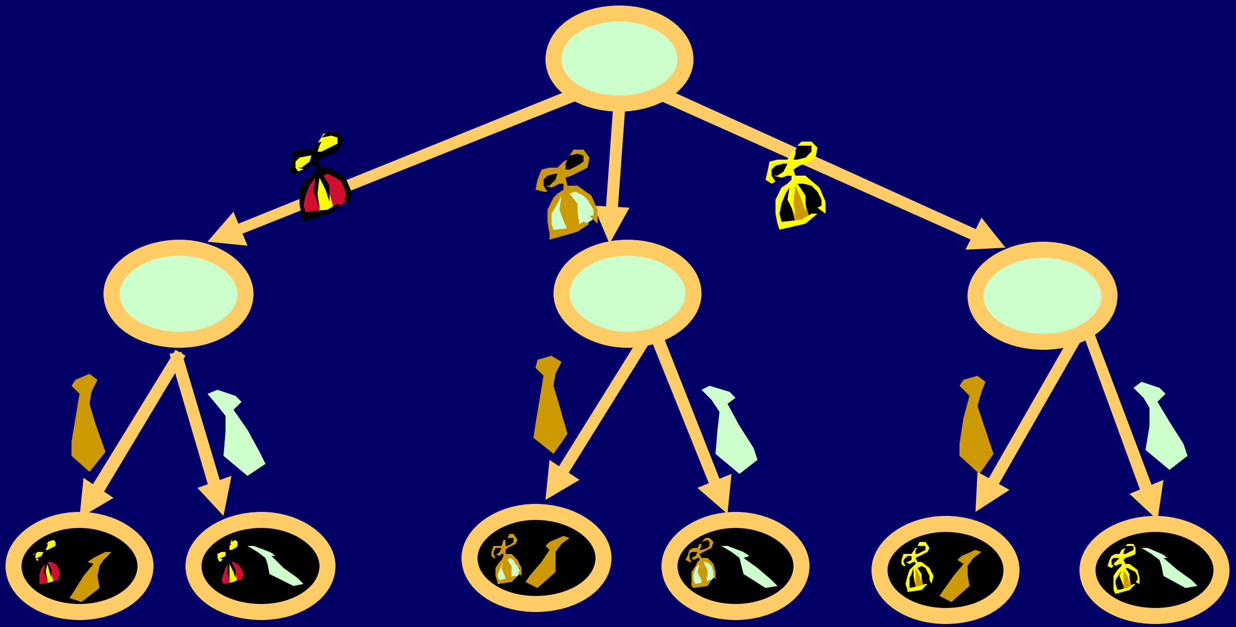
Great Theoretical Ideas In Computer Science

Steven Rudich CS 15-251 Spring 2005

Lecture 2 Jan 13, 2005 Carnegie Mellon University

Induction II:
Inductive Pictures



Inductive Proof:

“Standard” Induction

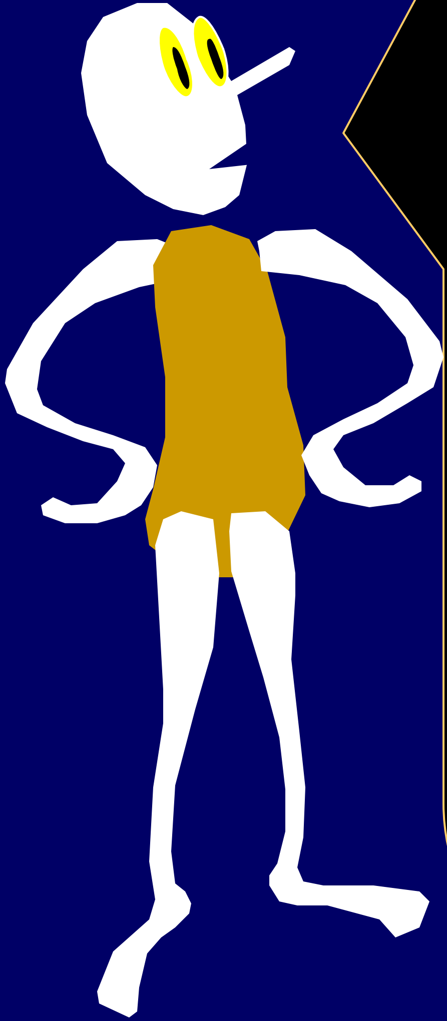
“Least Counter-example”

“All Previous” Induction

Inductive Definition:

Recurrences

Recursive Programming



Theorem? (k ¸ 0)

$1+2+4+8+\ldots+2^k = 2^{k+1} -1$

Try it out on small examples:

$2^0 = 2^1 -1$

$2^0 + 2^1 = 2^2 -1$

$2^0 + 2^1 + 2^2 = 2^3 -1$

S_k ´ “$1+2+4+8+...+2^k = 2^{k+1} -1$” Use induction to prove ∀k ¸ 0, S_k

Establish “Base Case”: S_0. We have already check it.

Establish “Domino Property”: ∀k ¸ 0, S_k) S_{k+1}

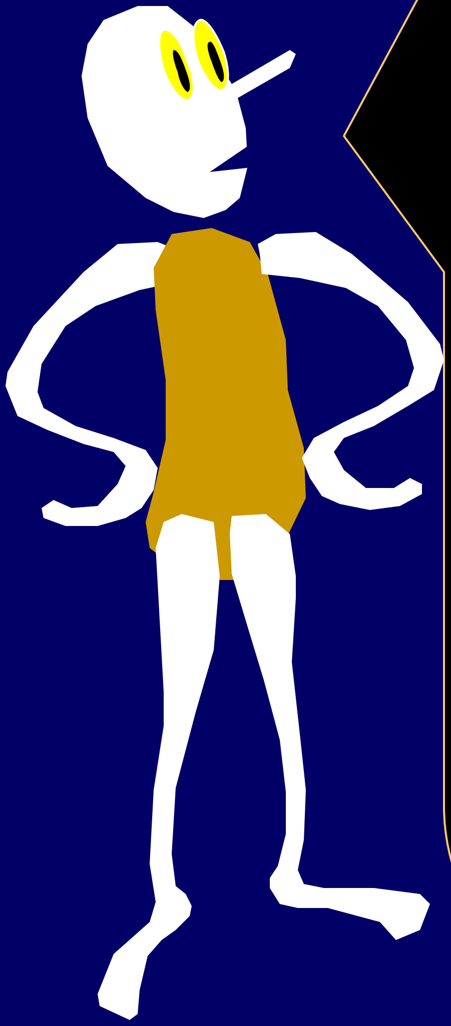
“Inductive Hypothesis” S_k:

$1+2+4+8+...+2^k = 2^{k+1} -1$

Add 2^{k+1} to both sides:

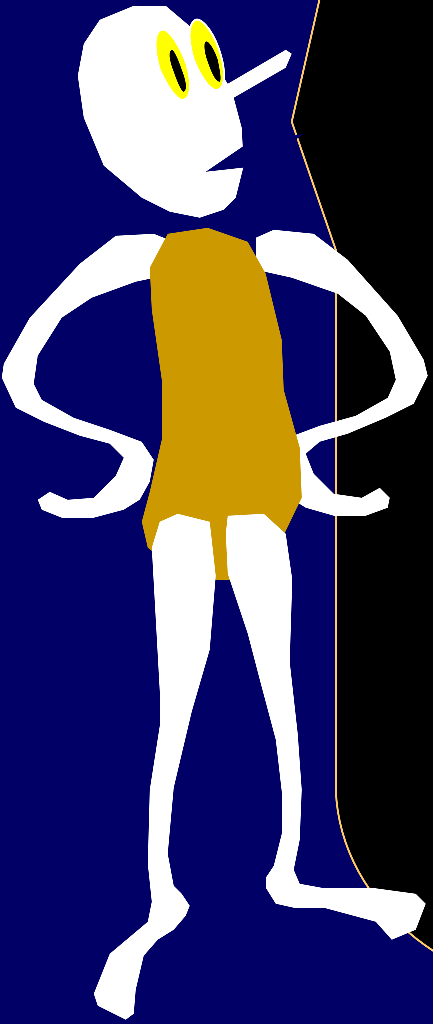
$1+2+4+8+...+2^k + 2^{k+1}= 2^{k+1} +2^{k+1} -1$

$1+2+4+8+...+2^k + 2^{k+1}= 2^{k+2} -1$



FUNDAMENTAL LEMMA OF THE POWERS OF TWO:

The sum of the first n powers of 2, is one less than the next power of 2.

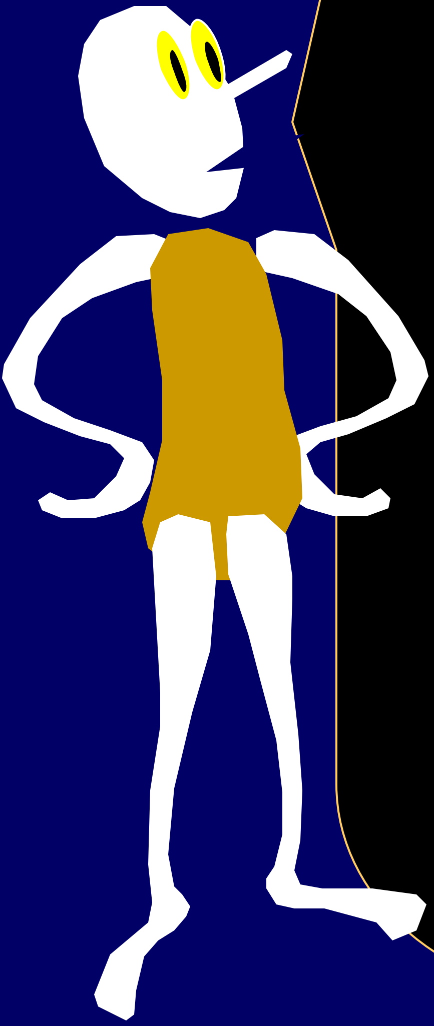


Yet another way of packaging inductive reasoning is to define an "invariant".

Invariant *(adj.)*

1. Not varying; constant.

2. (mathematics) Unaffected by a designated operation, as a transformation of coordinates.

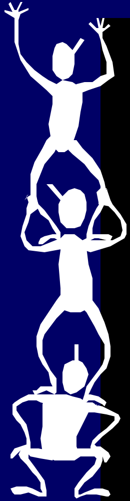


Yet another way of packaging inductive reasoning is to define an “invariant”.

Invariant *(adj.)*

3. *(programming)* A rule, such as the ordering an ordered list or heap, that applies throughout the life of a data structure or procedure.

Each change to the data structure must maintain the correctness of the invariant.



Invariant Induction

Suppose we have a time varying world state: W_0, W_1, W_2, …

Each state change is assumed to come from a list of permissible operations.

We seek to prove that statement S is true of all future worlds.

Argue that S is true of the initial world.

Show that if S is true of some world – then S remains true after one permissible operation is performed.

Odd/Even Handshaking Theorem:

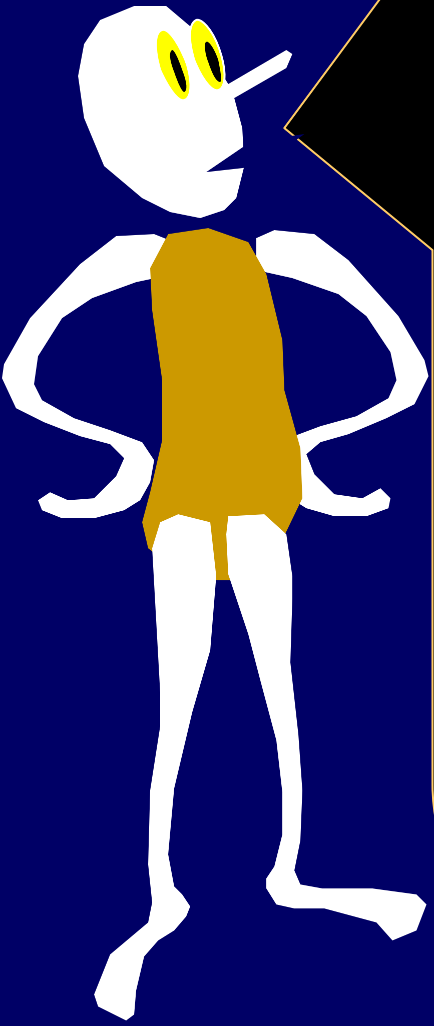
At any party, at any point in time, define a person's parity as ODD/EVEN according to the number of hands they have shaken.

Statement: The number of people of odd parity must be even.

Initial case: Zero hands have been shaken at the start of a party, so zero people have odd parity.

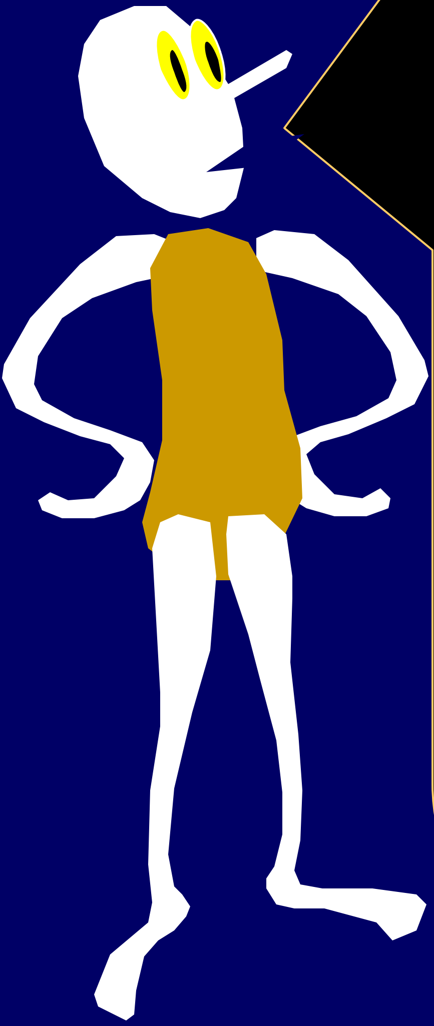
If 2 people of different parities shake, then they both swap parities and the odd parity count is unchanged.

If 2 people of the same parity shake, they both change. But then the odd parity count changes by 2, and remains even.



