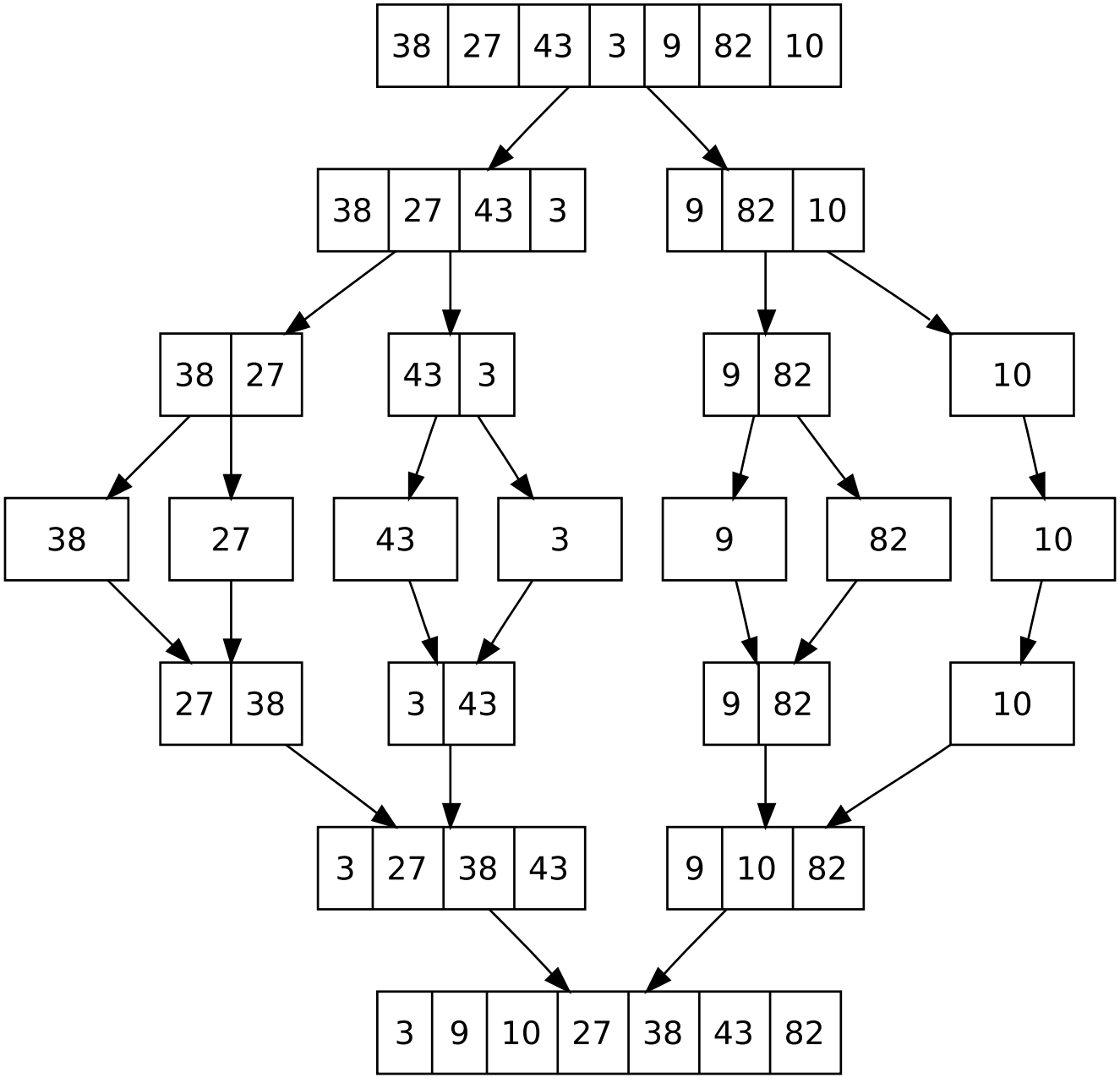
Programmeren (Ectrie)

