الگوریتم های بازگشتی
Recursive Algorithms

مطالب مورد بحث

- مقدمه
- بررسی چند مساله نمونه
  - محاسبه فاکتوریل عدد N
  - معمای برج هانوی (Tower of Hanoi)
  - محاسبه بزرگترین مقسوم علیه مشترک به روش اقلیدسی
  - دنباله اعداد فیبوناتچی
  - فرکتال ها

مقدمه

- یکی از تکنیک های حل مساله، استفاده از الگوریتم های بازگشتی می باشد.
- الگوریتم هایی که با استفاده از توابع بازگشتی طراحی و پیاده سازی می شوند.
  - تابع بازگشتی مستقیم: در داخل بدنه تابع، مجددا همان تابع، با پارامترهای متفاوتی فراخوانی می شود (فراخوانی صریح).
  - تابع بازگشتی غیر مستقیم (با واسطه): در داخل بدنه تابع، تابع دیگری فراخوانی می شود که در داخل آن، مجددا تابع اولیه با پارامترهای متفاوتی فراخوانی میشود. ممکن است تعداد توابع میانی بیش از یکی باشد. یعنی یک زنجیره فراخوانی داشته باشیم که نهایتا به فراخوانی مجدد تابع اولیه منجر شوند.

مقدمه

- هر تابع بازگشتی شامل دو قسمت است که در هر بار فراخوانی تابع، تنها یکی از این دو قسمت اجرا می شود.
  - قسمت پایه: این حالت منجر به فراخوانی مجدد تابع نمی شود.
  - قسمت بازگشتی: این حالت منجر به فراخوانی بازگشتی تابع می شود.
- دنباله فراخوانی های بازگشتی یک تابع نهایتا باید به قسمت پایه برسد، در غیر اینصورت ← حلقه پایان ناپذیری از فراخوانی های بازگشتی

مقدمه

- ویژگی مسائلی که با این روش قابل حل هستند
  - بازگشت پذیری
    - پاسخ یک نمونه از مساله را بتوان با استفاده از پاسخ نمونه های کوچکتر همان مساله بدست آورد.
  - ذات بازگشتی
    - روابط ریاضی و محاسباتی

مقدمه

- مزایای الگوریتم های بازگشتی
  - سازگاری با طبیعت مساله: برخی مسائل، ذاتا در تعریف خود، از مفهوم بازگشتی استفاده می کنند که توصیف آنها را ساده تر می کند.
    - بسیاری از مفاهیم، توابع و فرمول های ریاضی دارای طبیعت بازگشتی هستند.
  - خوانایی زیاد
  - سادگی
  - کوتاهی کد الگوریتم از نظر تعداد دستورات
- عیب الگوریتم های بازگشتی
  - استفاده از توابع بازگشتی منجر به فراخوانی های مکرر توابع می شود ← از نظر زمان اجرا و میزان مصرف حافظه، سربار قابل توجهی ایجاد می شود.

مقدمه

- اگر برای مساله ای بتوان یک الگوریتم بازگشتی ارائه نمود، آنگاه برای آن مساله می توان یک الگوریتم غیر بازگشتی (Iterative) نیز ارائه نمود.
  - Jacobini و Bohm در ۱۹۶۶ نشان دادند که هر برنامه ای را می توان به برنامه ای تبدیل کرد که در آن تنها از ساختار دنباله (sequence)، انتخاب (مثلا if) و تکرار (مثلا while) استفاده شده باشد.
  - یعنی می توان از توابع استفاده نکرد.
    - دلایل استفاده از توابع: امکان استفاده مجدد از کد، افزایش قابلیت نگهداری، افزایش سرعت یافتگی برنامه، ...
- البته ممکن است کیفیت الگوریتم غیر بازگشتی مذکور، خیلی بد باشد (مثلا الگوریتم خیلی پیچیده و طولانی باشد)
- آیا عکس این مطلب نیز درست است ؟؟

بررسی چند مساله نمونه

- **بررسی چند مساله نمونه**
  - **محاسبه فاکتوریل عدد N**
  - **معمای برج هانوی (Tower of Hanoi)**
  - **محاسبه بزرگترین مقسوم علیه مشترک به روش اقلیدسی**
  - **دنباله اعداد فیبوناتچی**
  - **فرکتال ها**

محاسبه فاکتوریل عدد n

- تعریف بازگشتی

$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & n > 0 \end{cases}$$

- الگوریتم بازگشتی

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

قسمت پایه

قسمت بازگشتی

محاسبه فاکتوریل عدد n :: محاسبه مرتبه زمانی

- ورودی: عدد n که می خواهیم فاکتوریل آن را حساب کنیم.
- اندازه ورودی: مقدار عدد n
- عمل کلیدی: عمل ضرب
- محاسبه تابع پیچیدگی (T(n) موجود می باشد)
- $T(n) = 1 + T(n-1)$
- یک رابطه بازگشتی است.
- $T(n) = T(n-1)+1 = T(n-2)+1+1 = T(n-3)+1+1+1 = \dots$

$\rightarrow T(n) = T(0) + n$

$T(0) = 0$

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

مرتبه زمانی خطی

محاسبه فاکتوریل عدد n به روش غیر بازگشتی

- تعریف غیر بازگشتی

$$n! = \begin{cases} 1 & n = 0 \\ 1*2*\ldots(n-1)*n & n > 0 \end{cases}$$

- الگوریتم غیر بازگشتی

```
long int fact(int n) {
    long int result = 1;
    for(int i=2; i<=n ;i++)
        result *= i;
    return result;
}
```

معادل قسمت پایه الگوریتم بازگشتی

معادل قسمت بازگشتی الگوریتم بازگشتی

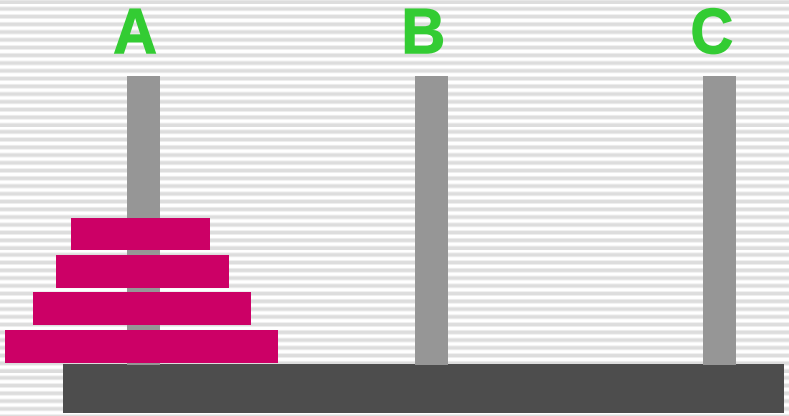
محاسبه محاسبه مرتبه زمانی الگوریتم غیر بازگشتی

- ورودی: عدد n که می خواهیم فاکتوریل آن را حساب کنیم.
- اندازه ورودی: مقدار عدد n
- عمل کلیدی: عمل ضرب
- محاسبه تابع پیچیدگی (T(n) موجود می باشد)
- مرتبه زمانی خطی $T(n) = n-1 \rightarrow T(n) = O(n)$
- هر دو الگوریتم بازگشتی و غیر بازگشتی، مرتبه زمانی خطی دارند. کدامیک بهتر به نظر می رسد؟

بررسی چند مساله نمونه

- بررسی چند مساله نمونه
  - محاسبه فاکتوریل عدد N
  - معمای برج هانوی (Tower of Hanoi)
  - محاسبه بزرگترین مقسوم علیه مشترک به روش اقلیدسی
  - دنباله اعداد فیبوناتچی
  - فرکتال ها

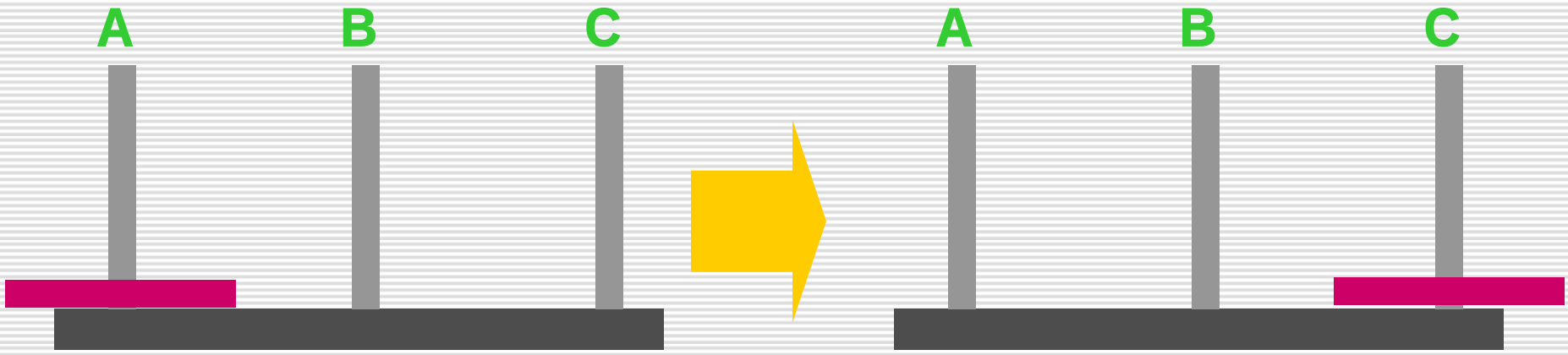
معمای برج هانوی (Tower of Hanoi)



- سیستمی داریم شامل ۳ میله و n دیسک که اندازه (شعاع) آنها با هم متفاوت است. تمام n دیسک بر روی یک میله و به ترتیب بزرگ به کوچک (از پایین به بالا) روی هم قرار گرفته اند.
- هدف: کلیه دیسک ها را از میله A به میله C منتقل کنیم بطوریکه باز هم به ترتیب بزرگ به کوچک (از پایین به بالا) روی هم قرار بگیرند، ضمنا در انجام این کار باید دو قانون را رعایت کرد:
  - هر بار تنها یک دیسک را می توان از میله ای به میله دیگر منتقل کرد.
  - هرگز نمی توان دیسکی را بر روی دیسک کوچکتر از خودش قرار داد.

