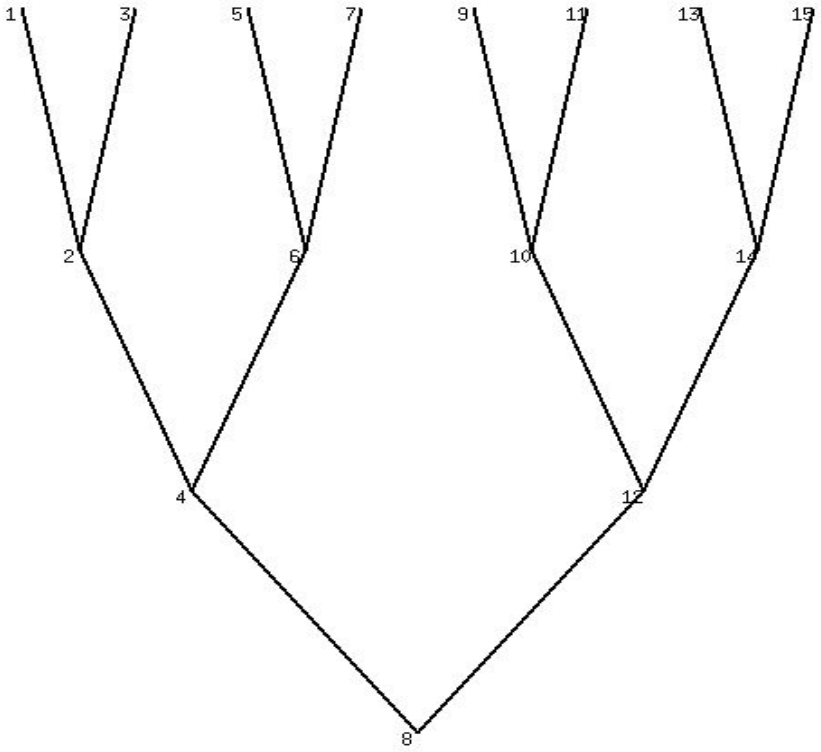
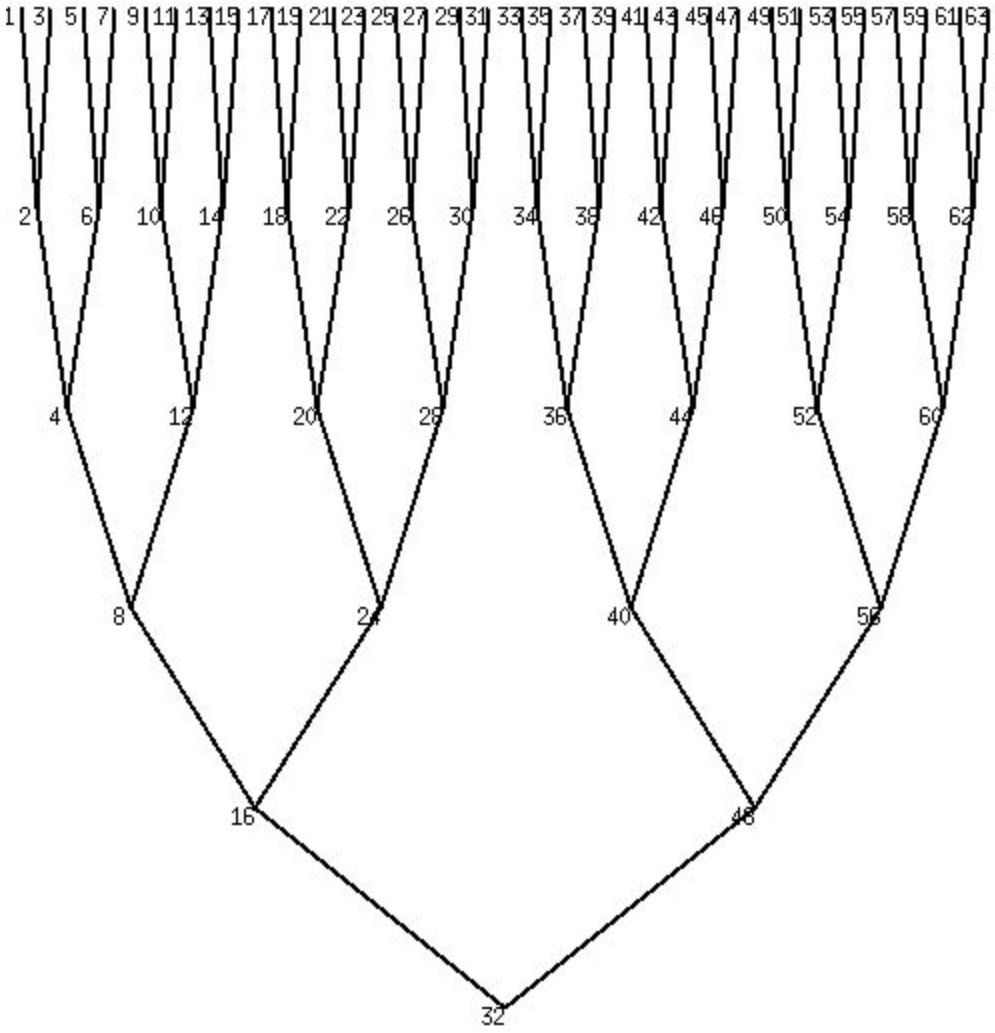
1. Consider a complete binary tree of height (levels) *h*. Such a tree has $2^{h+1} - 1$ vertices (nodes). E.g. a tree with level 1 has $2^2 - 1 = 3$ vertices, a tree of height 2 has $2^3 - 1 = 7$ vertices. Our goal is to not only draw the tree, but number the vertices in a certain order. The order we want is called *inorder*- where we number the left subtree first, then the root, then the right subtree. In such a tree, the leftmost vertex will have number 1, the rightmost $2^{h+1} - 1$, and the root of the whole tree would be 2^h. The inorder numbering of some trees is shown below. Modify our tree drawing program (of simplecpp graphics) such that the inorder number of the vertex is printed next to the vertex. Take the height *h* of the tree as input.

