تحلیل الگوریتم‌های بازگشتی

سید ناصر رضوی
www.snrazavi.ir
۱۳۹۵

توابع بازگشتی

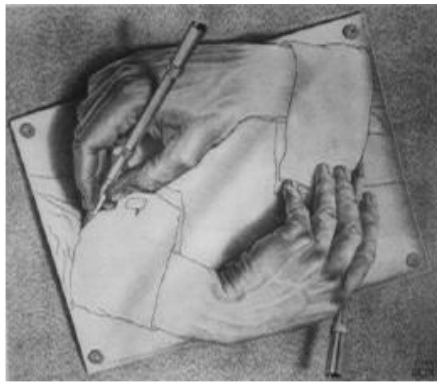
- توابع بازگشتی.
  - محاسبه‌ی فاکتوریل، ب.م.م.، درخت H، برجهای هانوی، دنباله‌ی فیبوناچی، مرتب‌سازی ادغامی
- تحلیل توابع بازگشتی.
- روش‌های حل روابط بازگشتی.
  - حدس و استقرا
  - جایگذاری مکرر
  - قضیه‌ی اصلی
  - معادله‌ی مشخصه
  - درخت بازگشت

الگوریتم‌های بازگشتی

مرور

- 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



- رابطه‌ی نزدیک با استقرای ریاضی.

فاکتوریل

- محاسبه‌ی فاکتوریل n.

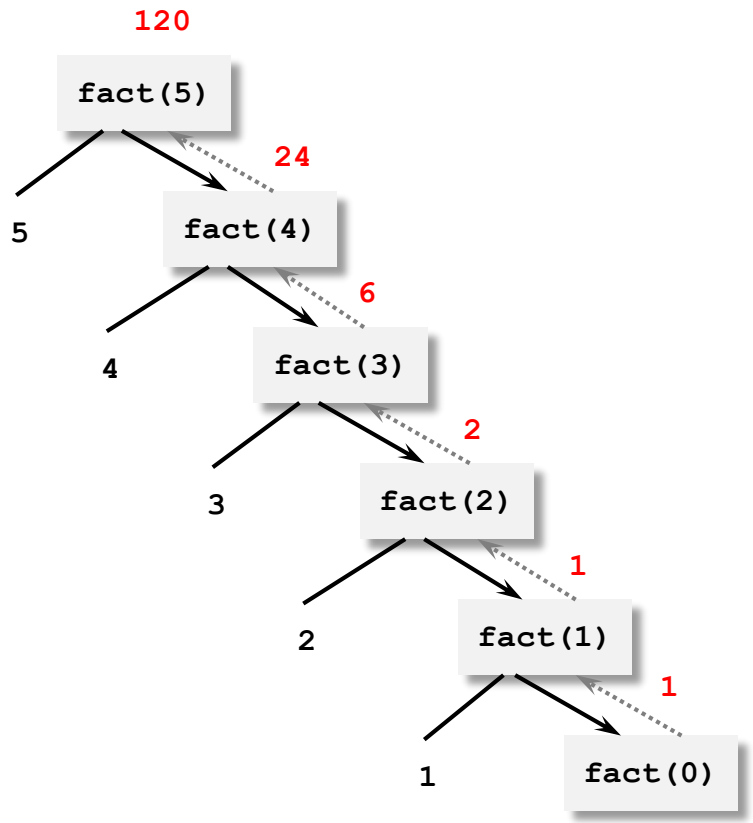
$$n! = 1 * 2 * \ldots * (n-1) * n$$

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

```
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** هر دو 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]

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

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

4032 = 3 * 1272 + 216

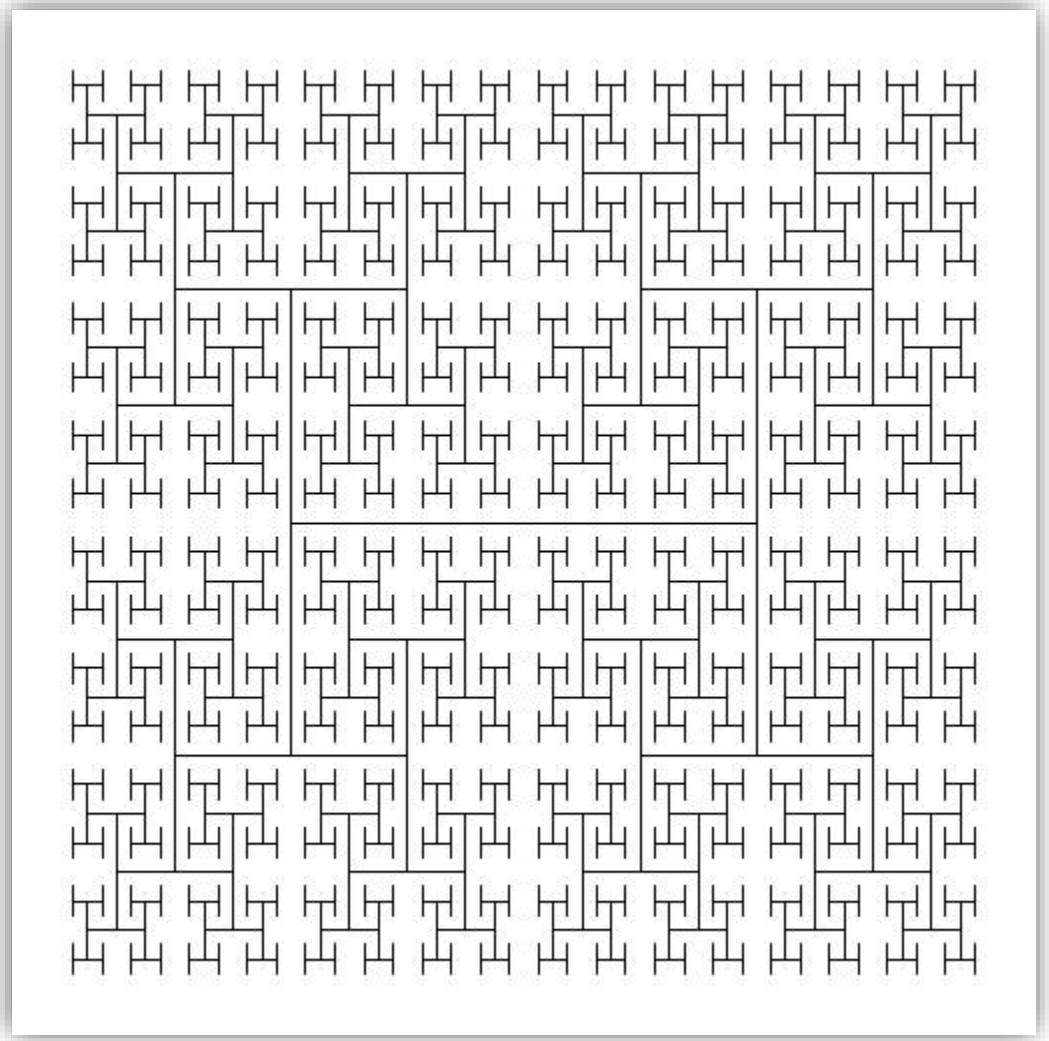
بزرگ‌ترین مقسوم علیه مشترک

- الگوریتم اقیلدس. [۳۰۰ سال قبل از میلاد]

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

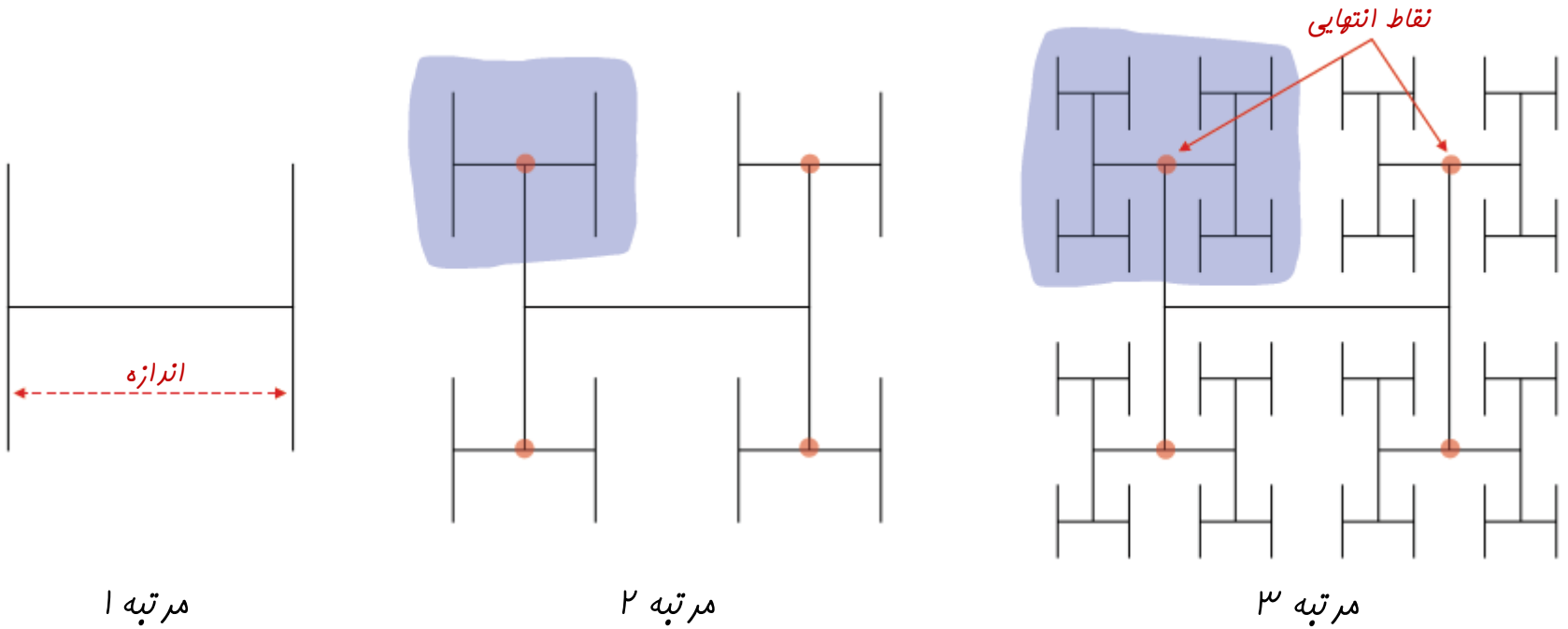
```
public static int gcd(int p, int q)
{
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