Inductive Definition of n!
[said n “factorial”]

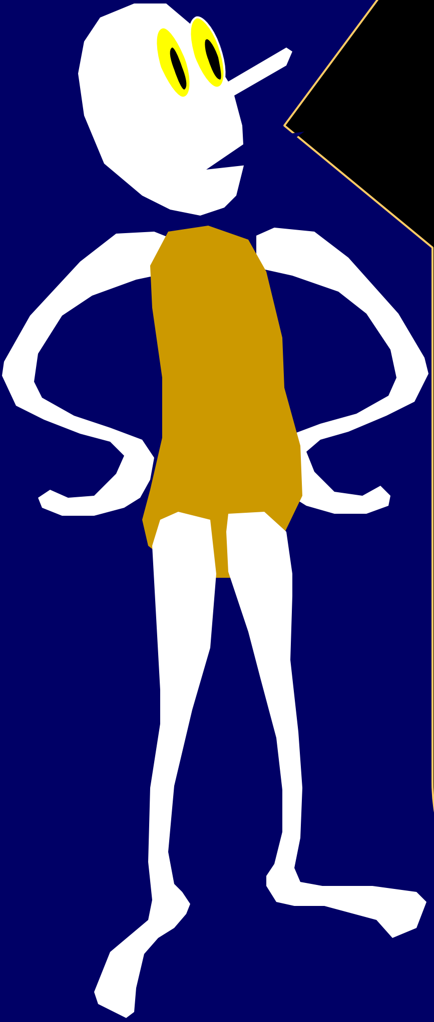
$0! = 1; n! = n*(n-1)!$



$$0! = 1;\ n! = n*(n-1)!$$

```
F:=1;
For x = 1 to n do
        F:=F*x;
Return F
```

Program for n! ?



$$0! = 1; \; n! = n*(n-1)!$$

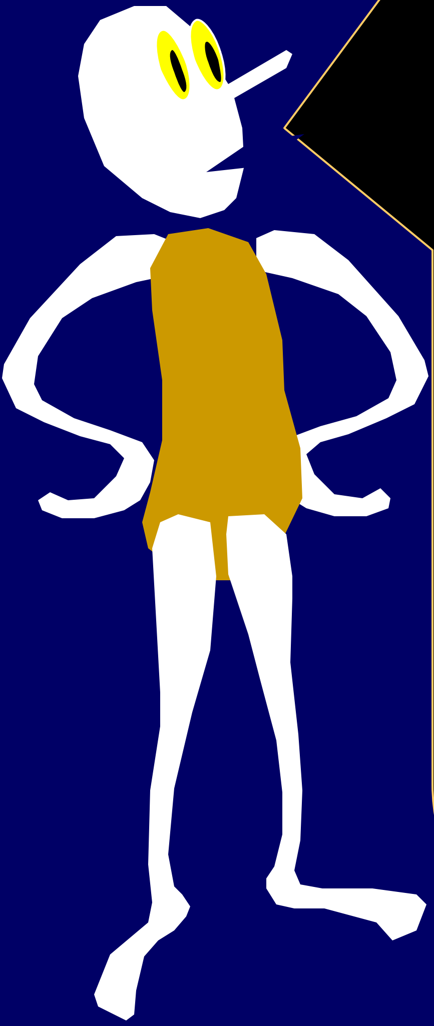
```
F:=1;
For x = 1 to n do
        F:=F*x;
Return F
```

n=0 returns 1

n=1 returns 1

n=2 returns 2

Program for n! ?



$0! = 1; n! = n*(n-1)!$

```
F:=1;
For x = 1 to n do
        F:=F*x;
Return F
```

Loop Invariant: F=x!

True for x=0. If true after k times through – true after k+1 times through.

Inductive Definition of T(n)

$T(1) = 1$

$T(n) = 4T(n/2) + n$

Notice that T(n) is inductively defined for positive powers of 2, and undefined on other values.

Inductive Definition of T(n)

$T(1) = 1$

$T(n) = 4T(n/2) + n$

Notice that T(n) is inductively defined for positive powers of 2, and undefined on other values.

$T(1)=1 \quad T(2)=6 \quad T(4)=28 \quad T(8)=120$

Guess a closed form formula for T(n). Guess G(n)

$G(n) = 2n^2 - n$

Let the domain of G be the powers of two.

Two equivalent functions?

$G(n) = 2n^2 - n$

Let the domain of G be the powers of two.

$T(1) = 1$

$T(n) = 4\ T(n/2) + n$

Domain of T are the powers of two.

Inductive Proof of Equivalence

$G(n) = 2n^2 - n$

$T(1) = 1$

$T(n) = 4\, T(n/2) + n$

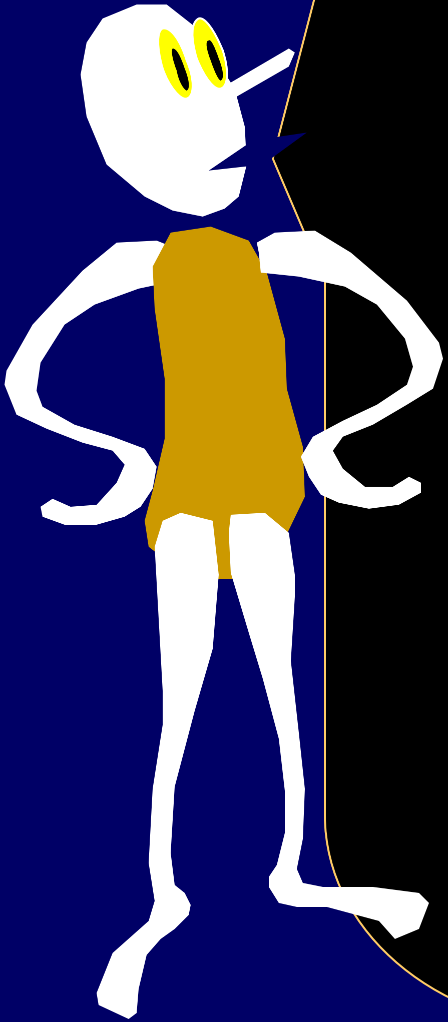
Base: $G(1) = 1$ and $T(1) = 1$

Induction Hypothesis:

$T(x) = G(x)$ for $x < n$

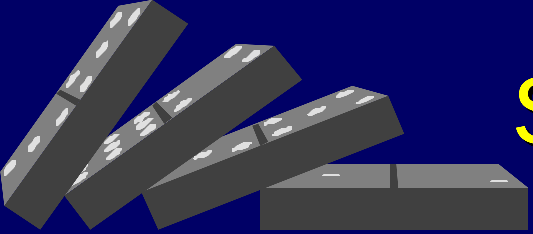
Hence: $T(n/2) = G(n/2) = 2(n/2)^2 - n/2$

$$\begin{aligned} T(n) &= 4\, T(n/2) + n \\ &= 4\, G(n/2) + n \\ &= 4\, [2(n/2)^2 - n/2] + n \\ &= 2n^2 - 2n + n \\ &= 2n^2 - n \\ &= G(n) \end{aligned}$$



We inductively proved the assertion that G(n) =T(n).

Giving a formula for T with no sums or recurrences is called solving the recurrence T.



Solving Recurrences
Guess and Verify

Guess: $G(n) = 2n^2 - n$

Verify: $G(1) = 1$ and $G(n) = 4\ G(n/2) + n$

Similarly: $T(1) = 1$ and $T(n) = 4\ T(n/2) + n$

So $T(n) = G(n)$

Technique 2
Guess Form and Calculate Coefficients

Guess: $T(n) = an^2 + bn + c$ for some a,b,c

Calculate: $T(1) = 1 \Rightarrow a + b + c = 1$

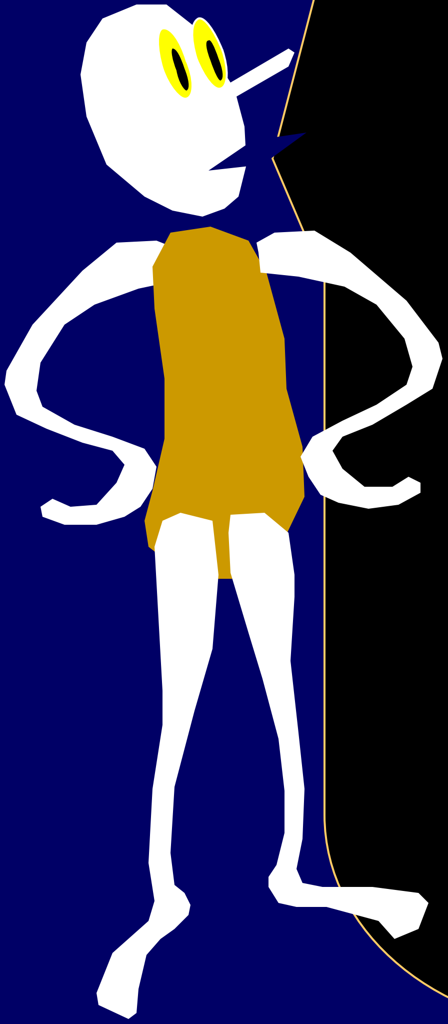
$$T(n) = 4\,T(n/2) + n$$

$$\Rightarrow\ an^2 + bn + c = 4\,[a(n/2)^2 + b(n/2) + c] + n$$

$$= an^2 + 2bn + 4c + n$$

$$\Rightarrow\ (b+1)n + 3c = 0$$

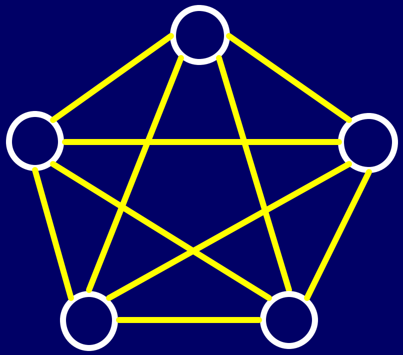
Therefore: $b=-1 \qquad c=0 \qquad a=2$

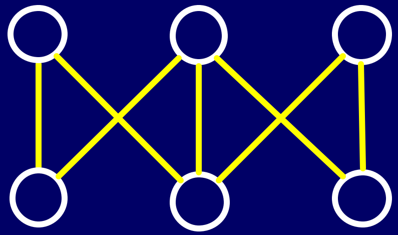


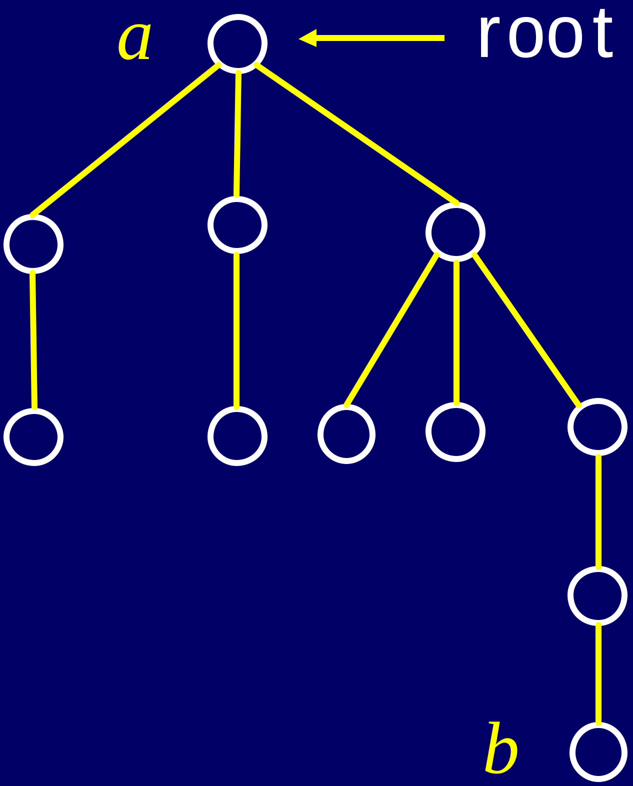
A computer scientist not only deals with numbers, but also with

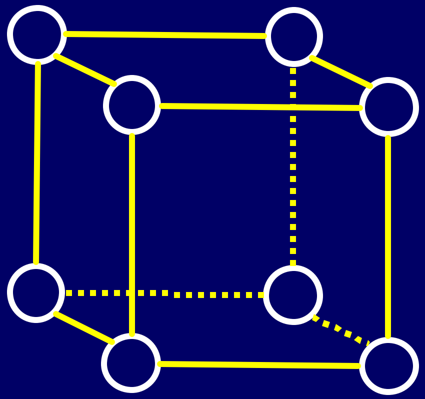
- Finite Strings of symbols
- Very visual objects called graphs
- And especially, especially the special graphs called trees

GRAPHS









Definition: Graphs

A graph G = (V,E) consists of a finite set V of vertices (nodes) and a finite set E of edges. Each edge is a set {a, b} of two different vertices.

A graph may not have self loops or multiple edges.

Definition: Directed Graphs

A graph G = (V,E) consists of a finite set V of vertices (nodes) and a finite set E of edges. Each edge is an ordered pair <a,b> of two different vertices.

Unless we say otherwise, our directed graphs will not have multi-edges, or self loops.

Definition: Tree

A tree is a directed graph with one special node called the root and the property that each node must a unique path from the root to itself.

Child: If <u,v>2E, we sav is a child of u

Parent: If <u,v>2E, we say u is the parent of u

Leaf: If u has no children, we say u is leaf.

Siblings: If u and v have the same parent, they are siblings.

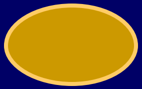
Descendants of u: The set of nodes reachable from u (including u).

Sub-tree rooted at u: Descendants of u and all the edges between them where u has been designated as a root.

