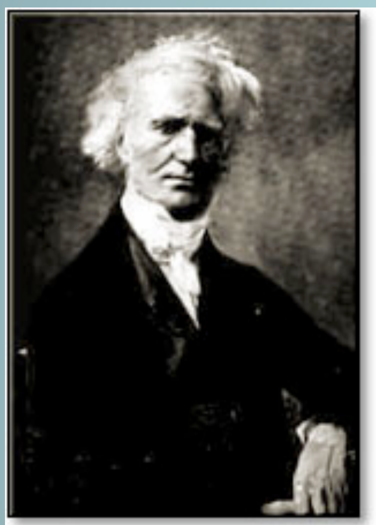
Introduction to Recursion



Jacques Binet
(from Wikipedia)

The classic recursion relation is given by:

F(1) = 1

F(2) = 1

F(n) = F(n-1) + F(n-2)

Which gives rise to the Fibonacci sequence

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …….

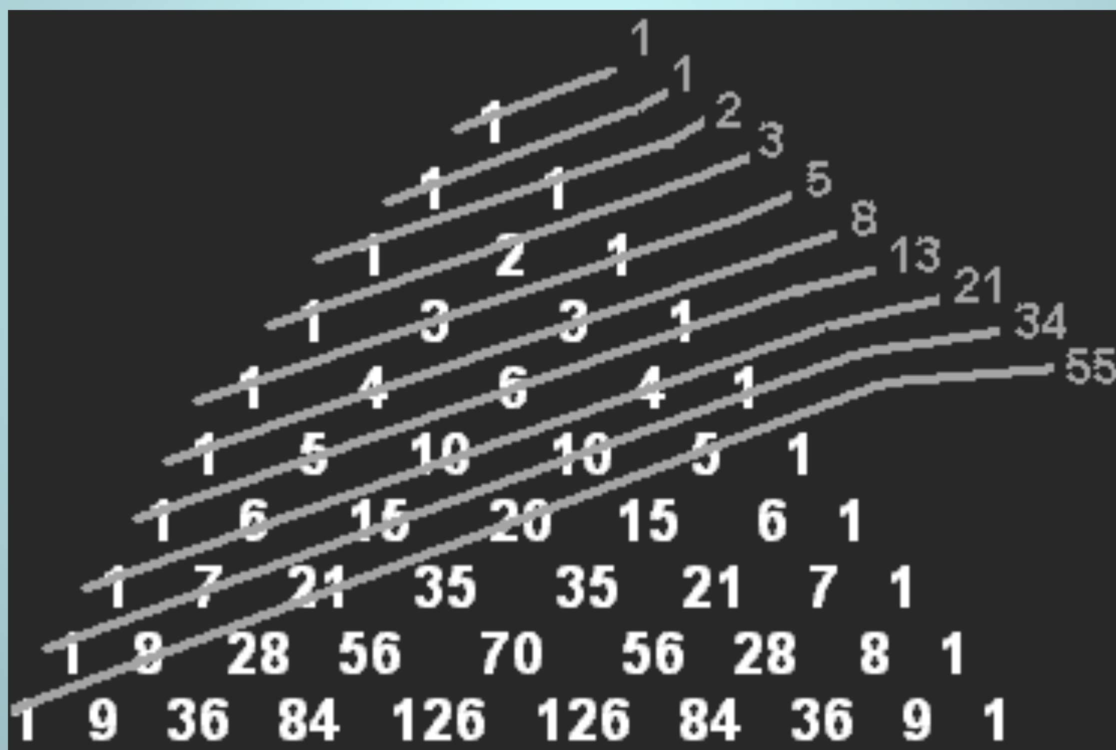
It has the closed-form solution (Binet's Formula):

$$F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} = \frac{\varphi^n - (-1/\varphi)^n}{\sqrt{5}},$$

where $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.6180339887\ldots$ is the golden ratio.

Fibonacci Numbers in Pascal's Triangle

The Fibonacci numbers are the sums of the terms along specific diagonals.



Methods of Solving Recurrence Relations

- Repeated substitution
- Analysis of the recursion tree
- Applying one of the Master Theorems
- Guess an upper bound and prove it

Example 1: Linear Search in an Array

- Recursively examine the first element in the array and then search the remaining elements.

e_1	e_2	e_3	.	.	.	.	.	.	e_n

- $T(n) = T(n-1) + c$
- T(n), the time to search n elements, is the time to examine 1 element, plus the time to search the remaining n-1 elements.

Linear Search (continued)

- Repeated substitution yields:

T(n) = T(n-1) + c (1)

But, T(n-1) = T(n-2) + c, from above

Substituting back in:

T(n) = T(n-2) + c + c

Gathering like terms yields:

T(n) = T(n-2) + 2c (2)

Linear Search (continued)

- Substitution number 3:

$T(n) = T(n-2) + 2c$

$T(n-2) = T(n-3) + c$

$T(n) = T(n-3) + c + 2c$

$T(n) = T(n-3) + 3c \qquad (3)$

- Substitution number 4:

$T(n) = T(n-4) + 4c \qquad (4)$

Example 1 – list of intermediates

Result at i^{th} substitution	i
$T(n) = T(n-1) + 1c$	1
$T(n) = T(n-2) + 2c$	2
$T(n) = T(n-3) + 3c$	3
$T(n) = T(n-4) + 4c$	4

Linear Search (continued)

- The k^{th} substitution yields:

  $T(n) = T(n-k) + kc$
- The two variables, k and n, are related.
- If d is a small constant $T(d)$ can be calculated from the algorithm itself, and is also a constant, i.e., $O(1)$.

Linear Search (continued)

- For this example let us stop at $T(0)$.
- Let $n-k = 0$. Thus $n=k$
- Substituting n for k yields:

$$T(n) = T(n-n) + nc$$

$$T(n) = T(0) + nc = nc + c_0 = O(n),$$

where $T(0)$ is some constant c_0.

Example 2: Binary Search

- Algorithm: check the element in the middle of the list. Then search the lower or upper half of the list.
- $T(n) = T(n/2) + c$

  where c is the cost of checking the middle element, and is $O(1)$, a constant.

Binary Search (continued)

Three repeated substitutions yield:

$T(n) = T(n/2) + c$ (1)

but $T(n/2) = T(n/4) + c$, so

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

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

$T(n/4) = T(n/8) + c$

$T(n) = T(n/8) + c + 2c$

$T(n) = T(n/8) + 3c$ (3)

Binary Search (continued)

Result at i^{th} unwinding	i
$T(n) = T(n/2) + c$	1
$T(n) = T(n/4) + 2c$	2
$T(n) = T(n/8) + 3c$	3
$T(n) = T(n/16) + 4c$	4

Binary Search (continued)

Result at i^{th} substitution			i
T(n)	= T(n/2) + c	$= T(n/2^1) + 1c$	1
T(n)	= T(n/4) + 2c	$= T(n/2^2) + 2c$	2
T(n)	= T(n/8) + 3c	$= T(n/2^3) + 3c$	3
T(n)	= T(n/16) + 4c	$= T(n/2^4) + 4c$	4

Binary Search (continued)

- After k substitutions:
- $T(n) = T(n/2^k) + kc$
- Let $T(1) = c_0$
- Let $n = 2^k$
- Then $\log_2 n = k$
- Substituting to get rid of the k:

$T(n) = T(1) + c \log_2 n$

$T(n) = c \log_2 n + c_0$

$T(n) = O(\log n)$