درخت H



درخت H

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

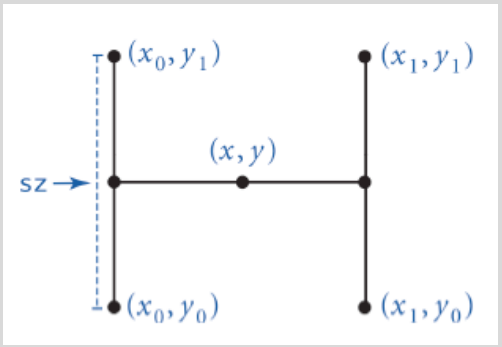


ترسیم بازگشتی: درخت H

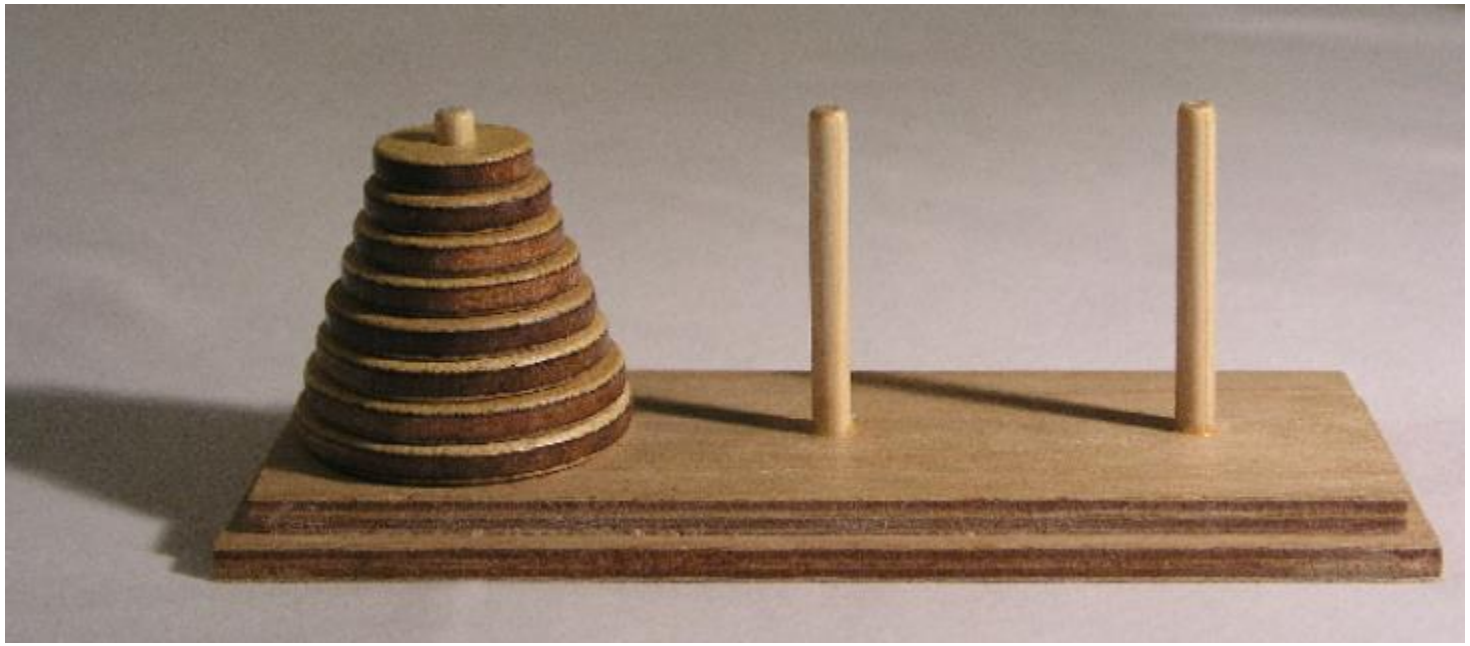

```
public class HTree
{
   public static void draw(int n, double sz, double x, y)
   {
      if (n == 0) return;
      double x0 = x - sz/2; double x1 = x + sz/2;
      double y0 = y - sz/2; double y1 = y + sz/2;

      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, 0.5, 0.5, 0.5);
   }
}
```

ترسیم H

ترسیم ۴ درخت H
به صورت بازگشتی



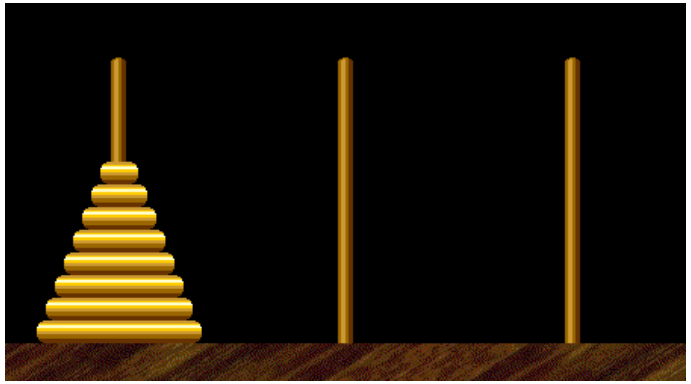
برجهای هانوی



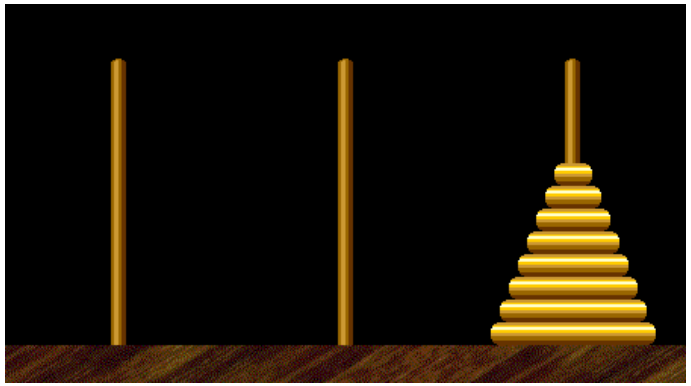
برجهای هانوی

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

شروع



پایان

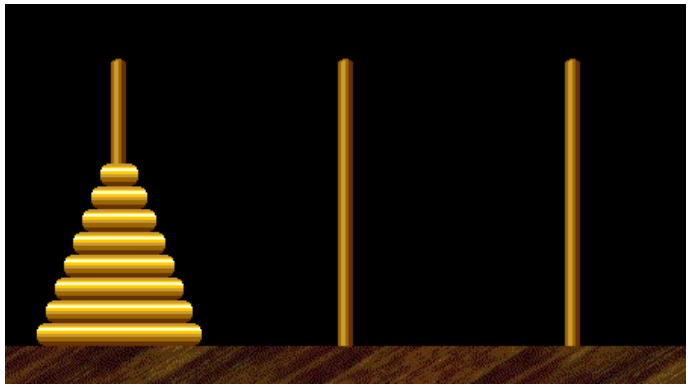


افسانه‌ی برجهای هانوی

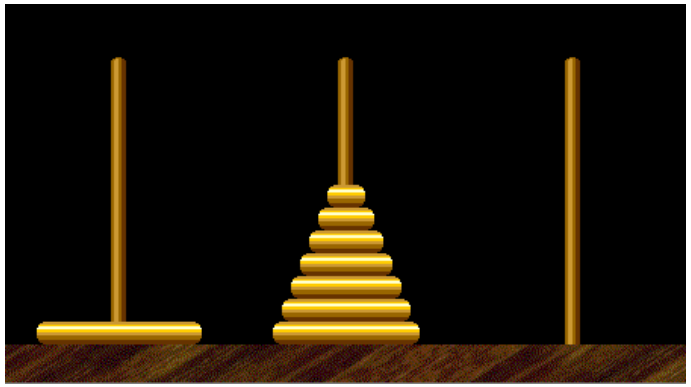
- س. دنیا چه زمانی به پایان می‌رسد؟
  - ۶۴ دیسک طلا بر روی سه میله‌ی الماس در ابتدای خلقت.
  - با اتمام کار یک گروه خاص از کشیش‌ها که مسئول انتقال این دیسک‌ها هستند، دنیا به پایان می‌رسد.