Classic Visualization: Tree

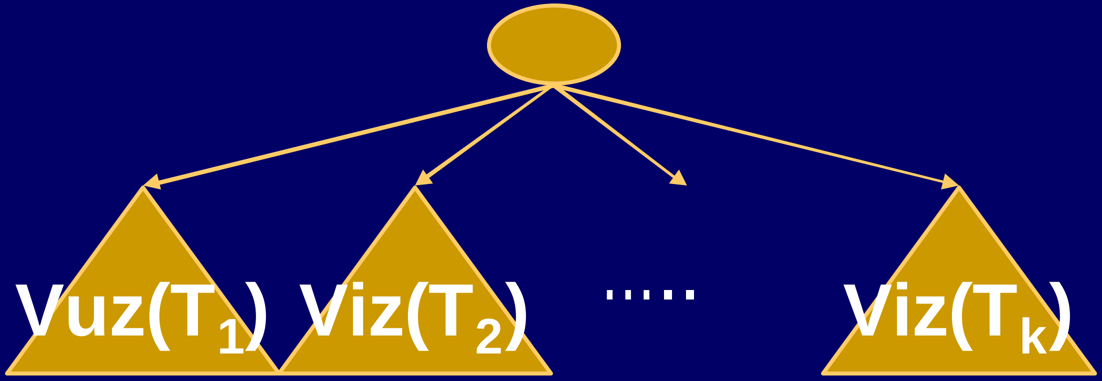
Inductive rule:

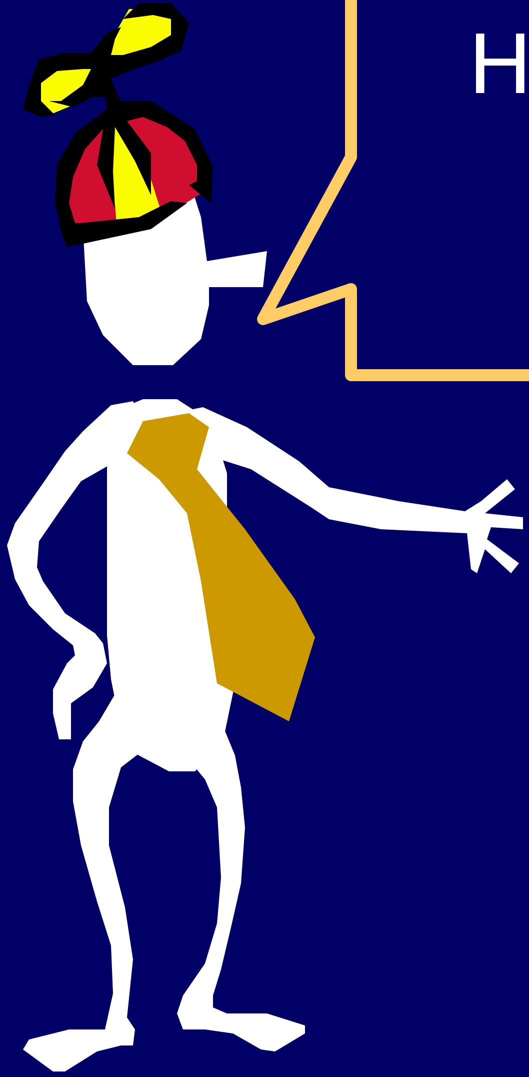
If G is a single node

Viz(G) = 

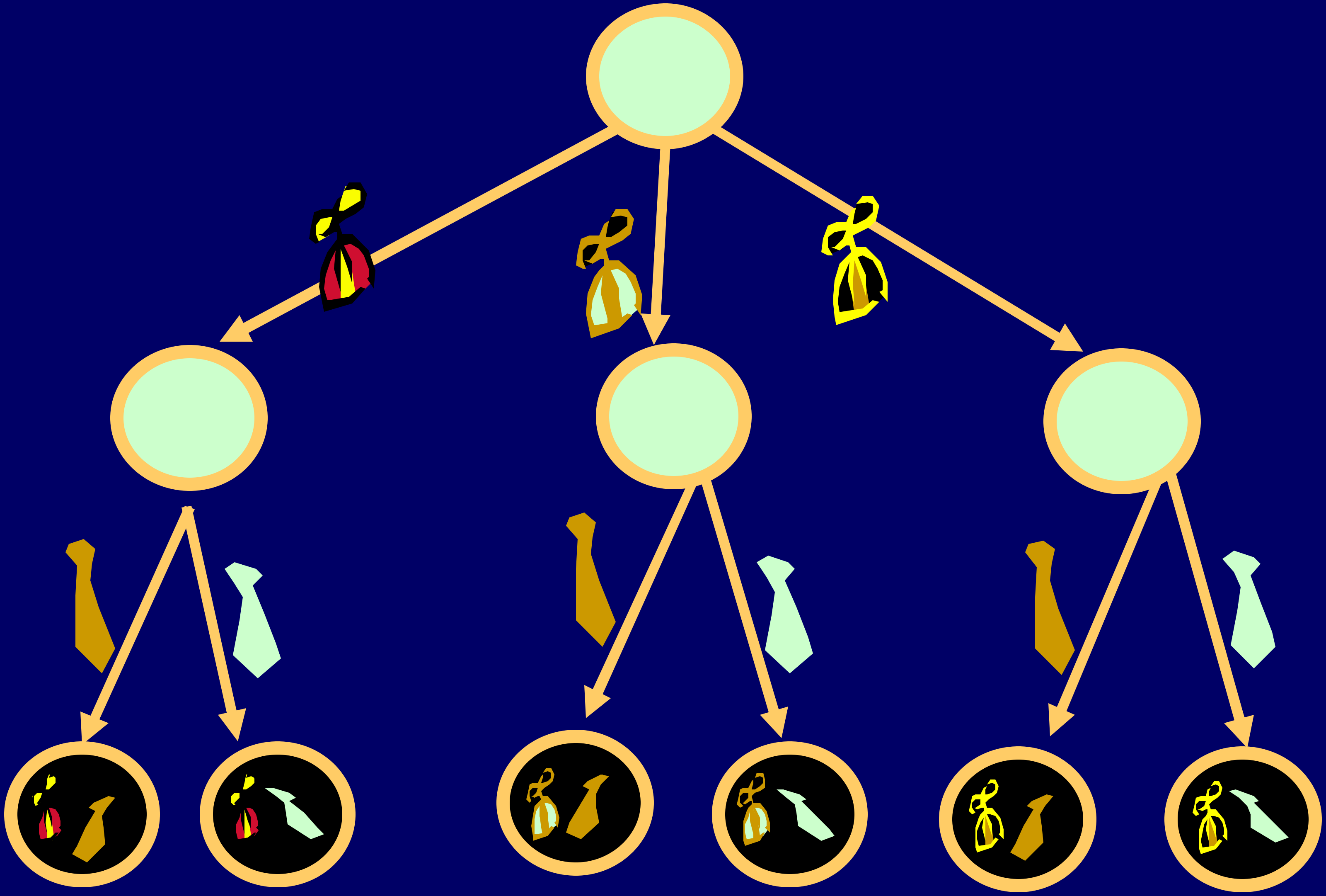
If G consists of root r with sub-trees $T_1, T_2, \ldots, T_k$

Viz(G) =

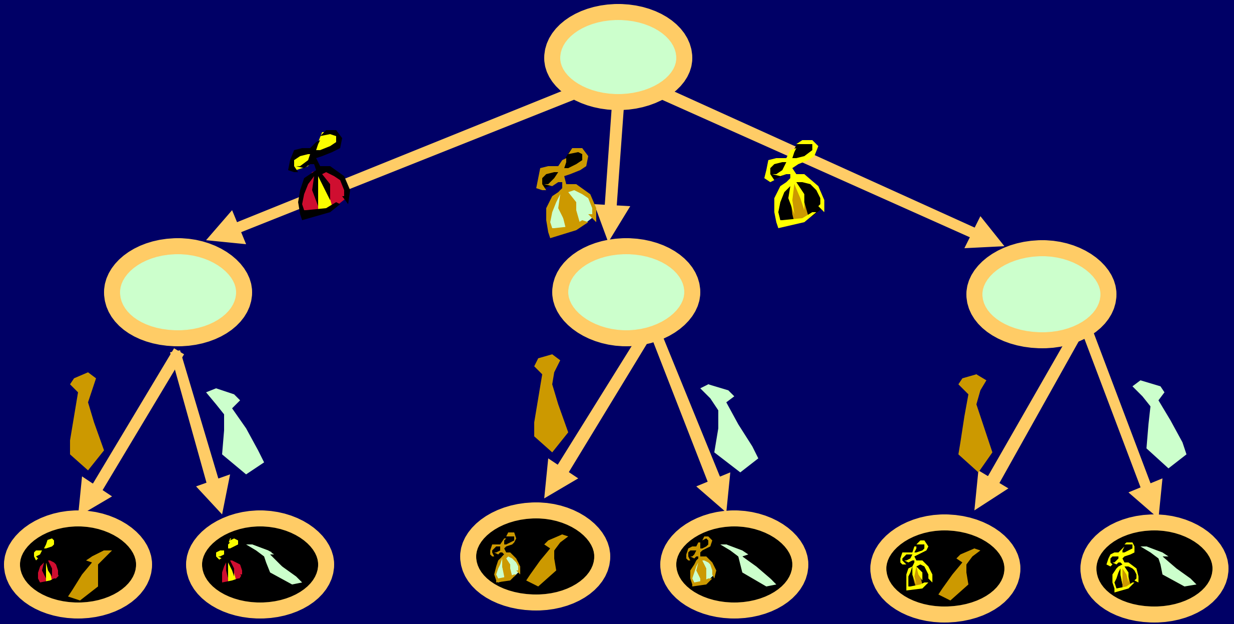




I own 3 beanies and 2 ties. How many beanie/tie combos might I wear to the ball tonight?



Choice Tree



A choice tree is a tree with an object called a “choice” associated with each edge and a label on each leaf.

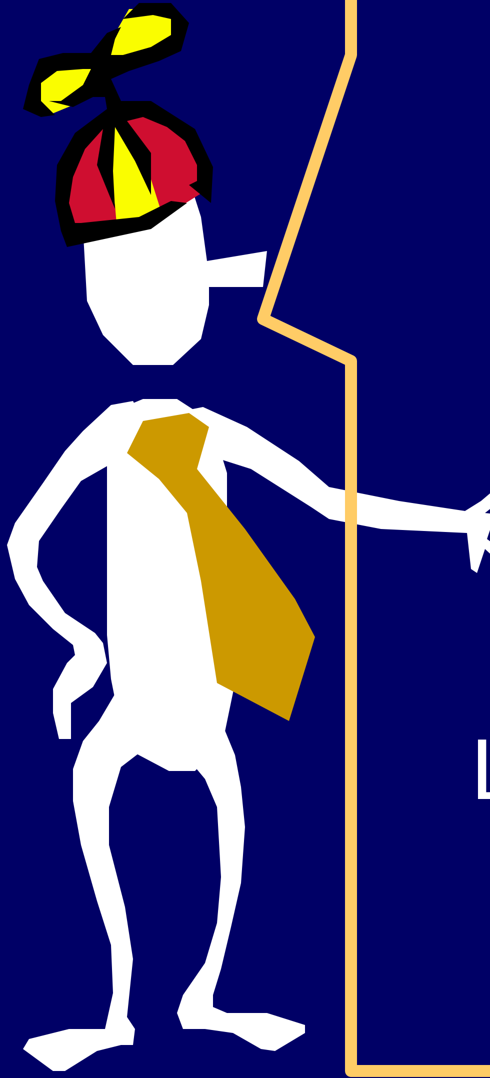
Definition: Labeled Tree

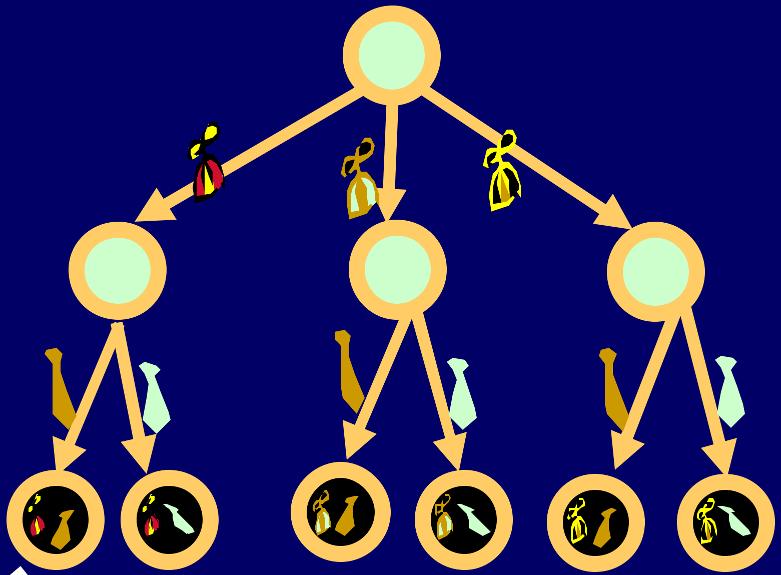
A tree node labeled by S is a tree T = <V,E> with an associated function

Label$_1$: V to S

A tree edge labeled by S is a tree T = <V,E> with an associated function

Label$_2$: E to S





was very illuminating.

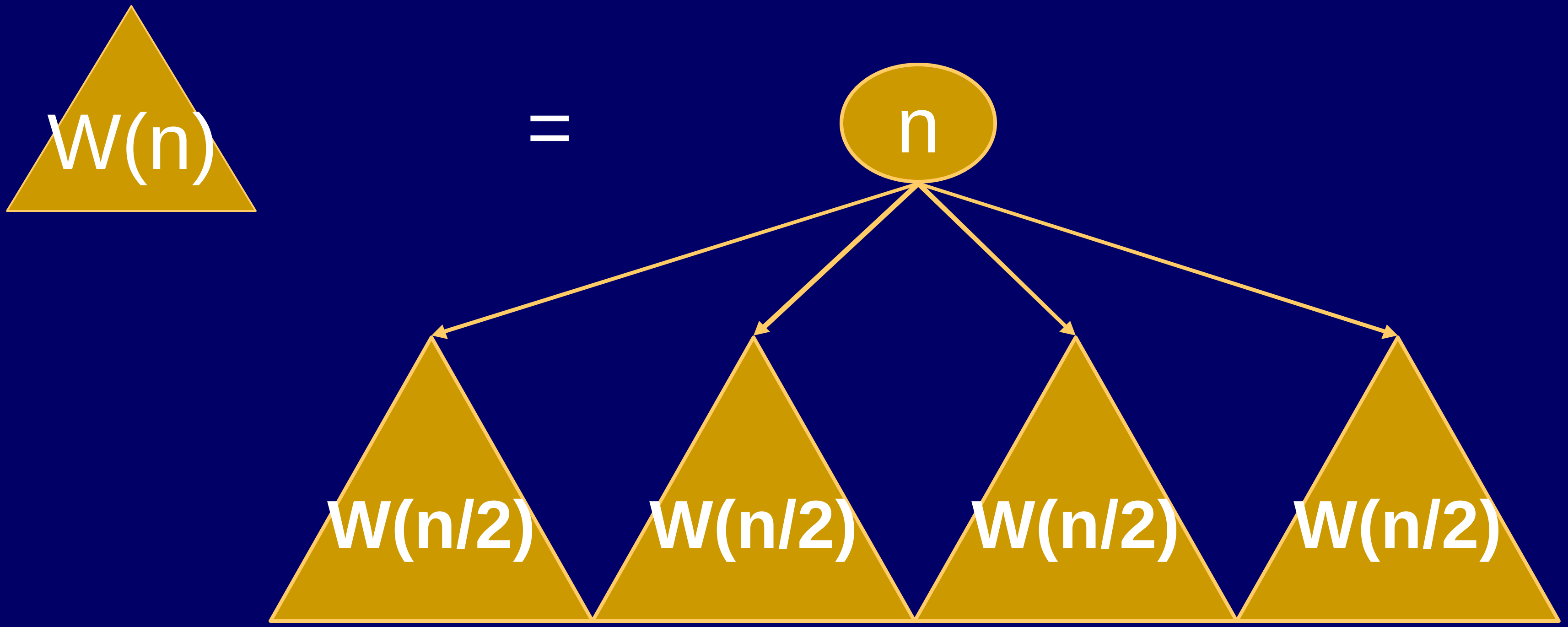
Let’s do something similar to illuminate the nature of $T(1)=1$; $T(n)= 4T(n/2) + n$

T(1)=1; T(n)= 4T(n/2) + n

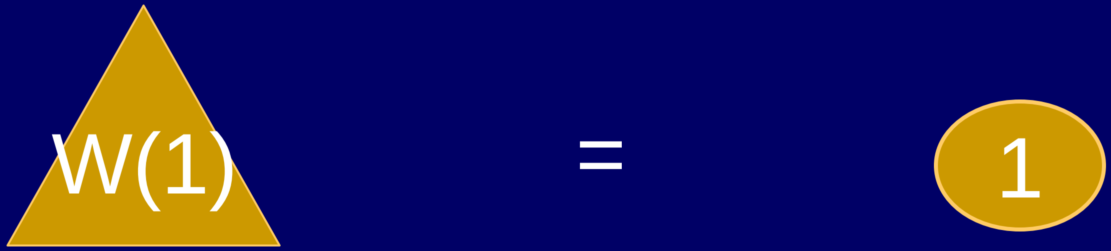
For each n (power of 2), we will define a tree W(n) node labeled by Natural numbers. W(n) will give us an incisive picture of T(n).

