Aula 06: Laboratório - Recursão (parte 2)

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

3Q-20107

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES®

founded in 1964 by N. J. A. Sloane

1,2,6,24,120 Search Hints

(Greetings from The On-Line Encyclopedia of Integer Sequences!)

Search: **seq:1,2,6,24,120**

Displaying 1-10 of 348 results found. page 1 2 3 4 5 6 7 8 9 10 ... 35

Sort: relevance | references | number | modified | created Format: long | short | data

A000142 Factorial numbers: n! = 1*2*3*4*...*n (order of symmetric group S_n, number of permutations of n letters). +20 1721

(Formerly M1675 N0659)

1, **1**, **2**, **6**, **24**, **120**, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600, 6227020800, 87178291200, 1307674368000, 20922789888000, 355687428096000, 6402373705728000, 121645100408832000, 2432902008176640000, 51090942171709440000, 1124000727777607680000 (list; graph; refs; listen; history; text; internal format)

OFFSET 0,3

COMMENTS

```
The earliest publication that discusses this sequence appears to be the
  Sepher Yezirah [Book of Creation], circa AD 300. (See Knuth, also the
  Zeilberger link) - N. J. A. Sloane, Apr 07 2014
For n >= 1, a(n) is the number of n X n (0,1) matrices with each row and
  column containing exactly one entry equal to 1.
This sequence is the BinomialMean transform of A000354. (See A075271 for
  definition.) - John W. Layman, Sep 12 2002. This is easily verified from
  the Paul Barry formula for A000354, by interchanging summations and using
  the formula: Sum_k (-1)^k C(n-i, k) = KroneckerDelta(i,n). - David Callan,
  Aug 31 2003
Number of distinct subsets of T(n-1) elements with 1 element A, 2 elements
  B,..., n - 1 elements X (e.g., at n = 5, we consider the distinct subsets
  of ABBCCCDDDD and there are 5! = 120). - Jon Perry, Jun 12 2003
n! is the smallest number with that prime signature. E.g., 720 = 2^4 * 3^2 *
  5. - Amarnath Murthy, Jul 01 2003
a(n) is the permanent of the n X n matrix M with M(i, j) = 1. - Philippe
  Deléham, Dec 15 2003
Given n objects of distinct sizes (e.g., areas, volumes) such that each
  object is sufficiently large to simultaneously contain all previous
  objects, then n! is the total number of essentially different arrangements
  using all n objects. Arbitrary levels of nesting of objects are permitted
  within arrangements. (This application of the sequence was inspired by
  considering leftover moving boxes.) If the restriction exists that each
  object is able or permitted to contain at most one smaller (but possibly
  nested) object at a time, the resulting sequence begins 1,2,5,15,52 (Bell
  Numbers?). Sets of nested wooden boxes or traditional nested Russian dolls
  come to mind here. - Rick L. Shepherd, Jan 14 2004
From Michael Somos, Mar 04 2004; edited by M. F. Hasler, Jan 02 2015:
  (Start)
Stirling transform of [2, 2, 6, 24, 120, ...] is A052856 = [2, 2, 4, 14, 76,
```