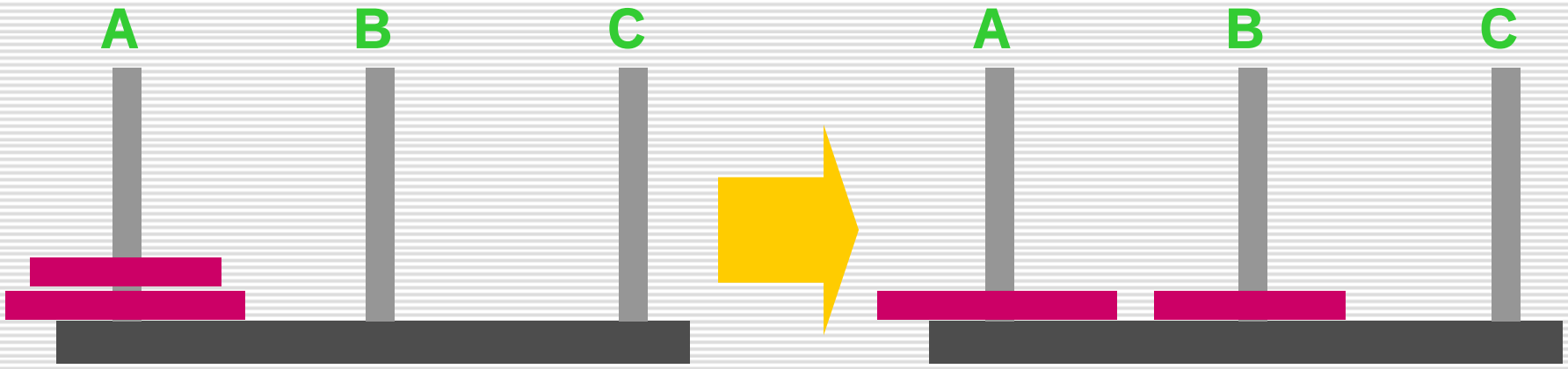
معمای برج هانوی

- راه حل برای حالت n=1
- بدیهی: دیسک بالایی را از A به C منتقل کن.



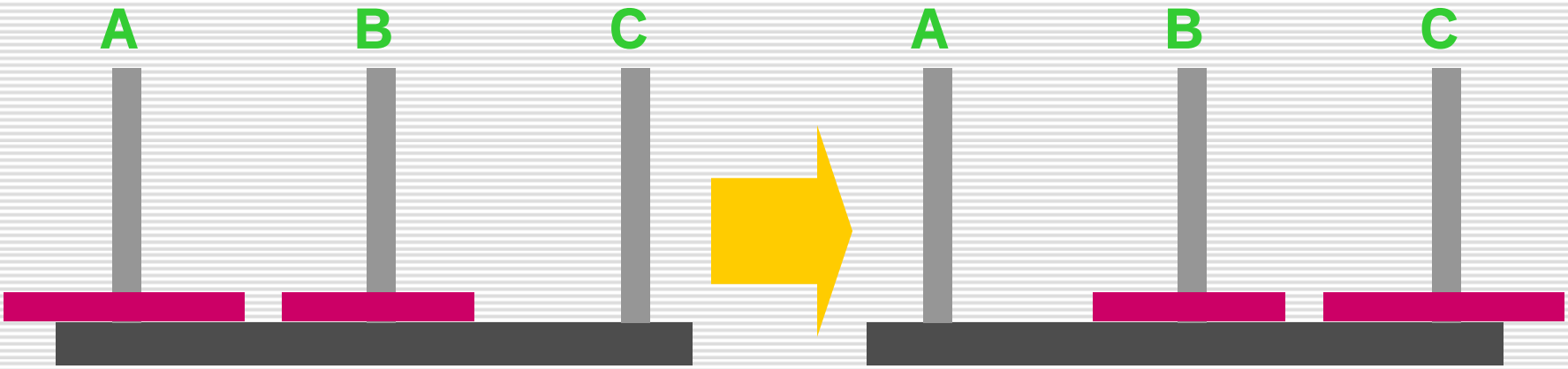
معمای برج هانوی

- راه حل برای حالت n=2
- گام اول: دیسک بالایی را از A به B منتقل کن.



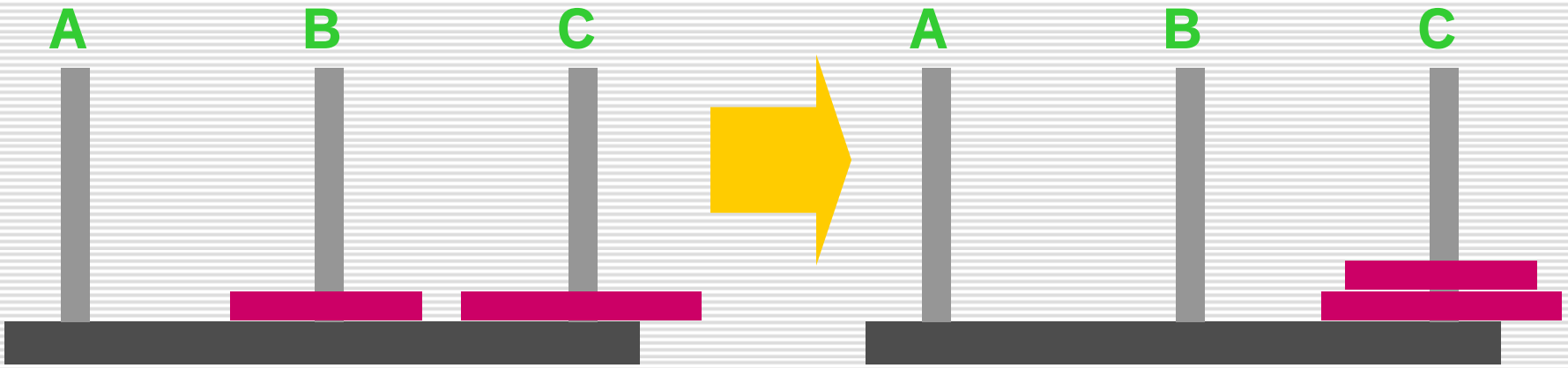
معمای برج هانوی

- گام دوم: دیسک بزرگ را از A به C منتقل کن.



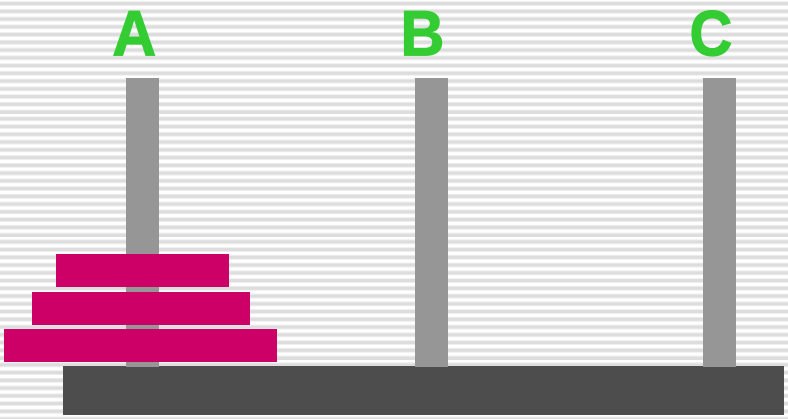
معمای برج هانوی

- گام سوم: دیسک کوچک را از B به C منتقل کن.
- پایان. (میله B بعنوان میله کمکی و محل موقت استفاده شد)