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

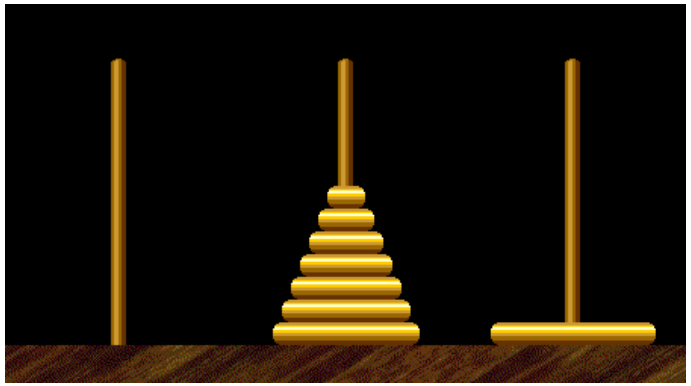
برجهای هانوی: راه‌حل بازگشتی



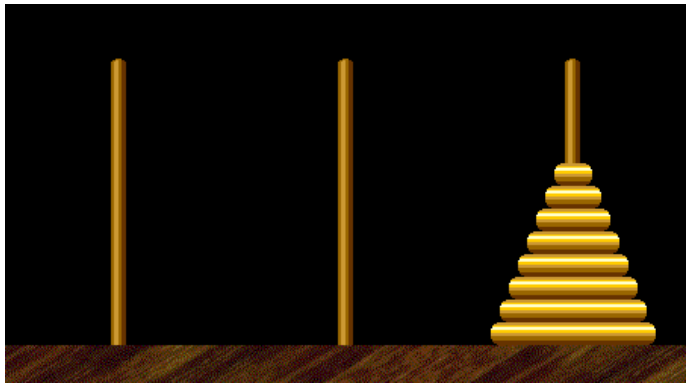
(۱) انتقال 1 − N دیسک به میله‌ی وسط



(۲) انتقال بزرگترین دیسک به میله‌ی راست



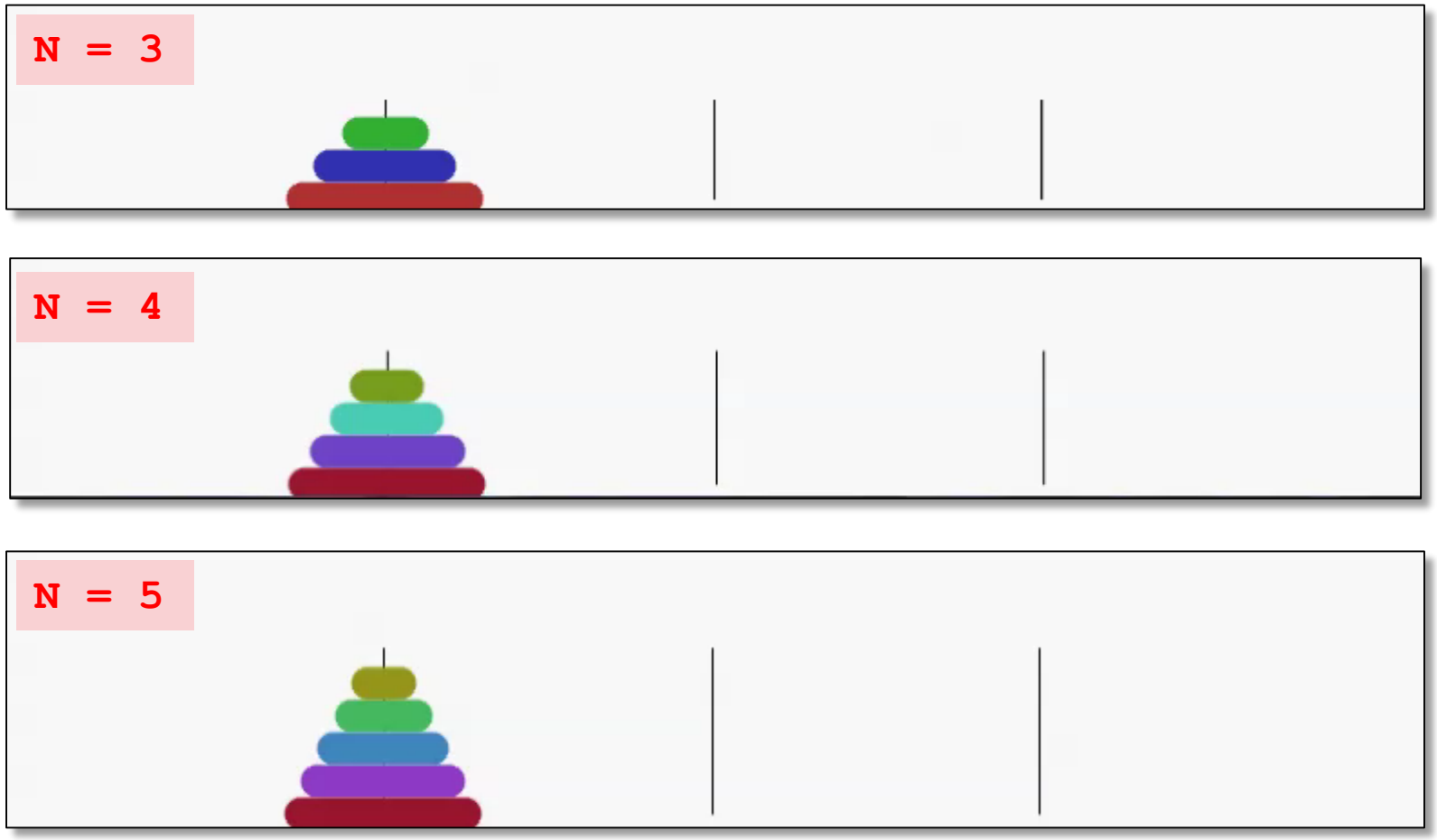
(۳) انتقال 1 − N دیسک به میله‌ی راست



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

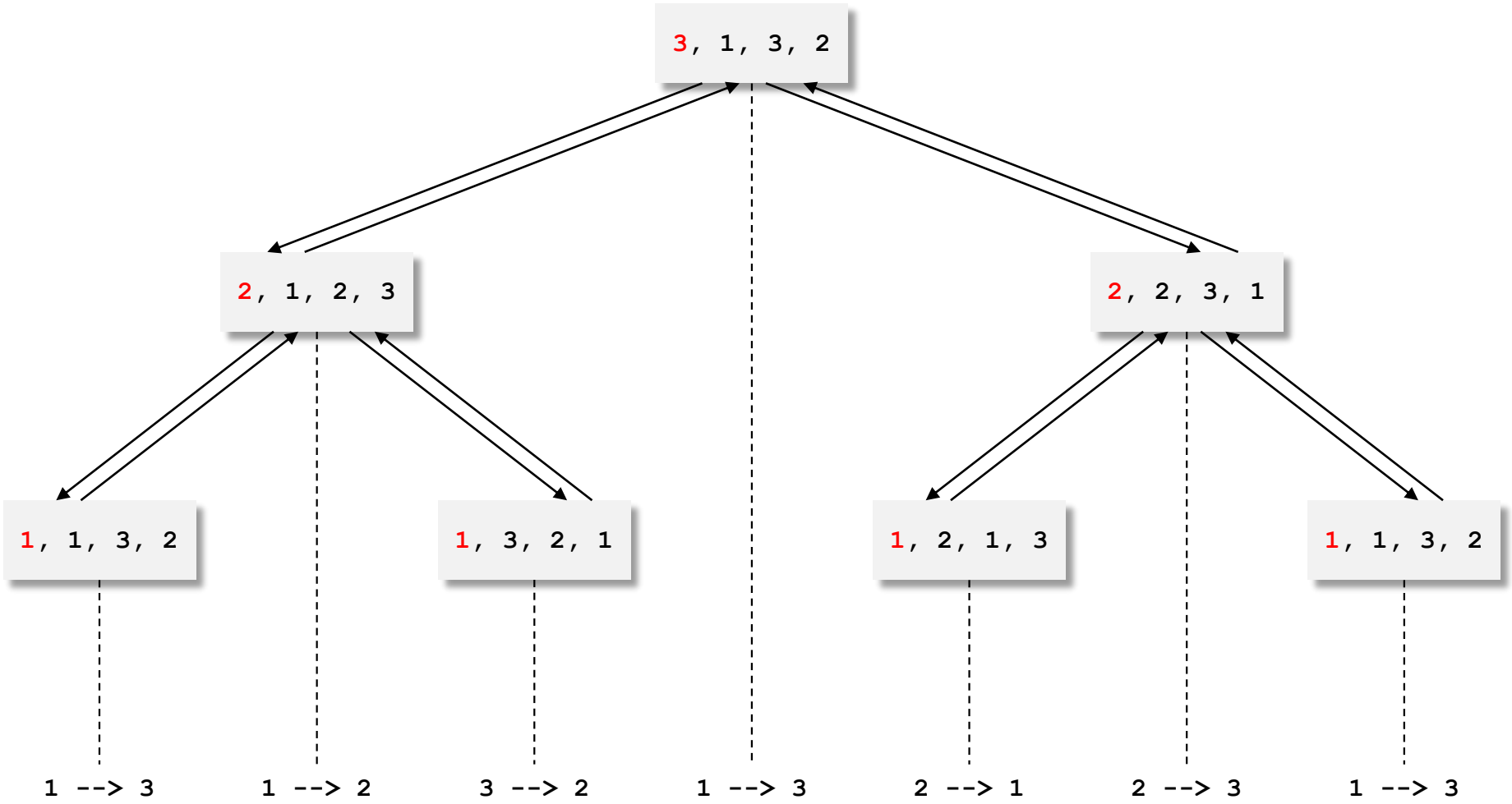


برجهای هانوی: راه‌حل بازگشتی


```
%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

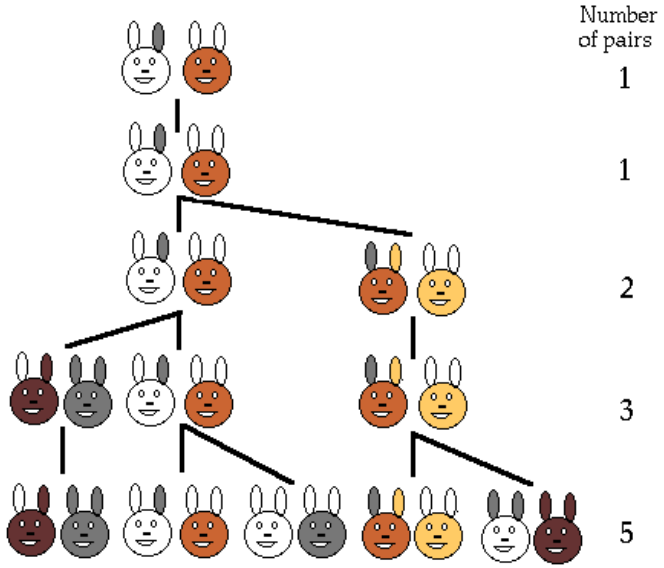
- تعداد جابجایی‌های لازم برای انتقال **N** دیسک؟
  - برای حل یک مسئله با **N** دیسک به **$2^N - 1$** جابجایی نیاز است !!! [با ما باشید]

- بالاخره سرنوشت دنیا چه می‌شود؟