(1) Fatorial de um número inteiro

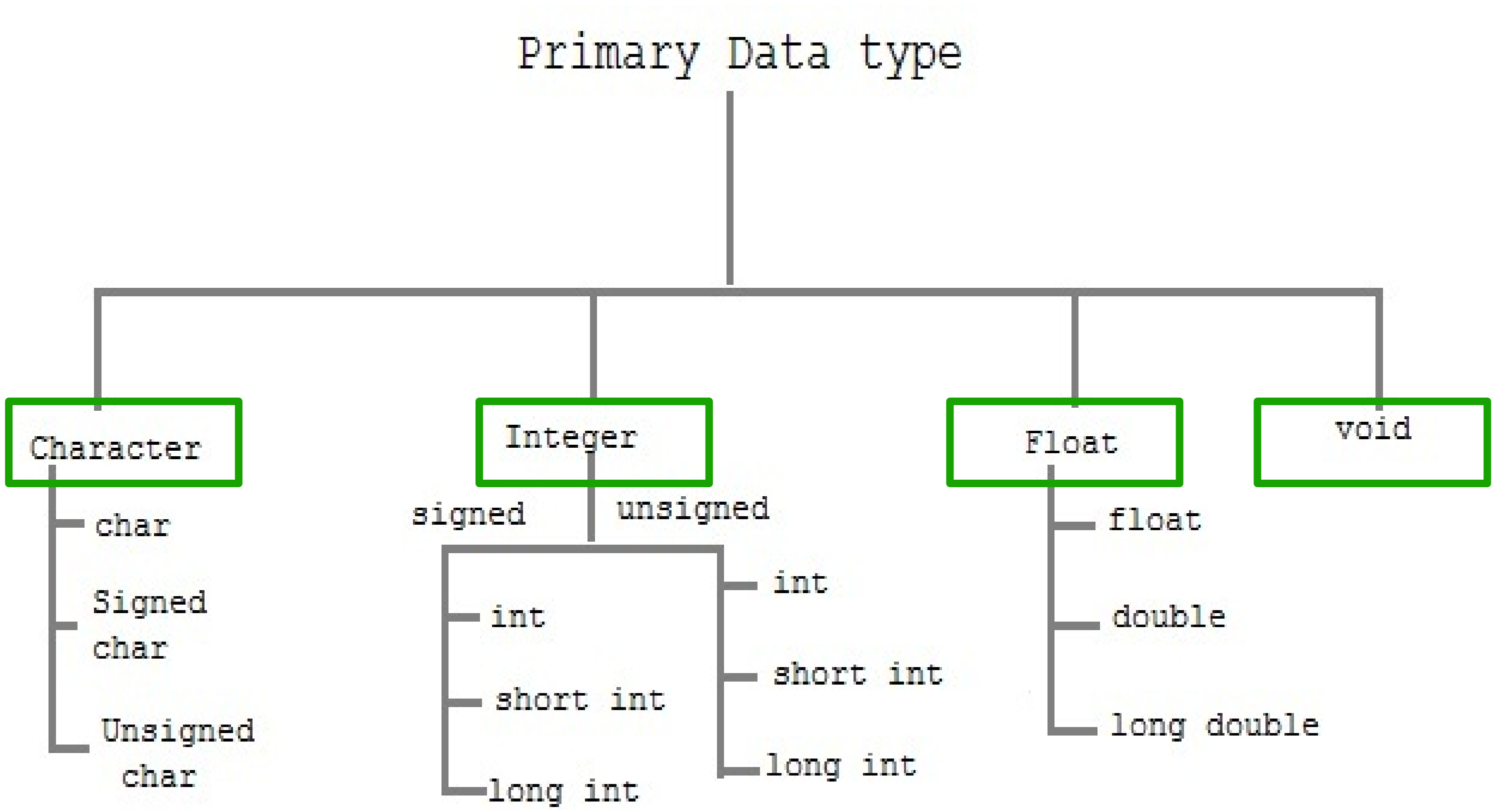
$$\text{fatorial}(n) = \begin{cases} 1 & \text{, se n=0} \\ n \times \text{fatorial}(n-1) & \text{, caso contrario} \end{cases}$$

Fatorial de um número

```c
#include <stdio.h>

int fatorial(int n) {
    if (n==0)
        return 1;
    else
        return fatorial(n-1)*n;
}

int main()
{
    int num;

    scanf("%d", &num);
    printf("%d\n", fatorial(num) );

    return 0;
}
```

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000

Teste para num=20
a resposta deve ser **2432902008176640000**



(*) Fonte: http://www.studytonight.com/c/datatype-in-c.php

Fatorial de um número

```c
#include <stdio.h>

long int fatorial(int n) {
    if (n==0)
        return 1;
    else
        return fatorial(n-1)*n;
}

int main()
{
    int num;

    scanf("%d", &num);
    printf("%ld\n", fatorial(num) );

    return 0;
}
```

Número de vezes em que a função **Fatorial** é chamada?

Fatorial de um número

```c
#include <stdio.h>

long int fatorial(int n) {
    if (n==0)
        return 1;
    else
        return fatorial(n-1)*n;
}

int main()
{
    int num;

    scanf("%d", &num);
    printf("%ld\n", fatorial(num) );

    return 0;
}
```

Número de vezes em que a função **Fatorial** é chamada? **n+1**

Fatorial de um número

```
$ gcc fatorial.c -o fatorial.exe

$ ./fatorial.exe
17
355687428096000

$ ./fatorial.exe
18
6402373705728000

$ ./fatorial.exe
19
121645100408832000

$ ./fatorial.exe
20
2432902008176640000
```

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000

Fatorial de um número

```
$ gcc -Wall -pg fatorial.c -o fatorial.exe

$ ./fatorial.exe

$ gprof fatorial.exe  > fatorial.txt
```

Wall ← Warnings all
pg ← para uso com o gprof
(gera um arquivo gmoun.out)

Um arquivo fatorial.txt é gerado.