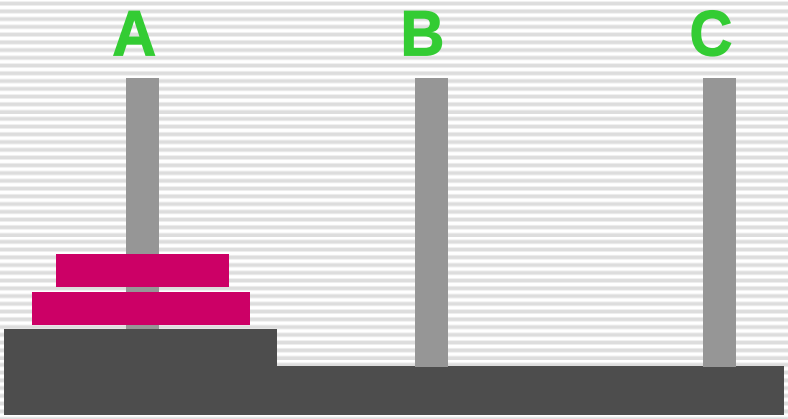
معمای برج هانوی

- راه حل برای حالت n=3 با استفاده از حالت n=2



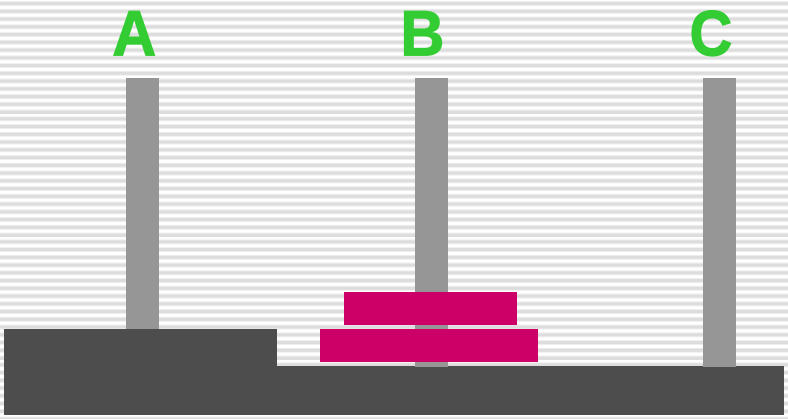
معمای برج هانوی

- بزرگترین دیسک را نادیده بگیر (تا مساله شبیه حالت قبل شود).



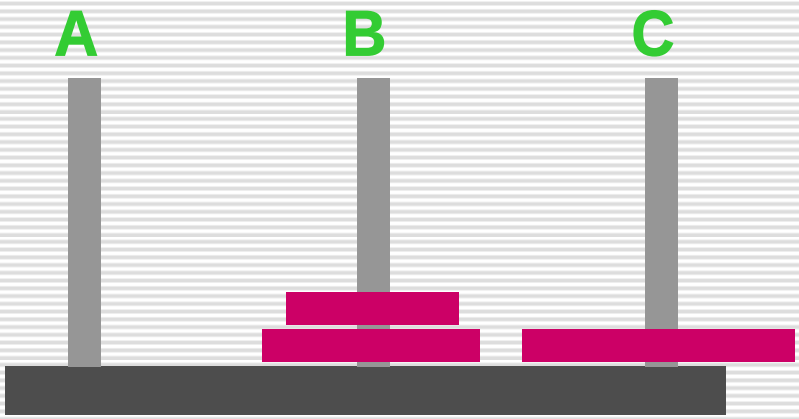
معمای برج هانوی

- دو دیسک بالایی را به روشی که قبلا برای n=2 گفته شد، از A به B منتقل کن (بکمک میله C)



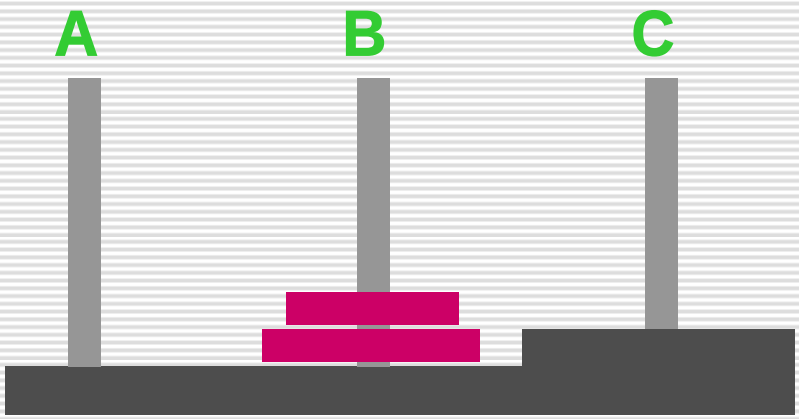
معمای برج هانوی

- بزرگترین دیسک را از A به C منتقل کن.



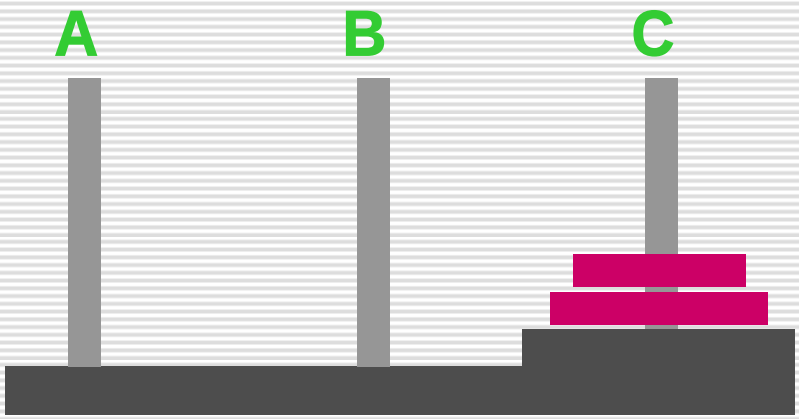
معمای برج هانوی

- بزرگترین دیسک را نادیده بگیر.



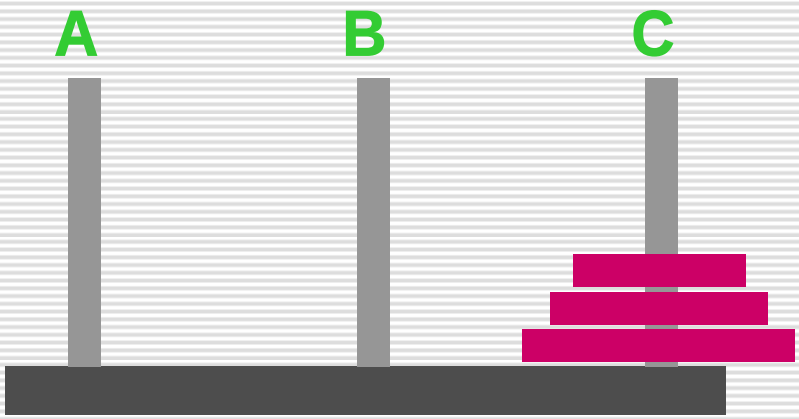
معمای برج هانوی

- دو دیسک دیگر را به روشی که قبلا برای n=2 گفته شد، از B به C منتقل کن (بکمک میله A)



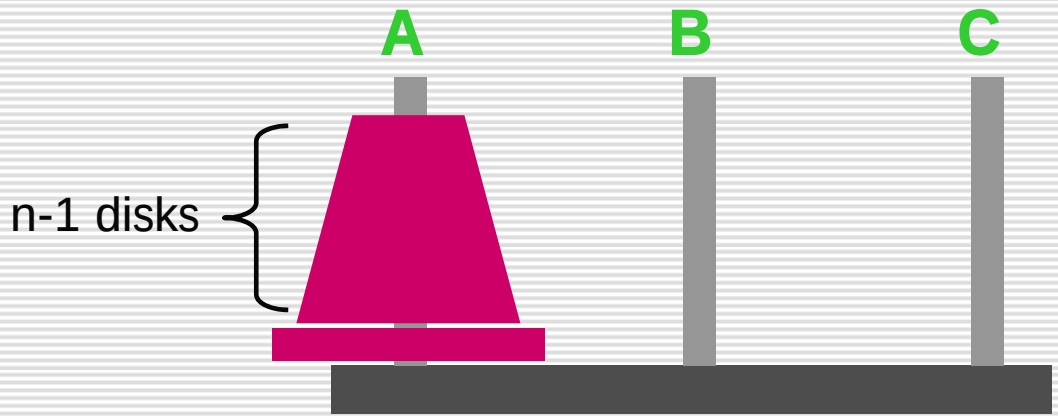
معمای برج هانوی

- پایان.



