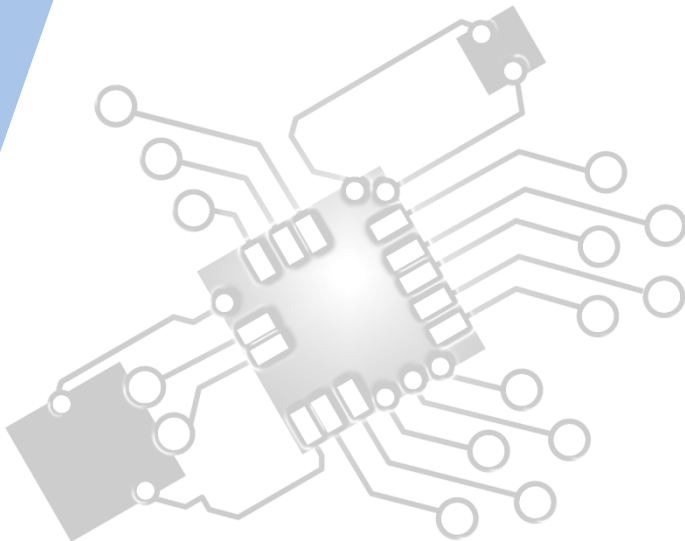




# ***Abstract Data Structures***

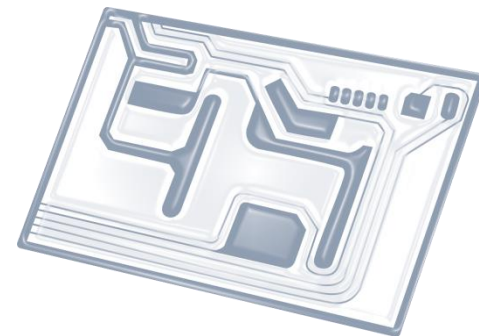
**IB Computer Science**



*Content developed by  
**Dartford Grammar School**  
Computer Science Department*



# HL Topics 1-7, D1-4



1: System design



2: Computer Organisation



3: Networks



4: Computational thinking



5: Abstract data structures



6: Resource management



7: Control



D: OOP

# HL only 5 Overview

## Thinking recursively

- 5.1.1 Identify a situation that requires the use of recursive thinking
- 5.1.2 Identify recursive thinking in a specified problem solution
- 5.1.3 Trace a recursive algorithm to express a solution to a problem

## Abstract data structures

- 5.1.4 Describe the characteristics of a two-dimensional array
- 5.1.5 Construct algorithms using two-dimensional arrays
- 5.1.6 Describe the characteristics and applications of a stack
- 5.1.7 Construct algorithms using the access methods of a stack
- 5.1.8 Describe the characteristics and applications of a queue
- 5.1.9 Construct algorithms using the access methods of a queue
- 5.1.10 Explain the use of arrays as static stacks and queues

## Linked lists

- 5.1.11 Describe the features and characteristics of a dynamic data structure
- 5.1.12 Describe how linked lists operate logically
- 5.1.13 Sketch linked lists (single, double and circular)

## Trees

- 5.1.14 Describe how trees operate logically (both binary and non-binary)
- 5.1.15 Define the terms: parent, left-child, right-child, subtree, root and leaf
- 5.1.16 State the result of inorder, postorder and preorder tree traversal
- 5.1.17 Sketch binary trees

## Applications

- 5.1.18 Define the term dynamic data structure
- 5.1.19 Compare the use of static and dynamic data structures
- 5.1.20 Suggest a suitable structure for a given situation