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00        1     0.00     0.00  fatorial

```

```

granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                                    20             fatorial [1]
                0.00    0.00       1/1           main [7]
[1]      0.0    0.00    0.00       1+20      fatorial [1]
                                    20             fatorial [1]
```

n=20

Fatorial de um número

```
GPROF(1)                          GNU                          GPROF(1)

NAME
       gprof - display call graph profile data

SYNOPSIS
       gprof [ -[abcDhilLrsTvwxyz] ] [ -[ACeEfFJnNOpPqQZ][name] ]
        [ -I dirs ] [ -d[num] ] [ -k from/to ]
        [ -m min-count ] [ -R map_file ] [ -t table-length ]
        [ --[no-]annotated-source[=name] ]
        [ --[no-]exec-counts[=name] ]
        [ --[no-]flat-profile[=name] ] [ --[no-]graph[=name] ]
        [ --[no-]time=name] [ --all-lines ] [ --brief ]
        [ --debug[=level] ] [ --function-ordering ]
        [ --file-ordering map_file ] [ --directory-path=dirs ]
        [ --display-unused-functions ] [ --file-format=name ]
        [ --file-info ] [ --help ] [ --line ] [ --min-count=n ]
        [ --no-static ] [ --print-path ] [ --separate-files ]
        [ --static-call-graph ] [ --sum ] [ --table-length=len ]
        [ --traditional ] [ --version ] [ --width=n ]
        [ --ignore-non-functions ] [ --demangle[=STYLE] ]
        [ --no-demangle ] [--external-symbol-table=name]
        [ image-file ] [ profile-file ... ]

DESCRIPTION
       "gprof" produces an execution profile of C, Pascal, or Fortran77
       programs.  The effect of called routines is incorporated in the profile
       of each caller.  The profile data is taken from the call graph profile
       file (gmon.out default) which is created by programs that are compiled
       with the -pg option of "cc", "pc", and "f77".  The -pg option also
       links in versions of the library routines that are compiled for
       profiling.  "Gprof" reads the given object file (the default is
       "a.out") and establishes the relation between its symbol table and the
       call graph profile from gmon.out.  If more than one profile file is
```

(2) Números de Fibonacci

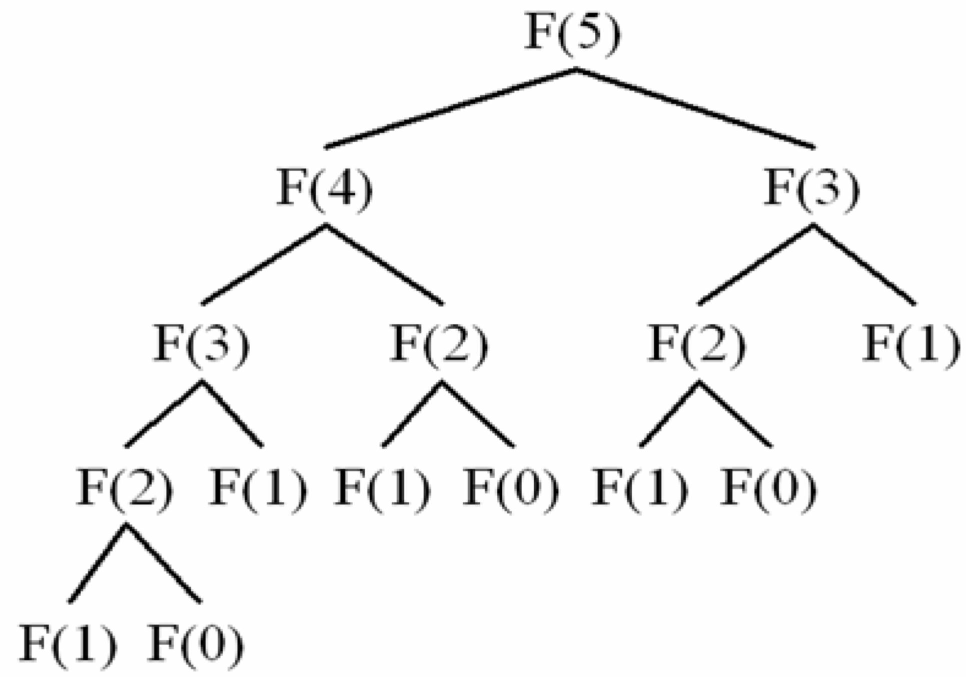
F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

$$Fib(n) = \begin{cases} 0 & , \text{ se } n = 0 \\ 1 & , \text{ se } n = 1 \\ Fib(n-1) + Fib(n-2) & , \text{ se } n > 1 \end{cases}$$