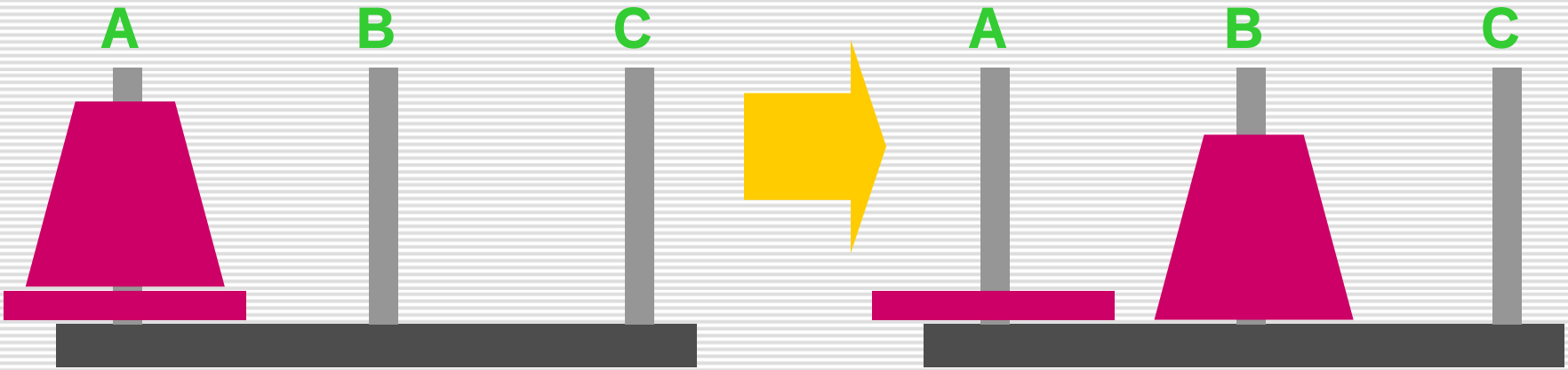
معمای برج هانوی

- بطور کلی برای حل مساله برای n دیسک از حل مساله برای n-1 دیسک ، استفاده می کنیم.
  - اگر n==1، تنها دیسک موجود را از A به C منتقل کن و کار تمام است.
  - در غیر اینصورت:
    - طی سه گام مساله را حل کن.



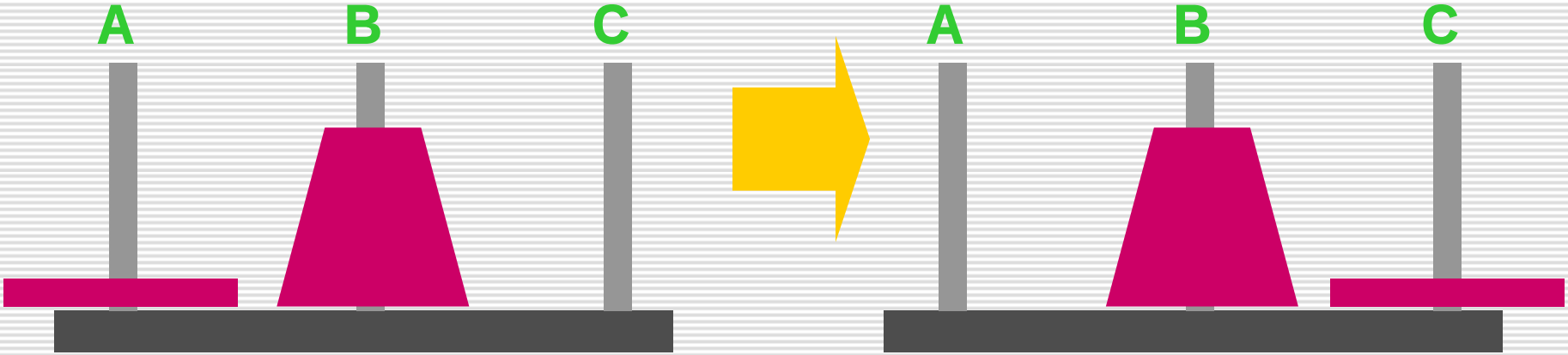
معمای برج هانوی

- گام اول: n-1 دیسک بالایی را بکمک میله C به میله B منتقل کن.



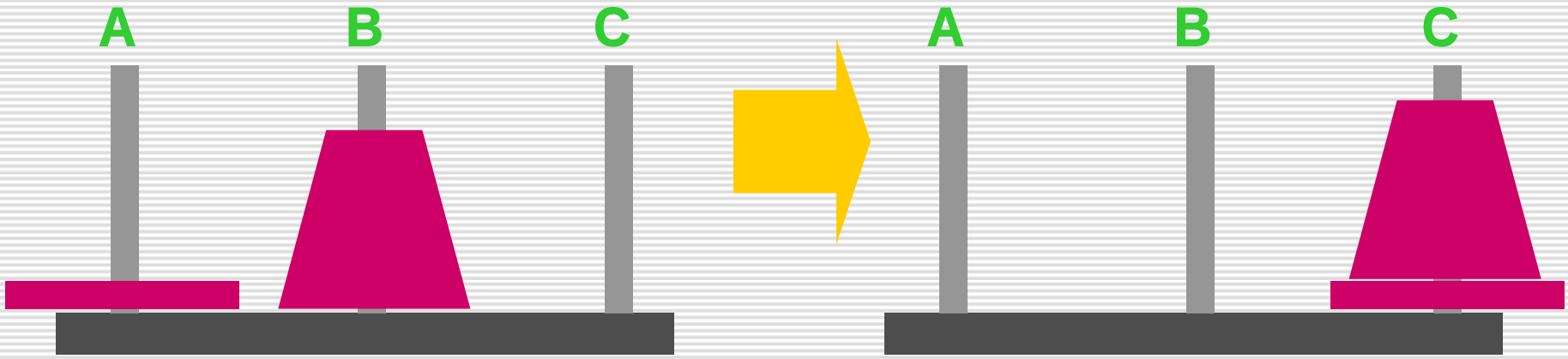
معمای برج هانوی

- گام دوم: بزرگترین دیسک را از A به C منتقل کن.



معمای برج هانوی

- گام سوم: n-1 دیسک دیگر را بکمک میله A به میله C منتقل کن.
- پایان.



معمای برج هانوی

- درستی الگوریتم پیشنهادی را می توان با استقرا بر روی n، اثبات نمود.

معمای برج هانوی

- الگوریتم

```
void hanoi (int n, pole A, pole B, pole C) {
    if(n==1) move top disk from A to C;
    else {
        hanoi (n-1, A, C, B);
        move top disk from A to C;
        hanoi (n-1, B, A, C);
    }
}
```

معمای برج هانوی :: محاسبه مرتبه زمانی

- ورودی: تعداد دیسکها ، یعنی n
- اندازه ورودی: مقدار n
- عمل کلیدی: عمل move (جابجایی یک دیسک)
- محاسبه تابع پیچیدگی (T(n)) موجود است.

$$T(n) = \begin{cases} 1 & n = 1 \\ 2\,T(n-1) + 1 & n > 1 \end{cases}$$

معمای برج هانوی :: محاسبه مرتبه زمانی

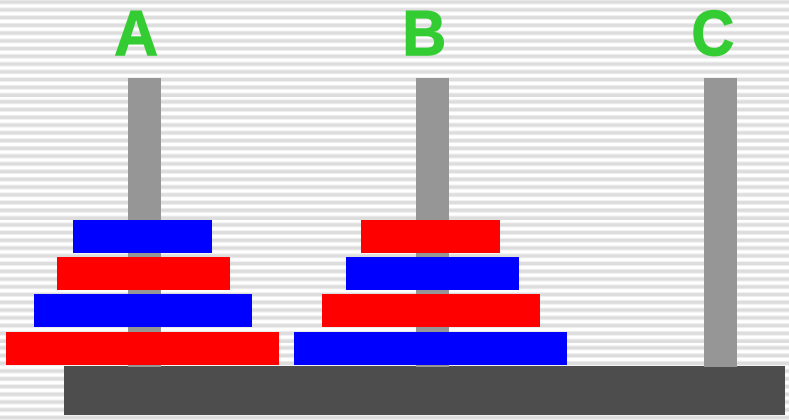
- $T(n) = 2T(n-1)+1 = 2\,(2T(n-2)+1) + 1 = 2^2T(n-2) + 3$
  $= 2^3T(n-3) + 7 = 2^4T(n-4)+15 = \ldots = 2^k\,T(n-k) + 2^K - 1$
- $T(1) = 1$
- $n-k=1 \rightarrow k=n-1$
- $T(n) = 2^{n-1}T(1) + 2^{n-1} - 1$

$$T(n) = 2^n - 1$$
$$\rightarrow T(n)=O(2^n)$$

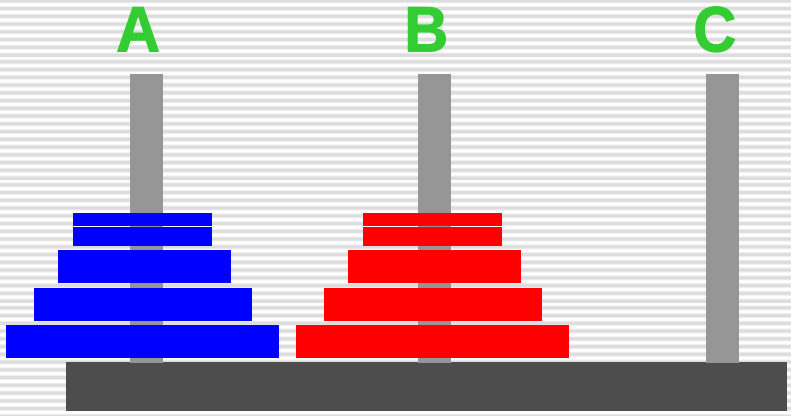
- مرتبه زمانی الگوریتم ارائه شده، نمایی (exponential) می باشد.

