OBSERVE THAT THE FUNCTION $x \rightarrow nx$ MAPS

$$[0,1) \longrightarrow [0,n)$$

AND HENCE THE SUBINTERVALS

$$\left[0, \tfrac{1}{n}\right) \longrightarrow [0,1)$$

$$\left[\tfrac{1}{n}, \tfrac{2}{n}\right) \longrightarrow [1,2)$$

$$\vdots$$

$$\left[\tfrac{i-1}{n}, \tfrac{i}{n}\right) \longrightarrow [i-1, i)$$

$$\vdots$$

$$\left[\tfrac{n-1}{n}, 1\right) \longrightarrow [n-1, n)$$

THUS $x \in \left[\tfrac{i-1}{n}, \tfrac{i}{n}\right)$ IS MAPPED TO $nx \in [i-1, i)$,
$\therefore \lfloor nx \rfloor = i-1$, $\therefore \lfloor nx \rfloor + 1 = i$.

THUS ALL ELEMENTS IN THE $i^{TH}$ SUBINTERVAL ARE PLACED IN THE $i^{TH}$ BUCKET $B[i]$

THE COST OF ALL OPERATIONS OTHER THAN (5) IS OBVIOUSLY $\Theta(n)$.

IF THERE ARE $n_j$ ELEMENTS IN BUCKET $j$ THEN THE (AVERAGE) RUN TIME OF BUCKETSORT IS

$$t(n) = \Theta(n) + \sum_{j=1}^{n} \Theta(n_j^2) .$$

OUR ASSUMPTION IMPLIES THAT ON AVERAGE, FOR ALL $n_i = 1$. THUS $\sum_{j=1}^{n} \Theta(n_j^2) = \Theta(n)$, WHENCE

$$t(n) = \Theta(n) .$$

(SEE TEXT FOR MORE RIGOROUS TREATMENT.)

# Lower Bounds & Computational Complexity

Consider the set of all algorithms (known & unknown) which solve some problem $P$ in all its instances.

Our goals are twofold:

1.) Find an algorithm which solves $P$ in (worst case) time $O(f(n))$ for some function $f(n)$ which we aim to reduce as far as possible.

2.) Prove that any algorithm which solves $P$ must run in (worst case) time $\Omega(g(n))$ for some function $g(n)$ which we aim to increase as far as possible.

Here $n$ denotes the 'size' of an instance of problem $P$.

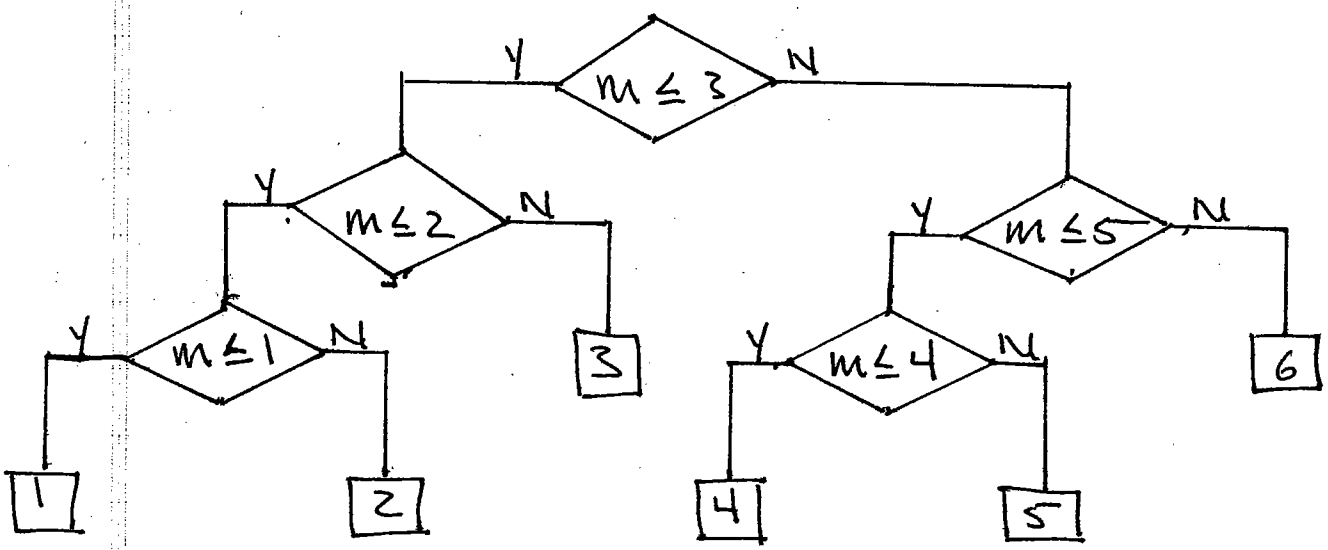
We are happy when $f(n) = \Theta(g(n))$ for then we know we have the best possible algorithm to solve $P$ (apart from improvements in hidden constants.)

