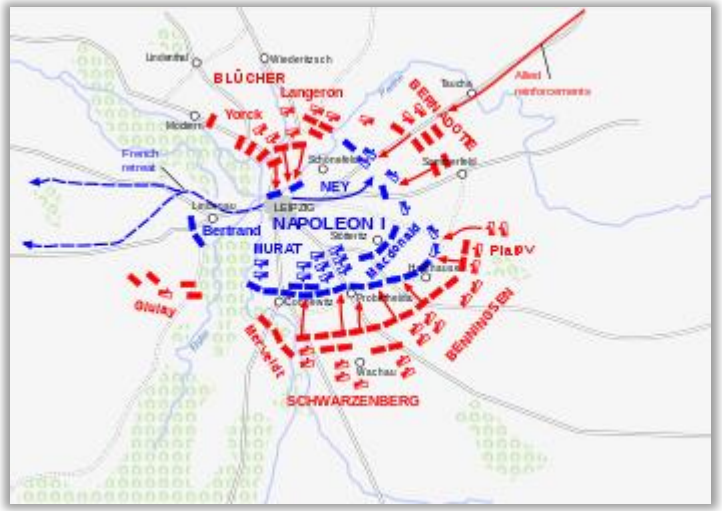
تقسیم و حل

سید ناصر رضوی n.razavi@tabrizu.ac.ir
۱۳۹۵

تقسیم و حل: رهیافت بالا به پایین

- تقسیم و حل. [استراتژی به کار رفته به وسیله‌ی ناپلئون در نبرد لایپزیگ، ۱۸۱۳]
  - یک نمونه را به تعدادی نمونه‌ی کوچک‌تر تقسیم کن.
  - هر یک از نمونه‌های کوچک‌تر را به صورت بازگشتی حل کن.
  - راه‌حل نمونه‌های کوچک‌تر را برای به دست آوردن راه‌حل نمونه‌ی اصلی ترکیب کن.