معمای برج هانوی ۲ رنگی

- حالت اولیه

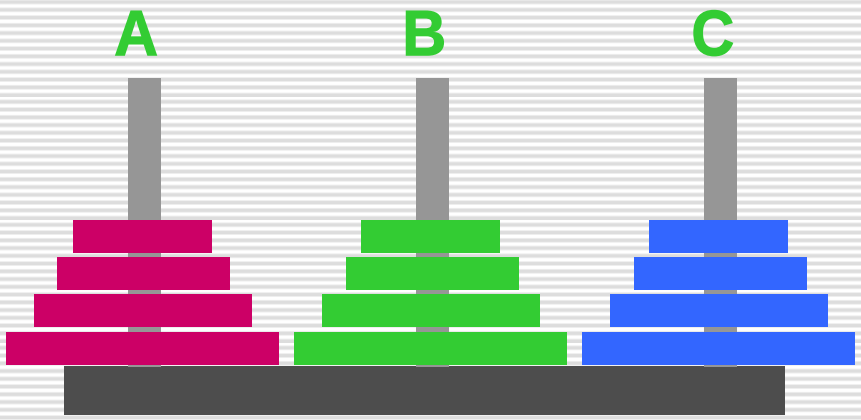


- حالت هدف

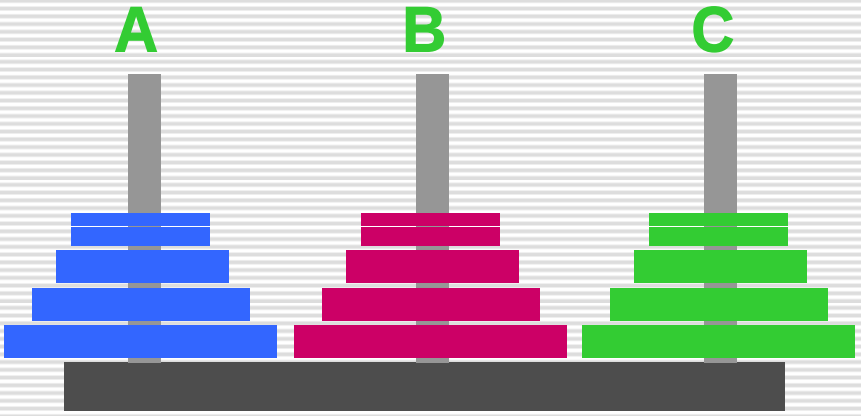


معمای برج هانوی ۳ رنگی

- حالت اولیه



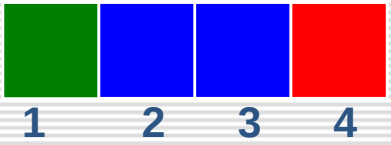
- حالت هدف



معمای مربع های رنگی

- n مربع داریم که رنگ هر یک از آنها یکی از مقادیر قرمز و سبز و آبی است. می خواهیم با تغییر دادن رنگ مربع ها، همه آنها را همرنگ نماییم. در این کار باید قواعد زیر را رعایت نماییم.
  - الف) اگر می خواهیم رنگ مربعی را به رنگ X تغییر دهیم، در سمت چپ آن مربع، نباید مربعی با رنگ X وجود داشته باشد. (منظور از سمت چپ کل مربع های سمت چپ می باشد و نه فقط مربع مجاور سمت چپ)
  - ب) اگر می خواهیم رنگ مربعی را تغییر دهیم و رنگ آن مربع در حال حاضر X می باشد، نباید در سمت چپ آن مربع، مربعی با رنگ X وجود داشته باشد. (منظور از سمت چپ کل مربع های سمت چپ می باشد و نه فقط مربع مجاور سمت چپ)

معمای مربع های رنگی



- مثال: یک نمونه از مساله فوق: n=4 و وضعیت اولیه مربع ها
  - در این حالت نمی توانیم مربع ۲ را سبز کنیم، چون سمت چپ آن مربعی با رنگ سبز وجود دارد (مربع ۱) و قاعده الف محدودیت ایجاد می کند.
  - همچنین در حالت فوق نمی توانیم مربع ۳ را قرمز کنیم چون مربع ۳ آبی است و در سمت چپ آن یک مربع آبی دیگر وجود دارد (مربع ۲) و قاعده ب محدودیت ایجاد می کند.

بررسی چند مساله نمونه

- بررسی چند مساله نمونه
  - محاسبه فاکتوریل عدد N
  - معمای برج هانوی (Tower of Hanoi)
  - محاسبه بزرگترین مقسوم علیه مشترک به روش اقلیدسی
  - دنباله اعداد فیبوناتچی
  - فرکتال ها

محاسبه ب.م.م. به روش اقلیدسی

- محاسبه ب.م.م. دو عدد صحیح غیرمنفی a و b با فرض a>=b
- تعریف بازگشتی

$$\gcd(a,b) = \begin{cases} a & b = 0 \\ \gcd(b, a \bmod b) & \text{else} \end{cases}$$

- gcd: greatest common divisor
- مثال: ب.م.م. دو عدد ۴۰ و ۱۸
- gcd(40, 18) = gcd(18, 4) = gcd(4, 2) = gcd(2, 0) = 2

محاسبه ب.م.م. به روش اقلیدسی

- الگوریتم بازگشتی

```
int gcd(int a, int b) {
    if (b == 0) return a;
    else return gcd(b, a % b);
}
```

محاسبه ب.م.م. به روش اقلیدسی

- نکته ۱: این الگوریتم بطور ضمنی، دنباله ای از اعداد تولید می کند که نهایتا به صفر می رسند.
- gcd(40, 18) ➔ 40, 18, 4, 2, 0
- $a_0, a_1, a_2, \ldots, a_m$
- $a_0 = a$, $a_1 = b$, $a_2 = a \bmod b$, … $a_m=0$
- $a_j = a_{j-2} \% a_{j-1}$ $j= 2,..,m$

- نکته ۲: برای اعداد طبیعی a و b:

$(a \bmod b) <= (a-1)/2$

محاسبه ب.م.م. به روش اقلیدسی:: محاسبه مرتبه زمانی

- ورودی: اعداد a و b
- اندازه ورودی: مقدار a ، یعنی n=a
- عمل کلیدی: عمل تعیین باقیمانده (mod)

محاسبه ب.م.م. به روش اقلیدسی:: محاسبه مرتبه زمانی

- محاسبه تابع پیچیدگی
  - تعداد اجرای عمل کلیدی، علاوه بر مقدار a به مقدار b نیز بستگی دارد.
    - gcd(40,18) = gcd(18,4)=gcd(4,2)=gcd(2,0)=2
    - gcd(40,10) = gcd(10,0) = 10
  - در نتیجه به محاسبه W(n) می پردازیم.
  - می توان گفت حجم کار انجام شده توسط الگوریتم به a%b بستگی دارد، اما از آنجا که ما بدنبال تعیین حد بالا هستیم، و از آنجا که a%b<=(a-1)/2، بجای a%b توجه خود را بر a متمرکز می کنیم.

محاسبه ب.م.م. به روش اقلیدسی:: محاسبه مرتبه زمانی

- $W(n) = 1 + W(n / 2)$
- $W(n) = W(n/2)+1 = W(n/4)+1+1 = W(n/8)+1+1+1=...$
- $W(n) = W(n/2^k)+k$
- $W(1) = 1$
- $n/2^k=1 \rightarrow k=\log^n_2$

$\rightarrow W(n) = 1+\log^n_2$

$\rightarrow W(n)=O(\log^n_2)$

مرتبه زمانی بدترین حالت، لگاریتمی می باشد.

بررسی چند مساله نمونه

- بررسی چند مساله نمونه
  - محاسبه فاکتوریل عدد N
  - معمای برج هانوی (Tower of Hanoi)
  - محاسبه بزرگترین مقسوم علیه مشترک به روش اقلیدسی
  - دنباله اعداد فیبوناتچی
  - فرکتال ها

محاسبه ب.م.م. به روش اقلیدسی

- تعریف بازگشتی

$$f\,ib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2) & \text{else} \end{cases}$$

Fibonacci sequence :

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$

- الگوریتم بازگشتی

```
long int fib(int n) {
    if ( n <= 1) return n;
    else return fib(n-1)+fib(n-2);
}
```

دنباله اعداد فیبوناتچی

- محاسبه پیچیدگی زمانی
  - ورودی: عدد n که می خواهیم مقدار fib(n) را حساب کنیم.
  - اندازه ورودی: مقدار عدد n
  - عمل کلیدی: عمل جمع
  - تابع پیچیدگی:

$$T(n) = 1 + T(n-1) + T(n-2)$$

دنباله اعداد فیبوناتچی

$T(n) = 1 + T(n-1) + T(n-2)$

$<= 1+T(n-1)+T(n-1) <= 2T(n-1) +1$

$<= 2(2T(n-2)+1) +1 <= \dots$

$T(n) <= 2^kT(n-k) + 2^k-1$

$n-k = 0 \rightarrow k = n$

$T(0) = 0$

$\rightarrow T(n) <= 2^n$

$\rightarrow T(n) = O(2^n)$

مرتبه زمانی نمایی

دنباله اعداد فیبوناتچی

- الگوریتم غیر بازگشتی

```
long int fib(int n) {
    if ( n <= 1) return n;
    long int F1 = 0,    F2 = 1, result ;
    for(int i=1;i<n;i++) {
            result = F1 + F2;
            F1 = F2;
            F2 = result;
    }
    return result;
}
```