```c
#include <stdio.h>

long int Fib(int n) {
    if (n==0)
        return 0;
    if (n==1)
        return 1;
    else
        return Fib(n-1) + Fib(n-2);
}

int main() {
    int num;
    scanf("%d", &num);
    printf("%ld\n", Fib(num));
}
```

Números de Fibonacci

Fib (5)
    Fib (4)
        Fib (3)
            Fib (2)
                Fib (1)
                Fib (0)
            Fib (1)
        Fib (2)
            Fib (1)
            Fib (0)
    Fib (3)
        Fib (2)
            Fib (1)
            Fib (0)
        Fib (1)



Números de Fibonacci

Fib (6)

- Fib (5)
  - Fib (4)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
    - Fib (2)
      - Fib (1)
      - Fib (0)
  - Fib (3)
    - Fib (2)
      - Fib (1)
      - Fib (0)
    - Fib (1)
- Fib (4)
  - Fib (3)
    - Fib (2)
      - Fib (1)
      - Fib (0)
    - Fib (1)
  - Fib (2)
    - Fib (1)
    - Fib (0)

Fib (7)

- Fib (6)
  - Fib (5)
    - Fib (4)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
      - Fib (2)
        - Fib (1)
        - Fib (0)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
  - Fib (4)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
    - Fib (2)
      - Fib (1)
      - Fib (0)
- Fib (5)
  - Fib (4)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
    - Fib (2)
      - Fib (1)
      - Fib (0)
  - Fib (3)
    - Fib (2)
      - Fib (1)
      - Fib (0)
    - Fib (1)

Fib (8)

- Fib (7)
  - Fib (6)
    - Fib (5)
      - Fib (4)
        - Fib (3)
          - Fib (2)
            - Fib (1)
            - Fib (0)
          - Fib (1)
        - Fib (2)
          - Fib (1)
          - Fib (0)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
    - Fib (4)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
      - Fib (2)
        - Fib (1)
        - Fib (0)
  - Fib (5)
    - Fib (4)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
      - Fib (2)
        - Fib (1)
        - Fib (0)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
- Fib (6)
  - Fib (5)
    - Fib (4)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
      - Fib (2)
        - Fib (1)
        - Fib (0)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
  - Fib (4)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
    - Fib (2)
      - Fib (1)
      - Fib (0)

Fib (9)

- Fib (8)
  - Fib (7)
    - Fib (6)
      - Fib (5)
        - Fib (4)
          - Fib (3)
            - Fib (2)
              - Fib (1)
              - Fib (0)
            - Fib (1)
          - Fib (2)
            - Fib (1)
            - Fib (0)
        - Fib (3)
          - Fib (2)
            - Fib (1)
            - Fib (0)
          - Fib (1)
      - Fib (4)
        - Fib (3)
          - Fib (2)
            - Fib (1)
            - Fib (0)
          - Fib (1)
        - Fib (2)
          - Fib (1)
          - Fib (0)
    - Fib (5)
      - Fib (4)
        - Fib (3)
          - Fib (2)
            - Fib (1)
            - Fib (0)
          - Fib (1)
        - Fib (2)
          - Fib (1)
          - Fib (0)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
  - Fib (6)
    - Fib (5)
      - Fib (4)
        - Fib (3)
          - Fib (2)
            - Fib (1)
            - Fib (0)
          - Fib (1)
        - Fib (2)
          - Fib (1)
          - Fib (0)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
    - Fib (4)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
      - Fib (2)
        - Fib (1)
        - Fib (0)
- Fib (7)
  - Fib (6)
    - Fib (5)
      - Fib (4)
        - Fib (3)
          - Fib (2)
            - Fib (1)
            - Fib (0)
          - Fib (1)
        - Fib (2)
          - Fib (1)
          - Fib (0)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
    - Fib (4)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
      - Fib (2)
        - Fib (1)
        - Fib (0)
  - Fib (5)
    - Fib (4)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
      - Fib (2)
        - Fib (1)
        - Fib (0)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)

Números de Fibonacci