1: System design

2: Computer Organisation



3: Networks

4: Computational thinking



5: Abstract data structures

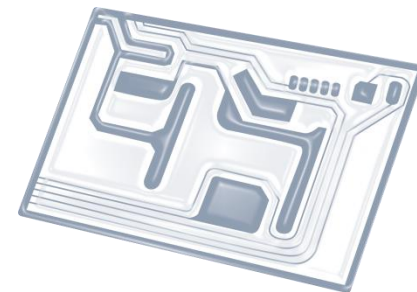
6: Resource management



7: Control

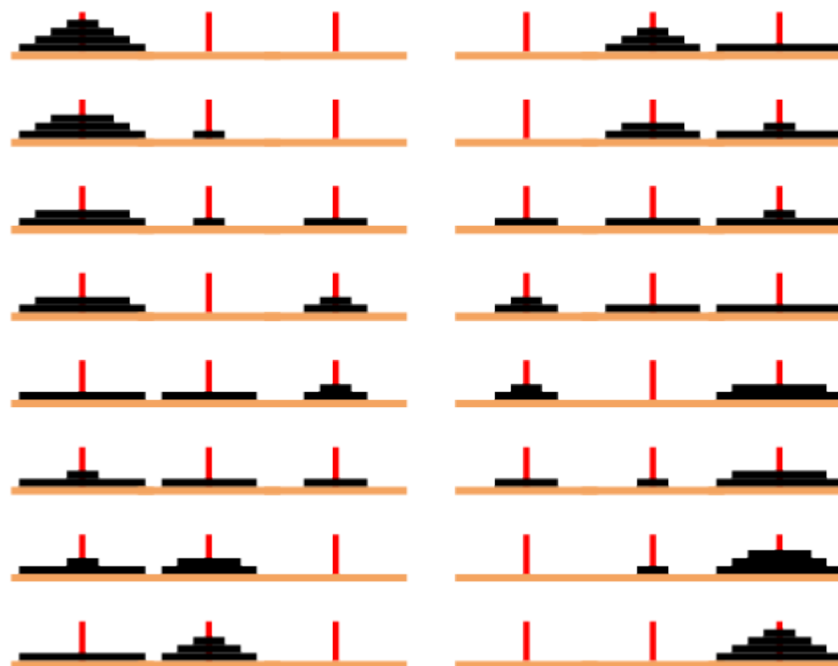
D: OOP





# Topic 5.1.2

Identify **recursive thinking** in a specified problem solution





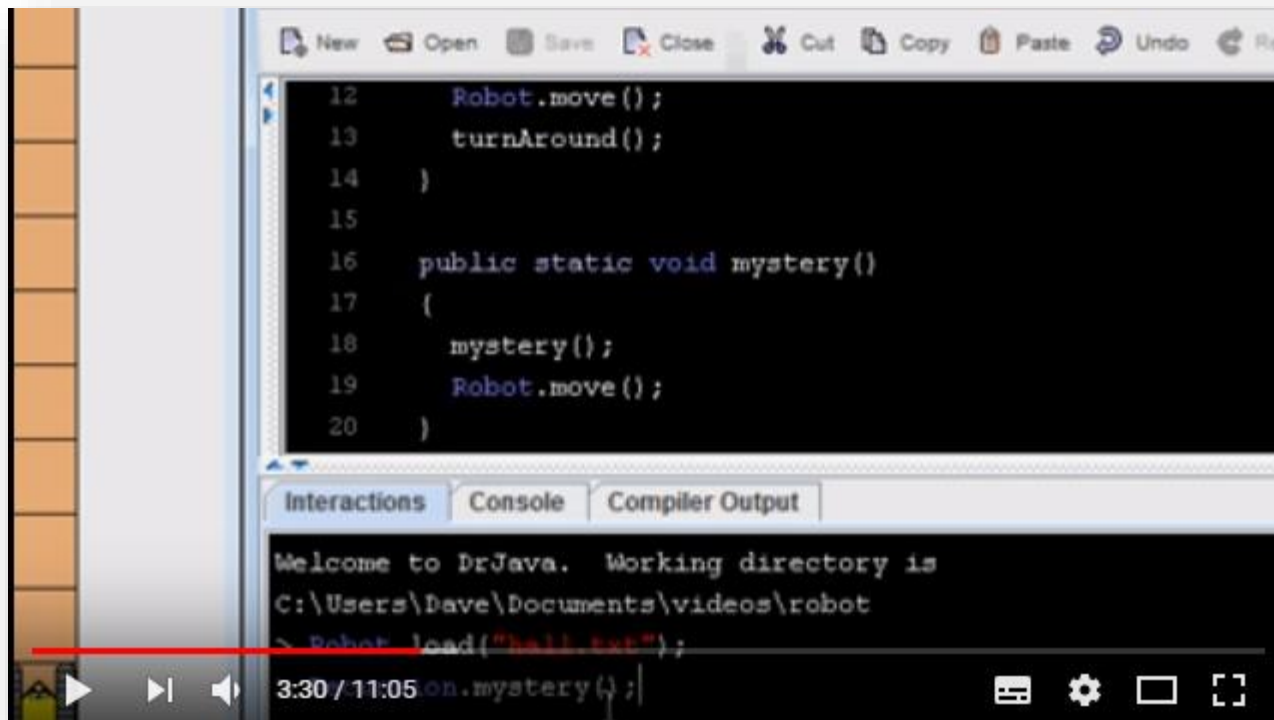
## *Exam note!*



This topic should really be studied in both **pseudo code** (Paper 1) and **Java** (Paper 2) as it links with **topic D.4**.

Students can expect both **algorithmic** and more **theory based questions** from this topic; answers could be a written paragraph or writing a pseudo code/Java method.