(1) is called <u>Algorithmics</u> by some authors, while (2) is the theory of <u>computational complexity</u>. The function $g(n)$ in (2) is called a <u>lower bound</u> on the complexity of problem P.

## Decision Trees / Information Theoretic Lower Bounds

<u>Ex.</u> Let m be an integer in the range $1 \le m \le 6$. Problem: Determine the value of m by asking a sequence of yes/no questions.

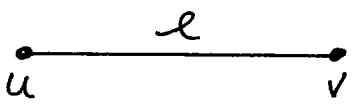
This problem is similar to binary search!



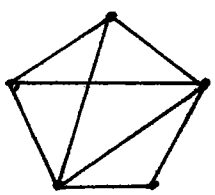
APPARENTLY THE ANSWER CAN BE OBTAINED BY ASKING NO MORE THAN 3 QUESTIONS. (will 2 SUFFICE ? )

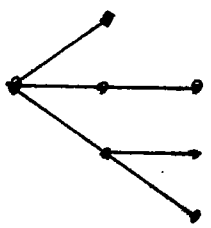
REVIEW: GRAPHS - TREES - ROOTED TREES.

A GRAPH $G = (V, E)$ is A PAIR OF SETS CALLED VERTICES ($V$) AND EDGES ($E$). EACH EDGE JOINS A (UNIQUE) PAIR OF (DISTINCT) VERTICES. TWO VERTICES WHICH ARE JOINED BY AN EDGE ARE CALLED ADJACENT

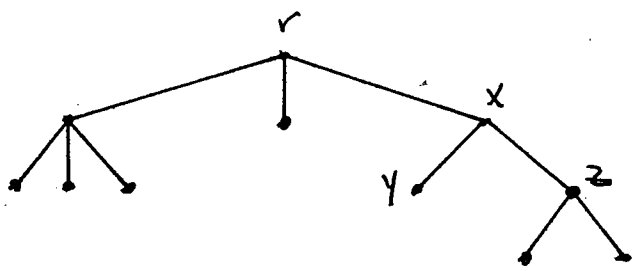



A PATH IN G is A SEQUENCE OF CONSECUTIVELY ADJACENT VERTICES. G is CALLED CONNECTED IF EVERY PAIR OF VERTICES IN G ARE JOINED BY A PATH. A CYCLE is A CLOSED PATH, i.e. A PATH IN WHICH THE INITIAL AND TERMINAL VERTICES ARE IDENTICAL. G is CALLED ACYCLIC IF IT CONTAINS NO CYCLES. A TREE is A CONNECTED ACYCLIC GRAPH.

GRAPH: 

TREE: 

A <u>ROOTED TREE</u> IS A TREE IN WHICH ONE VERTEX HAS BEEN DISTINGUISHED AS THE <u>ROOT</u>. THE VERTICES IN SUCH A TREE ARE OFTEN CALLED <u>NODES</u>. THE <u>DEPTH</u> OF A NODE IS ITS DISTANCE FROM THE ROOT. (DISTANCE MEANS SHORTEST PATH LENGTH.)




