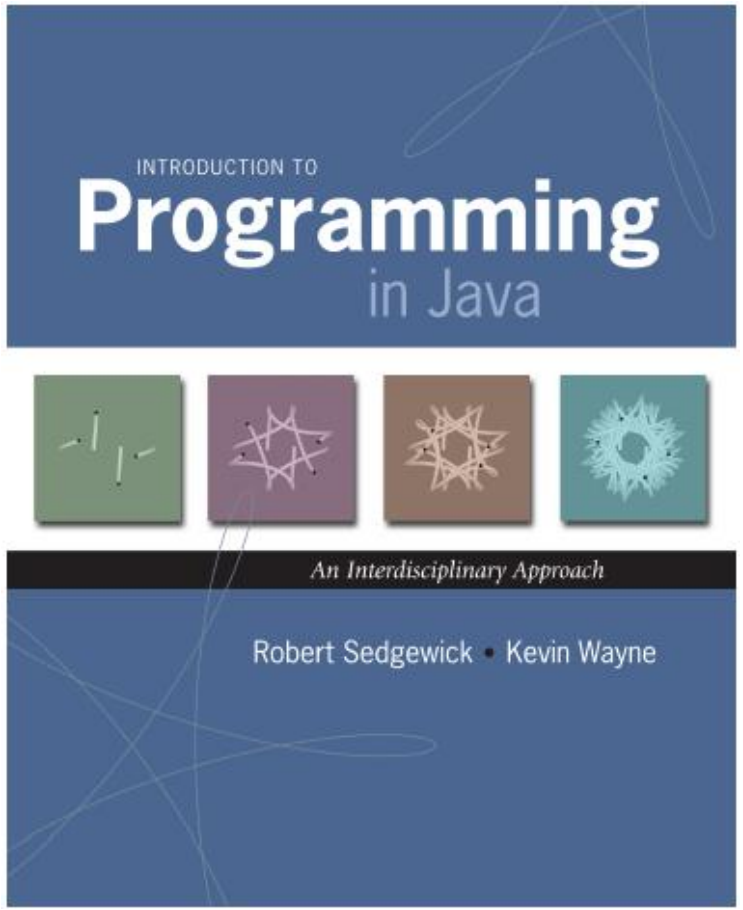
توابع و کتابخانه‌ها: توابع بازگشتی

سید ناصر رضوی **www.snrazavi.ir**
۱۳۹۶

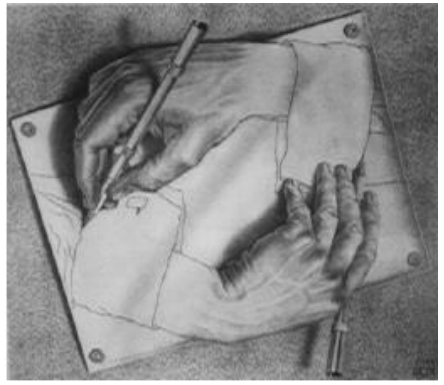
۳-۲ توابع بازگشتی



توابع بازگشتی

- تابع بازگشتی. تابعی که خودش را به صورت مستقیم یا غیرمستقیم فراخوانی می‌کند.

- مزایای یادگیری توابع بازگشتی.
  - آشنایی با یک سبک جدید تفکر (تفکر بازگشتی)
  - آشنایی با یک الگوی قدرتمند برنامه‌نویسی



- رابطه نزدیک با استقرای ریاضی.

فاکتوریل

- محاسبه فاکتوریل n.

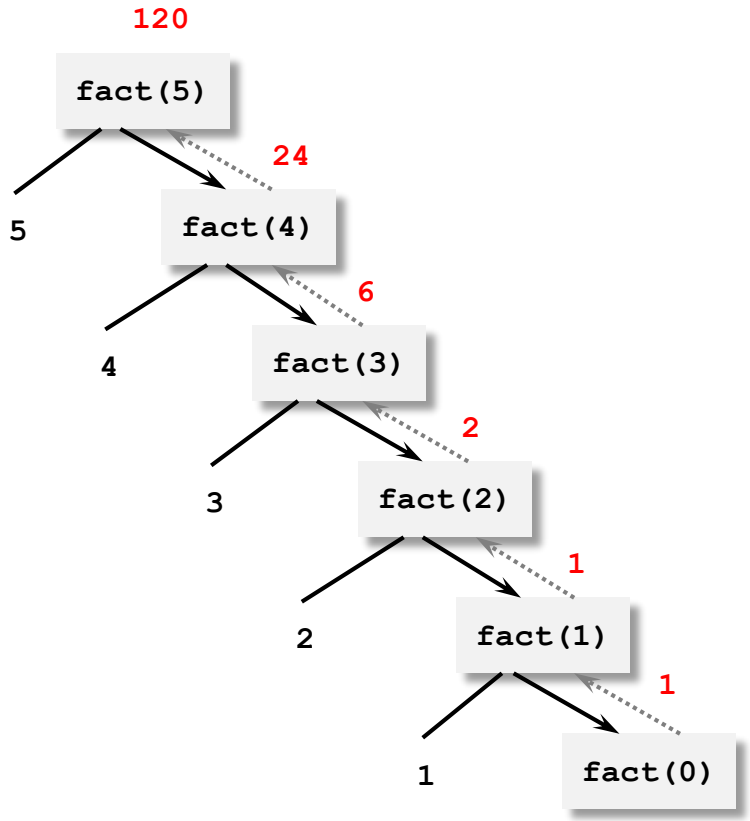
```
N! = 1 * 2 * ... * (N - 1) * N
```

$$N! = \begin{cases} 1 & N = 0 \\ N \times (N-1)! & N \geq 1 \end{cases}$$

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

ختم بازگشتی

فراخوانی بازگشتی



بزرگ‌ترین مقسوم علیه مشترک

- ب.م.م. یافتن بزرگ‌ترین عدد صحیحی که p و q بر آن بخش‌پذیرند.

- مثال.

gcd(4032, 1272) = 24.