Lecture 7: Sorting continued, searching

Tommi Tervonen

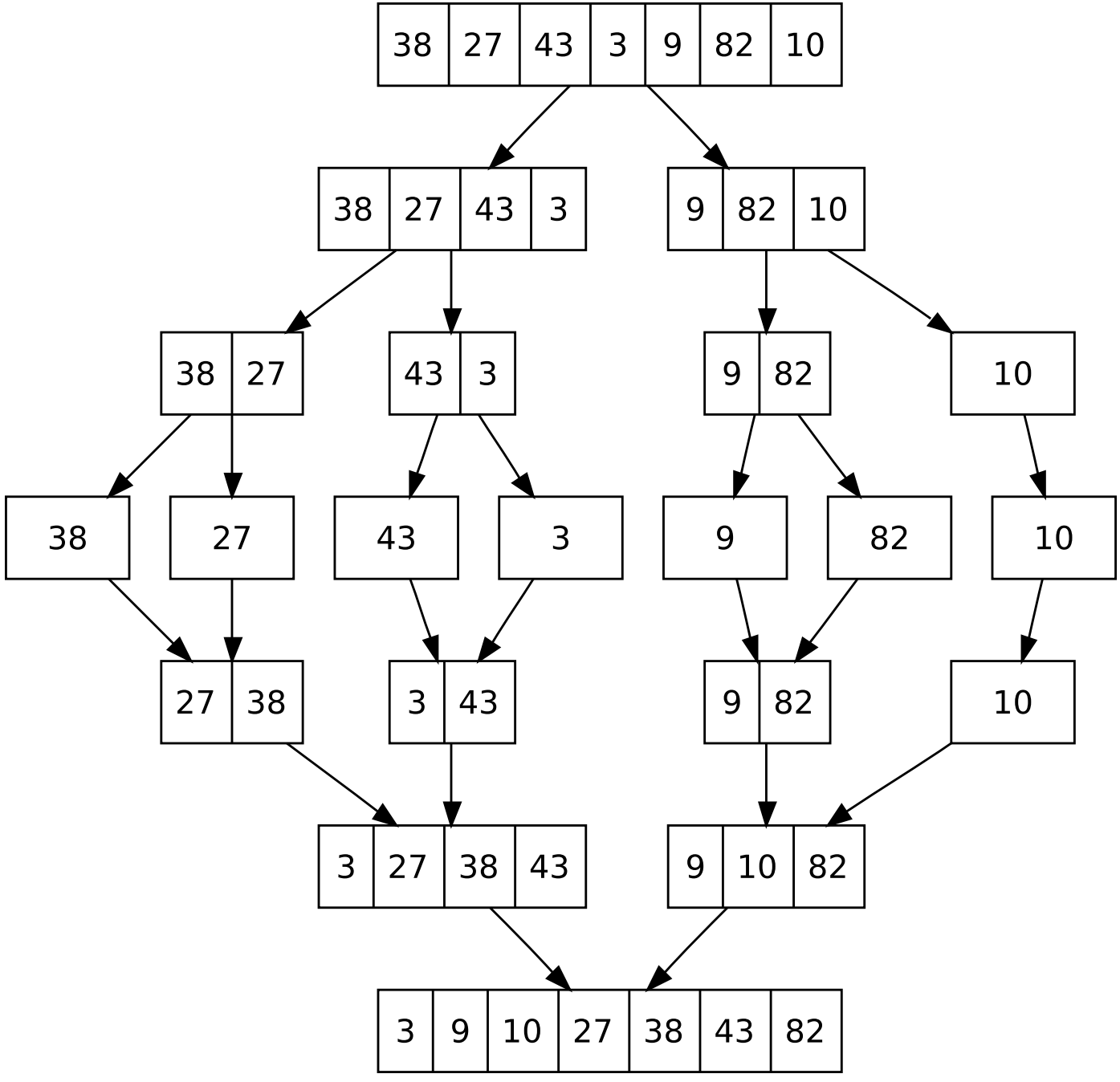
Econometric Institute, Erasmus University Rotterdam

Mergesort



Mergesort: algorithms

```
function A = mergeSort(A)
  if (length(A) > 1)
  % if not, it's our trivial case
    middle = floor(length(A) / 2);
    leftList = A(1:middle);
    rightList = A((middle+1):length(A));
    leftList = mergeSort(leftList);
    rightList = mergeSort(rightList);
    A = merge(leftList, rightList);
  end
end
```


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

```
function c = merge(a, b)
  lena = length(a);
  lenb = length(b);
  c=zeros(1,lena+lenb);
  inda = 1; % index to move along vector 'a'
  indb = 1; % index to move along vector 'b'
  indc = 1; % index to move along vector 'c'

  while ((inda <= lena) && (indb <= lenb))
    if a(inda) < b(indb)
      c(indc) = a(inda);
      inda = inda + 1;
    else
      c(indc) = b(indb);
      indb = indb + 1;
    end
    indc = indc + 1;
  end
  ...
```

```
    ...
    % copy any remaining elements of the 'a' into 'c'
    while (inda <= lena)
      c(indc) = a(inda);
      indc = indc + 1;
      inda = inda + 1;
    end
    % copy any remaining elements of the 'b' into 'c'
    while (indb <= lenb)
      c(indc) = b(indb);
      indc = indc + 1;
      indb = indb + 1;
    end
end
```

Mergesort: analysis

- Each step, two recursion steps with half size input (divide)
- After division, merge the lists: O(n)

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

- Does NOT sort in place, but requires O(n) memory

Quicksort

1. Choose a pivot element (e.g. first)
2. Partition array so, that elements left are $\leq$ pivot, and elements right are $>$ pivot
3. Sort recursively until size < 2

Video: Quicksort with Hungarian folk dancers

QS complexity analysis: worst case

$$
\begin{aligned}
T(n) &= T(n-1) + O(n) \\
&= \sum_{k=1}^{n} O(k) \\
&= O(\sum_{k=1}^{n} k) \\
&= O(n^2)
\end{aligned}
$$

QS complexity analysis: base case

$$
\begin{aligned}
T(n) &= 2T(n/2) + O(n) \\
&= O(n \log_2 n)
\end{aligned}
$$

QS analysis: guaranteed partitioning

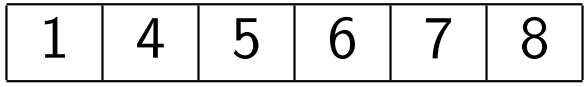
$$
\begin{aligned}
T(n) &= T(9n/10) + T(n/10) + n \\
&= O(n \log_{10/9} n) \\
&= O(n \log\ n)
\end{aligned}
$$

- Similarly for any 1-to-x partitioning, where x is a constant

Randomized quicksort

```
function A = randomizedPartition(A, p, r)
  i = p + (round(rand(1) * (r-p)));
  swap(A, i, p); % pseudo-code
  A = partition(A, p, r);
end
```

Binary search



- Min-heap
- BST
- Array

Where we started

- Elementary imperative programming: variables, control structures, decisions, operators, methods
- Elementary OO programming: classes, data hiding

Where we are now

- Computational complexity
- Memory organization
- Fundamental algorithms: sorting (insertion, bubble, heap, merge, quick)
- Fundamental data structures: array, stack, queue, linked list, (binary) tree, heap
- Program correctness: pre- and post-conditions, loop invariant, halting problem
- ... Matlab!

What next?

- Other courses: to use what you have learned in this one
- Voortgezet Programmeren: object oriented programming, abstract data types, software development

C u in the exam!