  - اگرهر جابه‌جایی تنها یک ثانیه طول بکشد، 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 ممکن 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 متولد می‌کنند 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, …

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

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

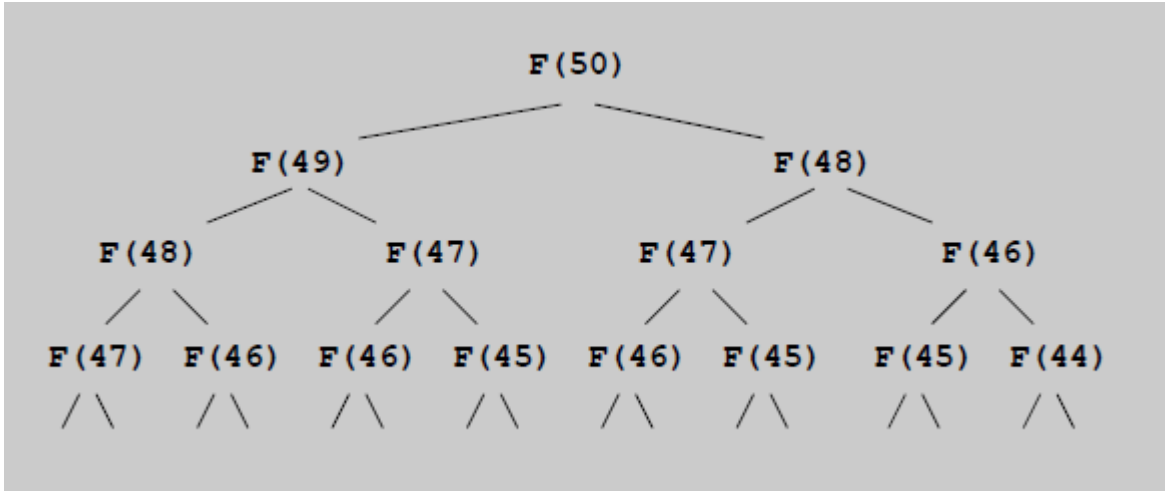
اعداد فیبوناچی

- س. آیا راه‌حل بازگشتی برای محاسبه‌ی **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 می شود
F(49): ۱ بار فراخوانی می شود
F(48): ۲ بار فراخوانی می شود
F(47): ۳ بار فراخوانی می شود
F(46): ۵ بار فراخوانی می شود
F(45): ۸ بار فراخوانی می شود
...
F(1): ۱۲۵۷۶۲۶۹۰۲۵ بار فراخوانی می شود



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 برای محاسبه‌ی اعداد فیبوناچی، تکنیک برنامه‌ریزی پویا نام دارد. [با ما باشید]