دنباله اعداد فیبوناتچی

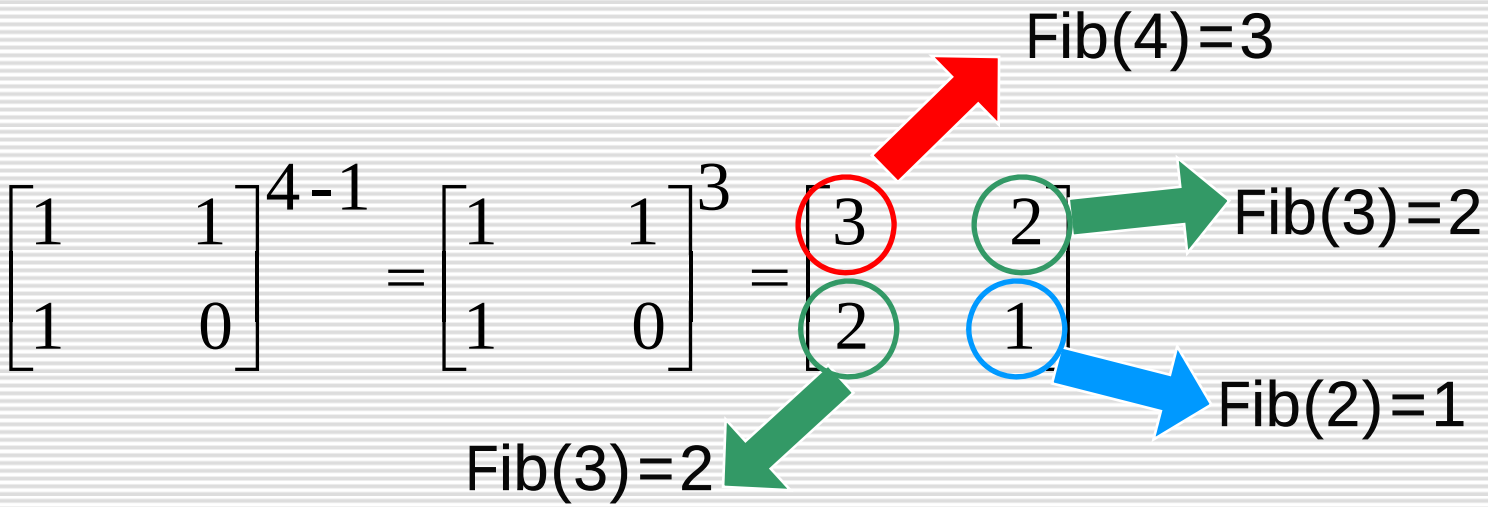
- محاسبه مرتبه زمانی الگوریتم غیر بازگشتی
- ورودی: عدد n
- اندازه ورودی: مقدار عدد n
- عمل کلیدی: عمل جمع (result = F1+F2)
- تابع پیچیدگی:
- $T(n) = n-1$
- مرتبه زمانی:
- $T(n) = n-1 \rightarrow T(n) = O(n) \rightarrow$ مرتبه زمانی خطی

دنباله اعداد فیبوناتچی

- یک الگوریتم بازگشتی دیگر با استفاده از رابطه بازگشتی زیر

$$\begin{bmatrix} Fib(n) & Fib(n-1) \\ Fib(n-1) & Fib(n-2) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1}$$

- مثال:



دنباله اعداد فیبوناتچی

- برای محاسبه Fib(n) باید یک ماتریس ۲ در ۲ را به توان n-1 برسانیم. حاصل هر مرحله، باز هم یک ماتریس ۲ در ۲ است. بنابراین حجم کار در هر مرحله یکسان است. از طرفی، الگوریتمی برای توان رسانی با مرتبه زمانی لگاریتمی موجود است، در نتیجه مرتبه زمانی این الگوریتم نیز لگاریتمی خواهد بود.

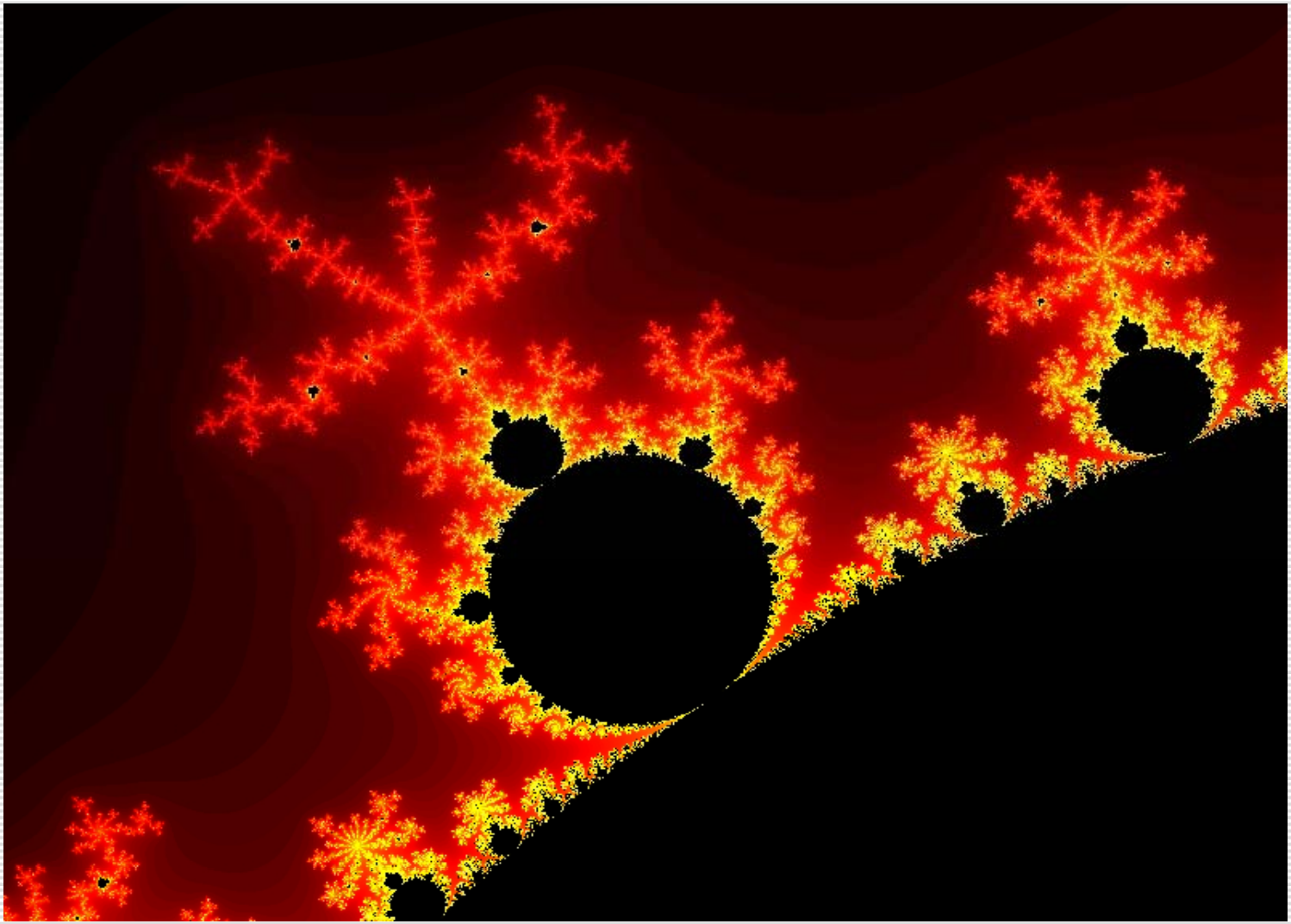
بررسی چند مساله نمونه

- بررسی چند مساله نمونه
  - محاسبه فاکتوریل عدد N
  - معمای برج هانوی (Tower of Hanoi)
  - محاسبه بزرگترین مقسوم علیه مشترک به روش اقلیدسی
  - دنباله اعداد فیبوناتچی
  - فرکتال ها