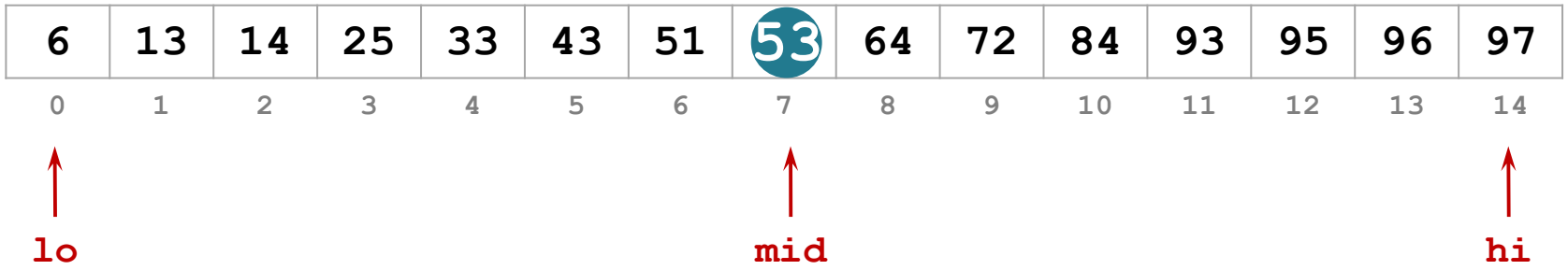


جستجوی دودویی

جستجوی دودویی

- هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

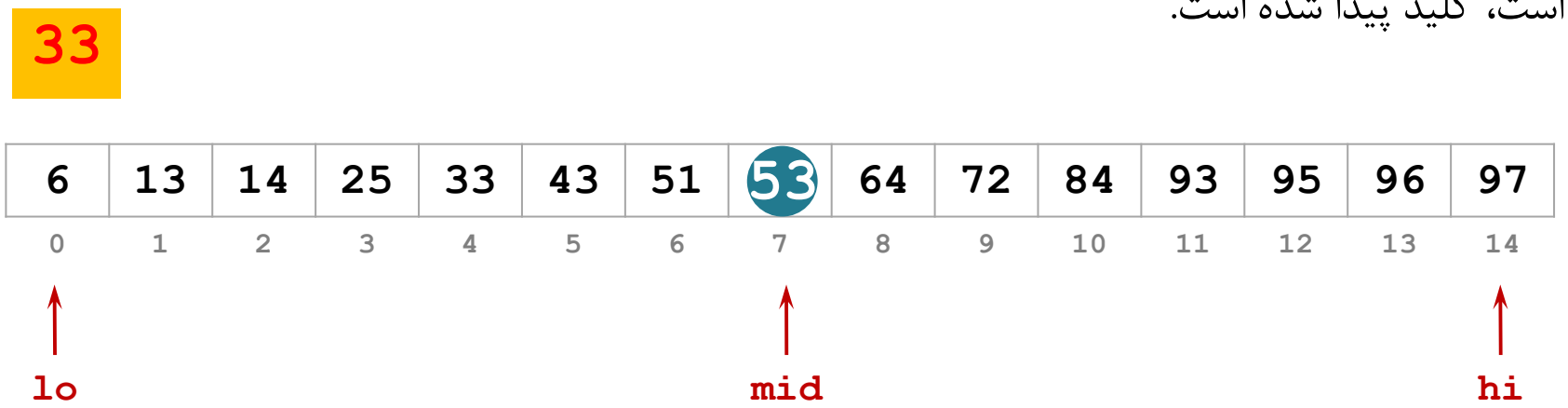
- جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن
  - اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.
  - اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.
  - اگر مساوی است، کلید پیدا شده است.



جستجوی دودویی

- هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

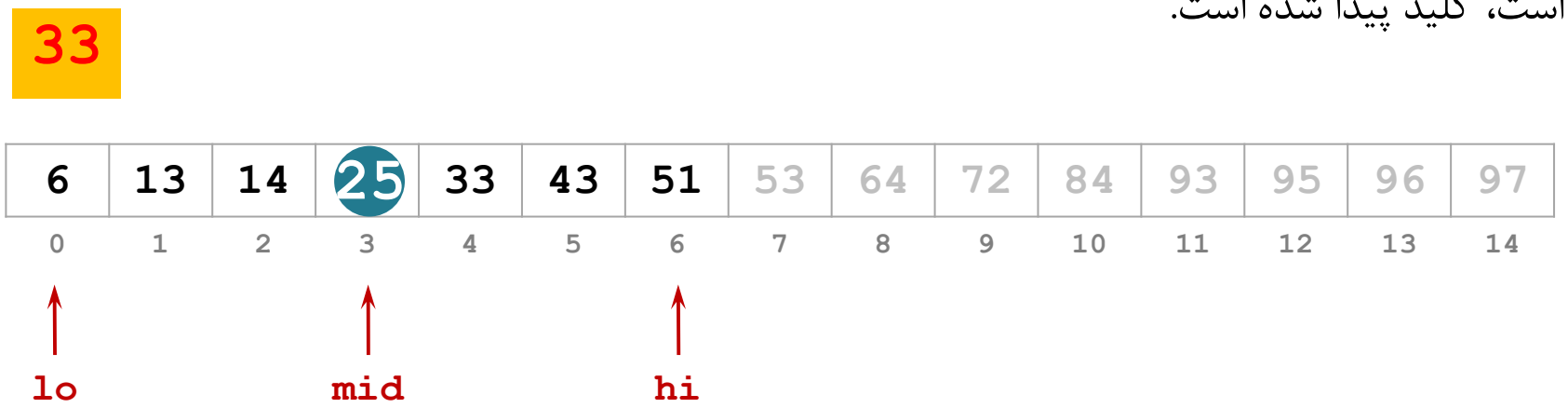
- جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن
  - اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.
  - اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.
  - اگر مساوی است، کلید پیدا شده است.