$4032 = 2^6 * 3^2 * 7^1$

$1272 = 2^3 * 3^1 * 53^1$

$gcd = 2^3 * 3^1$

- کاربردها.
  - ساده‌سازی اعداد کسری
  - رمزنگاری RSA

بزرگ‌ترین مقسوم علیه مشترک

- الگوریتم اقلیدس. [۳۰۰ سال قبل از میلاد]

$$\gcd(p,q) = \begin{cases} p & q = 0 \\ \gcd(q, p\%q) & \text{otherwise} \end{cases}$$

```
gcd(4032, 1272)     = gcd(1272, 216)
                    = gcd(216, 192)
                    = gcd(192, 24)
                    = gcd(24, 0)
                    = 24
```

```
4032 = 3 * 1272 + 216
```

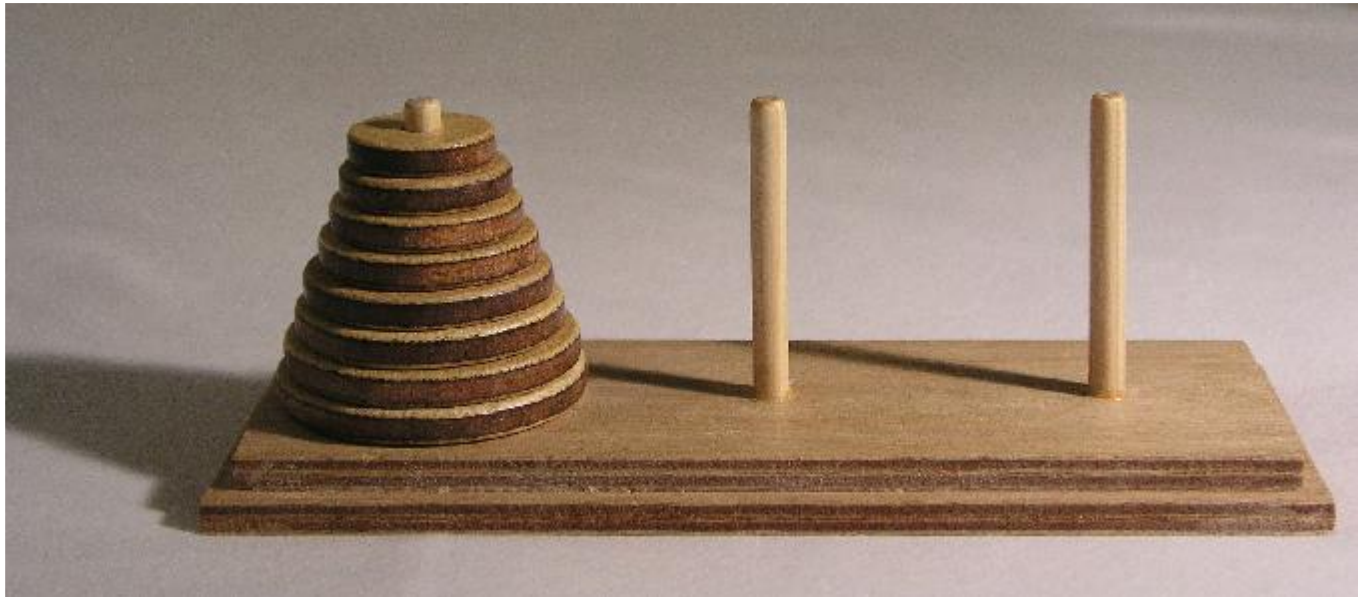
بزرگ‌ترین مقسوم علیه مشترک

- الگوریتم اقلیدس. [۳۰۰ سال قبل از میلاد]