# Video: Introducing Recursion

A screenshot of a video player interface. The main content area shows a Java code editor with a dark background. The code is as follows:

```
12 Robot.move();  
13 turnAround();  
14 }  
15  
16 public static void mystery()  
17 {  
18     mystery();  
19     Robot.move();  
20 }
```

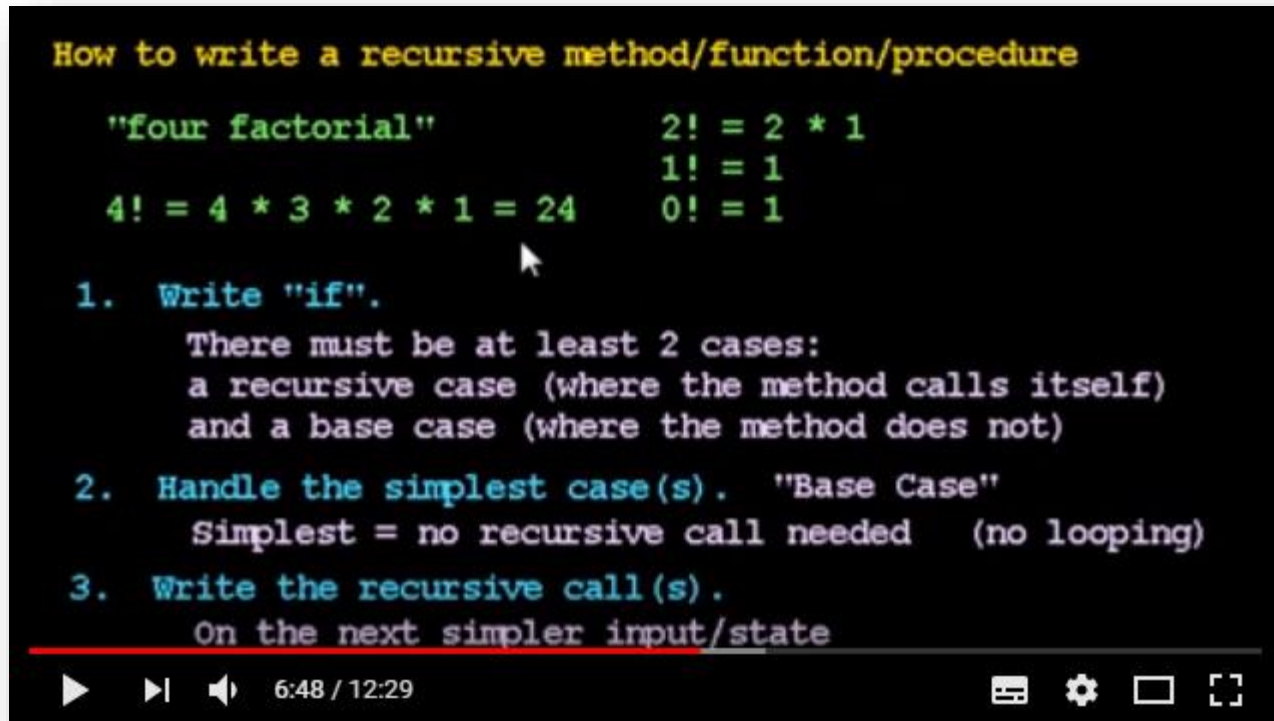
Below the code editor is a console window with tabs for 'Interactions', 'Console', and 'Compiler Output'. The 'Console' tab is active, showing the following text:

```
Welcome to DrJava. Working directory is  
C:\Users\Dave\Documents\videos\robot  
> Robot.load("hall.txt");  
3:30 / 11:05 on.mystery();
```