جستجوی دودویی

- هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

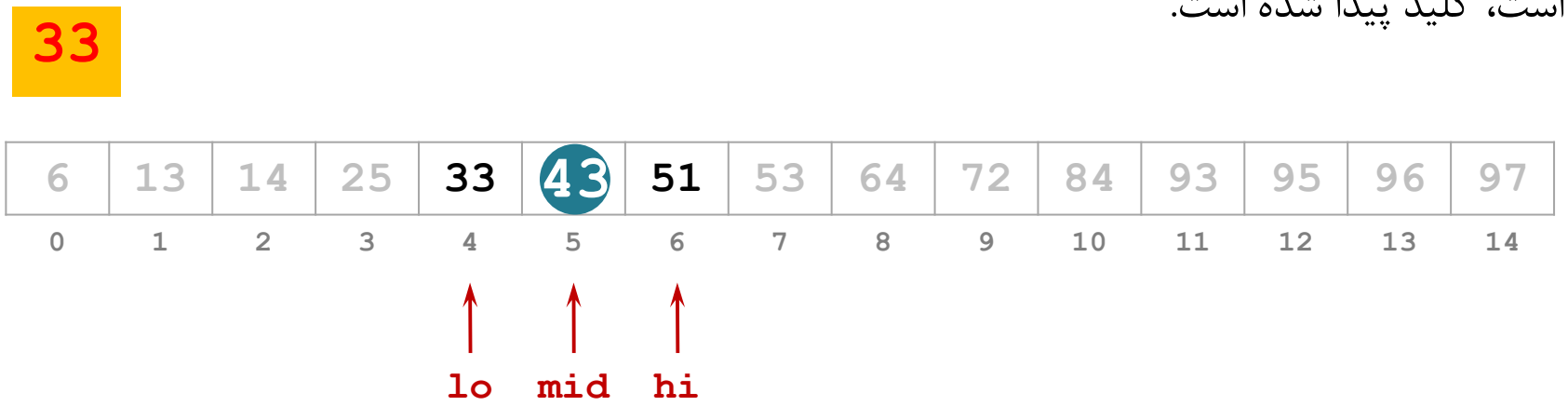
- جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن
  - اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.
  - اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.
  - اگر مساوی است، کلید پیدا شده است.



جستجوی دودویی

- هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

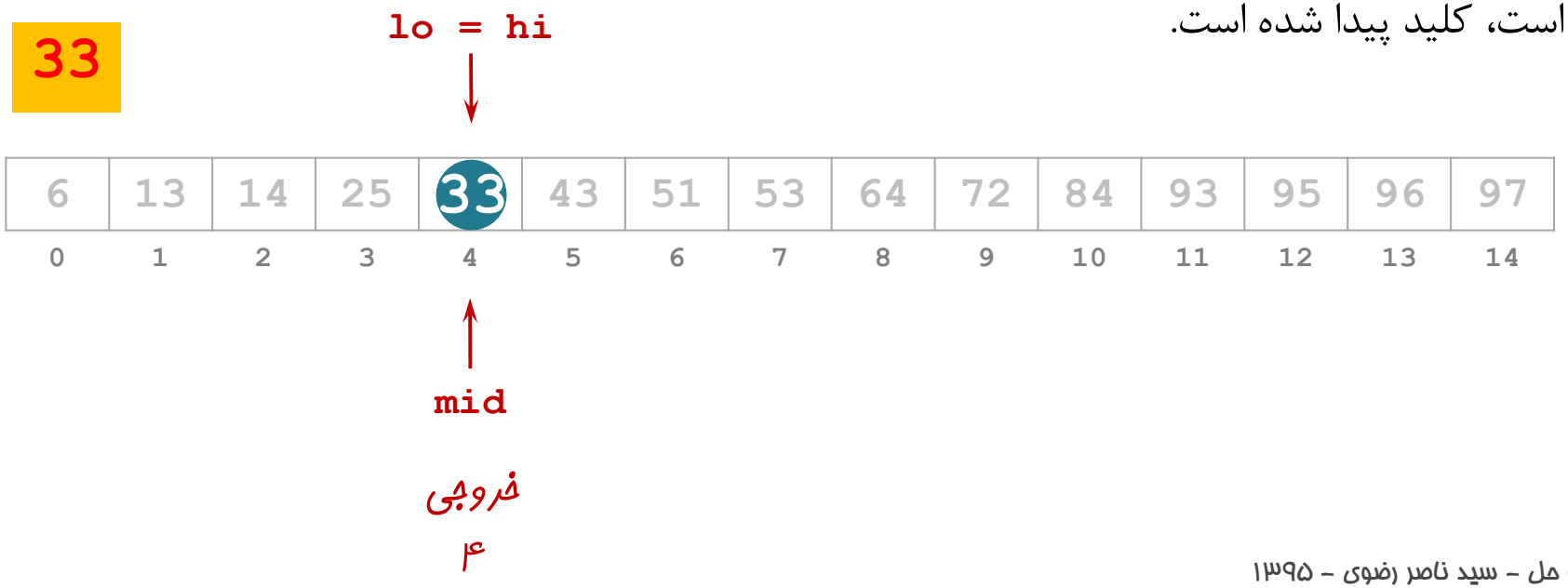
- جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن
  - اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.
  - اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.
  - اگر مساوی است، کلید پیدا شده است.