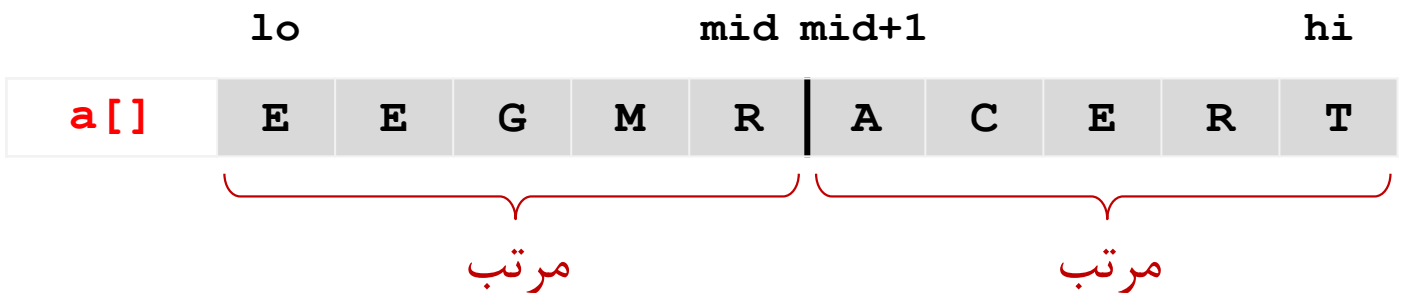
مرتب‌سازی ادغامی

- طرح اصلی. [تقسیم و حل]
  - آرایه را به دو نیمه تقسیم کن.
  - هر نیمه را به صورت بازگشتی مرتب کن.
  - دو نیمه‌ی مرتب را با هم ادغام کن.

ورودی	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
مرتب سازی نیمه اول	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
مرتب سازی نیمه دوم	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
ادغام	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

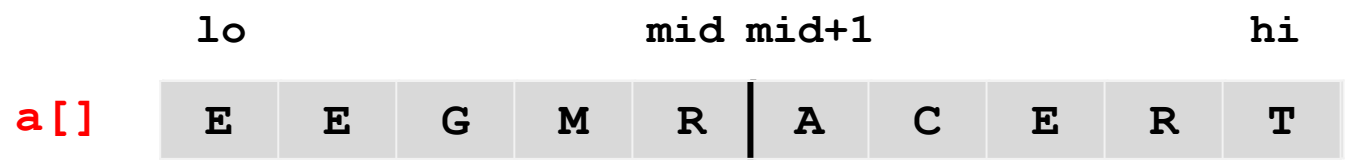
ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



ادغام

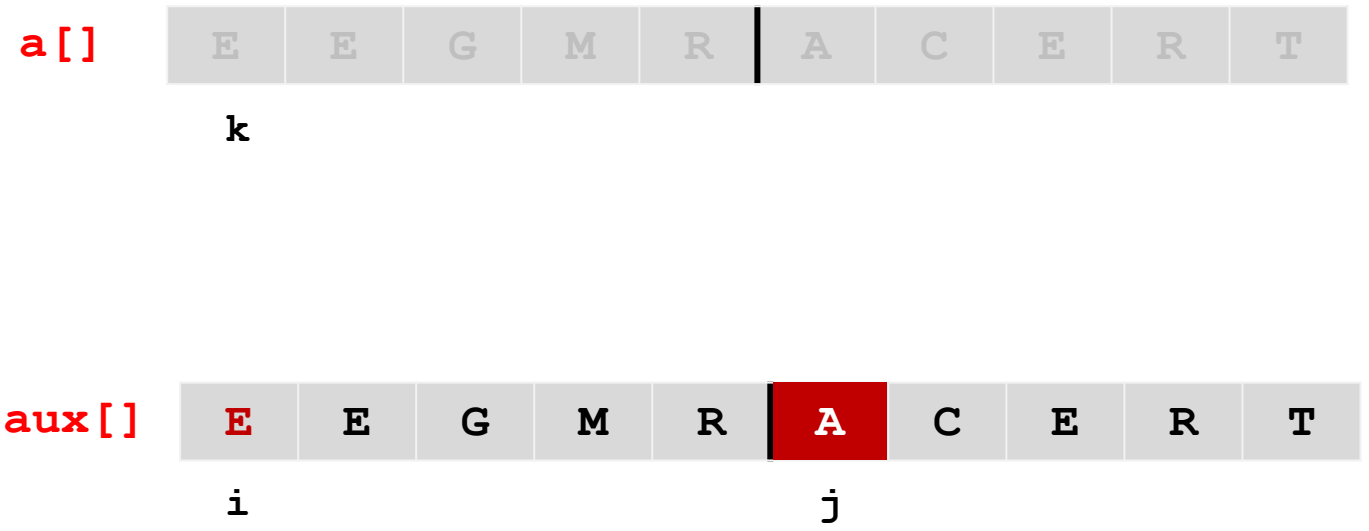
- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



کپی کردن عناصر در آرایه کمکی

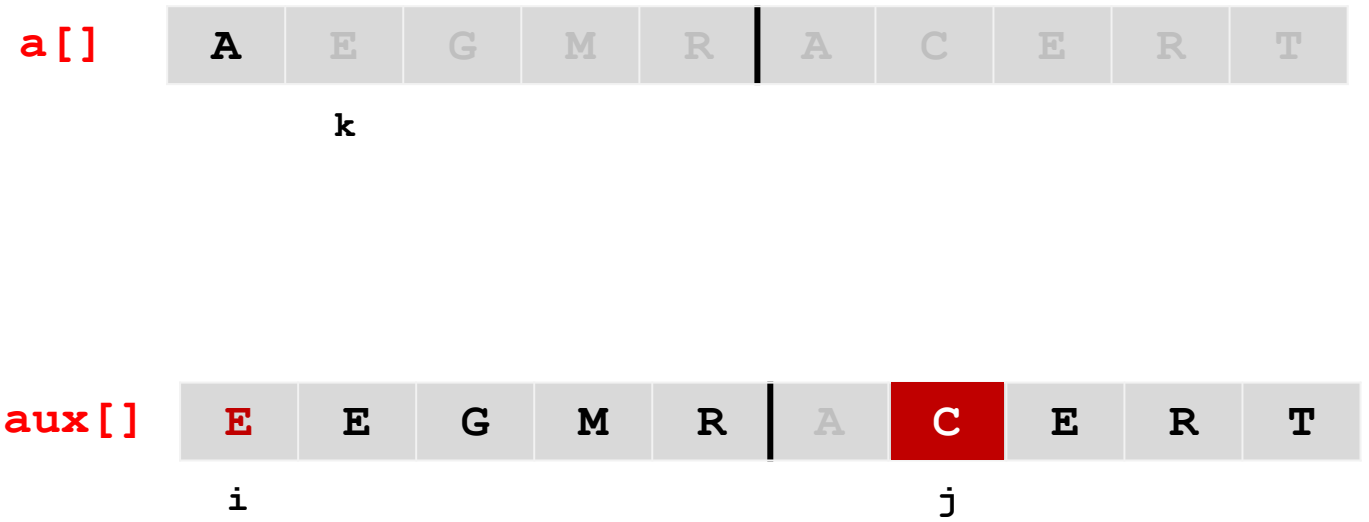
ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