$$\gcd(p,q) = \begin{cases} p & q = 0 \\ \gcd(q, p\%q) & \text{otherwise} \end{cases}$$

```
public static int gcd(int p, int q)
{
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

برج‌های هانوی

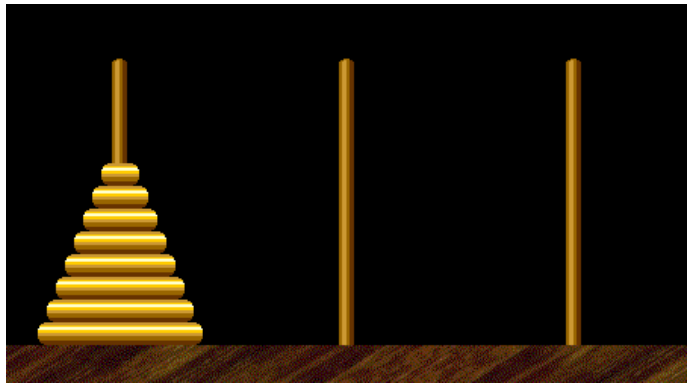


http://en.wikipedia.org/wiki/Image:Hanoiklein.jpg

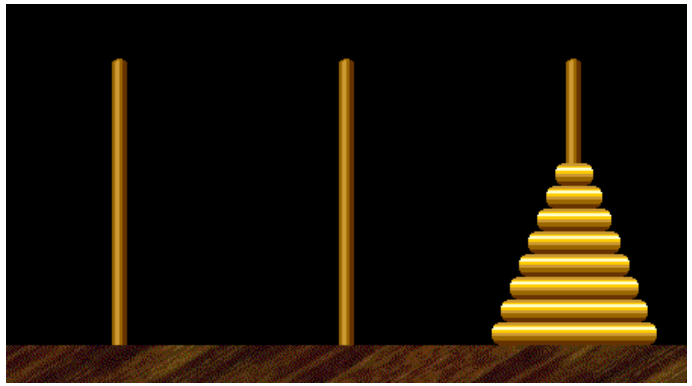
برج‌های هانوی

- هدف. انتقال دیسک‌ها از میله سمت چپ به میله سمت راست با حفظ ترتیب.
  - در هر حرکت تنها مجاز به انتقال یک دیسک هستیم.
  - هیچ‌گاه مجاز نیستیم یک دیسک بزرگ‌تر را بر روی یک دیسک کوچک‌تر قرار دهیم.

شروع



پایان

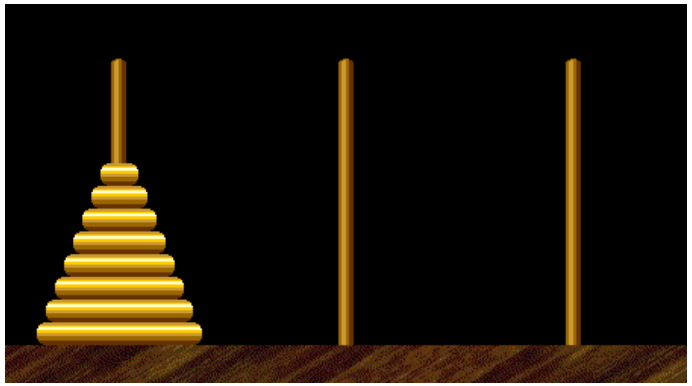


افسانه برج‌های هانوی

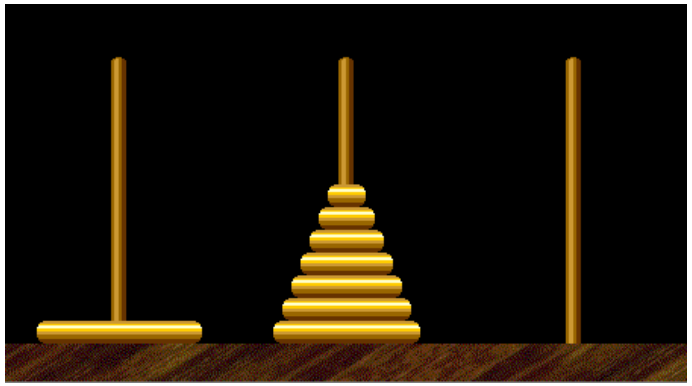
- س. دنیا چه زمانی به پایان می‌رسد؟
  - ۶۴ دیسک طلا بر روی سه میله الماس در ابتدای خلقت.
  - با اتمام کار یک گروه خاص از کشیش‌ها که مسؤول انتقال این دیسک‌ها هستند، دنیا به پایان می‌رسد.

- س. آیا می‌توان از کامپیوتر برای پیش‌بینی زمان پایان دنیا استفاده کرد؟

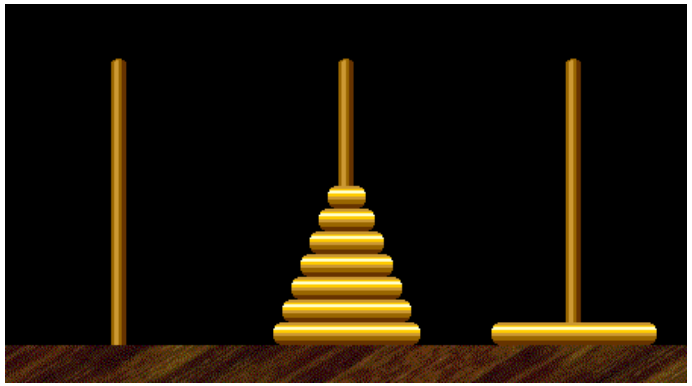
برج‌های هانوی: راه‌حل بازگشتی



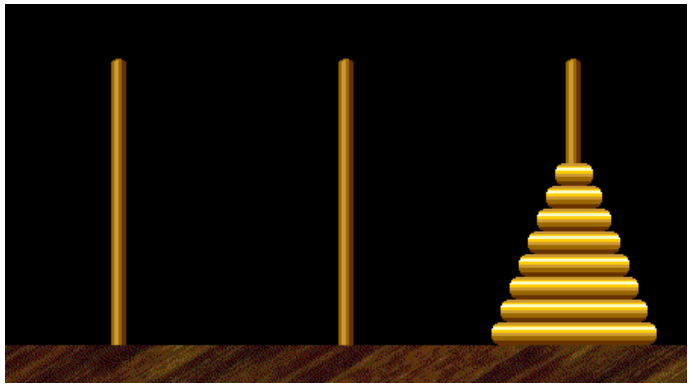
(۱) انتقال 1 – N دیسک به میله وسط



(۲) انتقال بزرگترین دیسک به میله راست



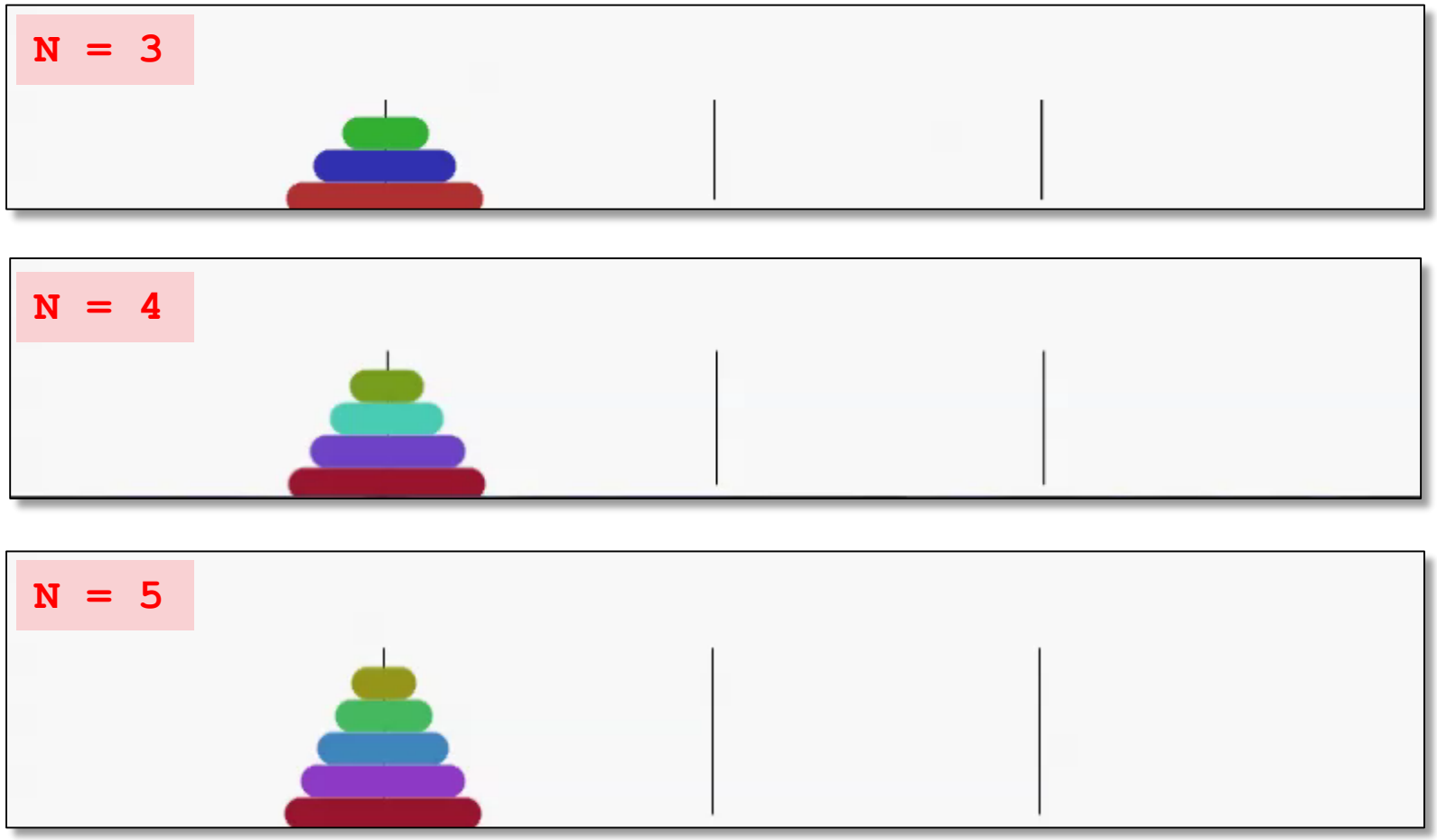
(۳) انتقال 1 – N دیسک به میله راست



برج‌های هانوی: راه‌حل بازگشتی