Tree of level h=3:



Tree of level h=5:



The rest of the programs in today's lab should be written in C++, not simplecpp

2. The Eratosthenes' Sieve for determining whether a number **n** is prime is as follows. We first write down the numbers 2, 3, … n on paper. We then start with the first uncrossed

number and cross out all its proper multiples. Then we look for next uncrossed number and cross out all its proper multiples, and so on. If **n** is not crossed out in this process, then it must be a prime. Using this idea alongwith arrays, write a program to find all the prime numbers upto a number **n**. Take **n** as input from user.

Sample input 1: 10	Sample output 1 : 2,3,5,7
Sample input 2: 20	Sample output 2 : 2, 3, 5, 7, 11, 13, 17, 19
Sample input 3: 30	Sample output 3 : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29

3. Tower of Hanoi is a mathematical puzzle. It consists of three PEGS (0 for Source , 1 for Destination and 2 for Auxiliary). Source peg consists of disks of different sizes such that they form a conical shape. The objective is to move all the disks from source to destination peg using an auxiliary peg with the following constraints :
   1) Only one disk can be moved at a time.
   2) We can never place a larger disk on top of a smaller disk.

Suppose, we are given n disks on the source peg. We need to display all the moves which are required to move the disks from source peg to destination peg. Let's formulate a recursive solution :
1) Move top n - 1 disks from source peg to auxiliary peg
2) Move the only disk present in source peg to destination peg
3) Move the n - 1 disks from auxiliary peg to destination peg

Write a program that takes n as an input and prints out the moves. Sample inputs and outputs are shown below.

Input
Enter number of disks
3
output:
Move disk 0 from peg 0 to peg 1
Move disk 1 from peg 0 to peg 2
Move disk 0 from peg 1 to peg 2
Move disk 2 from peg 0 to peg 1
Move disk 0 from peg 2 to peg 0
Move disk 1 from peg 2 to peg 1
Move disk 0 from peg 0 to peg 1

4. Given a non-empty array of integers, return true if there is a place to split the array so that the sum of the numbers on one side is equal to the sum of the numbers on the other side. Write a function to do this which also returns the index of the second part of the array in a reference argument.

[1, 1, 1, 2, 1] ⇒ true	because (1+1+1) == (2+1)
[2, 1, 1, 2, 1] ⇒ false	
[10, 10] ⇒ true	because 10 == 10
[5] ⇒ false	
[10, 0, 1, -1, 10]⇒ true	because 10 == (0 + 1 + -1 + 10)
[1, 1, 1, 3] ⇒ true	because (1+1+1) == 3

Try to do this problem in only two "scans" of the array.

5. Consider the leftmost and righmost appearances of some value in an array. We'll say that the "span" is the number of elements between the two inclusive. A single value has a span of 1. Returns the largest span found in the given array. (Efficiency is not a priority.)

maxSpan({1, 2, 1, 1, 3}) → 4
maxSpan({1, 4, 2, 1, 4, 1, 4}) → 6
treeInorder.cppmaxSpan({1, 4, 2, 1, 4, 4, 4}) → 6

6. Say that a "clump" in an array is a series of 2 or more adjacent elements of the same value. Return the number of clumps in the given array.

countClumps({1, 2, 2, 3, 4, 4}) → 2
countClumps({1, 1, 2, 1, 1}) → 2
countClumps({1, 1, 1, 1, 1}) → 1