جستجوی دودویی

- هدف. با داشتن یک آرایه‌ی مرتب و یک کلید، اندیس کلید را در آرایه پیدا کنید.

- جستجوی دودویی. کلید مورد نظر را با کلید وسط مقایسه کن
  - اگر کوچک‌تر است، نیمه‌ی چپ را جستجو کن.
  - اگر بزرگ‌تر است، نیمه‌ی راست را جستجو کن.
  - اگر مساوی است، کلید پیدا شده است.



جستجوی دودویی: پیاده‌سازی در جاوا

- پیاده‌سازی جستجوی دودویی چقدر آسان است؟
  - اولین پیاده‌سازی جستجوی دودویی سال ۱۹۴۶ ؛ اولین پیاده‌سازی بدون خطا سال ۱۹۶۲
  - خطای جاوا در **`Arrays.binarySearch()`** در سال ۲۰۰۶ کشف شد.

```
public static int binarySearch(int[] a, int key)
{
   int lo = 0, hi = a.length - 1;
   while (lo <= hi)
   {
      int mid = lo + (hi - lo) / 2;
      if      (key < a[mid]) hi = mid - 1;
      else if (key > a[mid]) lo = mid + 1;
      else return mid;
   }
   return -1;
}
```

جستجوی دودویی: پیاده‌سازی بازگشتی

- پیاده‌سازی جستجوی دودویی چقدر آسان است؟
  - اولین پیاده‌سازی جستجوی دودویی سال ۱۹۴۶ ؛ اولین پیاده‌سازی بدون خطا سال ۱۹۶۲
  - خطای جاوا در **`Arrays.binarySearch()`** در سال ۲۰۰۶ کشف شد.

```
public static int binarySearch(int[] a, int key, int lo, int hi)
{
    if (hi < lo) return -1;
    int mid = lo + (hi - lo) / 2;
    if (key < a[mid])
        return binarySearch(a, key, lo, mid - 1);
    else if (key > a[mid])
        return binarySearch(a, key, mid + 1, hi);
    else return mid;
}
```

جستجوی دودویی: تحلیل ریاضی

- گزاره. جستجوی دودویی برای جستجو در یک آرایه‌ی مرتب با اندازه‌ی **N**، حداکثر **lg N + 1** مقایسه انجام می‌دهد.

- تعریف.
  - **T(n)** = حداکثر تعداد مقایسه‌های جستجوی دودویی در یک زیرآرایه‌ی مرتب با اندازه‌ی **N**

- رابطه‌ی بازگشتی مربوط به جستجوی دودویی.

$$T(N) \le T(N/2) + 1,\ T(1) = 1$$

- اثبات.

$$\begin{aligned} T(N) &\le T(N/2) + 1 \\ &\le T(N/4) + 1 + 1 \\ &\le T(N/8) + 1 + 1 + 1 \\ &\ldots \\ &\le T(N/N) + 1 + 1 + \ldots + 1 \\ &= 1 + \lg N \end{aligned}$$