Inductive Definition Of Labeled Tree W(n)

$T(n) = n + 4\,T(n/2)$

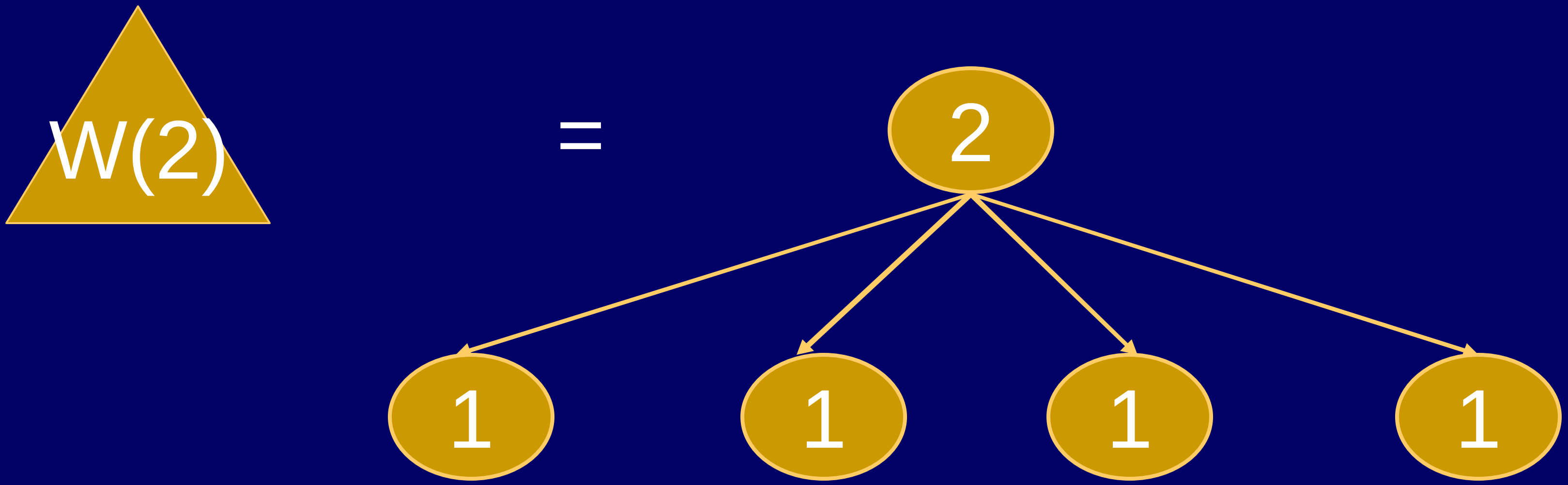


$T(1) = 1$

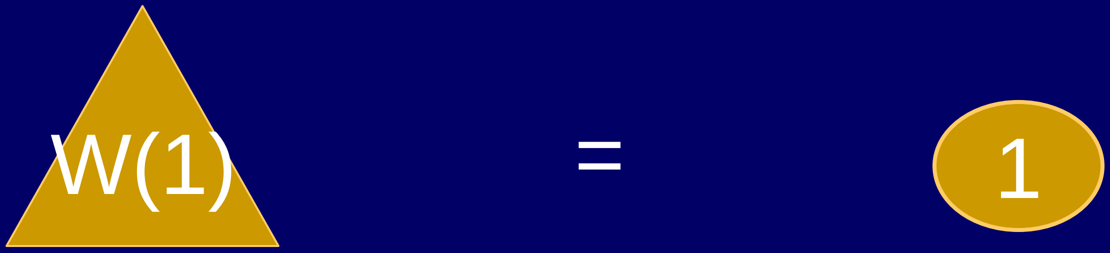


Inductive Definition Of Labeled Tree W(n)

T(2) = 6

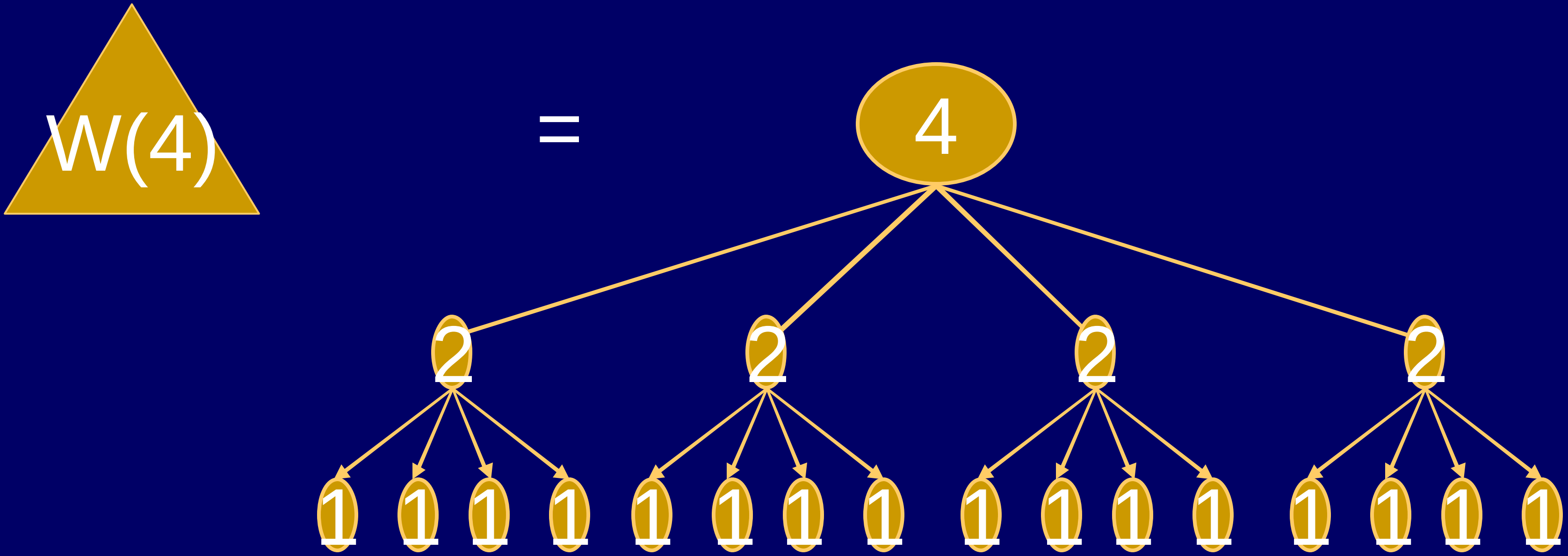


T(1) = 1



Inductive Definition Of Labeled Tree W(n)

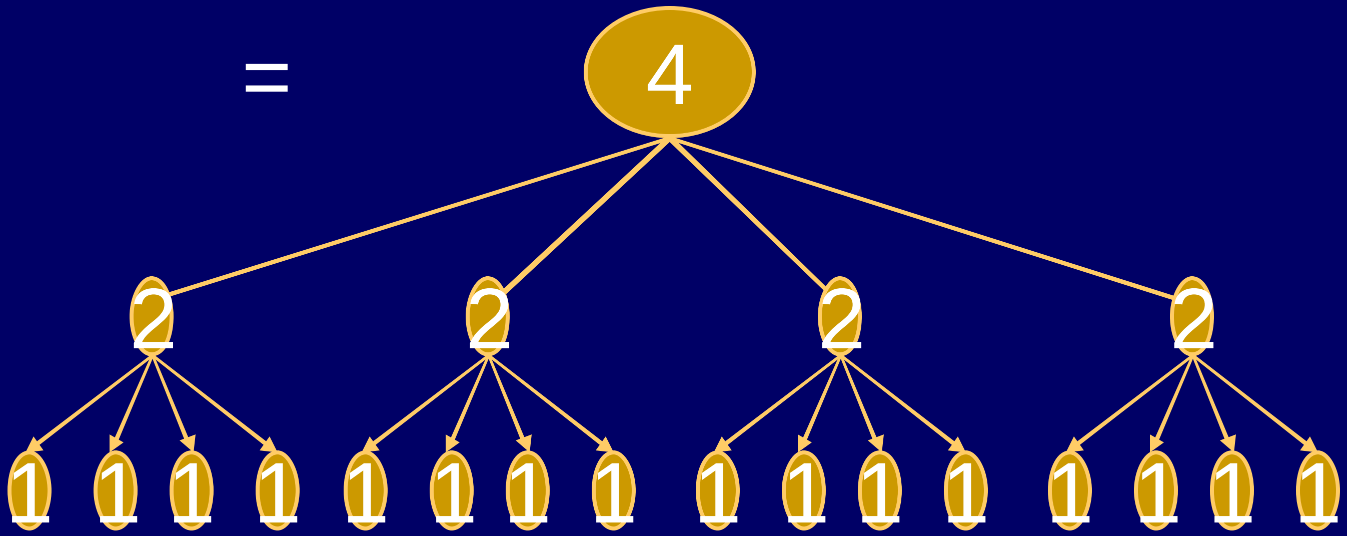
T(4) = 28

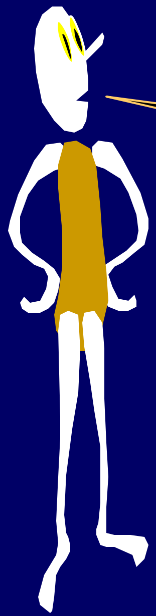


Inductive Definition Of Labeled Tree W(n)

T(4) = 28

=

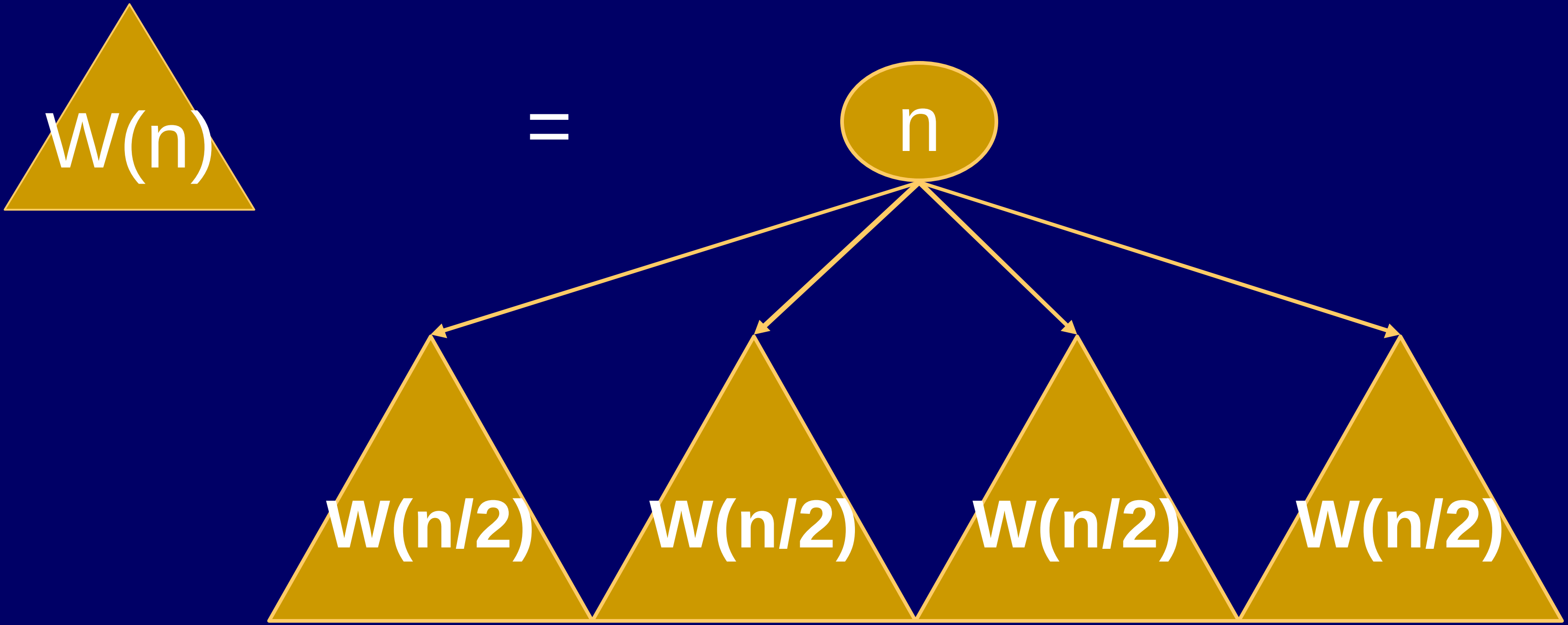




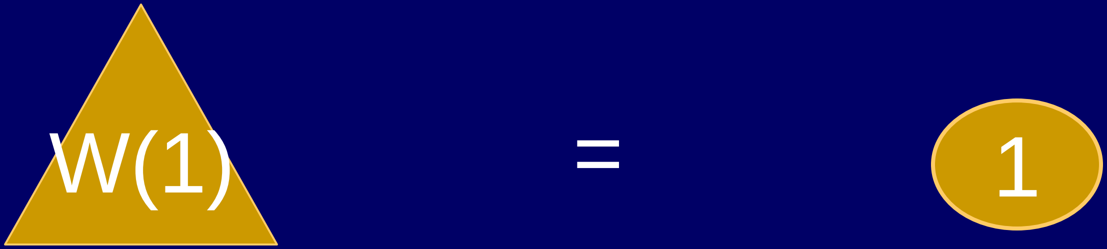
NOTE: When we sum all the node labels on W(n), we get T(n)