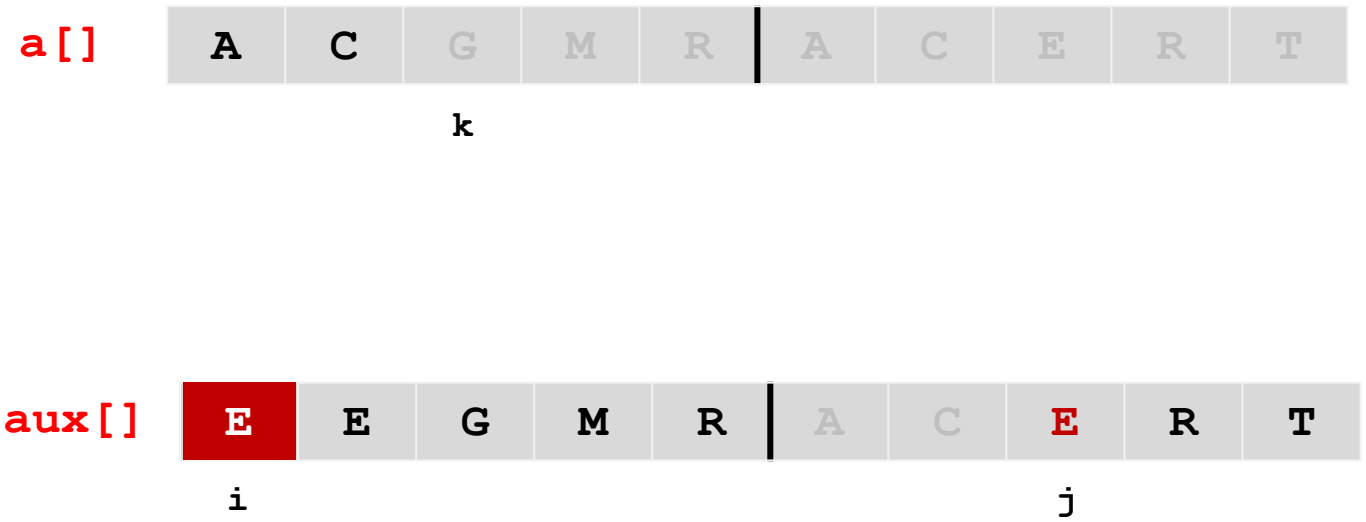
ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.

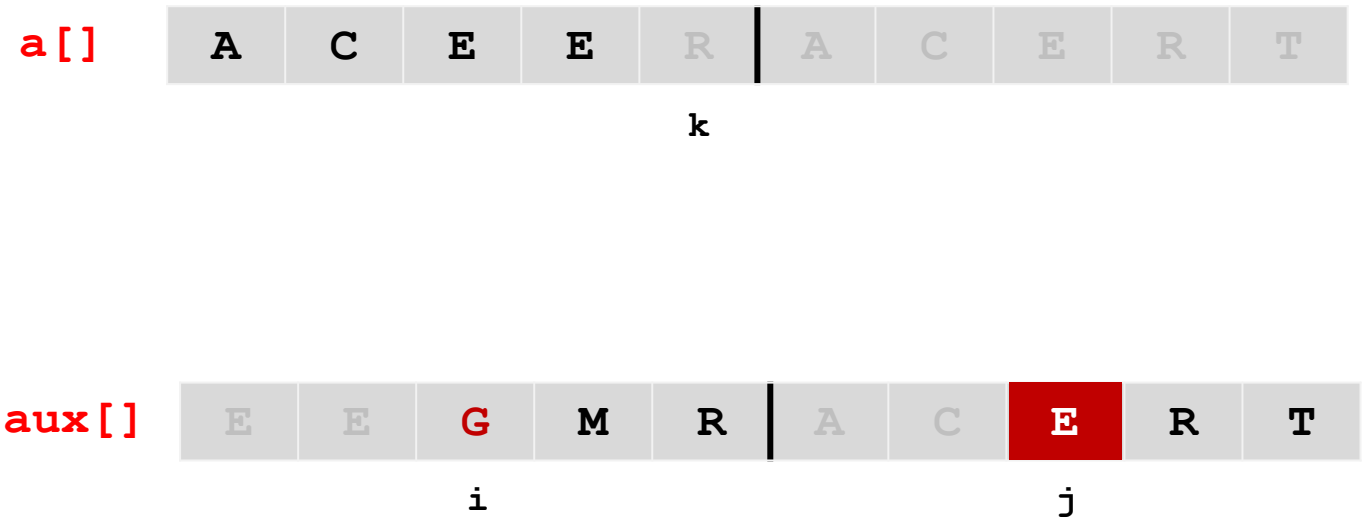


ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.

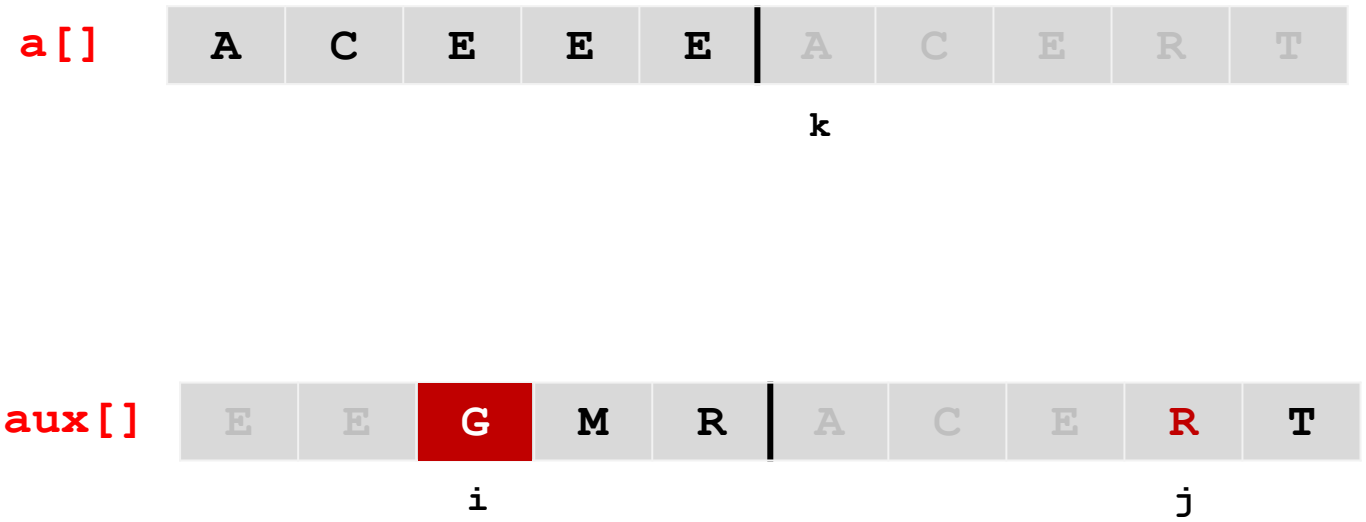
ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



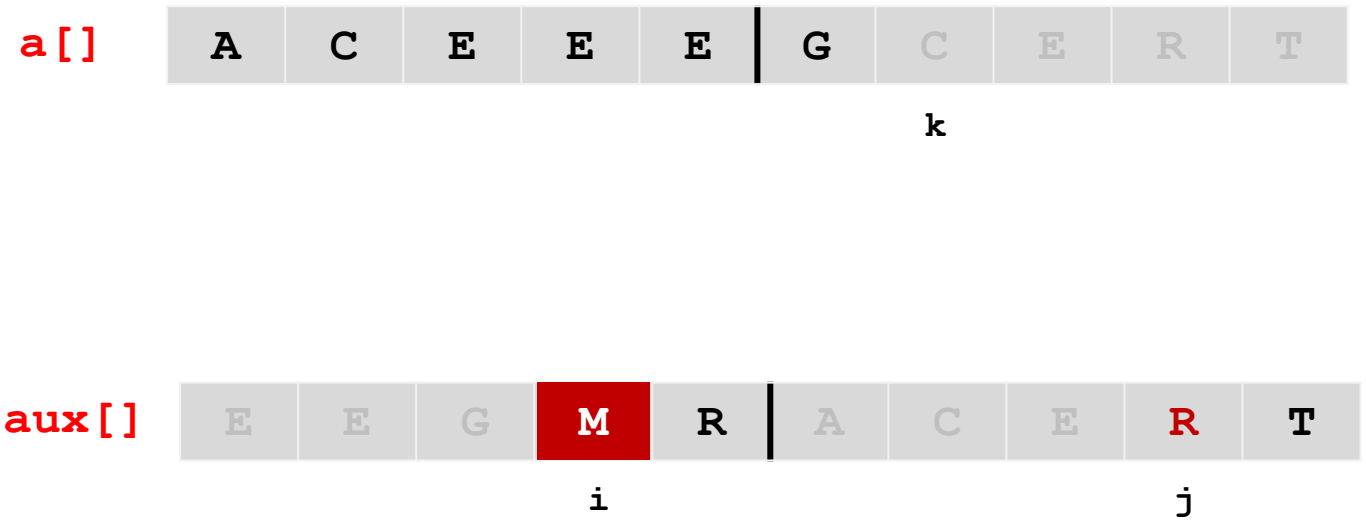
ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



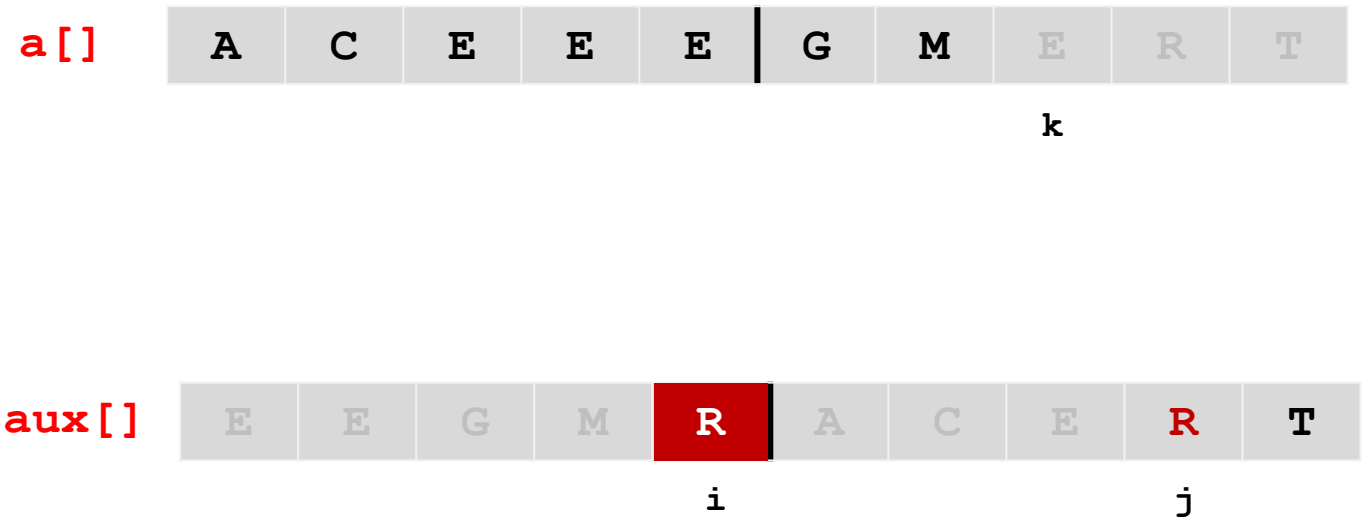
ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.

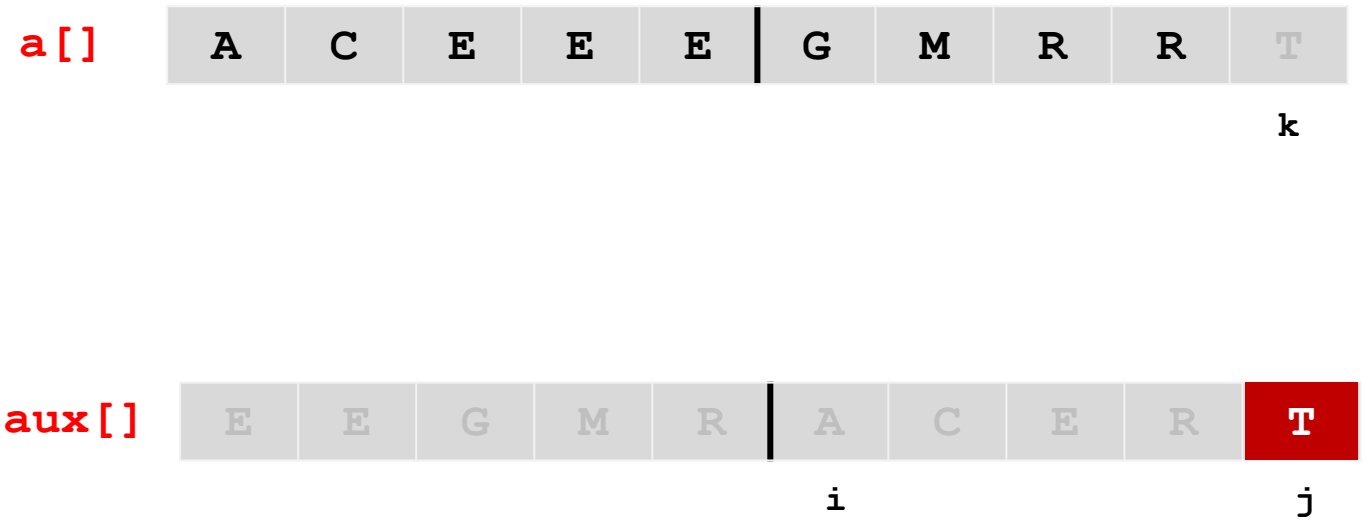


ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.

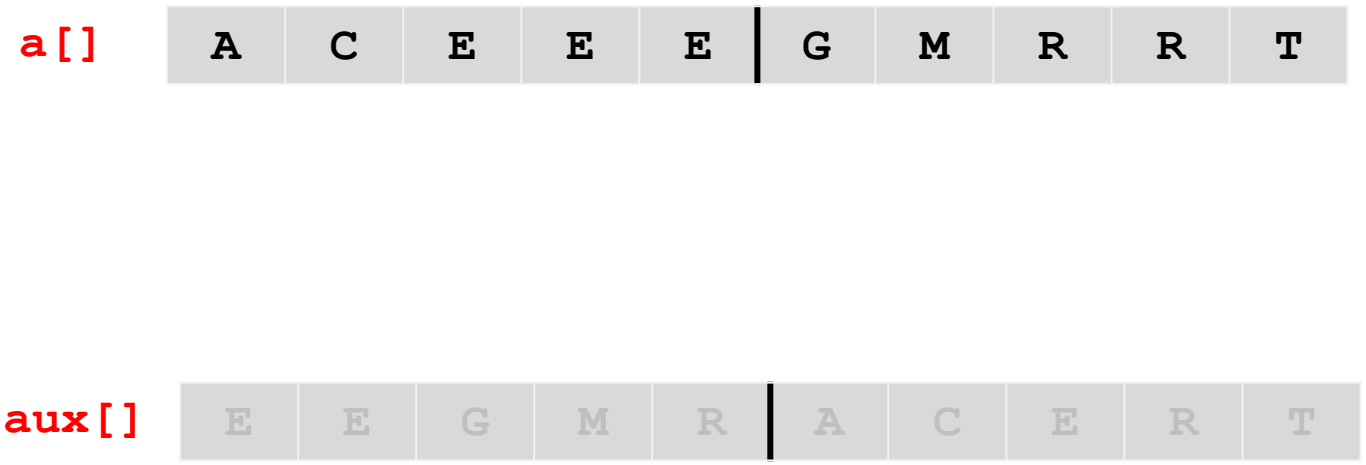
ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



ادغام

- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



ادغام: پیاده‌سازی در جاوا