مرتب‌سازی ادغامی

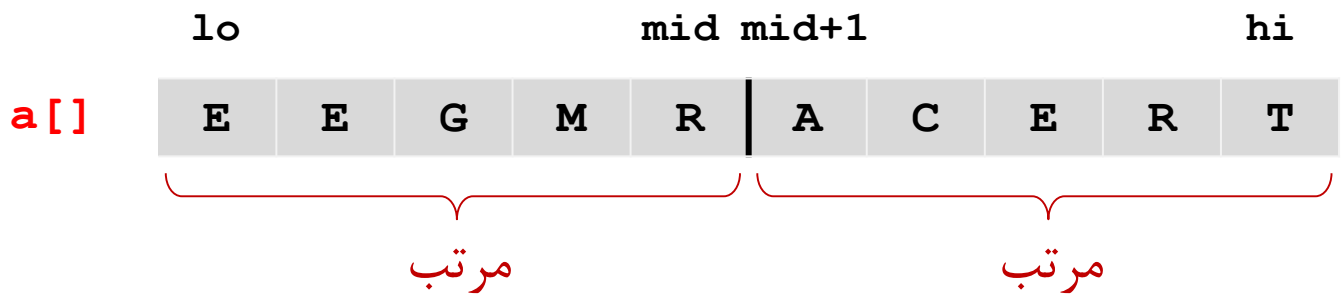
مرتب‌سازی ادغامی

- 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
مرتب‌سازی نیمه اول	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
مرتب‌سازی 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

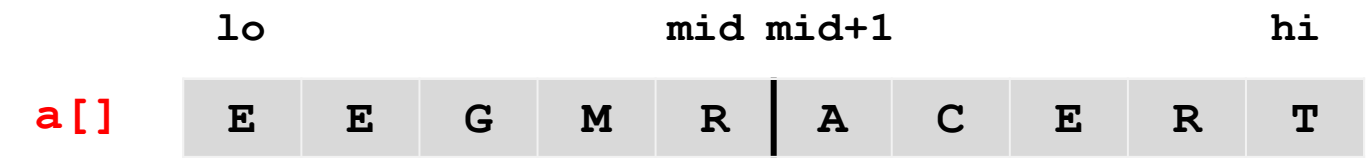
ادغام

- 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

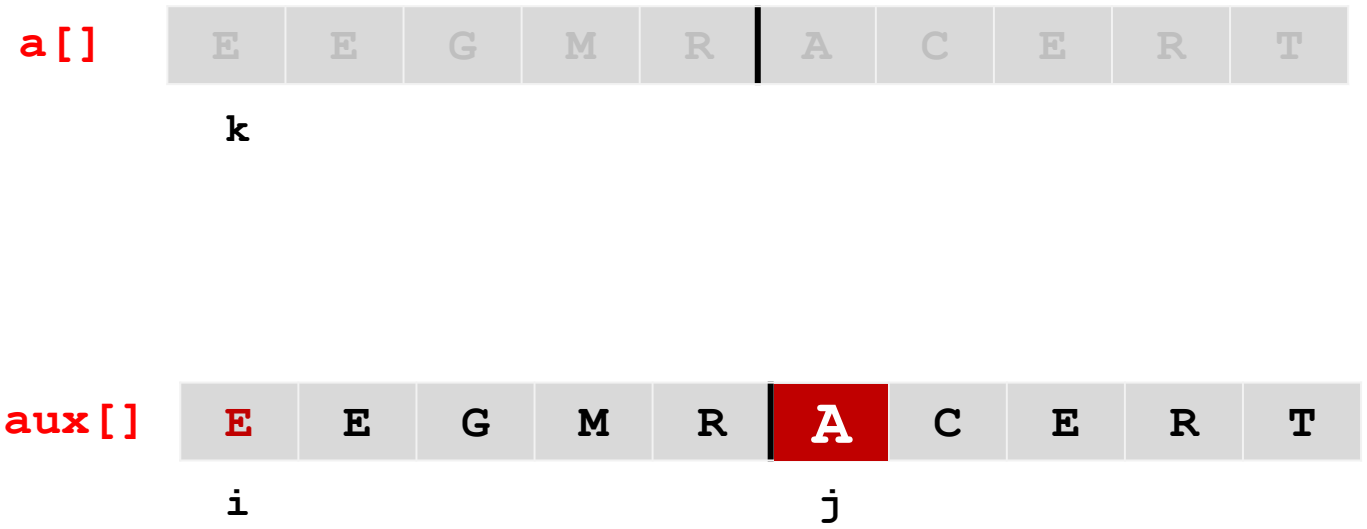
- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.