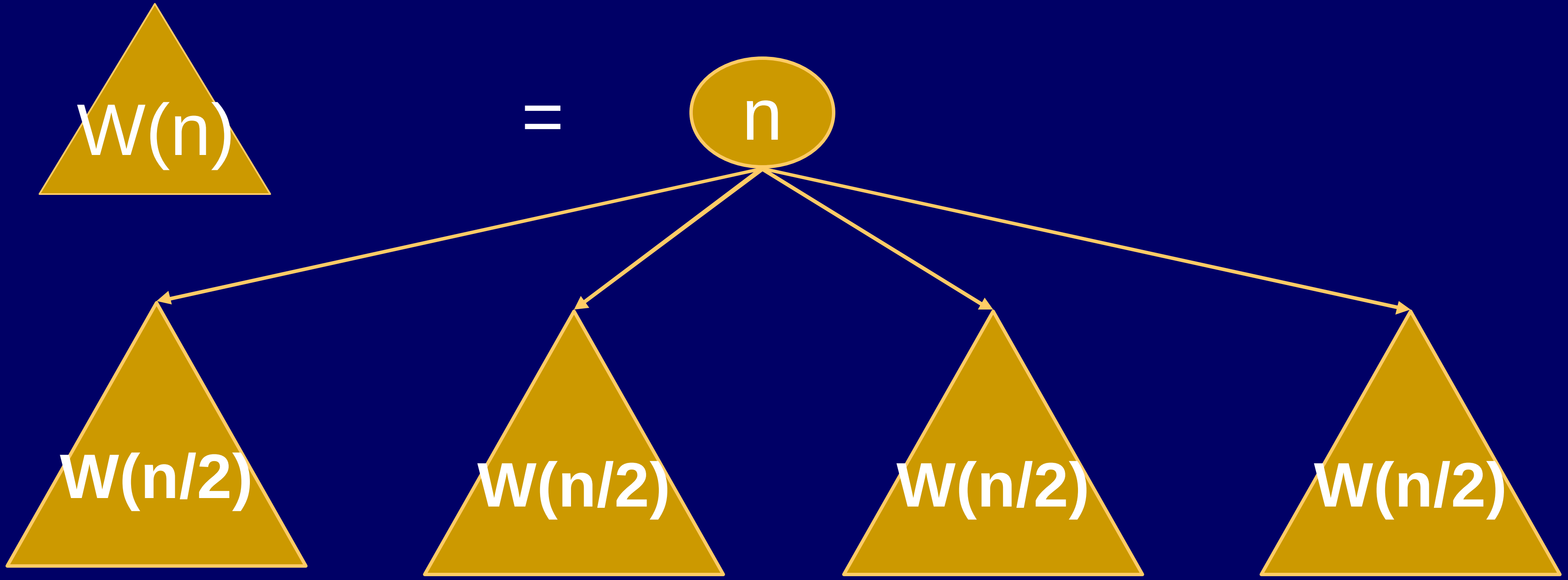
Invariant: LabelSum W(n) = T(n)

