# CM2035 Topic 2 Recursive Algorithms

Ian Sanders

April 2025 Session

UNIVERSITY OF LONDON

# About me

- I have a PhD in Computer Science from the University of Pretoria, South Africa.
- I have taught at universities in South Africa and the United States of America.
- I have typically taught courses about algorithms, data structures, and computer science theory but I have taught other topics too.
- I have done research in Computer Science Education.
- I am officially retired.

UNIVERSITY OF LONDON

# Recap

In the module videos we have seen

- understanding recursive algorithms,
- the structure of recursive algorithms,
- writing recursive algorithms, and
- analysing recursive algorithms.

UNIVERSITY OF LONDON

# Another example

Suppose we are asked to develop an algorithm to sum up a list of numbers stored in an array.
For example, if we are given the list of numbers 3, 5, 6, 8, 2, 1, 9, 4, 7 then our function should return 45.
The iterative algorithm is relatively easy to develop.

# Iterative sum

```
00   function sum(mylist, n)
         Input:   mylist, n where mylist is an array
         with n entries indexed 0...(n − 1).
         Output:   sum the sum of the numbers in mylist.
01       sum ← 0
02       for 0 ≤ i < n
03           sum ← sum + mylist[i]
04       return sum
```

# Iterative sum continued

Using what we have learnt in Topic 1 it can be shown that Iterative Sum is in $\Theta(n)$

**Exercises**

1. Make sure that you understand how the algorithm works.
2. Make sure you know how to do the analysis of the algorithm.

# Recursive sum

```
00   function sumr(mylist, n)
        Input:  mylist, n where mylist is an array
        with n entries indexed 0...(n − 1).
        Output:  sum the sum of the numbers in mylist.
01      if n == 0
02         return 0
03      else
04         return mylist[n − 1] + sumr(mylist, n − 1)
```

# Tracing this recursive algorithm

Suppose we have as our input list [3, 5, 4, 2]
The first call to the function is sumr([3, 5, 4, 2], 4)
If we now trace the recursive flow we will see.

sumr([3, 5, 4, 2], 4)
    sumr([3, 5, 4], 3)
        sumr([3, 5], 2)
            sumr([3], 1)
                sumr([], 0)
                return 0
            return $3 + 0 = 3$
        return $5 + 3 = 8$
    return $4 + 8 = 12$
return $2 + 12 = 14$

# Analysis of this recursive algorithm

The analysis of this algorithm is similar to that of the recursive factorial algorithm from the videos.
We get a similar *recurrence relation*.
$T(n) = T(n-1) + C_1$
$T(n-1) = T(n-2) + C_1$
...
$T(0) = C_2$
From this we get, $T(n) = C_1 * n + C_2$
This implies $T(n) \in \Theta(n)$

# Another approach

Recursion is an approach in computer science that solves problems by breaking them down into smaller instances of the same problem.
In the example above we solved the problem for a list of length $n$ in terms of solving the problem for a list of length $n-1$.
And we solved the problem of a list of length $n-1$ in terms of solving the problem for a list of length $n-2$.
And so on until we solved a very small problem directly.
Suppose now that we do something slightly different.

# A *divide and conquer approach*

We solve a problem of size $n$ by splitting it into two problems of size $n/2$ and solving those in the same way.
Below we look at using this approach to solve our sum of numbers problem.
In the discussion to come we will consider the length of the list to be a power of 2 but the approach will work (with minor modifications) for all lists.
Clearly the sum of a list of numbers = the sum of the left half of the list plus the sum of the right half of the list.
sum of [2, 4, 6, 1] = sum of [2, 4] + sum of [6, 1]
We use this idea in the recursive algorithm below.