کپی کردن عناصر در آرایه کمکی

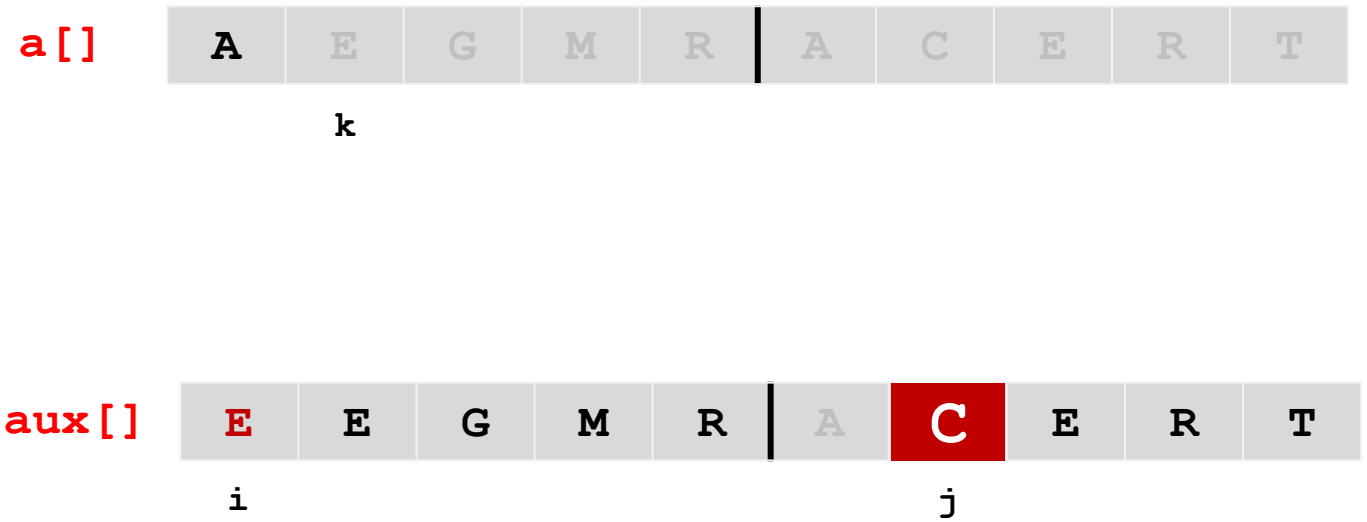
ادغام

- 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

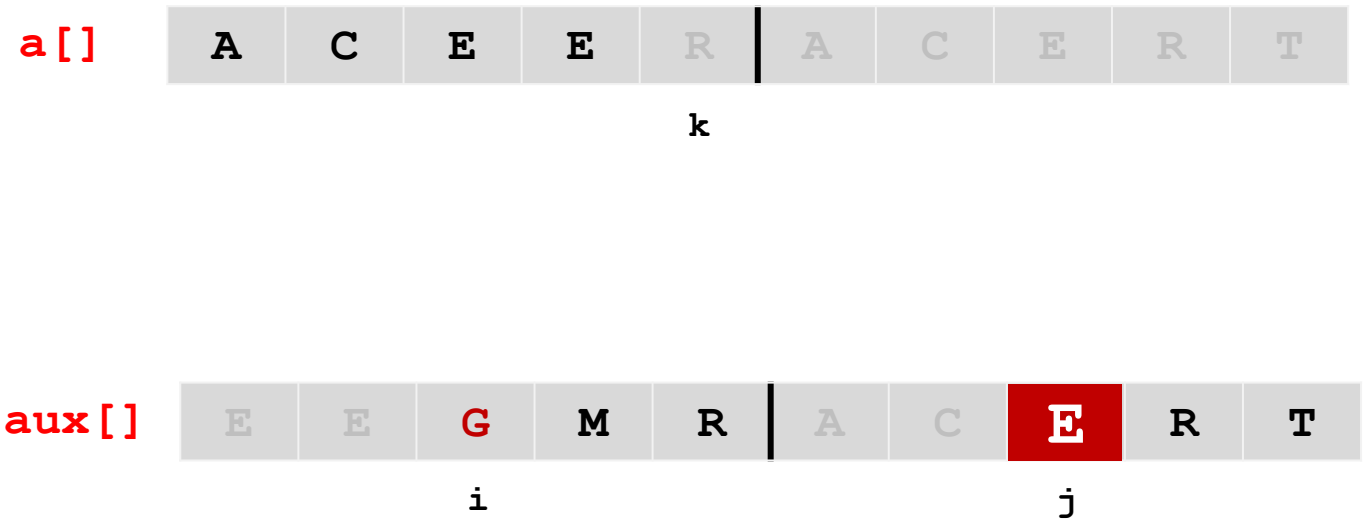
- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.

