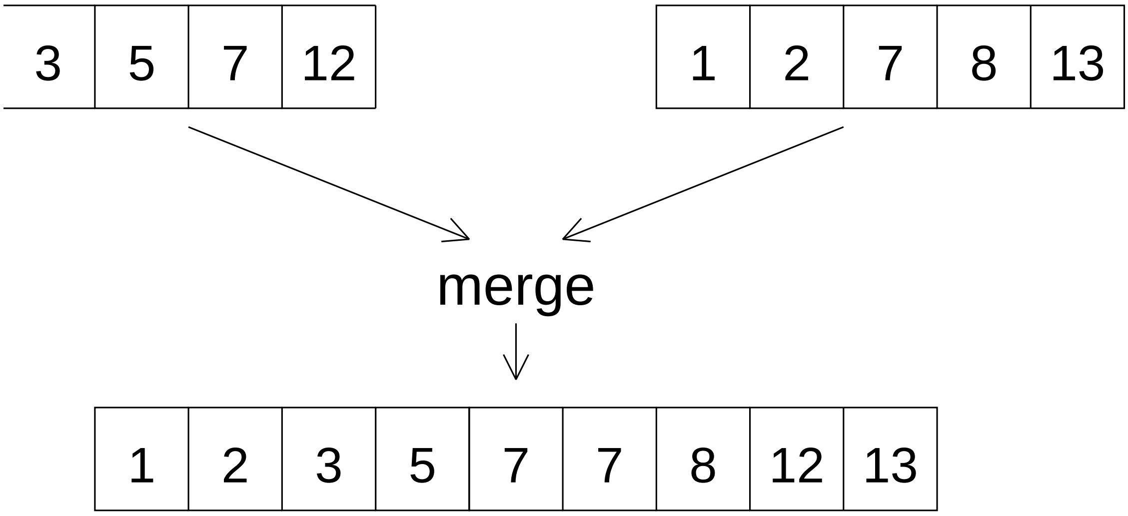
1. Merge Sort

Running time: $\Theta(n \log n)$,
where n is the number of elements to be sorted.

Apply the Divide & Conquer (& Combine) Principle



Merging two sorted lists



Specification

function merge L M
TYPE: int list $\longrightarrow$ int list $\longrightarrow$ int list
PRE: L and M are non-decreasingly sorted
POST: a non-decreasingly sorted permutation of the list L@M

Exercise

Redo all the functions in this chapter for α lists.

Construction

Variant: length(L) · length(M). (Exercise: try length(L) + length(M).)

Base cases
If L is empty, then the result is M.
If M is empty, then the result is L.

General case
Let L be x::xs and let M be y::ys.
If x < y, then x is the minimum of L and M,
and the result is x::zs, where zs is merge xs M.
If x >= y, then y is the minimum of L and M,
and the result is y::zs, where zs is merge L ys.

Note that the recursive calls *do* satisfy the pre-condition, and that the variant *does* get smaller.

SML program

```
fun  merge [ ] M = M
 |   merge L [ ] = L
 |   merge (L as x::xs) (M as y::ys) =
           if  x < y
           then  x :: (merge xs M)
           else  y :: (merge L ys)
```

Running time: $O(|L| + |M|)$

Splitting a list into two 'halves'

Specification

function split L
TYPE: α list $\longrightarrow$ (α list $*$ α list)
PRE: (none)
POST: (A,B) such that A@B is a permutation of L
while A and B are of the same length, up to one element

Note that the order of the elements in A and B is irrelevant!

Naive SML program

```
fun split L =
   let
      val t = (length L) div 2
   in
      ( List.take (L,t) , List.drop (L,t) )
   end
```

- Running time: $n + \lfloor \frac{n}{2} \rfloor + \lfloor \frac{n}{2} \rfloor = \Theta(n)$, where n is the length of L.
- How to realise split with a *single* traversal of L?!

Merge sort

Specification

function sort L
TYPE: int list $\longrightarrow$ int list
PRE: (none)
POST: a non-decreasingly sorted permutation of L

SML Program

Variant: length(L).