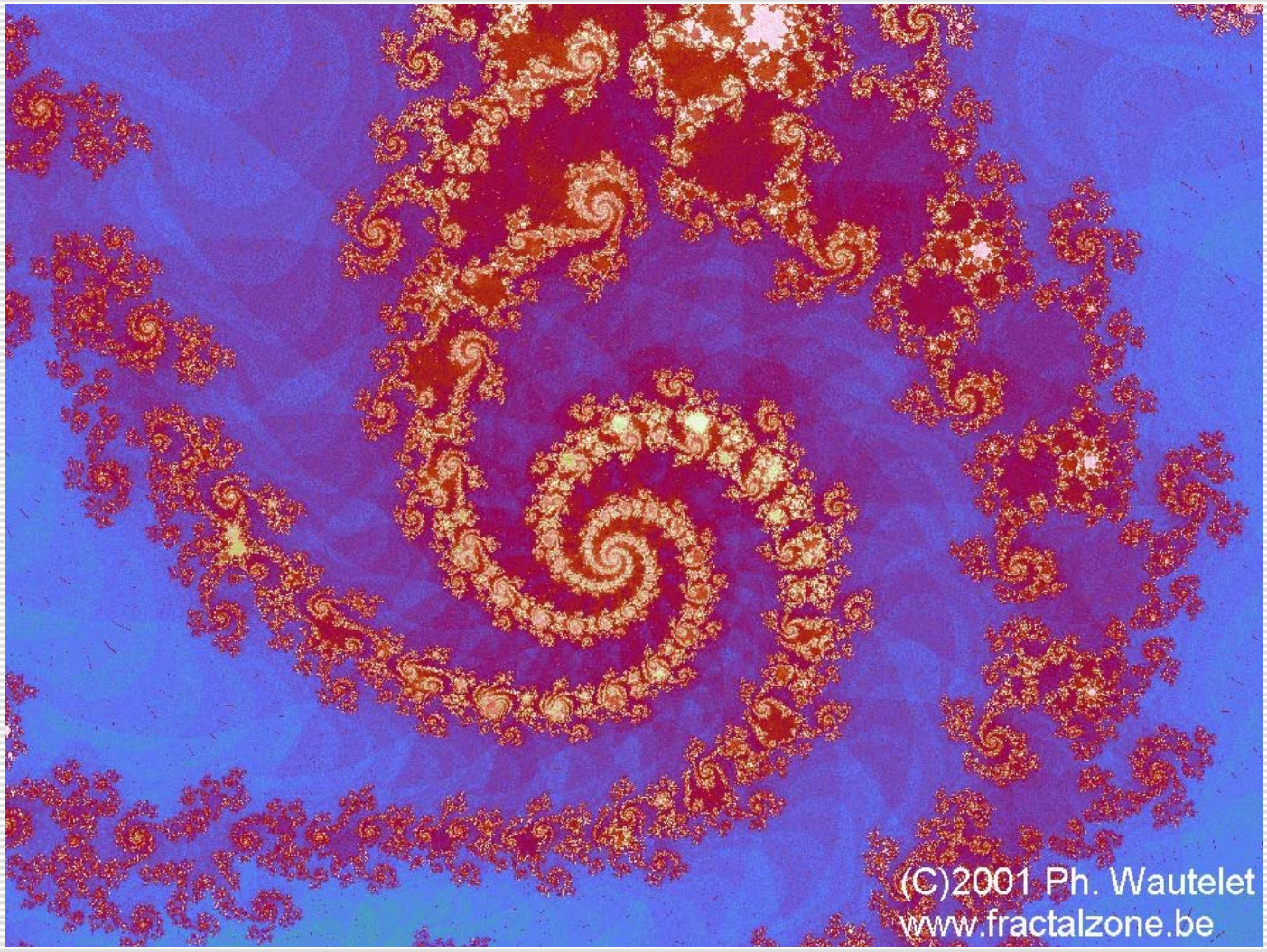
فرکتال ها

- یک شکل هندسی که می توان آن را به چند قسمت تقسیم کرد که هر یک از آن قسمت ها، از نظر ظاهری یا از نظر ساختار، یک کپی کوچکتر از شکل اولیه می باشد.
- مبتنی بر روشها و فرمولهای بازگشتی می باشند.
- چند نمونه از کاربردها در اینجا ذکر شده است.