$$T(n) \quad = \quad n + 4\,T(n/2)$$



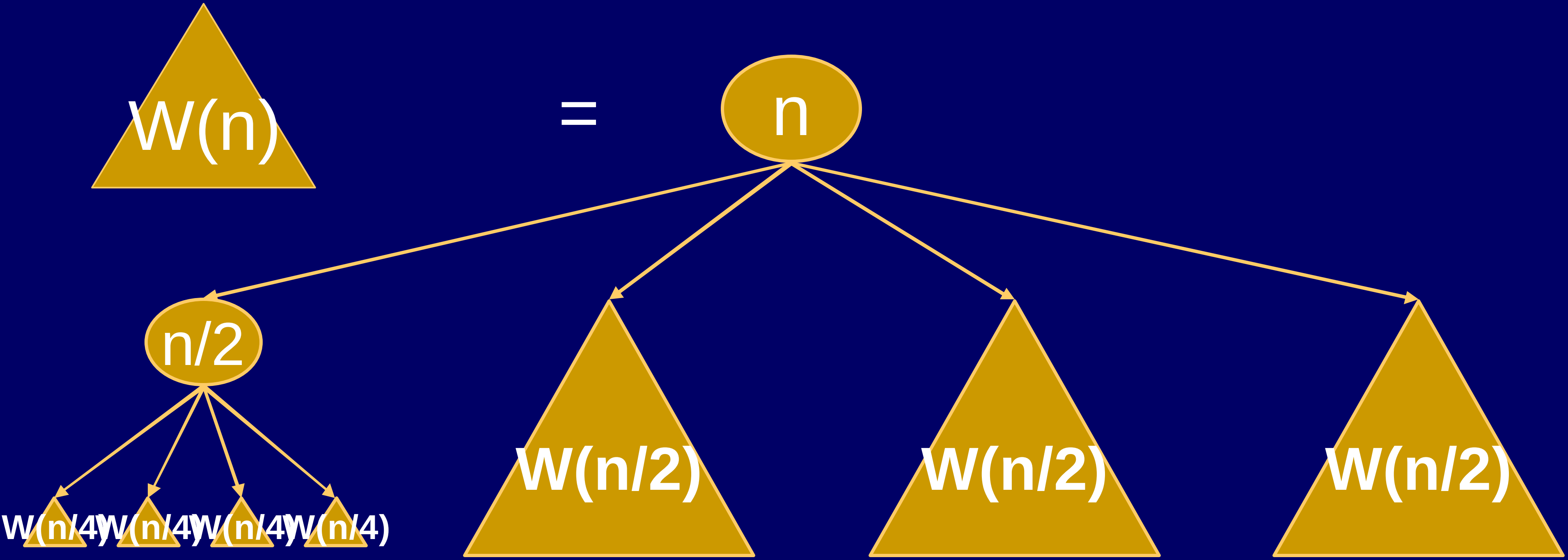
$$T(1) \quad = \quad 1$$

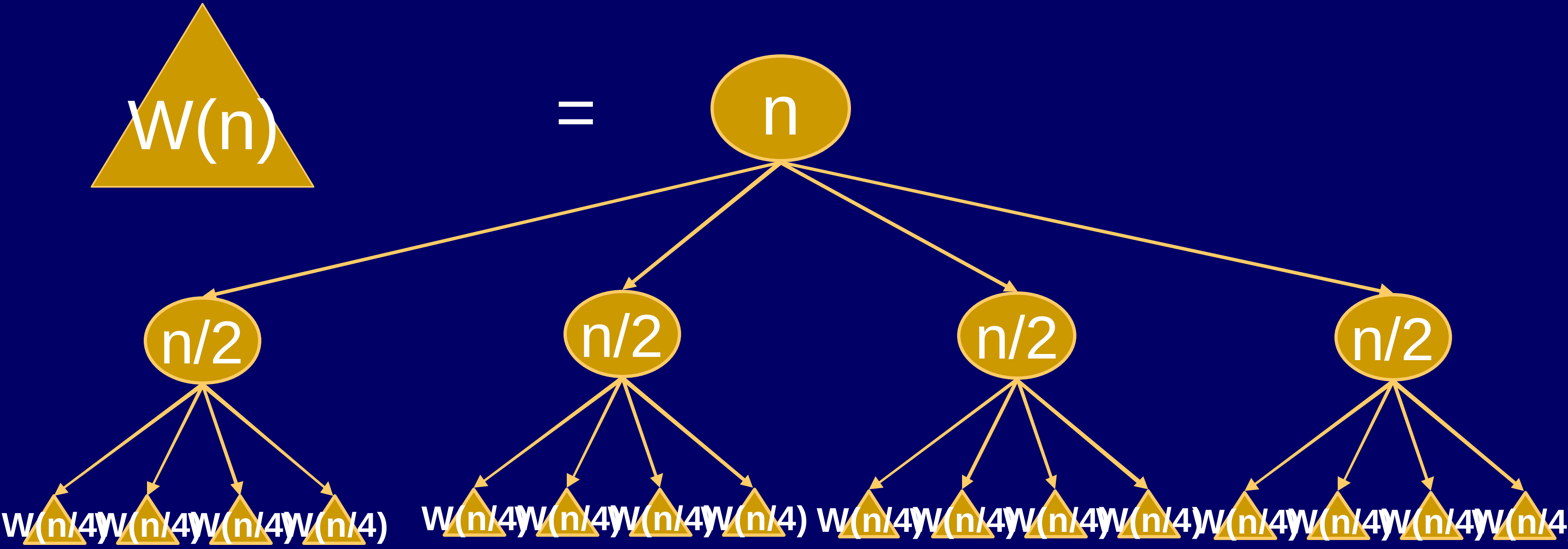


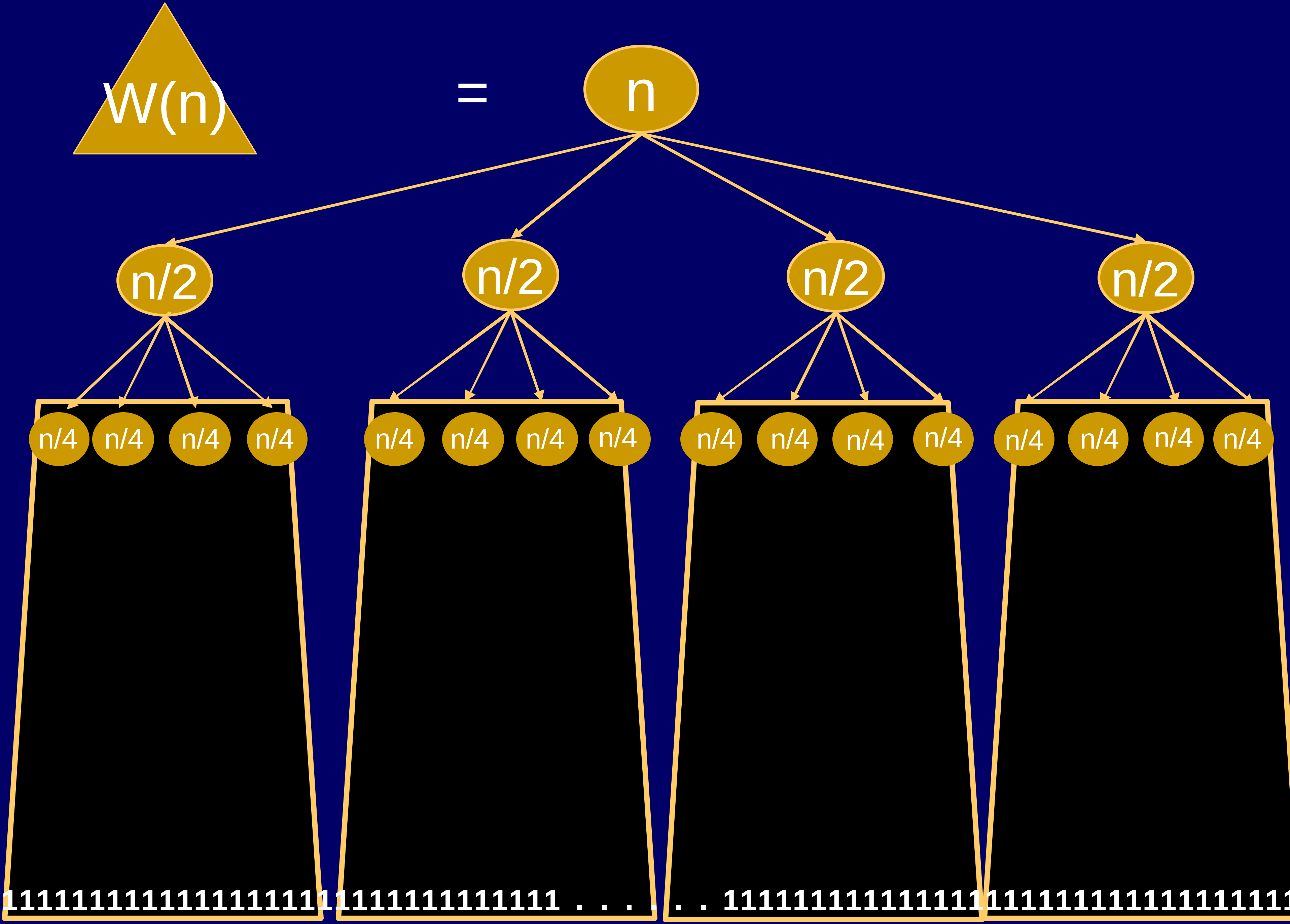


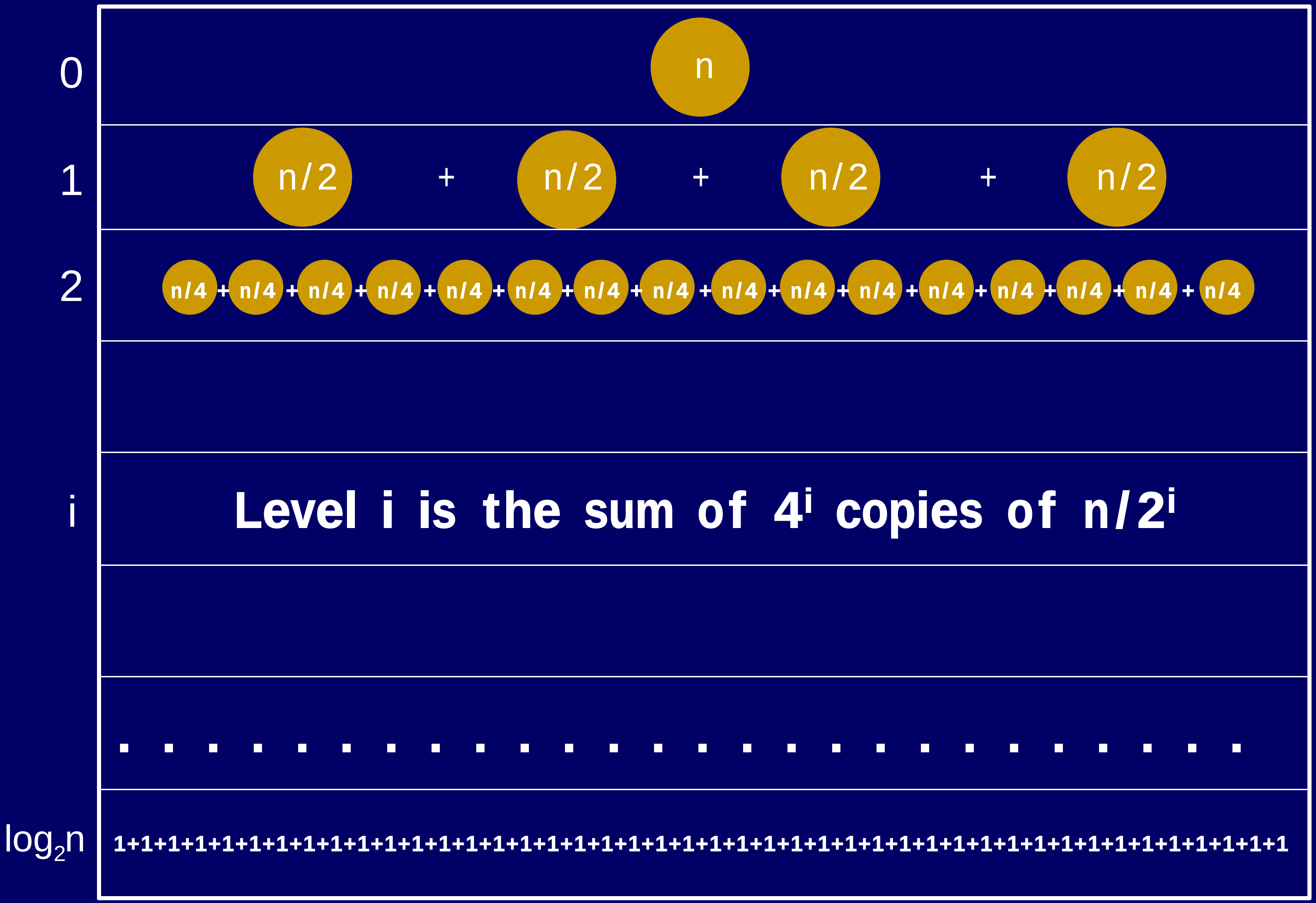
W(n) = n

W(n/2) W(n/2) W(n/2) W(n/2)

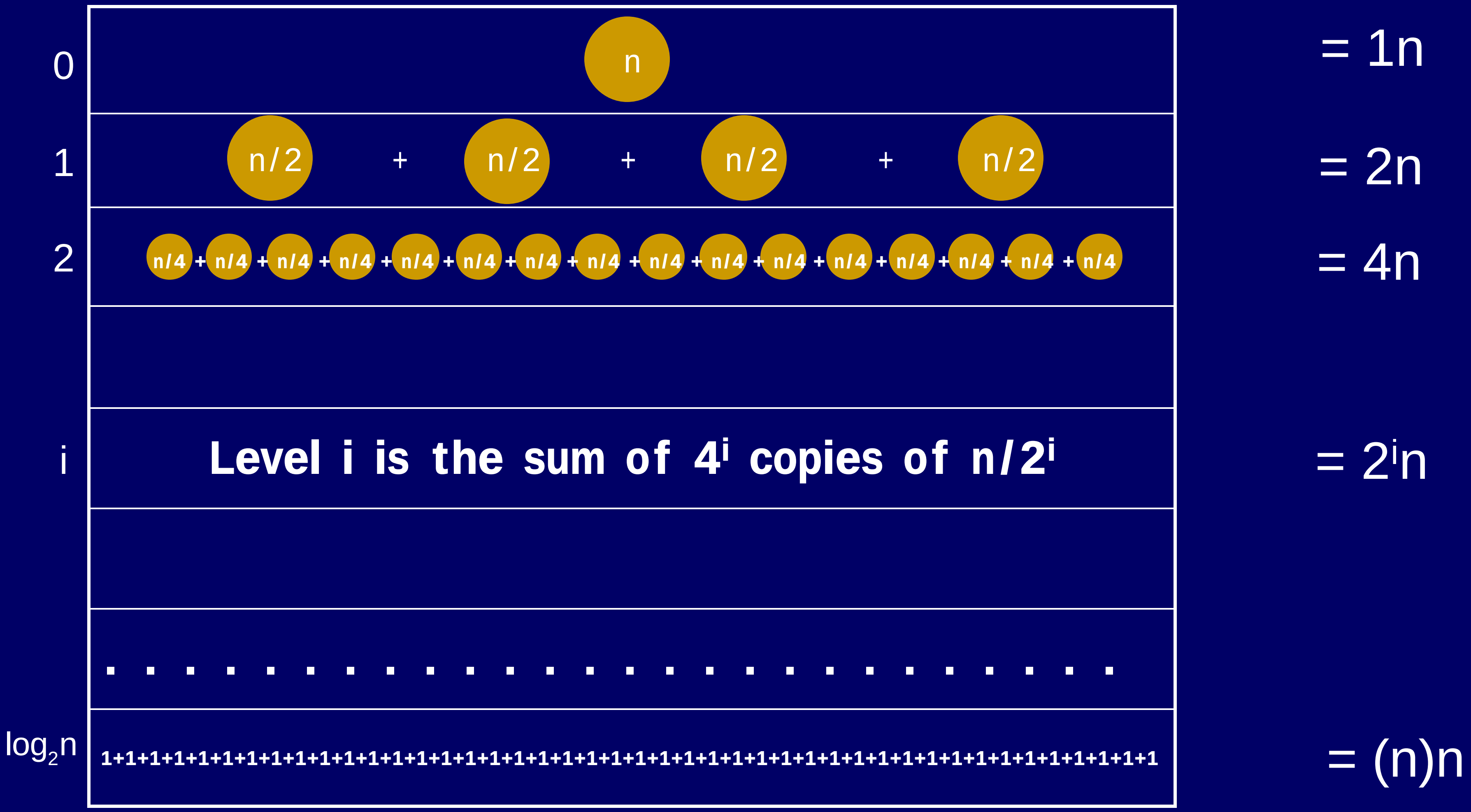

W(n)
=
n
n/2
W(n/2)
W(n/2)
W(n/2)
W(n/4)
W(n/4)
W(n/4)
W(n/4)


W(n)
=
n
n/2
n/2
n/2
n/2
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4)
W(n/4

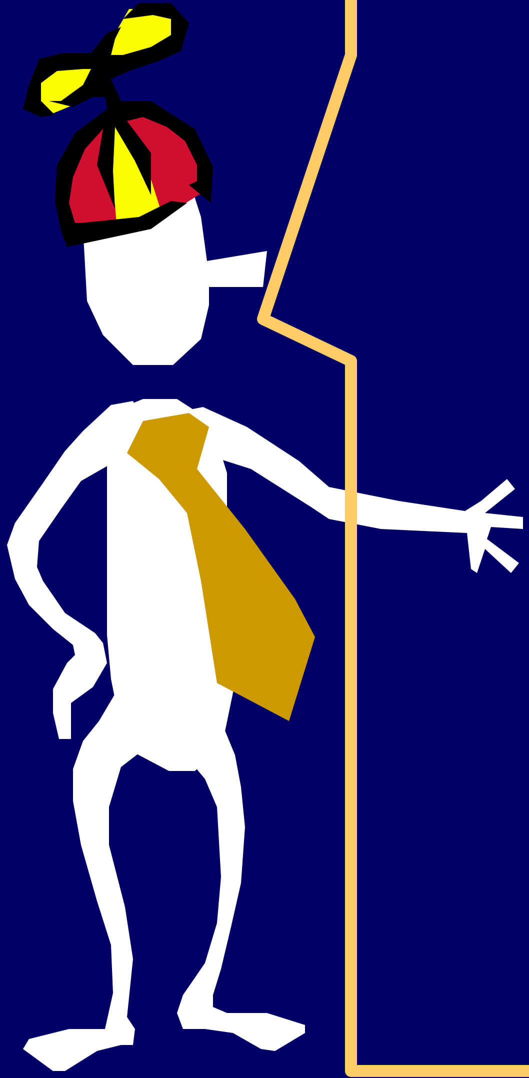

W(n)
=
n
n/2
n/2
n/2
n/2
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4
111111111111111111111111111111 111111111111111111111111111111



Level i is the sum of 4^i copies of $n/2^i$



$$n(1+2+4+8+ \ldots +n) = \qquad n(2n-1) = \qquad 2n^2-n$$



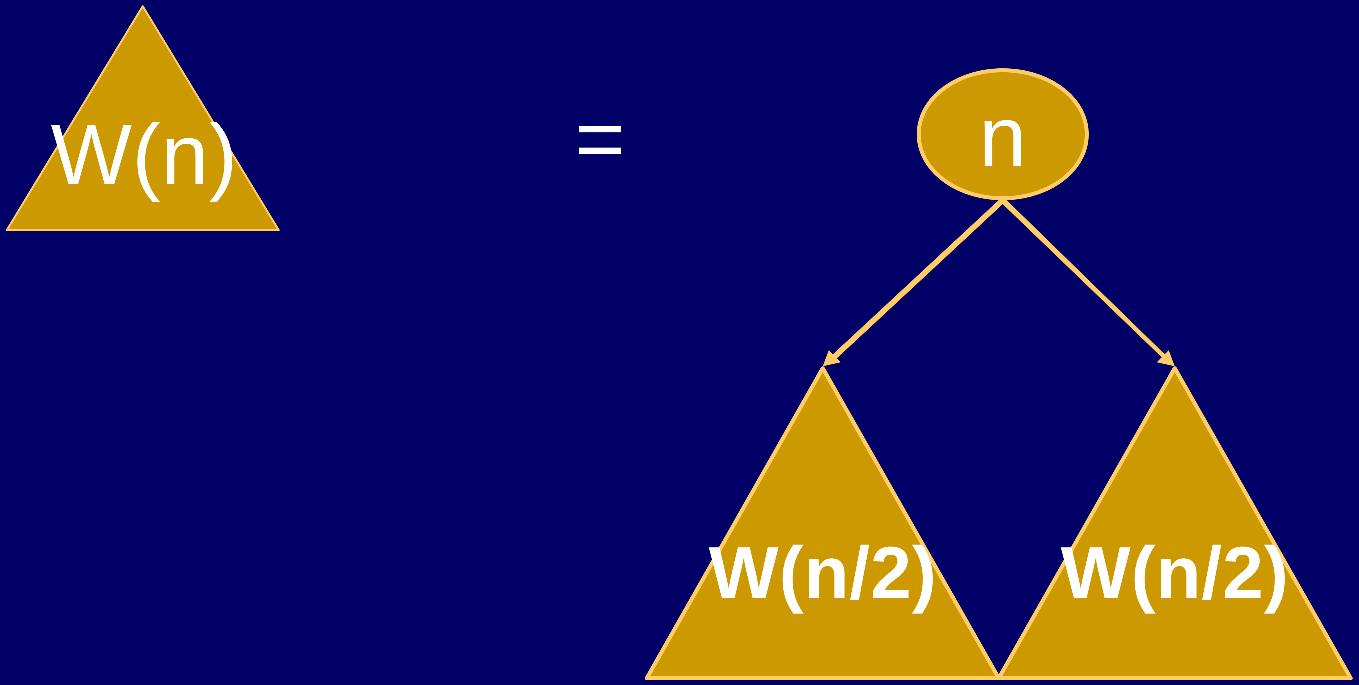
Instead of

$T(1)=1;\ T(n)=4T(n/2)+n$

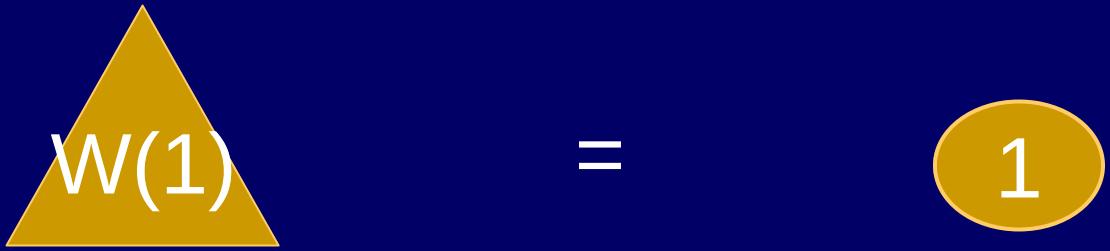
We could illuminate

$T(1)=1;\ T(n)=2T(n/2)+n$

$$T(n) = n + 2\,T(n/2)$$

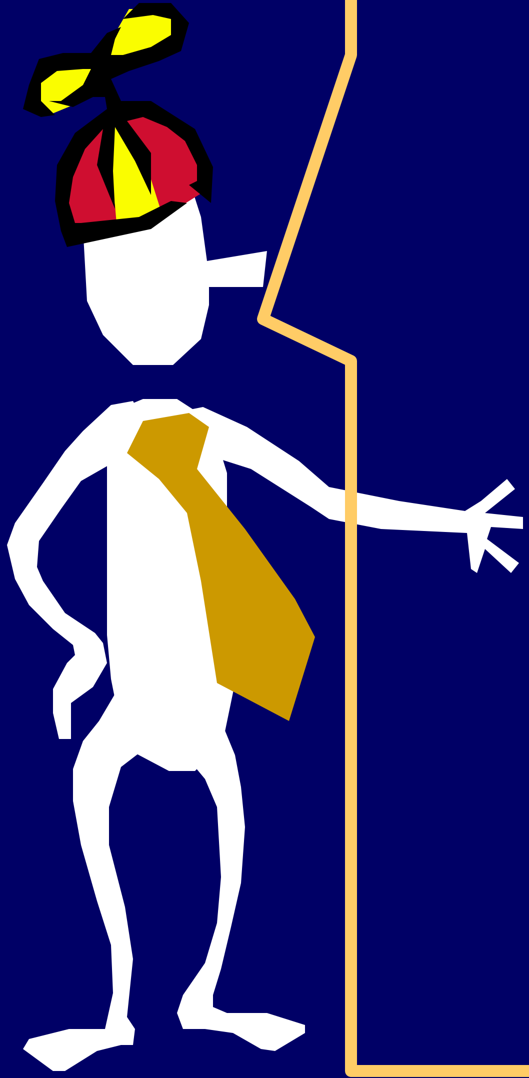


$$T(1) = 1$$



0	n
1	n/2 + n/2
2	
i	**Level i is the sum of 2^i copies of $n/2^i$**
	. . . . .
$\log_2 n$	**1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1**

0	n
1	n
2	
i	**n**
	. . . .
$\log_2 n$	n



$T(1)=1;\ T(n) = 2T(n/2) + n$

Has closed form: $n\log_2(n)$ where n is a power of 2

Representing a recurrence relation as a labeled tree is one of the basics tools you will have to put recurrence relations in closed form.