DEPTH

0

1

2

3

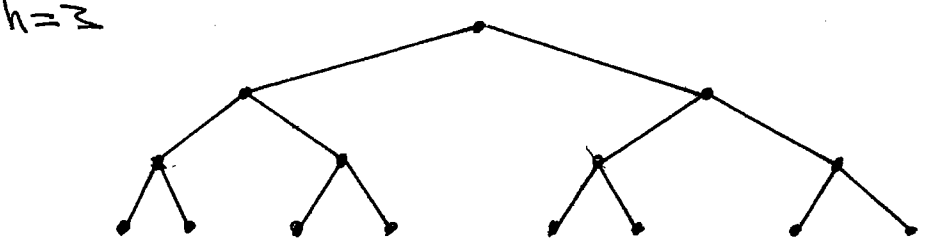
THE <u>CHILDREN</u> OF A NODE x ARE THOSE NODES ADJACENT TO x WHOSE DEPTH IS ONE MORE THAN THAT OF x. THE <u>PARENT</u> OF x IS THE (UNIQUE) NODE WHICH IS ADJACENT TO x AND HAS DEPTH ONE LESS THAN THAT OF x.

THE ROOT IS THE ONLY NODE WHICH HAS NO PARENT (SINCE THE ROOT HAS DEPTH 0.) IF A NODE y HAS NO CHILDREN, IT IS CALLED A <u>LEAF</u>. A NON-LEAF NODE IS CALLED AN <u>INTERNAL NODE</u>.

THE HEIGHT OF A ROOTED TREE IS ITS MAXIMUM NODE DEPTH, I.E. THE LENGTH OF A LONGEST DOWNWARD PATH FROM THE ROOT TO A LEAF. THE HEIGHT OF A NODE IS THE HEIGHT OF THE SUBTREE ROOTED AT THAT NODE.

A BINARY TREE IS A ROOTED TREE IN WHICH EACH NODE HAS AT MOST 2 CHILDREN. MORE GENERALLY A K-ARY TREE IS A ROOTED TREE IN WHICH EACH NODE HAS AT MOST K CHILDREN.

A COMPLETE BINARY TREE (CBT) IS A BINARY TREE IN WHICH ALL LEAVES ARE AT THE SAME DEPTH, AND EACH INTERNAL NODE HAS EXACTLY 2 CHILDREN




| DEPTH | # NODES |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

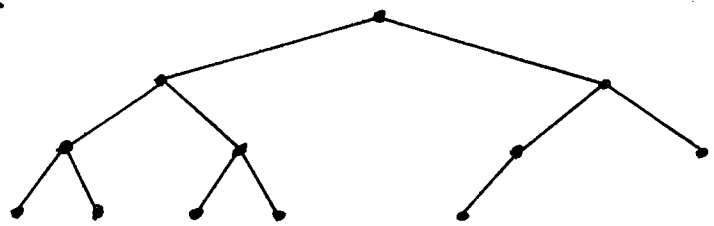
# NODES AT DEPTH $d = 2^d$

# LEAVES = # NODES AT DEPTH $h = 2^h$

∴ THE HEIGHT OF A CBT WITH $n$ LEAVES IS $h = \lg(n)$.

AN <u>ALMOST COMPLETE BINARY TREE</u> (ACBT) IS A BINARY TREE WHICH HAS THE MAXIMUM POSSIBLE NUMBER OF NODES AT EACH DEPTH, EXCEPT POSSIBLY THE LAST, WHICH IS FILLED FROM LEFT TO RIGHT.

$h = 3$



(AN ACBT IS THE BASIS OF THE HEAP DATA STRUCTURE.)

<u>EXERCISE</u>

PROVE THAT AN ACBT WITH $n$ LEAVES AND HEIGHT $h$ SATISFIES $h = \lceil \lg n \rceil$.

<u>THEOREM</u>

THE HEIGHT $h$ OF ANY BINARY TREE WITH $n$ LEAVES SATISFIES

$$h \geq \lceil \lg n \rceil .$$

<u>NOTATION:</u>

LET $L(T)$ AND $H(T)$ DENOTE THE NUMBER OF LEAVES AND THE HEIGHT OF A