فرکتال ها



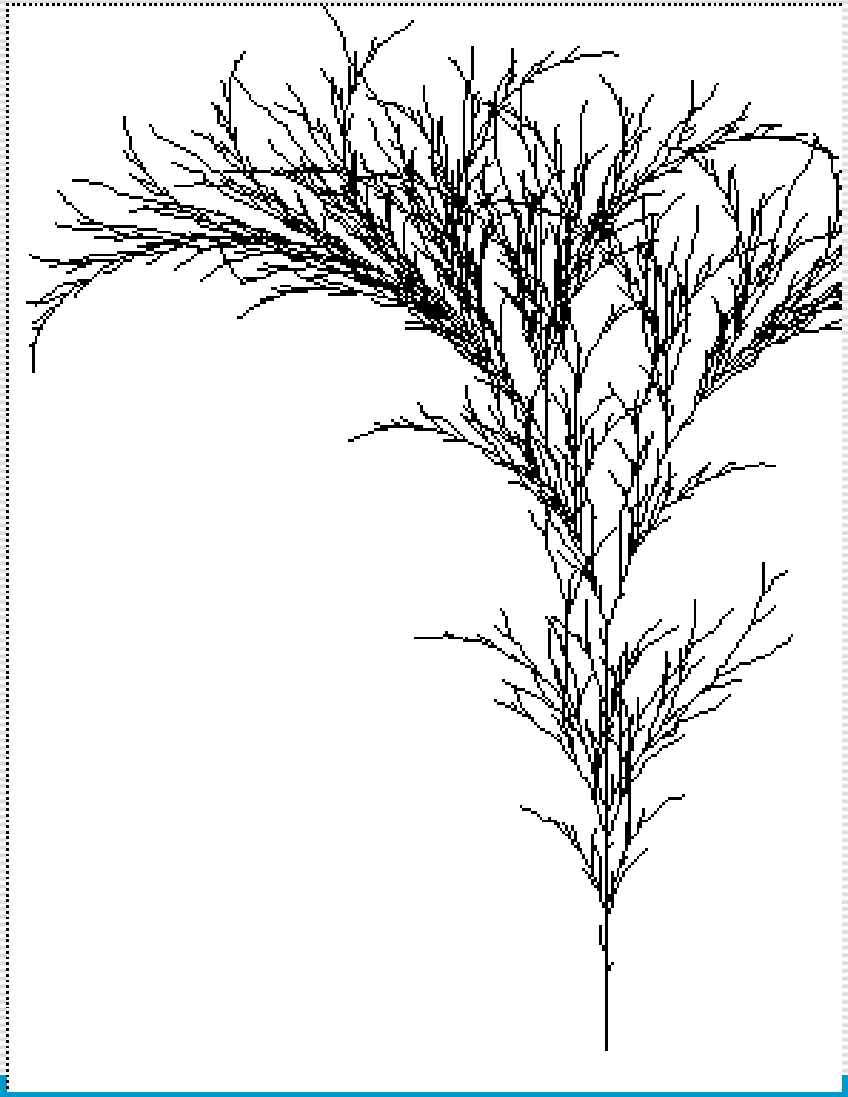
فرکتال ها



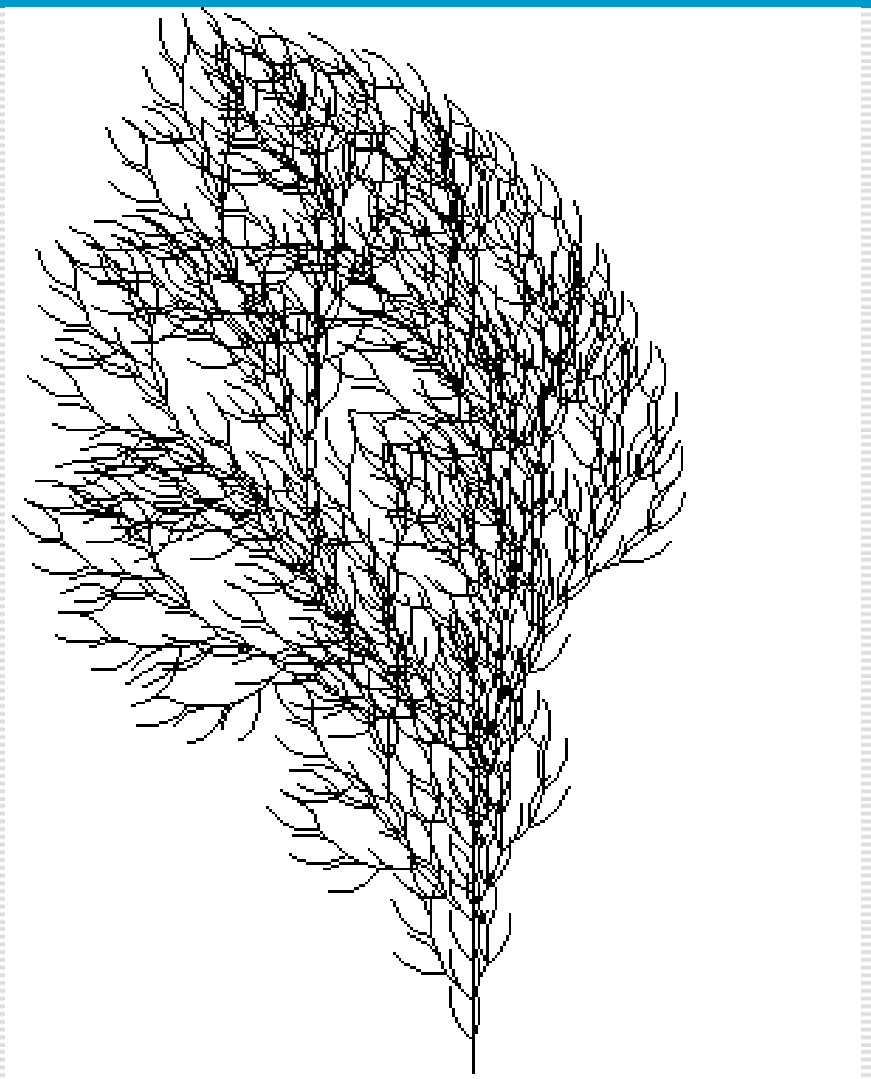
فرکتال ها

فرکتال ها

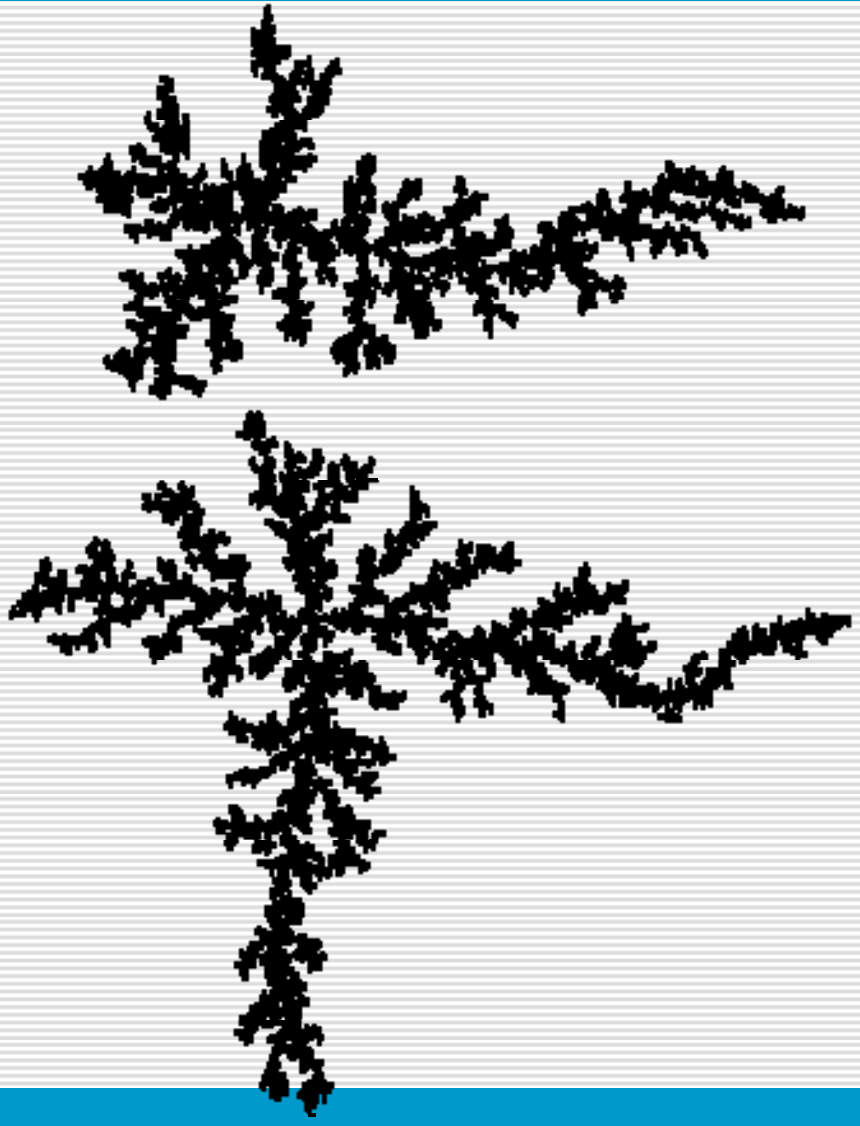
فرکتال ها



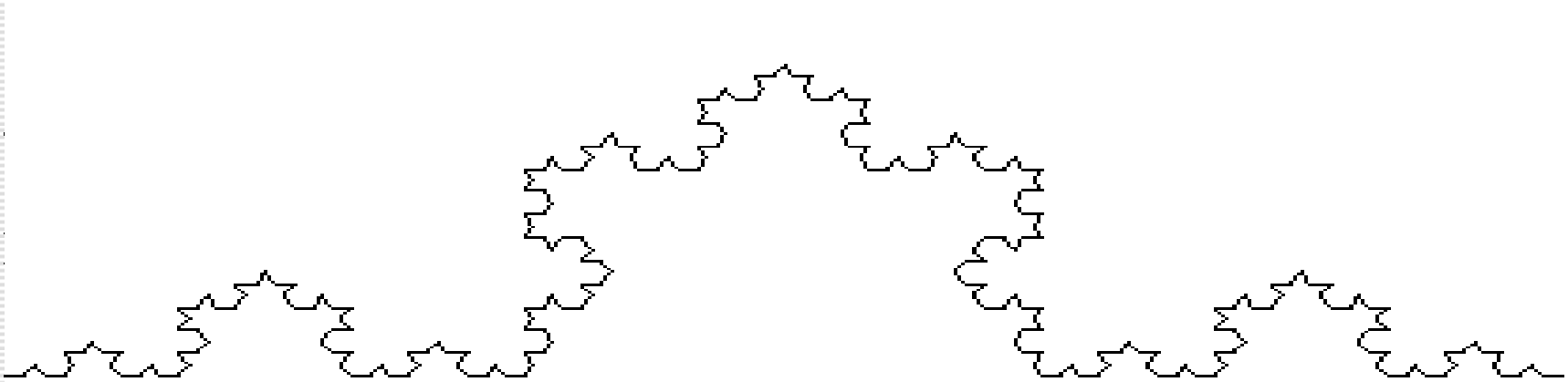
فرکتال ها



فرکتال ها

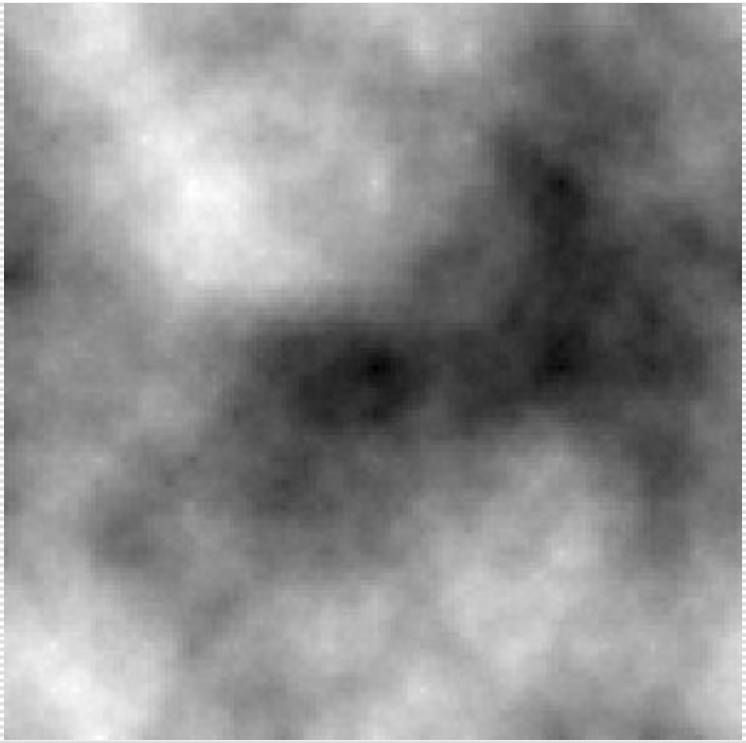


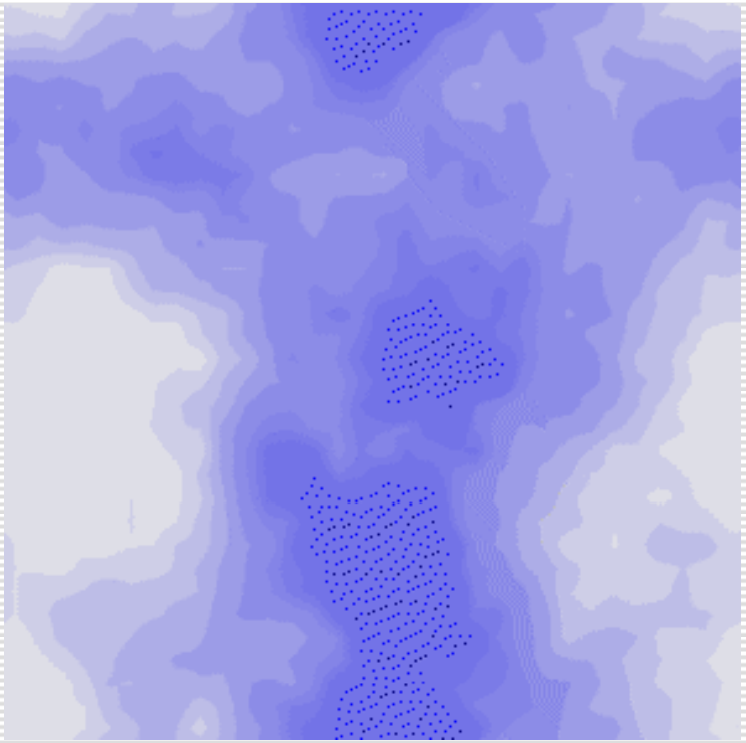
فرکتال ها



فرکتال ها

فرکتال ها





فرکتال ها