The video player controls at the bottom include a play button, a progress bar, and a timestamp of 3:30 / 11:05. There are also icons for settings, a full-screen button, and a close button.

Link (YouTube): [https://youtu.be/to7tAmiZ\\_Ic](https://youtu.be/to7tAmiZ_Ic)

# Video: How to write a recursive method



Link (YouTube): <https://youtu.be/MyzFdthuUcA>

# Video: **How recursion works**

```
"four factorial"      2! = 2 * 1
                     1! = 1
                     0! = 1
4! = 4 * 3 * 2 * 1 = 24
```

1. Write "if".  
There must be at least 2 cases:  
a recursive case (where the method calls itself)  
and a base case (where the method does not)
2. Handle the simplest case(s). "Base Case"  
Simplest = no recursive call needed (no looping)
3. Write the recursive call(s).  
On the next simpler input/state (maybe store result  
in a variable)
4. Assume the recursive call works.  
Ask yourself: What does it do?  
Ask yourself: How does it help?

0:01 / 11:40

Link (YouTube): <https://youtu.be/ozmE8G6YKww>



# Practice time: CodingBat

## Recursion-1 ☆☆☆☆☆☆☆☆☆

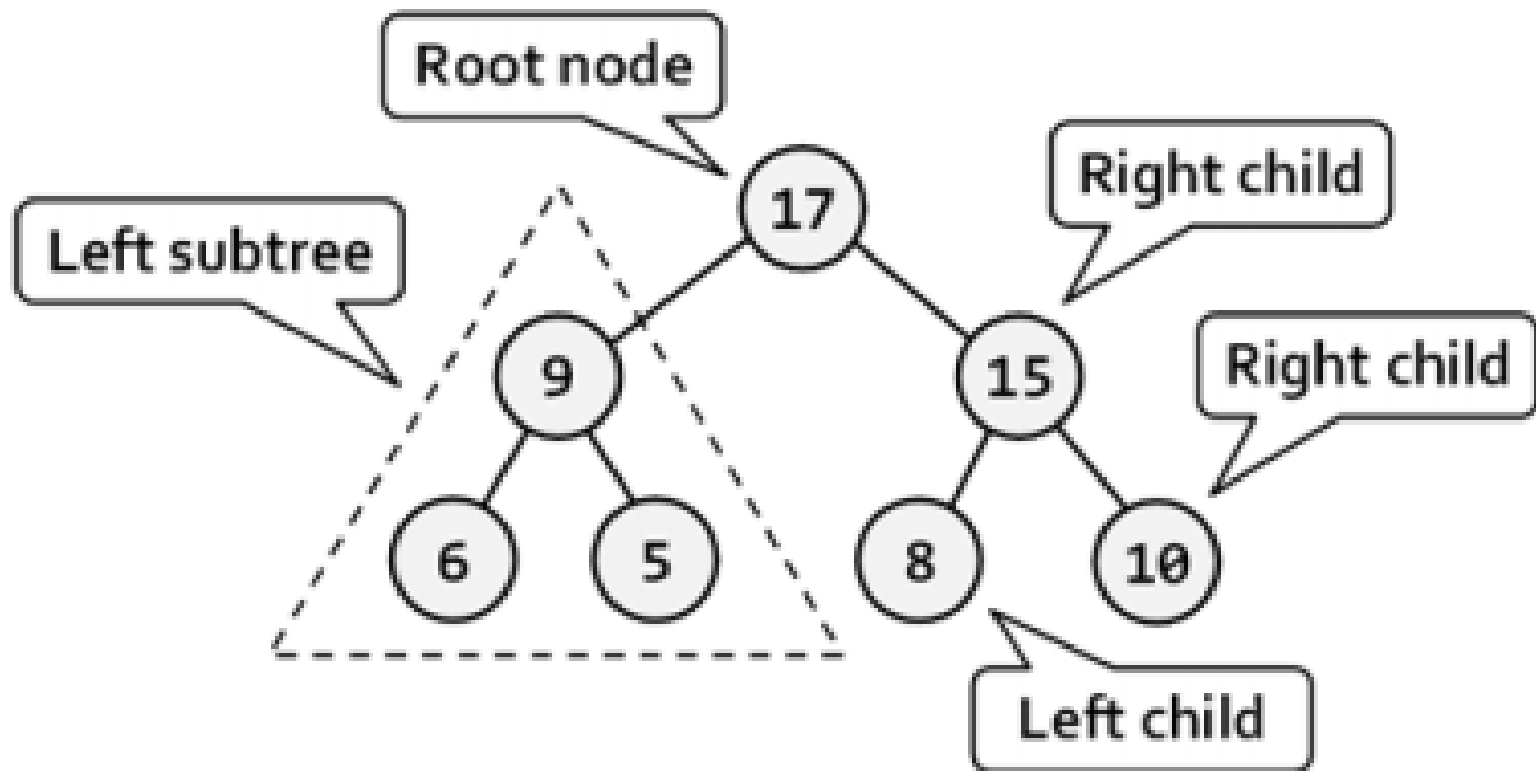
chance

Basic recursion problems. Recursion strategy: first test for one or two base cases that are so simple, the answer can be returned immediately. Otherwise, make a recursive call for a smaller case (that is, a case which is a step towards the base case). Assume that the recursive call works correctly, and fix up what it returns to make the answer.