```
public class TowersOfHanoi
{
   public static void moves(int N, int from, int to, int help)
   {
      if (N == 1)
         System.out.printf("%d --> %d\n", from, to);
      else
      {
        moves(N - 1, from, help, to);
        System.out.printf("%d --> %d\n", from, to);
        moves(N - 1, help, to, from);
      }

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      moves(N, 1, 3, 2);
   }
}
```

برج‌های هانوی: اجرای نمایشی

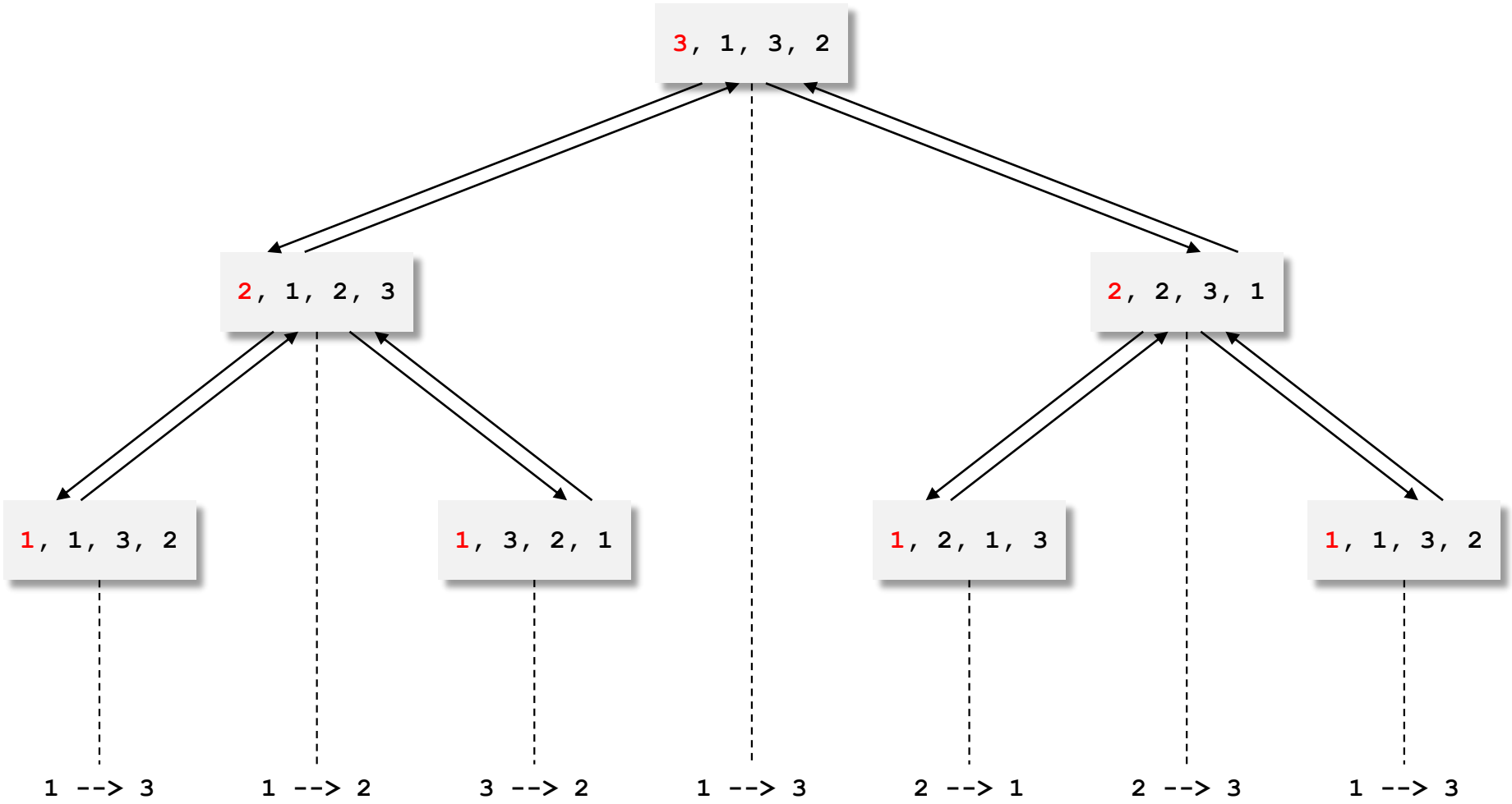


برج‌های هانوی: راه‌حل بازگشتی


```
%java TowersOfHanoi 3
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
```