```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
   assert isSorted(a, lo, mid);       // precondition: a[lo..mid]   sorted
   assert isSorted(a, mid+1, hi);     // precondition: a[mid+1..hi] sorted

   for (int k = lo; k <= hi; k++)
        aux[k] = a[k]

   int i = lo, j = mid + 1;
   for (int k = lo; k <= hi; k++)
   {
      if       (i > mid)                a[k] = aux[j++];
      else if  (j >  hi)                a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                              a[k] = aux[i++];
   }

   assert isSorted(a, lo, hi);        // postcondition: a[lo..hi] sorted
}
```

کپی

ادغام

جملات ادعایی: assert

- جمله‌ی ادعایی. دستوراتی برای آزمایش فرض‌هایی در برنامه.
  - کمک به تشخیص خطاهای منطقی.
  - مستندسازی کد برنامه.

- دستور **assert** در جاوا. باعث بروز استثنا می‌شود مگر آنکه عبارت بولی صحیح باشد.

```
assert isSorted(a, lo, hi);
```

- می‌توان دستورات **assert** را در زمان اجرا فعال یا غیر فعال نمود.

```
java -ea MyProgram;          // enable assertions
java -da MyProgram;          // disable assertions (default)
```

مرتب‌سازی ادغامی: پیاده‌سازی در جاوا


```
public class MergeSort
{
  private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
  {  /* see slide 39 */  }

  private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
  {
     if (hi <= lo) return;
     int mid = lo + (hi - lo) / 2;
     sort (a, aux, lo, mid);
     sort (a, aux, mid+1, hi);
     merge(a, aux, lo, mid, hi);
  }

  public static void sort(Comparable[] a)
  {
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length - 1);
  }
}
```

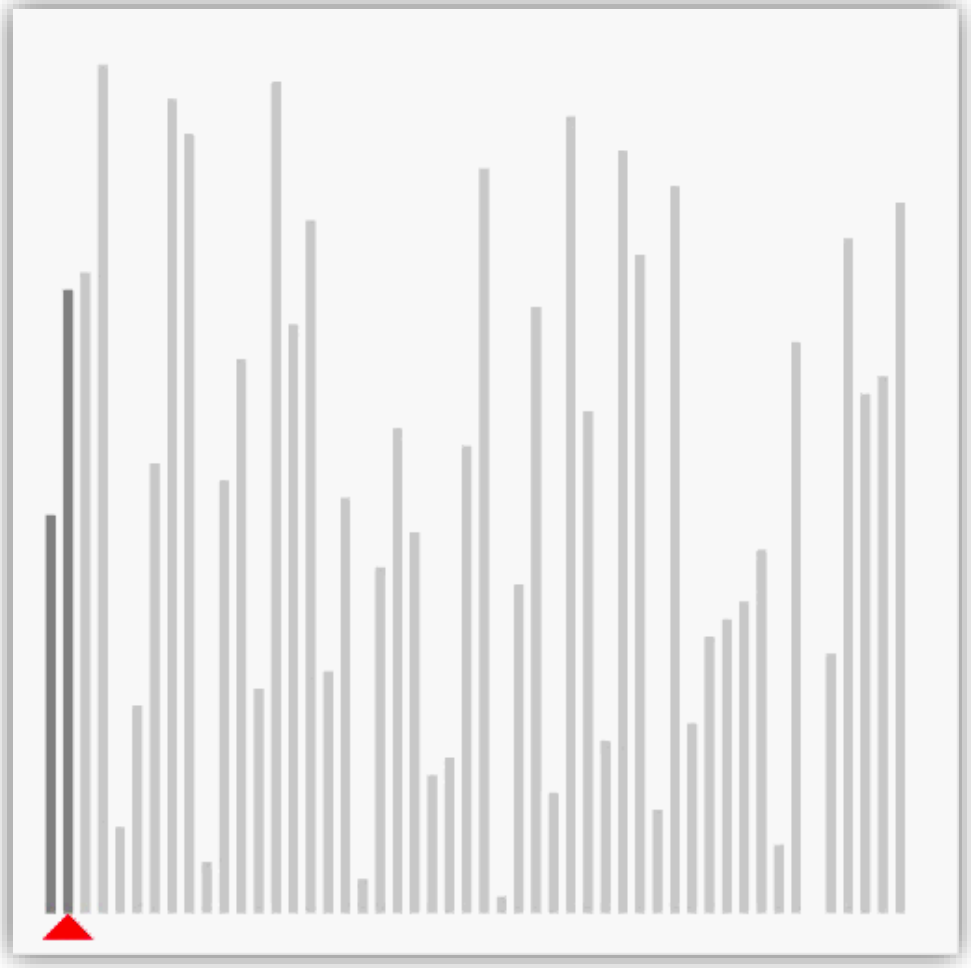
مرتب‌سازی ادغامی: ردیابی اجرا

	lo	hi	a[] 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	0,	1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	2,	3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	0,	3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	4,	5)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	6,	7)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	4,	7)	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
merge(a,	0,	7)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
merge(a,	8,	9)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
merge(a,	10,	11)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
merge(a,	8,	11)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a,	12,	13)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a,	14,	15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
merge(a,	12,	15)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a,	8,	15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge(a,	0,	15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

```
% java TraceMerge MERGESORTEXAMPLE
```

مرتب‌سازی ادغامی (اجرای نمایشی)

- مرتب‌سازی ۵۰ عنصر تصادفی.



```
% java Merge 50
```

مرتب‌سازی ادغامی: تحلیل تجربی

- تخمین زمان اجرا:
  - لپ‌تاپ: 10^8 مقایسه در ثانیه
  - ابر کامپیوتر: 10^{12} مقایسه در ثانیه

کامپیوتر	مرتب‌سازی درجی (N^2)			مرتب‌سازی ادغامی ($N \lg N$)		
	هزار	میلیون	میلیارد	هزار	میلیون	میلیارد
لپ‌تاپ	فوراً	۲/۸ ساعت	۳۱۷ سال	فوراً	۱ ثانیه	۱۸ دقیقه
ابر کامپیوتر	فوراً	۱ ثانیه	۱ هفته	فوراً	فوراً	فوراً

- نتیجه‌گیری. یک الگوریتم خوب بیشتر از یک ابرکامپیوتر ارزش دارد!

خلاصه

- چگونه الگوریتم‌های بازگشتی بنویسیم؟
  - حالت ختم بازگشتی و حالت بازگشتی را مشخص کنید.
  - از درخت بازگشتی برای دنبال کردن مراحل اجرا استفاده کنید.
  - از شکل و تصویر استفاده کنید.
- مزایای یادگیری الگوریتم‌های بازگشتی.
  - آشنایی با یک سبک جدید تفکر (تفکر بازگشتی)
  - آشنایی با یک الگوی قدرتمند برنامه‌نویسی
- تکنیک تقسیم و حل. یک راه‌حل ظریف و زیبا برای بسیاری از مسائل!

تحلیل الگوریتم‌های بازگشتی

محاسبه‌ی فاکتوریل n

- مراحل تحلیل یک الگوریتم بازگشتی.
  - تعین عمل اصلی
  - محاسبه‌ی تعداد دفعات اجرای عمل اصلی به صورت یک تابع بازگشتی از اندازه‌ی ورودی
  - حل رابطه‌ی بازگشتی به دست آمده [در ادامه]

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

ضرب T(n)

۰ ضرب

۱ ضرب

ضرب T(n - 1)

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

برجهای هانوی