- |               |               |               |
|---------------|---------------|---------------|
| ✓ factorial H | ✓ bunnyEars H | ✓ fibonacci   |
| ✓ bunnyEars2  | ✓ triangle    | ✓ sumDigits   |
| ✓ count7      | ✓ count8      | ✓ powerN      |
| ✓ countX      | ✓ countHi     | ✓ changeXY    |
| ✓ changePi    | ✓ noX         | ✓ array6      |
| ✓ array11     | ✓ array220    | ✓ allStar     |
| ✓ pairStar    | ✓ endX        | ✓ countPairs  |
| ✓ countAbc    | ✓ count11     | ✓ stringClean |
| ✓ countHi2    | ✓ parenBit    | ✓ nestParen   |
| ✓ strCount    | ✓ strCopies   | ✓ strDist     |

Try practicing writing recursive methods in Java using **CodingBat**, then practice writing them in IB pseudo code

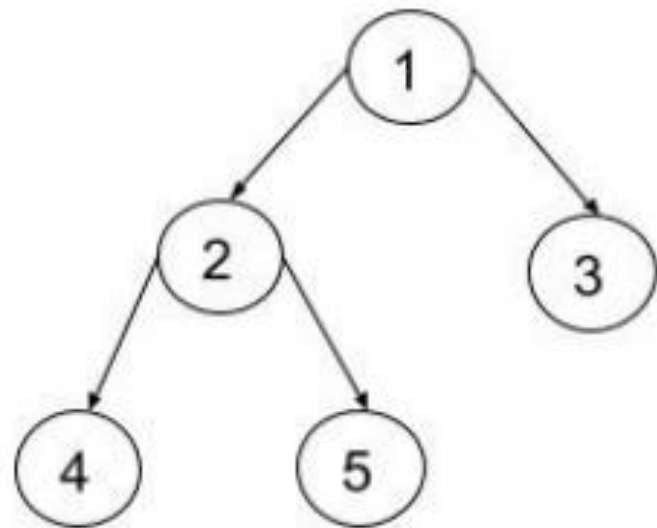
# Recursion & Binary Trees



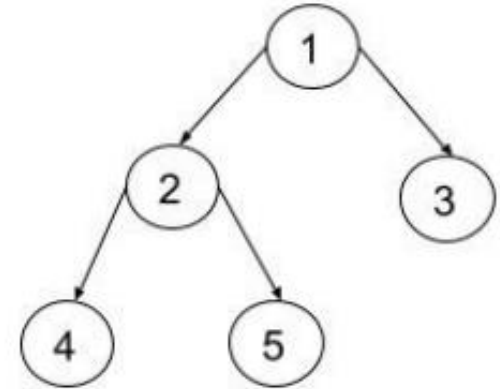
# Exercise 1: Find max sum

Implement **findMaxSum()** method that find the maximum sum of all paths (each path has their own sum and find max sum of those sums). For example, the path **1 → 2 → 5** makes the max sum of 8 and 8 is the result.

```
int findMaxSum(Node n)
{
    //insert code here
}
```