```
%java TowersOfHanoi 4
1 --> 2
1 --> 3
2 --> 3
1 --> 2
3 --> 1
3 --> 2
1 --> 2
1 --> 3
2 --> 3
2 --> 1
3 --> 1
2 --> 3
1 --> 2
1 --> 3
2 --> 3
```

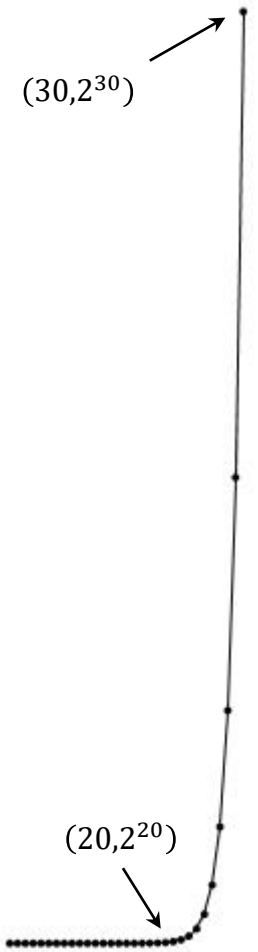
برج‌های هانوی: راه‌حل بازگشتی



برج‌های هانوی: ویژگی‌های راه‌حل

- تعداد جابه‌جایی‌های لازم برای انتقال N دیسک؟
  - برای حل یک مسئله با N دیسک به $2^N - 1$ جابجایی نیاز است !!!

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

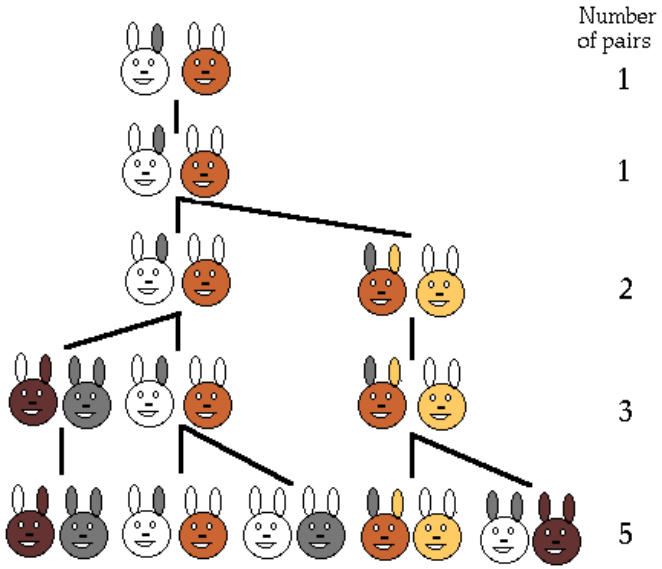


- پس بالاخره سرنوشت دنیا چه می‌شود؟
  - با این فرض که هر جابه‌جایی تنها به یک ثانیه زمان نیاز داشته باشد، دنیا ۵۸۵ میلیارد سال پس از خلقت آن به پایان می‌رسد.
  - هنوز بیش از ۵۷۰ میلیارد سال باقیمانده است.
  - برای این که خیالتان راحت شود، این راه‌حل سریع‌ترین راه‌حل است.

اعداد فیبوناچی

«فرض کنیم خرگوش‌هایی وجود دارند که هر جفت (یک نر و یک ماده) از آنها که به سن ۱ ماهگی رسیده باشند، به ازای هر ماه که از زندگی‌شان سپری شود یک جفت خرگوش به دنیا می‌آورند که آن‌ها هم از همین قاعده پیروی می‌کنند. حال اگر فرض کنیم این خرگوش‌ها هرگز نمی‌میرند و در آغاز یک جفت از این نوع خرگوش در اختیار داشته باشیم که به تازگی متولد شده‌اند، حساب کنید پس از **n** ماه چند جفت از این نوع خرگوش خواهیم داشت.»

لئوناردو فیبوناچی
(۱۱۷۰ – ۱۲۵۰)



اعداد فیبوناچی

- اعداد فیبوناچی.

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

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n \geq 2 \end{cases}$$

- پیاده‌سازی بازگشتی.

```
public static long F(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return F(n-1) + F(n-2);
}
```

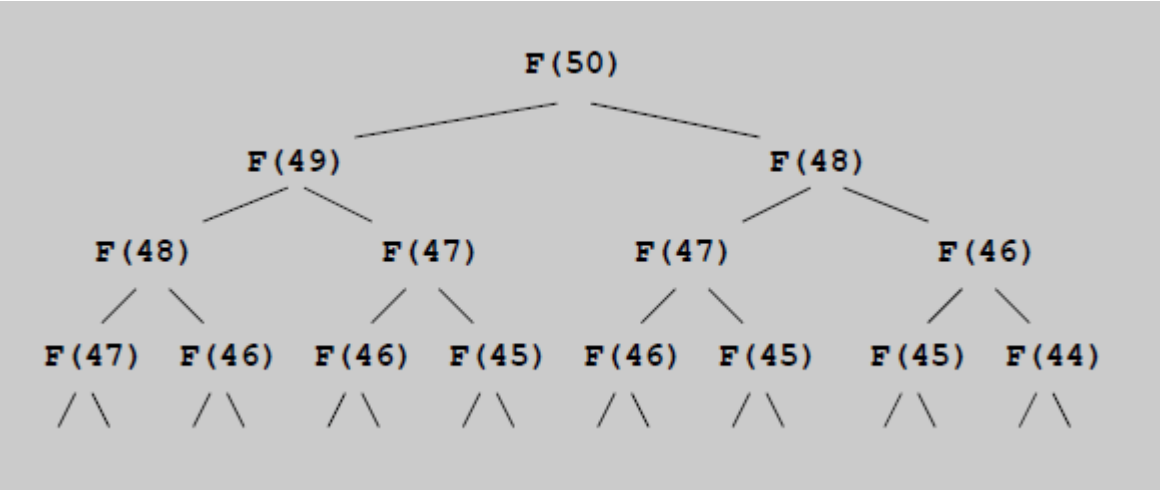
اعداد فیبوناچی

- س. آیا راه‌حل بازگشتی برای محاسبه **F(50)** کارا است؟

```
public static long F(int n) {
   if (n == 0) return 0;
   if (n == 1) return 1;
   return F(n-1) + F(n-2);
}
```

- ج. خیر، این کد به طرز شگفت‌انگیزی ناکارامد است.

F(50): ۱ بار فراخوانی می‌شود
F(49): ۱ بار فراخوانی می‌شود
F(48): ۲ بار فراخوانی می‌شود
F(47): ۳ بار فراخوانی می‌شود
F(46): ۵ بار فراخوانی می‌شود
F(45): ۸ بار فراخوانی می‌شود
...
F(1): ۱۲۵۷۶۲۶۹۰۲۵ بار فراخوانی می‌شود



اعداد فیبوناچی

- س. آیا روش سریع‌تری برای محاسبه **F(50)** وجود دارد؟
- ج. بله، کد زیر این کار را تنها با ۴۹ عمل جمع انجام می‌دهد.