The Lindenmayer Game

$\Sigma = \{a,b\}$

Start word: a

SUB(a) = ab SUB(b) = a

For each $w = w_1\ w_2\ \ldots\ w_n$

$\mathrm{NEXT}(w) = \mathrm{SUB}(w_1)\mathrm{SUB}(w_2)..\mathrm{SUB}(w_n)$

The Lindenmayer Game

SUB(a) = ab SUB(b) = a

For each w = w_1 w_2 … w_n

NEXT(w) = SUB(w_1)SUB(w_2)..SUB(w_n)

Time 1: a

Time 2: ab

Time 3: aba

Time 4: abaab

Time 5: abaababa

The Lindenmayer Game

SUB(a) = ab SUB(b) = a

For each w = w_1 w_2 … w_n

NEXT(w) = SUB(w_1)SUB(w_2)..SUB(w_n)

Time 1: a

Time 2: ab

Time 3: aba

Time 4: abaab

Time 5: abaababa

How long are the strings as a function of time?

Aristid Lindenmayer (1925-1989)

1968 Invents L-systems in Theoretical Botany

Time 1: a

Time 2: ab

Time 3: aba

Time 4: abaab

Time 5: abaababa

FIBONACCI(n)
cells at time n

The Koch Game

$\Sigma = \{F,+,-\}$

Start word: F

SUB(F) = F+F--F+F SUB(+)=+ SUB(-)=-

For each $w = w_1\ w_2\ \ldots\ w_n$

$\text{NEXT}(w) = \text{SUB}(w_1)\text{SUB}(w_2)..\text{SUB}(w_n)$

The Koch Game

Gen 0:F

Gen 1: F+F--F+F

Gen 2: F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F

The Koch Game

Picture representation:

F draw forward one unit
+ turn 60 degree left
- turn 60 degrees right.

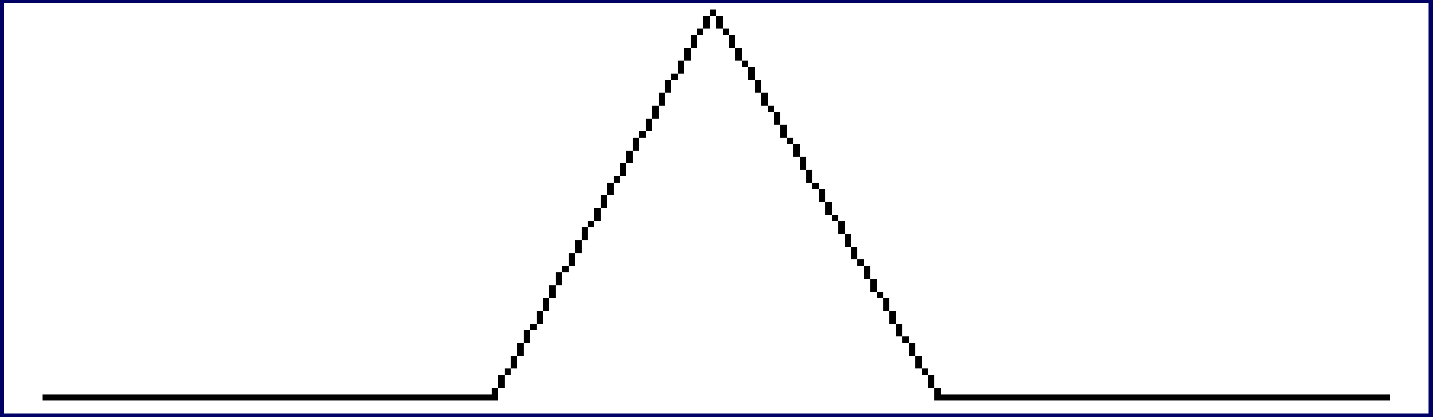
Gen 0: F

Gen 1: F+F--F+F

Gen 2: F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F

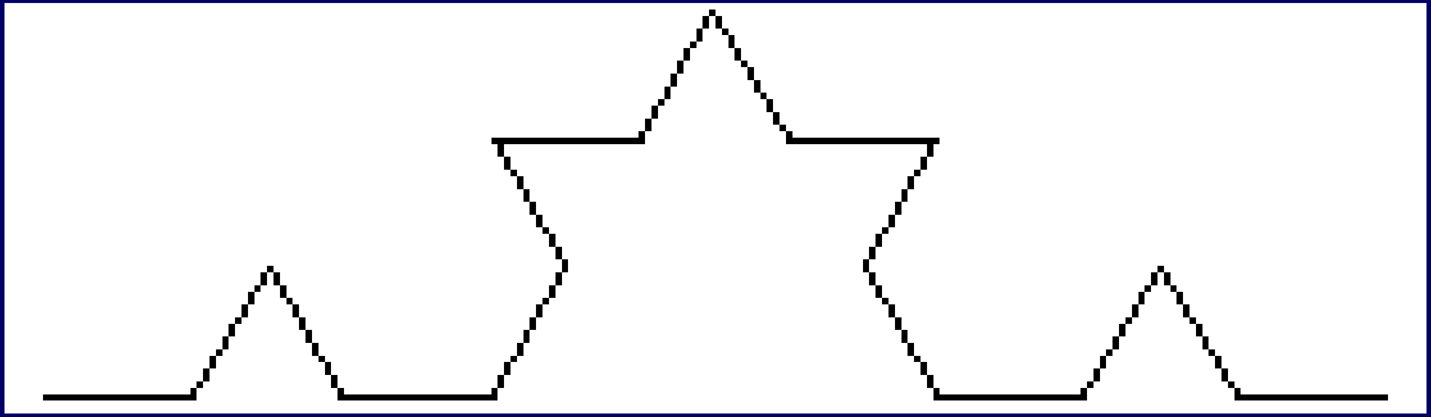
The Koch Game

F+F--F+F

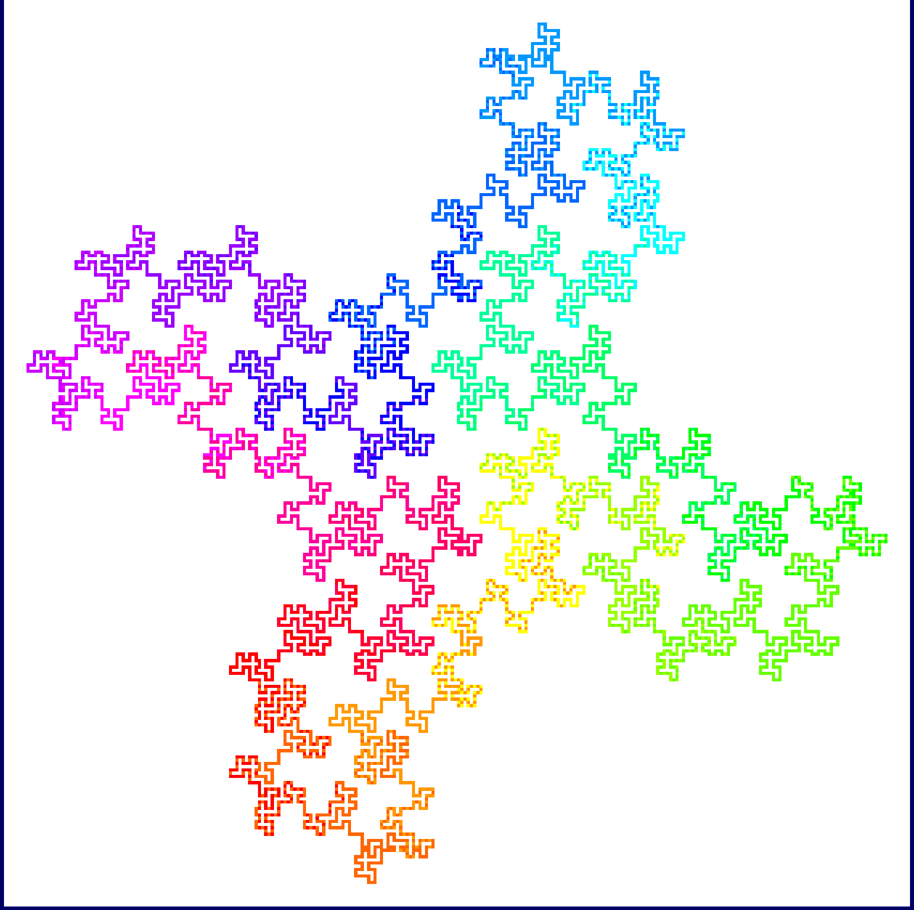


The Koch Game

F+F--F+F+F+F--F+F--F+F--F+F+F+F--F+F



Koch Curve

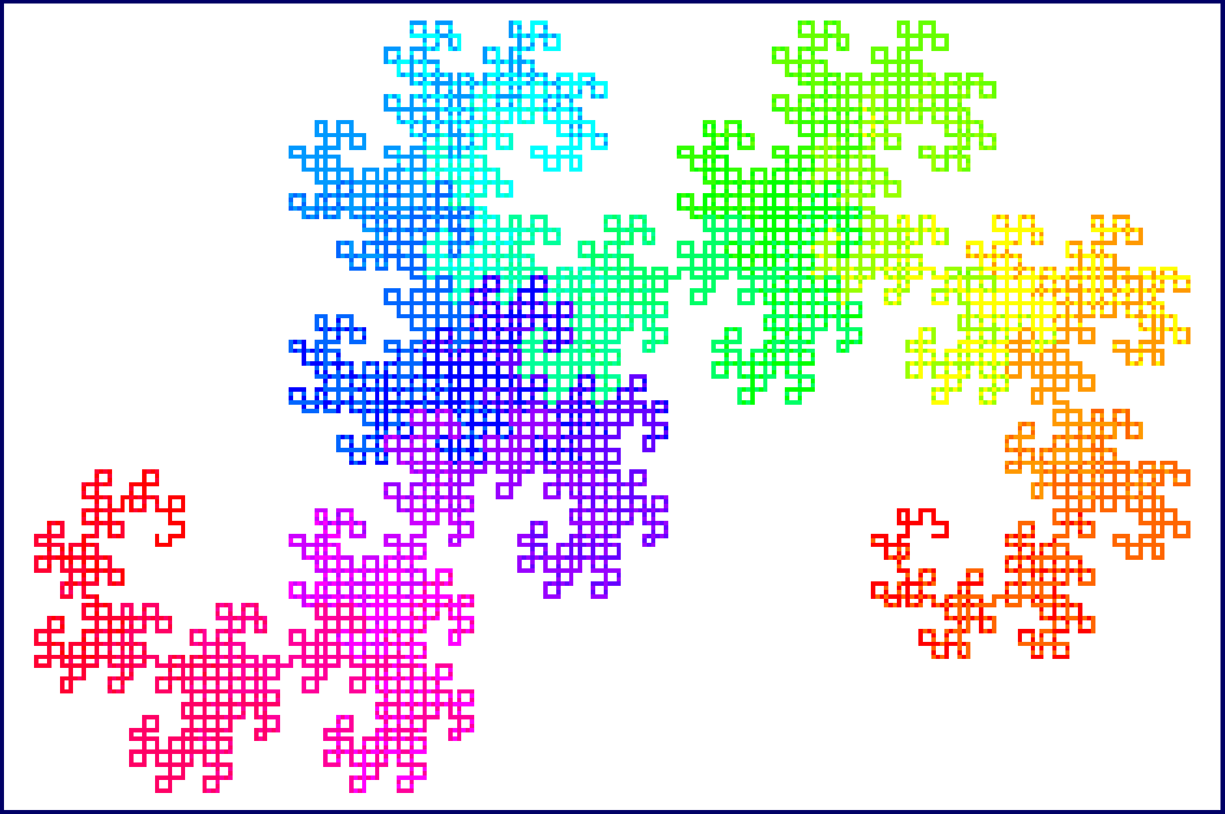


Dragon Game

SUB(X) =X+YF+

SUB(Y) = -FX-Y

Dragon Curve:

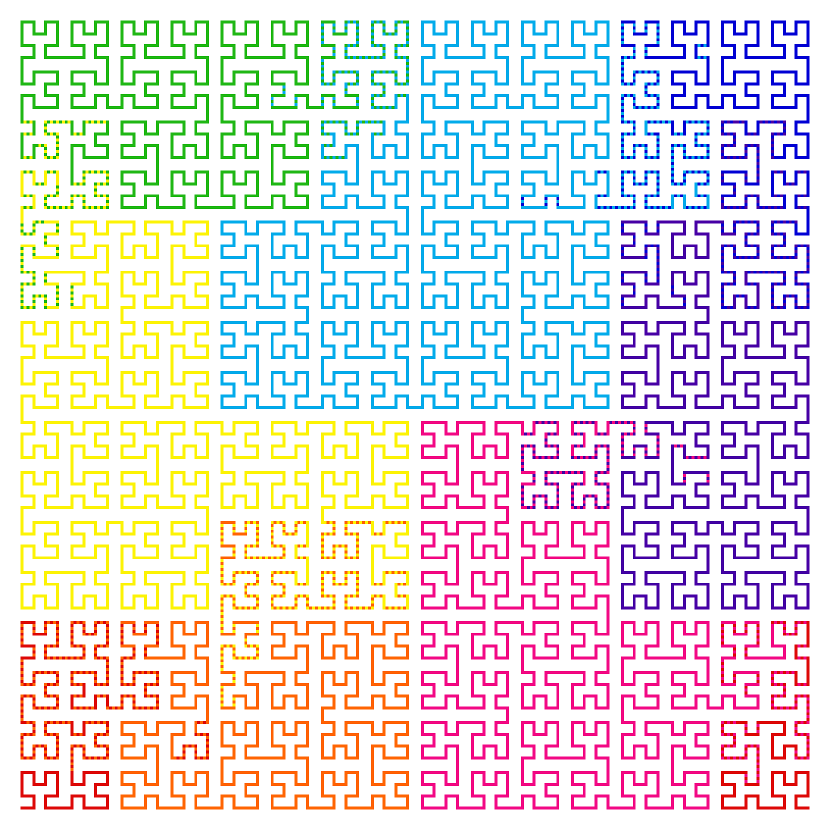


Hilbert Game

SUB(L)= +RF-LFL-FR+

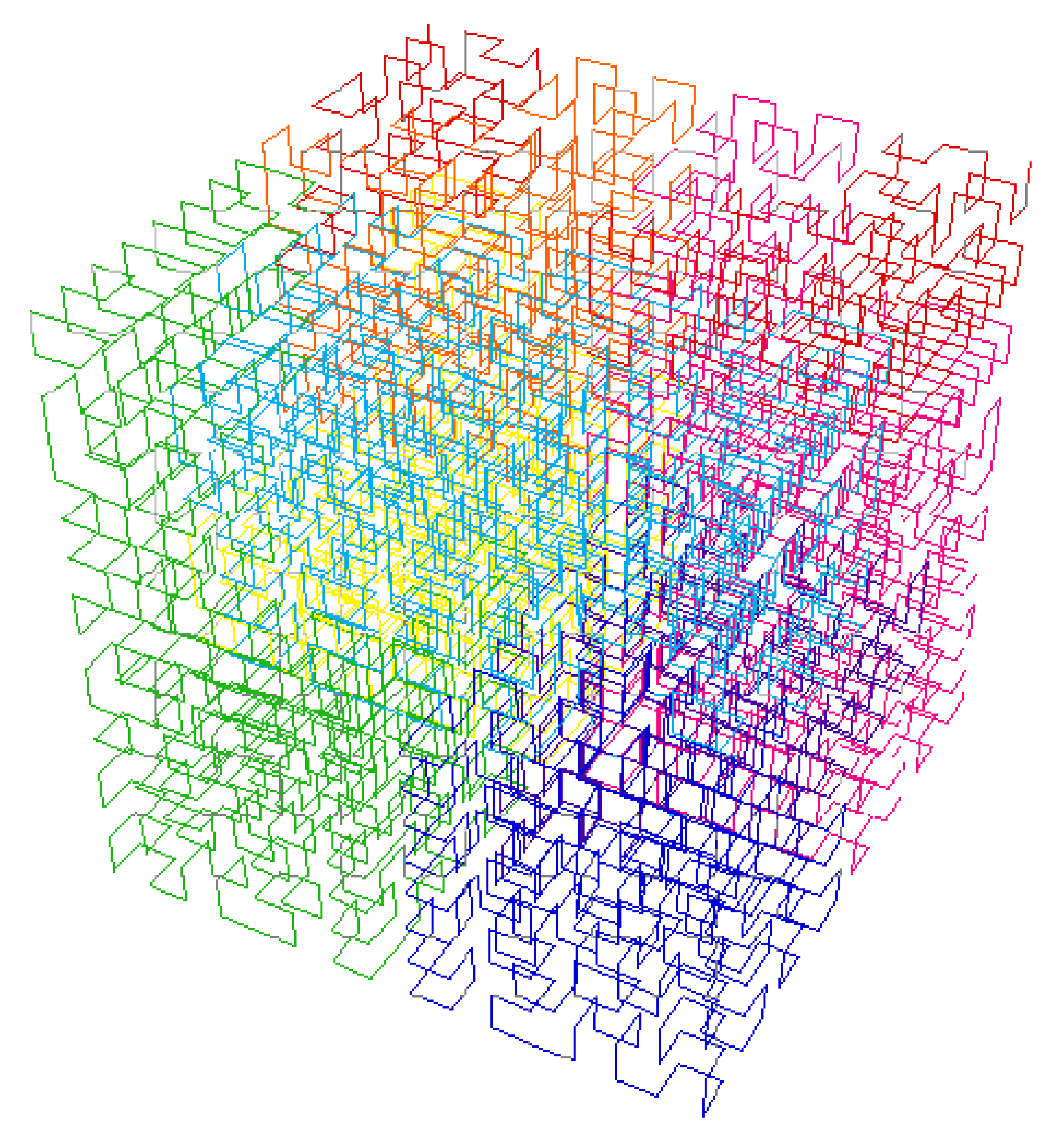
SUB(R)= -LF+RFR+FL-

Hilbert Curve:

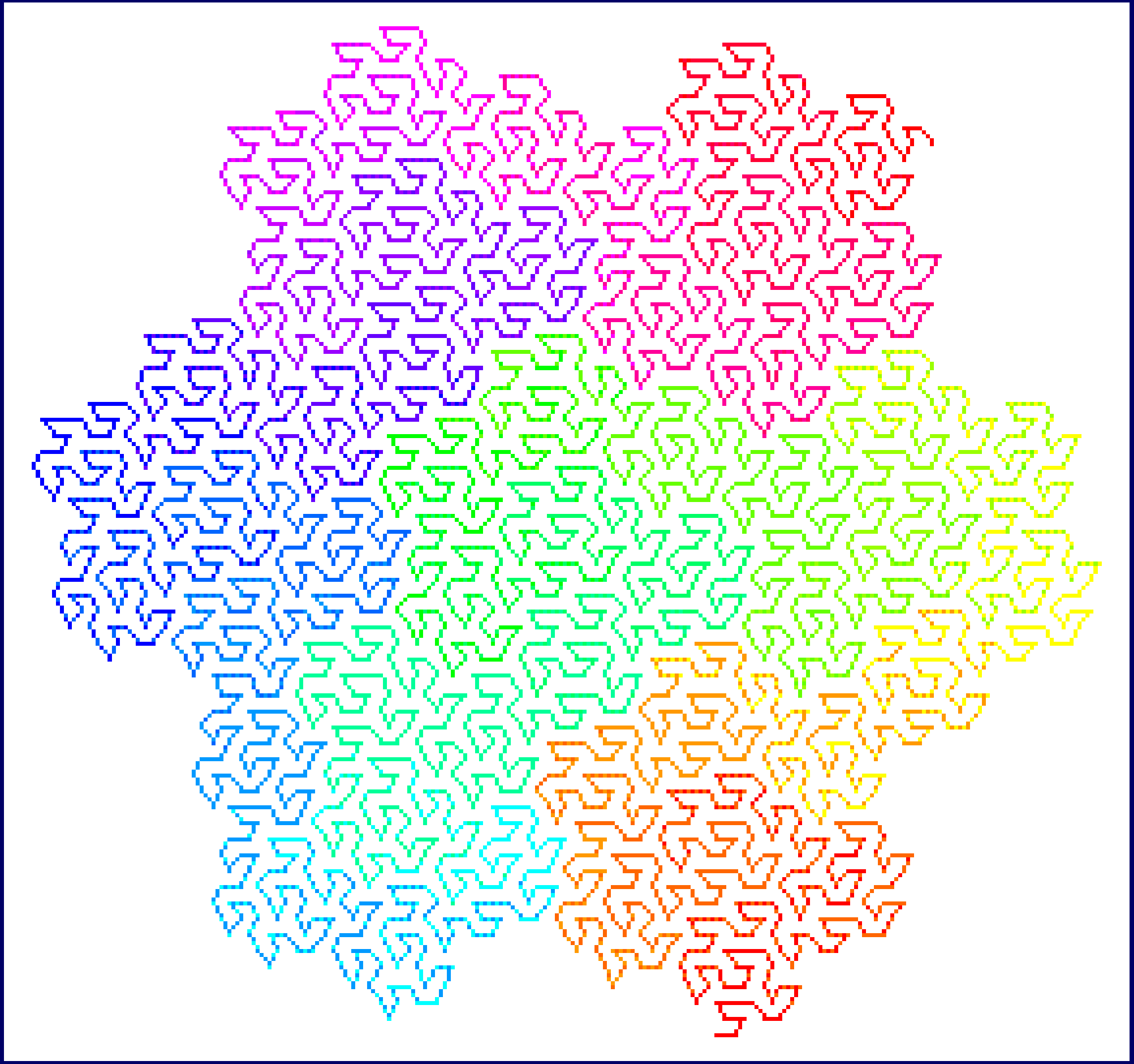


Note: Make 90 degree turns instead of 60 degrees.

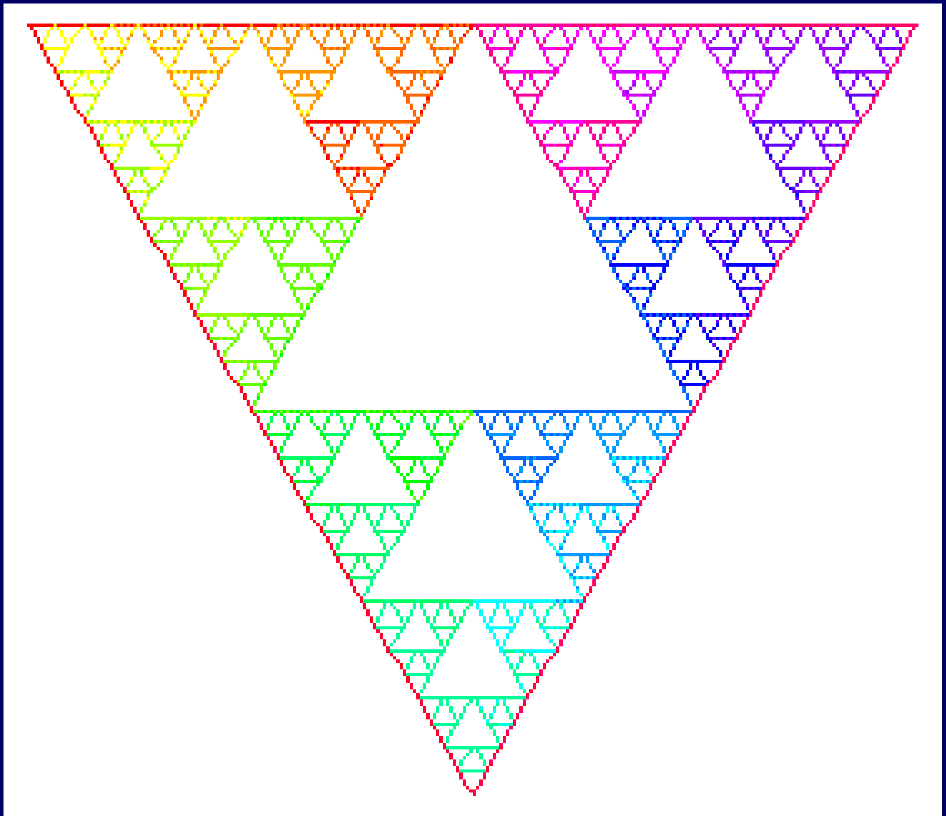
Hilbert’s Space Filling Curve



Peano-Gossamer Curve



Sierpinski Triangle

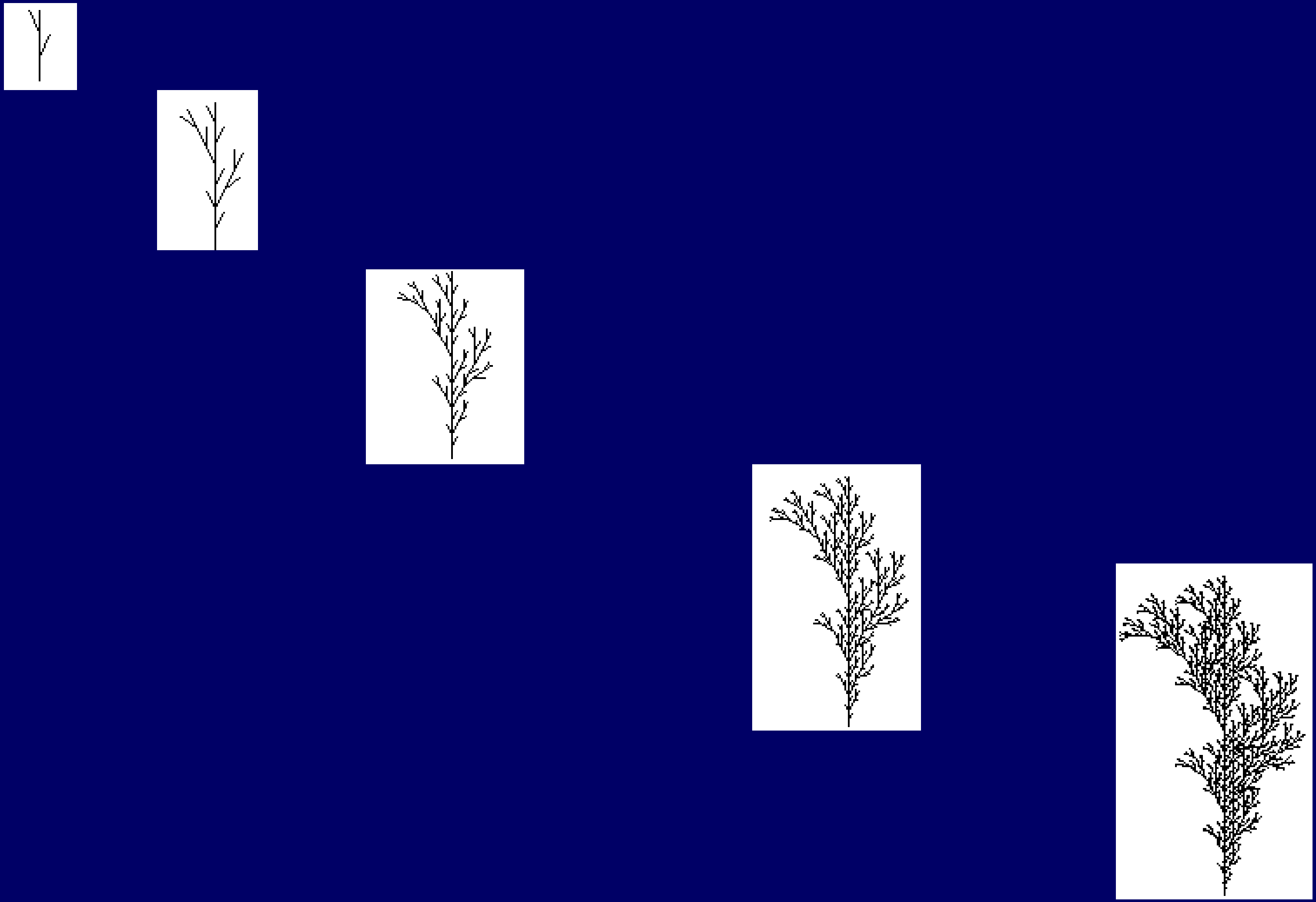


Lindenmayer 1968

SUB(F) = F[-F]F[+F][F]

Interpret the stuff inside brackets as a branch.

Lindenmayer 1968



Inductive Leaf