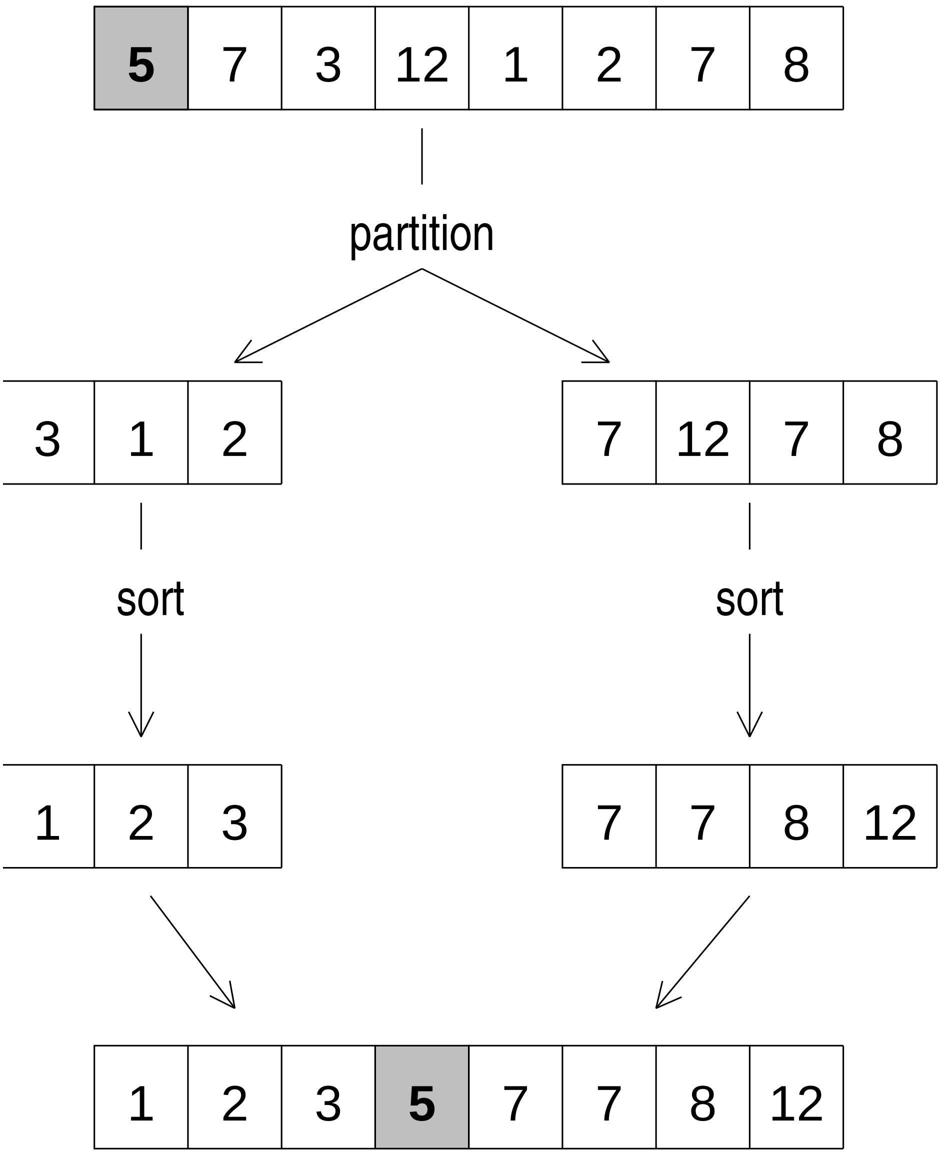
```
fun sort [ ] = [ ]
 |  sort [x] = [x]
 |  sort xs =
      let
         val (ys,zs) = split xs
      in
         merge (sort ys) (sort zs)
      end
```

Why is the base case sort [x] indispensable?!

2. Quicksort

A sorting method proposed by C.A.R. Hoare, in 1962.
Average-case running time: $\Theta(n \log n)$,
where n is the number of elements to be sorted.

Application of the Divide & Conquer Principle



Specification

The same as for merge sort!

SML program

```
fun sort [ ] = [ ]
 |  sort (x::xs) =
      let val (S,B) = partition (x,xs)
      in (sort S) @ (x :: (sort B))
      end
```

- Double recursion and no tail-recursion
- Average-case running time: $\Theta(n \log n)$
- Usage of X @ Y (concatenation), which is $\Theta(|X|)$

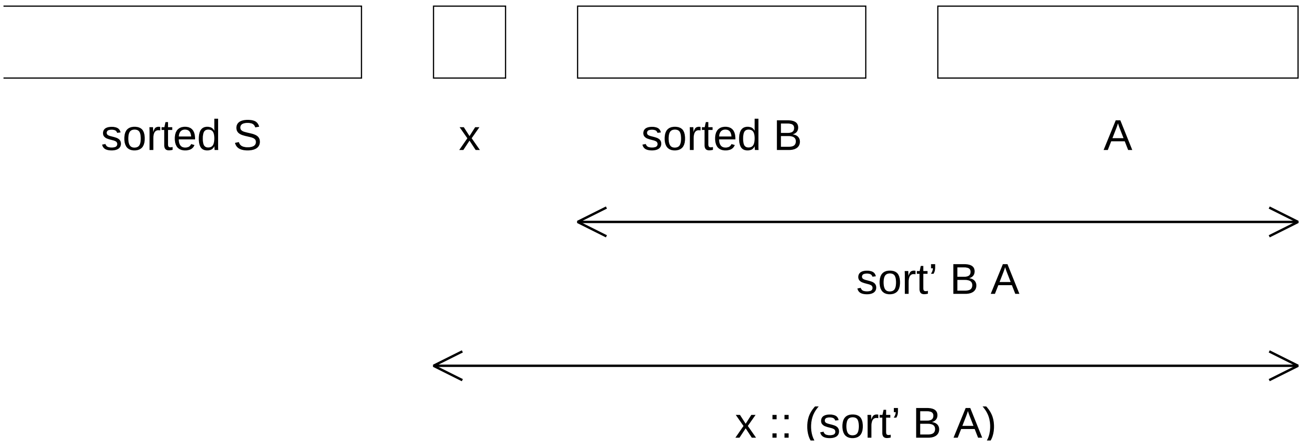
Help function: partition

```
function partition (p,L)
TYPE: int * int list → int list * int list
PRE:  (none)
POST: (S,B) where S has all x<p of L and B has all x≥p of L
```

```
fun partition (p,[ ]) = ([ ],[ ])
 |  partition (p,x::xs) =
      let val (S,B) = partition (p,xs)
      in  if x < p then (x::S,B)
          else (S,x::B)
      end
```

- Running time: $\Theta(|L|)$

Generalisation



function sort' L A
TYPE: int list → int list → int list
PRE: (none)
POST: (a non-decreasingly sorted permutation of L) @ A
(Exercise: try POST: A @ (a non-decreasingly sorted permutation of L))

```
local
   fun sort' [ ] A = A
    |  sort' (x::xs) A =
          let val (S,B) = partition (x,xs)
          in sort' S (x :: (sort' B A))
          end
in fun sort2 L = sort' L [ ]
end
```

- Double recursion, but one tail-recursion
- No usage of @ (no concatenation)
- Average-case running time: again $\Theta(n \log n)$, but less space consumption