```
public static long F(int n) {
   if (n == 0) return 0;
   long[] F = new long[n+1];
   F[0] = 0;
   F[1] = 1;
   for (int i = 2; i <= n; i++)
      F[i] = F[i-1] + F[i-2];
   return F[n];
}
```

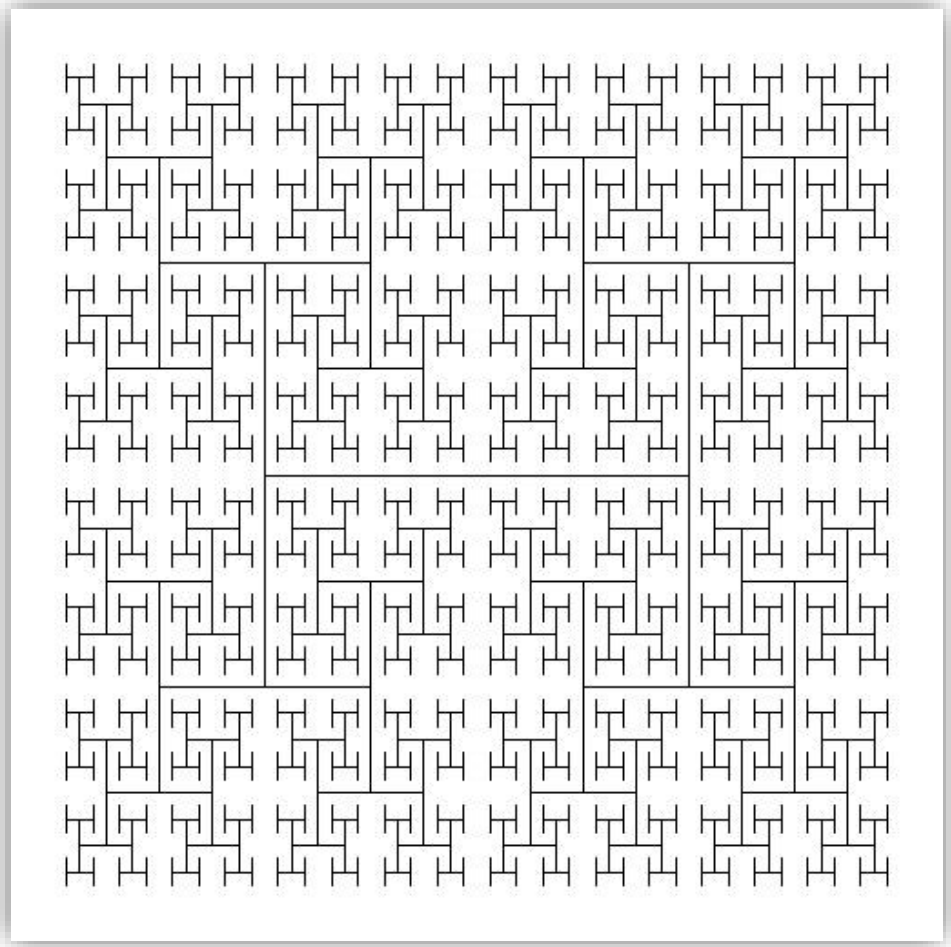
F(5)

0	1	2	3	4	5
0	1	1	2	3	5

- تکنیک طراحی الگوریتم. تکنیک بالا به منظور محاسبه اعداد فیبوناچی تکنیک برنامه‌ریزی پویا نام دارد.

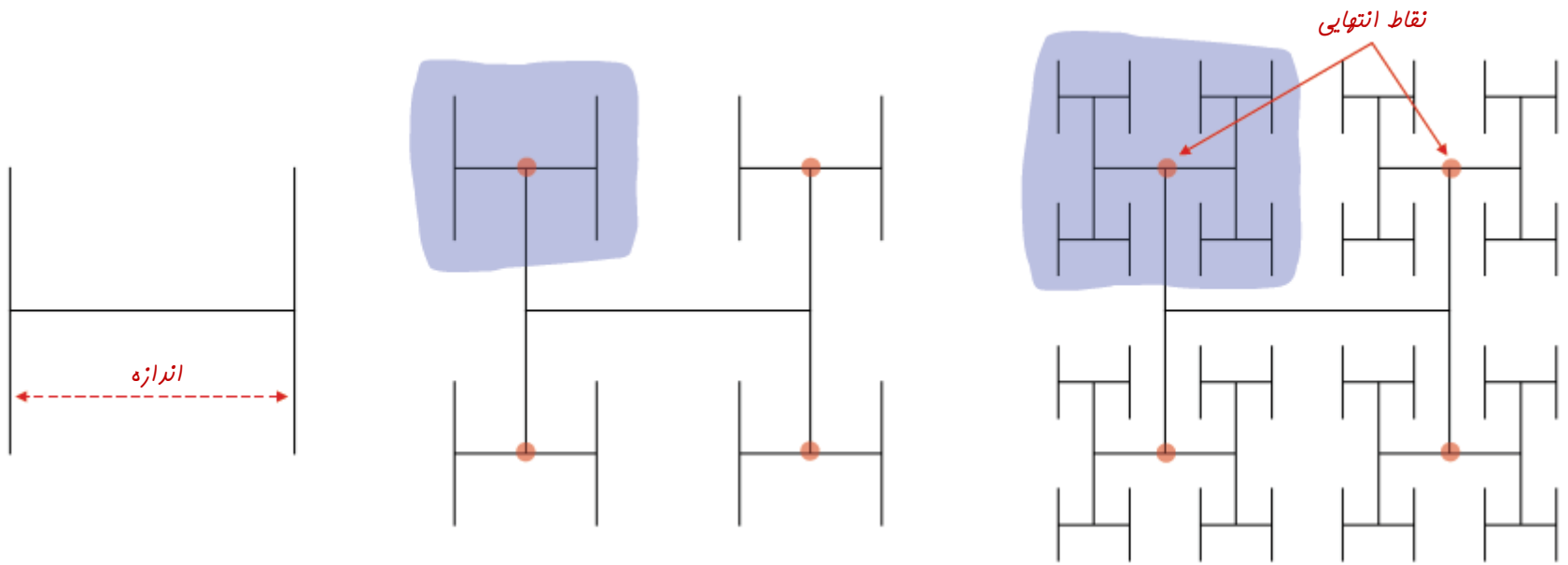
ترسیم بازگشتی

درخت H



درخت H

- درخت **H** از مرتبه **n**.
  - یک **H** ترسیم کنید.
  - به طور بازگشتی، چهار درخت **H** از مرتبه **n – 1** (با اندازه نصف) ترسیم کنید به طوری که هر کدام به یکی از نقاط انتهایی متصل باشند.

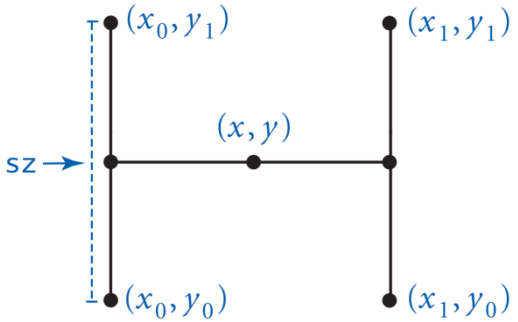


ترسیم بازگشتی: درخت H