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.

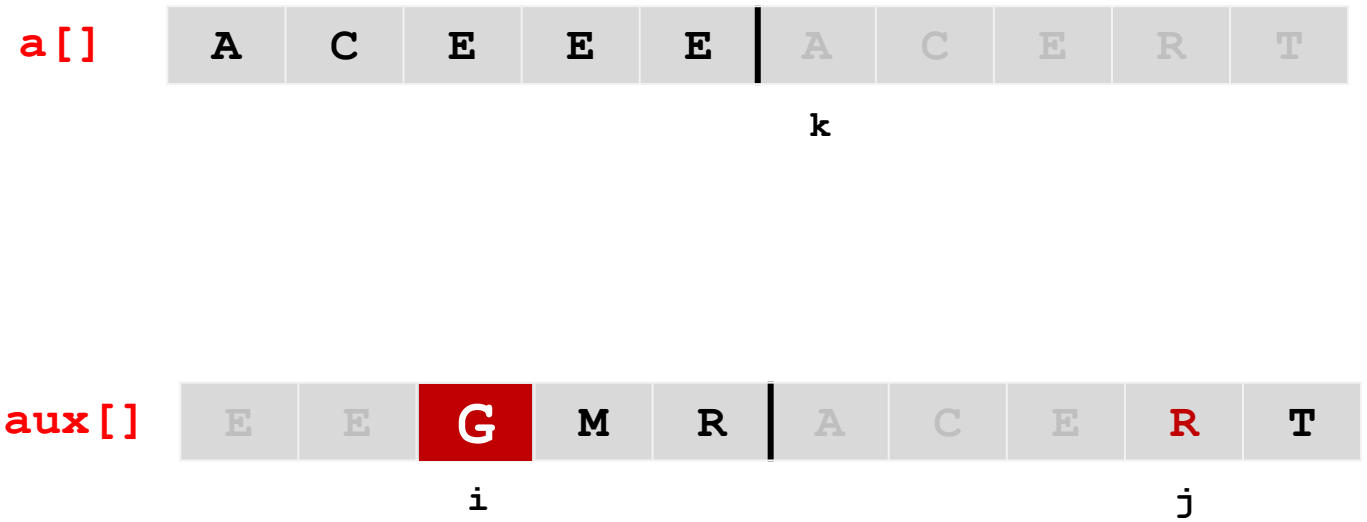
ادغام

- 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.