```
$ gcc -Wall -pg fibonacci.c -o fibonacci.exe

$ ./fibonacci.exe

$ gprof fibonacci.exe > fibonacci.txt
```

```
index % time    self  children    called     name
                                   14             Fib [1]
                0.00    0.00       1/1            main [7]
[1]      0.0    0.00    0.00       1+14       Fib [1]
                                   14             Fib [1]
```

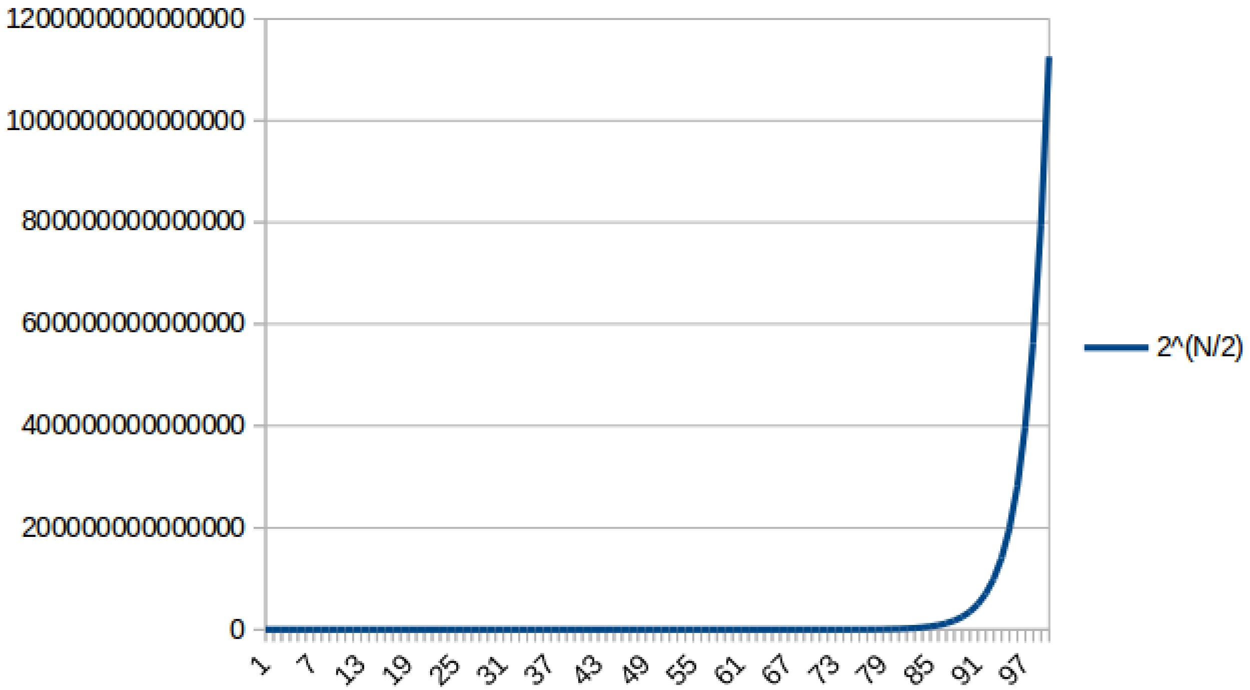
n=5

```
index % time    self  children    called     name
                                21890             Fib [1]
                0.00    0.00       1/1            main [7]
[1]      0.0    0.00    0.00       1+21890    Fib [1]
                                21890             Fib [1]
```

n=20

Números de Fibonacci

n	2^(N/2)
1	1.41
2	2.00
3	2.83
4	4.00
5	5.66
6	8.00
7	11.31
8	16.00
9	22.63
10	32.00
11	45.25
12	64.00
13	90.51
14	128.00
15	181.02
16	256.00
17	362.04
18	512.00
19	724.08
20	1024.00
21	1448.15
22	2048.00
23	2896.31
24	4096.00
25	5792.62
26	8192.00
27	11585.24
28	16384.00
29	23170.48
30	32768.00



Usando uma variável global para contar o número de chamados à função

```c
#include <stdio.h>

int count=0;

long int Fib(int n) {
    count++;
    if (n==0 || n==1)
        return n;
    else
        return Fib(n-1) + Fib(n-2);
}

int main() {
    int num;
    scanf("%d", &num);
    printf("%ld\n", Fib(num));
    printf("%d\n", count);
    return 0;
}
```

fibonacciContador.c

Versão com memória