```
public class HTree {
   public static void draw(int n, double sz, double x, double y) {
      if (n == 0) return;

      double x0 = x - sz/2, x1 = x + sz/2;
      double y0 = y - sz/2, y1 = y + sz/2;

      StdDraw.line(x0, y, x1, y);
      StdDraw.line(x0, y0, x0, y1);
      StdDraw.line(x1, y0, x1, y1);

      draw(n-1, sz/2, x0, y0);
      draw(n-1, sz/2, x0, y1);
      draw(n-1, sz/2, x1, y0);
      draw(n-1, sz/2, x1, y1);
}

   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      draw(n, .5, .5, .5);
   }
}
```

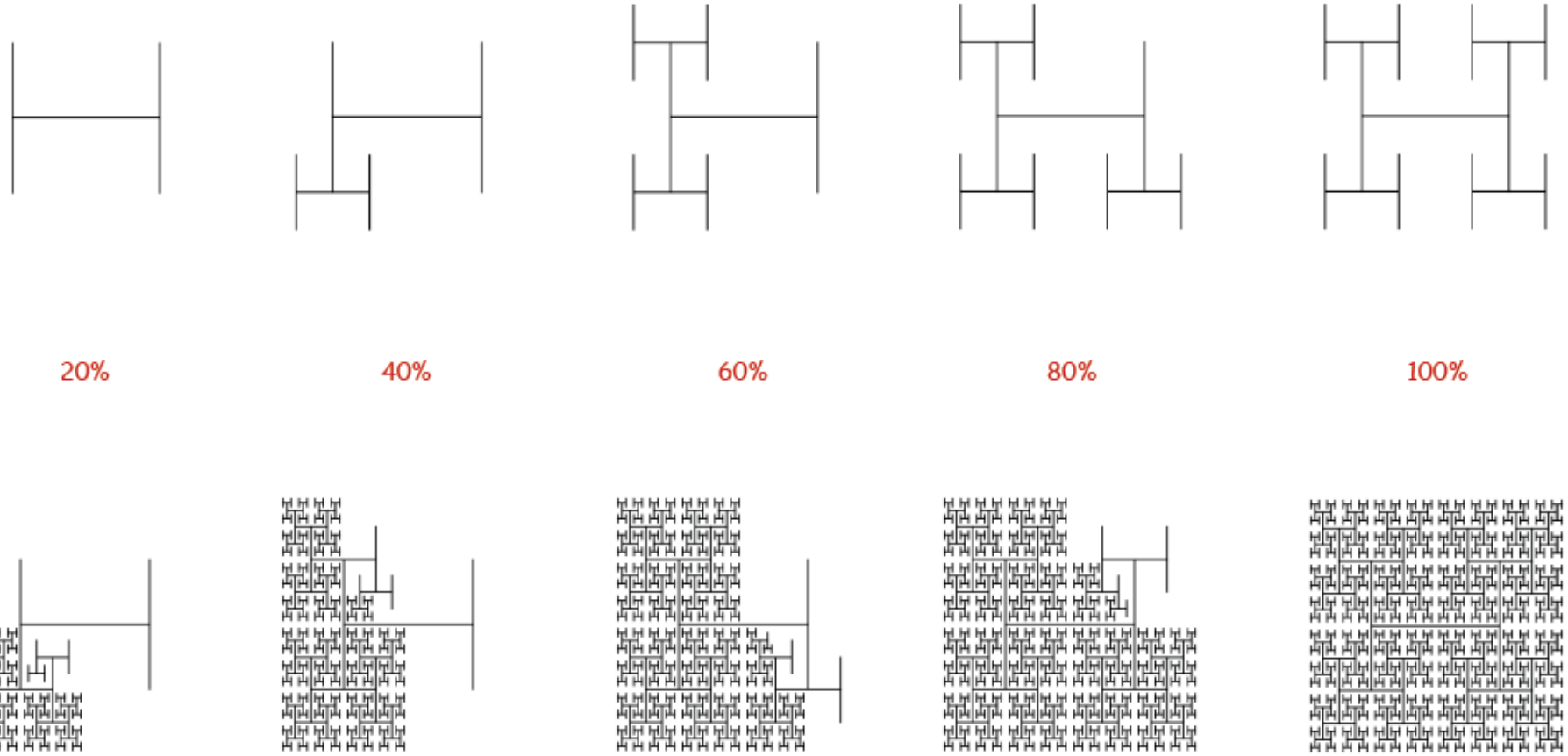
ترسیم H، به مرکز (x, y)

ترسیم چهار درخت H با اندازه نصف به صورت بازگشتی



درخت H

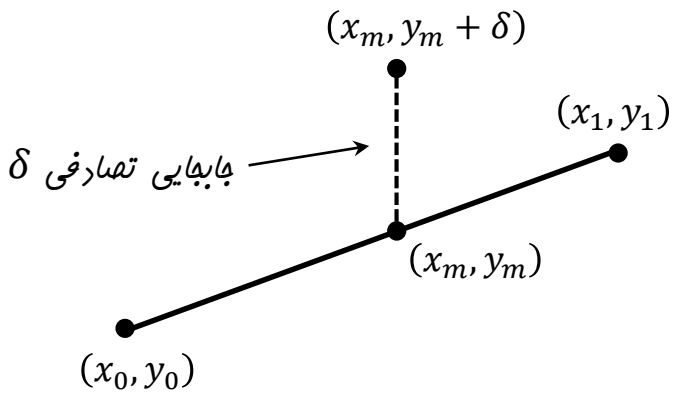
- ترسیم درخت H به صورت انیمیشن. پس از رسم هر H کمی مکث کن.



پل براونی

- پل براونی. یک مدل ریاضی به منظور تولید سطوح ناهموار و لبه‌های دندانه‌دار.
  - مانند کوهستان‌ها، کناره رودخانه‌ها، ابرها و موج‌ها و ...

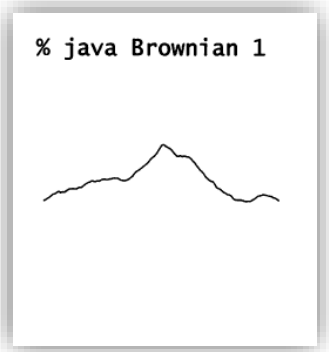
- الگوریتم بازگشتی. ترسیم در بازه $[x_0,x_1]$
  - اگر $x_1 - x_0 < .01$، ترسیم یک خط مستقیم.
  - محاسبه نقطه وسط به مختصات (x_m,y_m).
  - افزودن مقدار تصادفی δ به y_m.
  - فراخوانی بازگشتی در دو زیربازه و تقسیم واریانس به s.

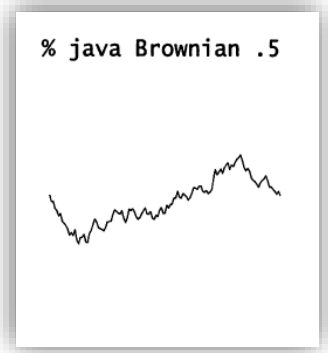


محاسبات پل براونی

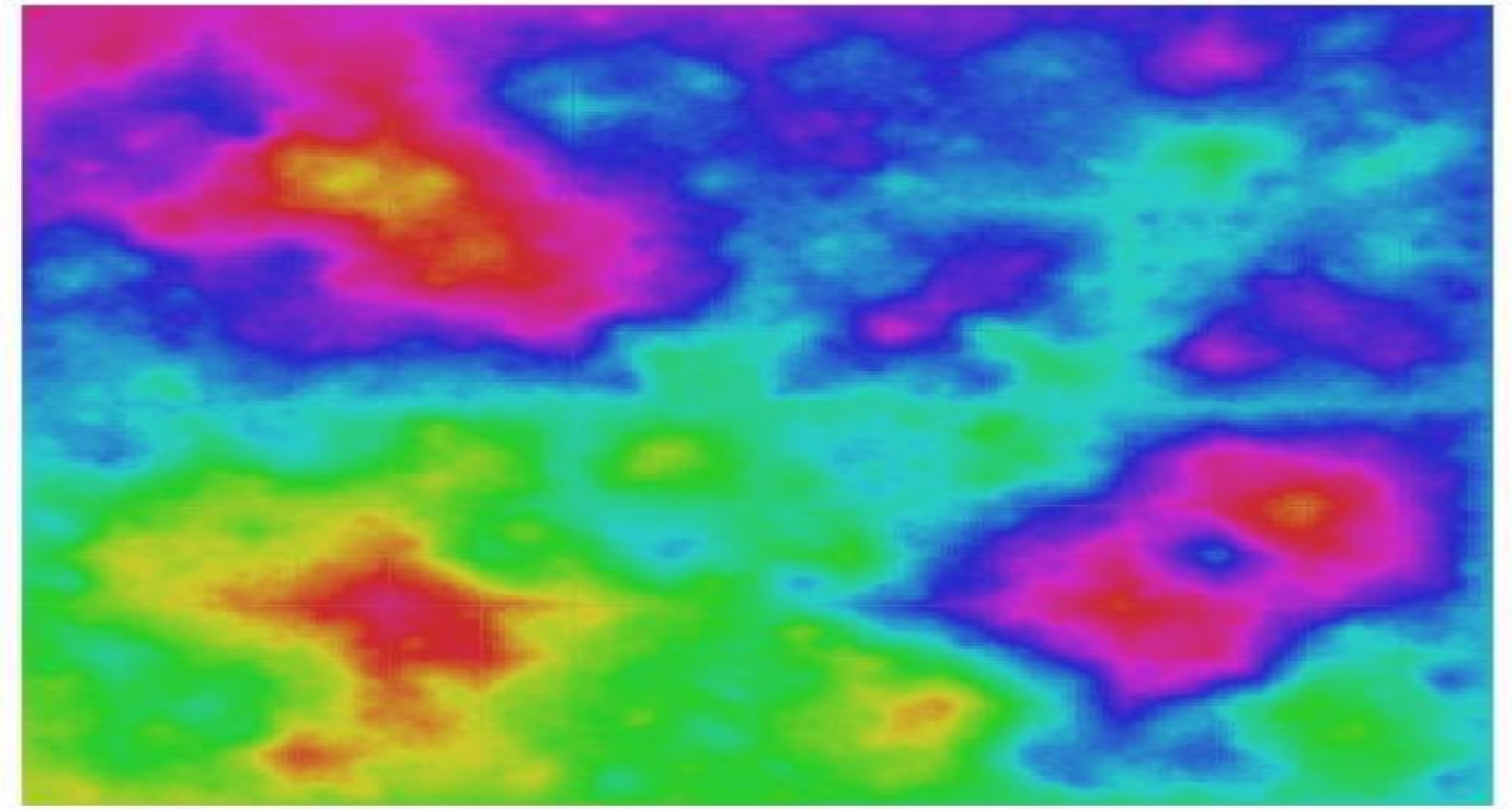
پل براونی