```
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);                    1
      else
      {
        moves(n - 1, from, help, to);                                T(n - 1)
        System.out.printf("%d --> %d\n", from, to);                     1
        moves(n - 1, help, to, from);                                T(n - 1)
      }
   }
```

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

مرتب‌سازی ادغامی


```
public class MergeSort
{
  private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
  {  /* see slide 81 */  }

  private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
  {
     if (hi <= lo) return;
     int mid = lo + (hi - lo) / 2;
     sort (a, aux, lo, mid);                                    T(n/2)
     sort (a, aux, mid+1, hi);                                  T(n/2)
     merge(a, aux, lo, mid, hi);                                 n - 1
  }
}
```

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(n/2) + n - 1 & n > 1 \end{cases}$$

حل روابط بازگشتی

حل روابط بازگشتی

- حدس و استقرا.
  - حدس جواب با استفاده از چند جمله‌ی اولیه
  - اثبات حدس به وسیله استقرا

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

حدس

```
T(1) = T(0) + 1 = 0 + 1 = 1
T(2) = T(1) + 1 = 1 + 1 = 2
T(3) = T(2) + 1 = 2 + 1 = 3
T(4) = T(3) + 1 = 3 + 1 = 4
...
T(n) = n
```

استقرا

پایه به ازای `n = 0` *برقرار است.*

```
T(0) = 0
```

گام استقرا.

```
T(n + 1) = T(n) + 1 = n + 1
```

حل روابط بازگشتی

- حدس و استقرا.
  - حدس جواب با استفاده از چند جمله‌ی اولیه
  - اثبات حدس به وسیله استقرا

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

حدس

```
T(2)  = T(1) + 1 = 1 + 1 = 2
T(4)  = T(2) + 1 = 2 + 1 = 3
T(8)  = T(4) + 1 = 3 + 1 = 4
T(16) = T(8) + 1 = 4 + 1 = 5
...
T(n)  = lg n + 1
```

استقرا

پایه به ازای **n = 1** برقرار است.

```
T(1) = 1 + lg 1 = 1 + 0 = 1
```

گام استقرا.

```
T(2n) = T(n) + 1
      = (lg n + 1) + 1
      = (lg n + lg 2) + 1
      = lg 2n + 1
```

حل روابط بازگشتی

- حدس و استقرا.
  - حدس جواب با استفاده از چند جمله‌ی اولیه
  - اثبات حدس به وسیله استقرا

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

حدس

```
T(2) = 2T(1) + 1 =  2 + 1 =  3
T(3) = 2T(2) + 1 =  6 + 1 =  7
T(4) = 2T(3) + 1 = 14 + 1 = 15
T(5) = 2T(4) + 1 = 30 + 1 = 31
...
T(n) = 2^n - 1
```

استقرا

پایه به ازای $n = 1$ *برقرار است.*

$$T(1) = 2^1 - 1 = 1$$

گام استقرا.

$$\begin{aligned} T(n+1) &= 2T(n) + 1 \\ &= 2(2^n - 1) + 1 \\ &= 2^{n+1} - 2 + 1 \\ &= 2^{n+1} - 1 \end{aligned}$$

حل روابط بازگشتی

- حدس و استقرا.
  - حدس جواب با استفاده از چند جمله‌ی اولیه
  - اثبات حدس به وسیله استقرا

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(n/2) + n - 1 & n > 1, n = 2^k \end{cases}$$

حدس

```
T(2)  = 2T(1) + 1  =  0 +  1 =  1
T(4)  = 2T(2) + 3  =  2 +  3 =  5
T(8)  = 2T(4) + 7  = 10 +  7 = 17
T(16) = 2T(8) + 15 = 34 + 15 = 49
...
T(n)  = ?
```

حدس راه‌حل همیشه امکان‌پذیر نیست!

حل روابط بازگشتی

- جایگذاری مکرر.

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

```
T(n) = T(n - 1) + 1
     = T(n - 2) + 1 + 1      = T(n - 2) + 2
     = T(n - 3) + 1 + 2      = T(n - 3) + 3
     = T(n - 4) + 1 + 3      = T(n - 4) + 4
                                ...
                             = T(n - i) + i
```

```
T(n - i) = T(0)      ⟹ n - i = 0
                     ⟹ i = n

T(n) = T(n - n) + n
     = T(0) + n
     = 0 + n
     = n
```

حل روابط بازگشتی

- جایگذاری مکرر.

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

$$\begin{aligned}
T(n) &= T(n/2^1) + 1 && = T(n/2^1) + 1 \\
&= T(n/2^2) + 1 + 1 && = T(n/2^2) + 2 \\
&= T(n/2^3) + 1 + 1 + 1 && = T(n/2^3) + 3 \\
&= T(n/2^4) + 1 + 1 + 1 + 1 && = T(n/2^4) + 4 \\
& && \ldots \\
& && = T(n/2^i) + i
\end{aligned}$$

$$T(n/2^i) = T(1) \Rightarrow n/2^i = 1$$

$$\Rightarrow \underline{i = \lg n}$$

$$\begin{aligned}
T(n) &= T(1) + \lg n \\
&= 1 + \lg n
\end{aligned}$$

حل روابط بازگشتی

- جایگذاری مکرر.

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

$$\begin{aligned}
T(n) &= 2T(n-1) + 1 && = 2^1T(n-1) + 2^1 - 1 \\
&= 2^1[2T(n-2) + 1] + 1 && = 2^2T(n-2) + 2^2 - 1 \\
&= 2^2[2T(n-3) + 1] + 3 && = 2^3T(n-3) + 2^3 - 1 \\
&= 2^3[2T(n-4) + 1] + 7 && = 2^4T(n-4) + 2^4 - 1 \\
& && \ldots \\
& && = 2^iT(n-i) + 2^i - 1
\end{aligned}$$

$$\begin{aligned}
T(n-i) = T(1) &\Rightarrow n - i = 1 \\
&\Rightarrow \underline{i = n - 1}
\end{aligned}$$

$$\begin{aligned}
T(n) &= 2^{n-1}\,T(1) + 2^{n-1} - 1 \\
&= 2 \cdot 2^{n-1} - 1 \\
&= 2^n - 1
\end{aligned}$$

حل روابط بازگشتی

- جایگذاری مکرر.

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(n/2) + n - 1 & n > 1, n = 2^k \end{cases}$$

$$\begin{aligned}
T(n) &= 2T(n/2) + n && = 2^1T(n/2^1) + 1\cdot n \\
&= 2^1[2T(n/2^2) + n/2^1] + n && = 2^2T(n/2^2) + 2\cdot n \\
&= 2^2[2T(n/2^3) + n/2^2] + 2n && = 2^3T(n/2^3) + 3\cdot n \\
&= 2^3[2T(n/2^4) + n/2^3] + 3n && = 2^4T(n/2^4) + 4\cdot n \\
& && \ldots \\
& && = 2^iT(n/2^i) + i\cdot n
\end{aligned}$$

$$T(n/2^i) = T(1) \Rightarrow n/2^i = 1$$
$$\Rightarrow \underline{i = \lg n}$$

$$\begin{aligned}
T(n) &= 2^{\lg n}\, T(1) + n \lg n \\
&= n \cdot 0 + n \lg n \\
&= n \lg n
\end{aligned}$$

حل روابط بازگشتی

- قضیه‌ی اصلی.

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^c) \qquad (a > 0, b > 1)$$

$$\Rightarrow \quad T(n) \in \begin{cases} \Theta\left(n^{\log_b a}\right) & \log_b a > c \\ \Theta(n^c \lg n) & \log_b a = c \\ \Theta(n^c) & \log_b a < c \end{cases}$$

$$T(n) = 4T(n/2) + \Theta(n^1) \overset{(1)}{\Rightarrow} T(n) \in \Theta(n^2) \qquad (a = 4, b = 2, c = 1) \Rightarrow \log_b a > c$$

$$T(n) = 4T(n/2) + \Theta(n^2) \overset{(2)}{\Rightarrow} T(n) \in \Theta(n^2 \lg n) \qquad (a = 4, b = 2, c = 2) \Rightarrow \log_b a = c$$

$$T(n) = 4T(n/2) + \Theta(n^3) \overset{(3)}{\Rightarrow} T(n) \in \Theta(n^3) \qquad (a = 4, b = 2, c = 3) \Rightarrow \log_b a < c$$