```c
#include <stdio.h>

int count=0;
long int vetorF[100]={0};

long int Fib(int n) {
    count++;

    if (n==0 || n==1)
        return n;
    if (vetorF[n]==0)
        vetorF[n] = Fib(n-1) + Fib(n-2);

    return vetorF[n];
}

int main() {
    int num;
    scanf("%d", &num);
    printf("%ld\n", Fib(num));
    printf("%d\n", count);
    return 0;
}
```

0	0	0	0	0	0	0	...

0	0	1	2	3	5	8	...

FibonacciComMemoria.c

(3) Palindromo

Vetor palindromo (Iterativo)

```c
#include <stdio.h>

int ehPalindromo(int V[], int N) {
    int i;
    for (i=0; i<N/2; i++) {
        printf("compara %d\n", V[i]);
        if ( V[i] != V[N-i-1] )
            return 0;
    }
    return 1;
}



int main() {
    int vetor[] = {1,2,3,4,999,4,3,2,1};
    int n = sizeof(vetor)/sizeof(vetor[0]);

    printf("Resposta: %d\n", ehPalindromo(vetor, n));
}
```

Crie sua versão recursiva

Vetor palindromo (Recursivo)

```c
#include <stdio.h>

int ehPalindromo(int V[], int N, int i) {
    if (i==N/2)
        return 1;
    if (V[i]!=V[N-i-1])
        return 0;
    else
        return ehPalindromo(V, N, i+1);
}

int main() {
    int vetor[] = {1,2,3,4,999,4,3,2,1};
    int n = sizeof(vetor)/sizeof(vetor[0]);

    printf("Resposta: %d\n", ehPalindromo(vetor, n, 0));
}
```

(4) Primorial

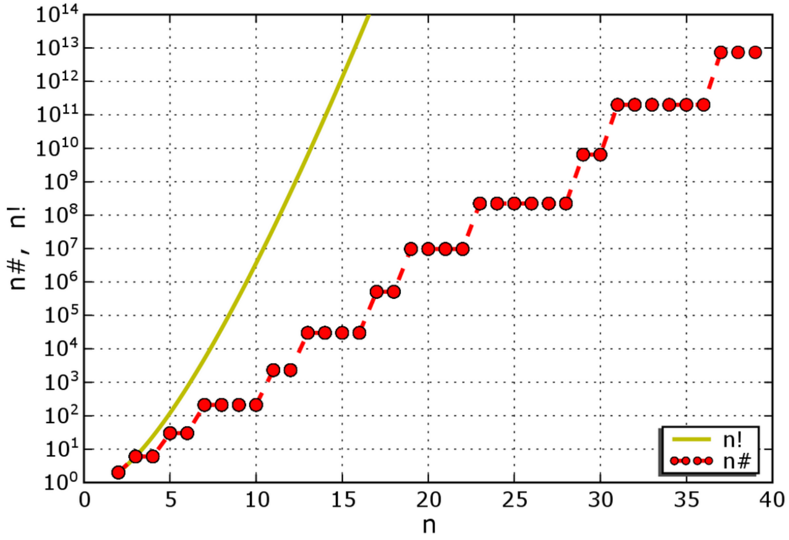
Primorial

O primorial de um número inteiro positivo **n** é o produto de todos os primos menores ou iguais a **n**.

- É denotado por **n#**

$$1\# = 1$$
$$2\# = 2$$
$$3\# = 2 \cdot 3 = 6$$
$$4\# = 2 \cdot 3 = 6$$
$$5\# = 2 \cdot 3 \cdot 5 = 30$$
$$6\# = 2 \cdot 3 \cdot 5 = 30$$
$$7\# = 2 \cdot 3 \cdot 5 \cdot 7 = 210$$

https://en.wikipedia.org/wiki/Primorial



- Crie uma função **recursiva** que, dado um número inteiro positivo, devolva o seu Primorial.

ehPrimo

```c
#include <stdio.h>

//Funcao valida para n inteiro e positivo
int ehPrimo(int n) {
    int i;

    for(i=2; i<n; i++)
        if (n%i==0)
            return 0;

    return 1;
}

int main() {
    int num;

    scanf("%d", &num);

    if ( ehPrimo(num) )
        printf("O numero %d eh primo\n", num);
    else
        printf("O numero %d nao eh primo\n", num);

    return 0;
}
```