```
public class Brownian {
    // midpoint displacement method
    public static void curve(double  x0, double y0,
                             double  x1, double y1,
                             double var, double  s)
    {
        // stop if interval is sufficiently small
        if (Math.abs(x1 - x0) < .01) {
            StdDraw.line(x0, y0, x1, y1);
            return;
        }
        double xm = (x0 + x1) / 2;
        double ym = (y0 + y1) / 2;
        ym = ym + StdRandom.gaussian(0, Math.sqrt(var));
        curve(x0, y0, xm, ym, var/s, s);
        curve(xm, ym, x1, y1, var/s, s);

    public static void main(String[] args)
    {
        double H = Double.parseDouble(args[0]);
        double s = Math.pow(2, 2*H);
        curve(0.0, 0.5, 1.0, 0.5, .01, s);
    }
}
```





ابر پلاسمایی



خلاصه

- چگونه می‌توان یک تابع بازگشتی نوشت؟
  - حالت پایه، فراخوانی بازگشتی.
  - ردیابی اجرای یک تابع بازگشتی.
  - استفاده از شکل.
- مزایای یادگیری توابع بازگشتی.
  - آشنایی با یک سبک جدید تفکر (تفکر بازگشتی)
  - آشنایی با یک الگوی قدرتمند برنامه نویسی
- تقسیم و حل. یک راه‌حل ظریف و زیبا برای بسیاری از مسائل مهم.