# A *divide and conquer approach*

```
00   function sumdc(mylist, l, r)
        Input:  mylist, n where mylist is an array
        with n entries indexed 0...(n − 1).
        Output:  sum the sum of the numbers in mylist.
01      if l == r
02         return mylist[l]
03      else
04         mid ← (l + r)//2
05         leftsum ← sumdc(mylist, l, mid)
06         rightsum ← sumdc(mylist, mid + 1, r)
07         return leftsum + rightsum
```

UNIVERSITY OF LONDON

# Tracing our new algorithm

The first call to the function is `sumdc([3, 5, 4, 2], 0, 3)`
In the first instantiation of `sumdc`
$l = 0$ and $r = 3$ and so $mid \leftarrow (0 + 3)//2 = 1$
Then in line 05 `sumdc` is called again
`sumdc([3, 5, 4, 2], 0, 1)`

UNIVERSITY OF LONDON

# Tracing our new algorithm

sumdc([3, 5, 4, 2], 0, 3)
    sumdc([3, 5, 4, 2], 0, 1)
        sumdc([3, 5, 4, 2], 0, 0)
        return mylist[0] = 3
        sumdc([3, 5, 4, 2], 1, 1)
        return mylist[1] = 5
    return $3 + 5 = 8$

    sumdc([3, 5, 4, 2], 2, 3)
        sumdc([3, 5, 4, 2], 2, 2)
        return 4
        sumdc([3, 5, 4, 2], 3, 3)
        return 2
    return $4 + 2 = 6$
return $8 + 6 = 14$

UNIVERSITY OF LONDON

# Exercise:

Trace the new recursive algorithm on the list of numbers
2, 4, 3, 7, 8, 1, 5, 9

UNIVERSITY
OF LONDON

# Analysis

Once again we can describe the work done to solve the problem in terms of the work done to solve a smaller problem.

$$T(n) = 2 \times T(\tfrac{n}{2}) + C_0$$
$$T(\tfrac{n}{2}) = 2 \times T(\tfrac{n}{4}) + C_0$$

...

$$T(1) = C_1$$

Suppose now that $n = 8$

$$
\begin{aligned}
T(8) &= 2 \times T(4) + C_0 && (1)\\
&= 2 \times (2 \times T(2) + C_0) + C_0 && (2)\\
&= 2 \times (2 \times (2 \times T(1) + C_0) + C_0) + C_0 && (3)\\
&= 2 \times (2 \times (2 \times C_1 + C_0) + C_0) + C_0 && (4)\\
&= 2 \times (4 \times C_1 + 2 \times C_0 + C_0) + C_0 && (5)\\
&= (8 \times C_1 + 4 \times C_0 + 2 \times C_0) + C_0 && (6)\\
&= 8 \times C_1 + 7 \times C_0 && (7)
\end{aligned}
$$

This approach can be done for other values of $N$ as well.
In general $T(n) = n \times A + B$
Our algorithm is thus in $\Theta(n)$.

UNIVERSITY OF LONDON

# Master Theorem

Apply the Master Theorem to find the time complexity of the given recurrence relation.
Identify $a$, $b$, and $f(n)$.
From $(T(n) = 2T(\frac{n}{2}) + 1)$, we have: $(a = 2)$, $(b = 2)$ and $(f(n) = 1)$
Calculate $n^{\log_b a}$: $n^{\log_b a} = n^{\log_2 2} = n^1 = n$
Compare $f(n)$ with $n^{\log_b a}$, $f(n) = 1$ and $n^{\log_b a} = n$
Since $(1 < (n^{\log_b a}))$, we are in Case 1 of the Master Theorem.
Apply Case 1 of the Master Theorem
$T(n) = \Theta(n^{\log_b a})$.
Therefore, $T(n) = \Theta(n)$.
The time complexity of the recurrence relation is $\Theta(n)$.

UNIVERSITY OF LONDON

# Comments

1. Recursive algorithms where the recursive call is in the last line of the recursive function are generally called *tail recursion*.
2. Recursive functions like the last one we looked at are said to have *embedded* recursion. Something happens before and after the recursive call[s].
3. Divide and conquer recursive algorithms are used to solve many problems in computer science.
   You will study some of them later in the module.