# Exercise 1: **Solution**

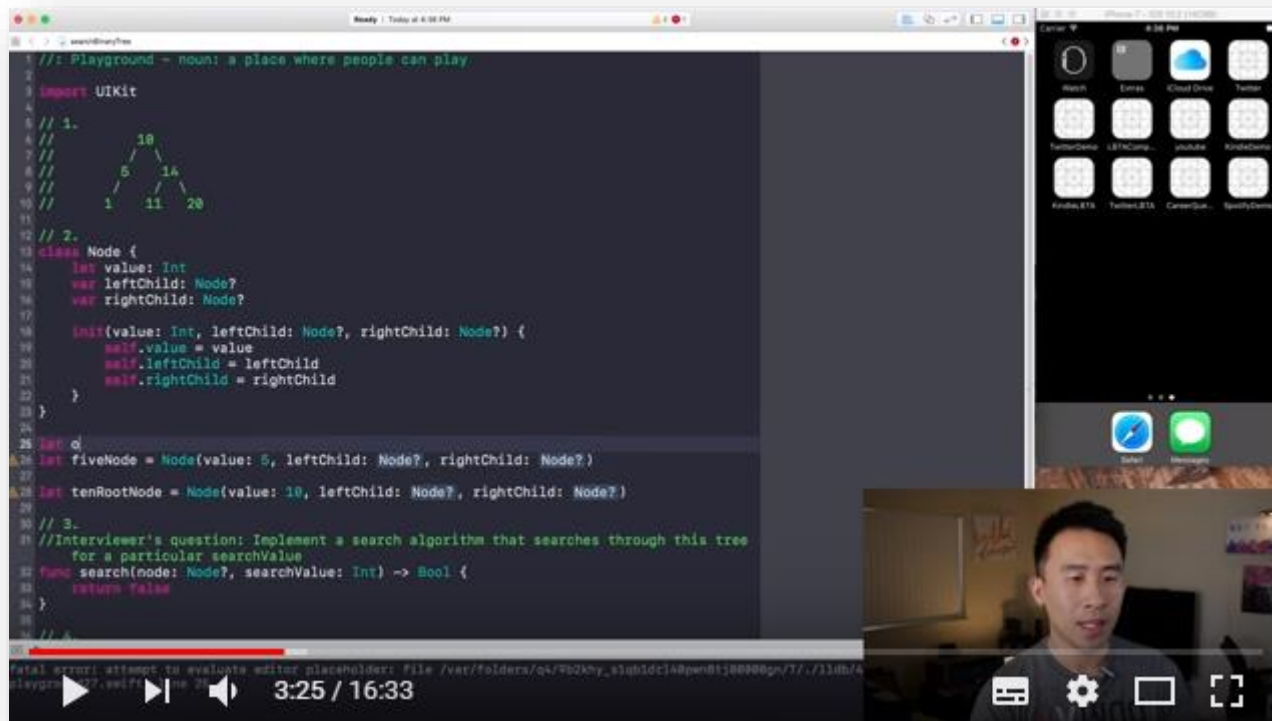


```

int findMaxSum(Node n) {
    if (n == null) return 0;
    else {
        int sumleft = findMaxSum(n.left);
        int sumright = findMaxSum(n.right);
        if (sumleft > sumright)
            return n.data + sumleft;
        else
            return n.data + sumright;
    }
}
  
```

In this case, the path 1->2->5 makes sum of 8; 1->2->4 makes sum of 7; and 1->3 makes sum of 4. Therefore, 8 is the result.

# Video: Recursively searching binary trees



Link (YouTube): <https://youtu.be/beQEimO3-3E>