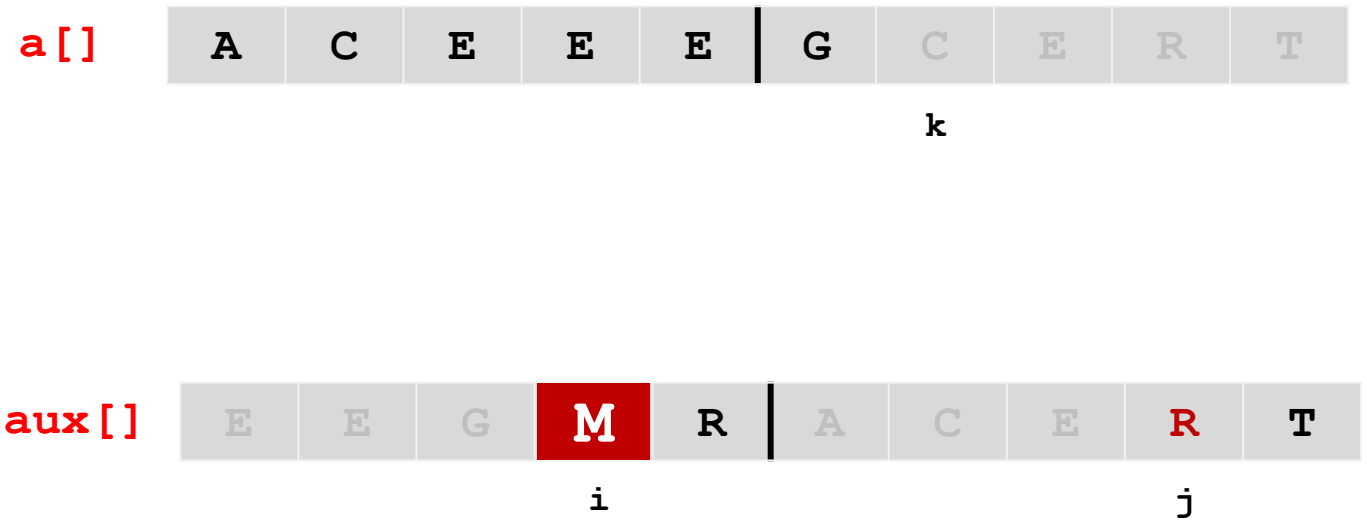
ادغام

- 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.



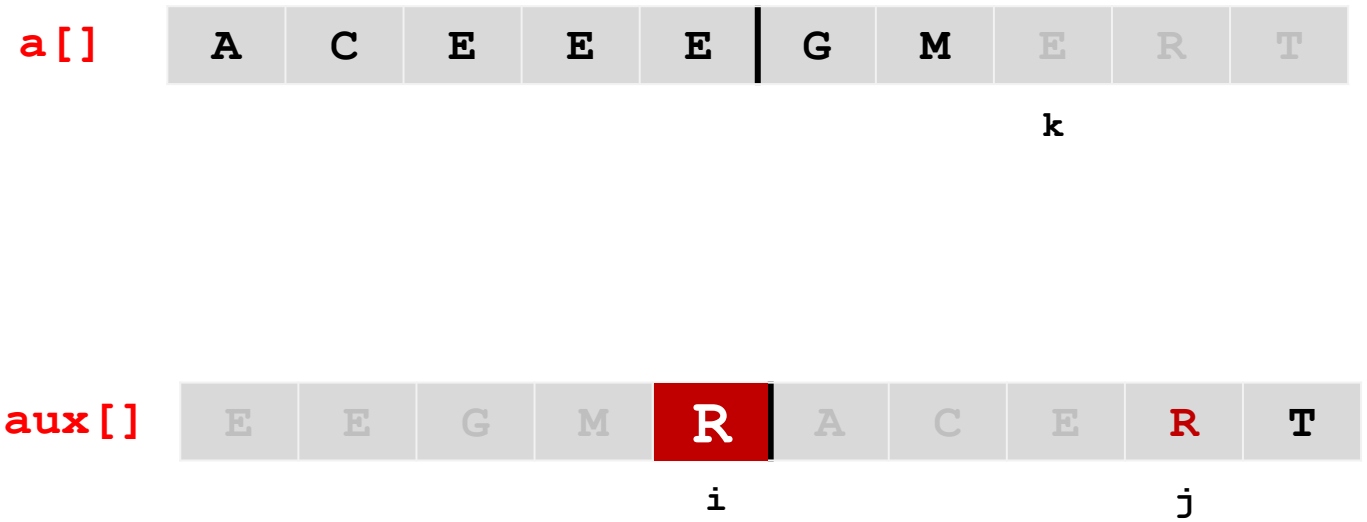
ادغام

- 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

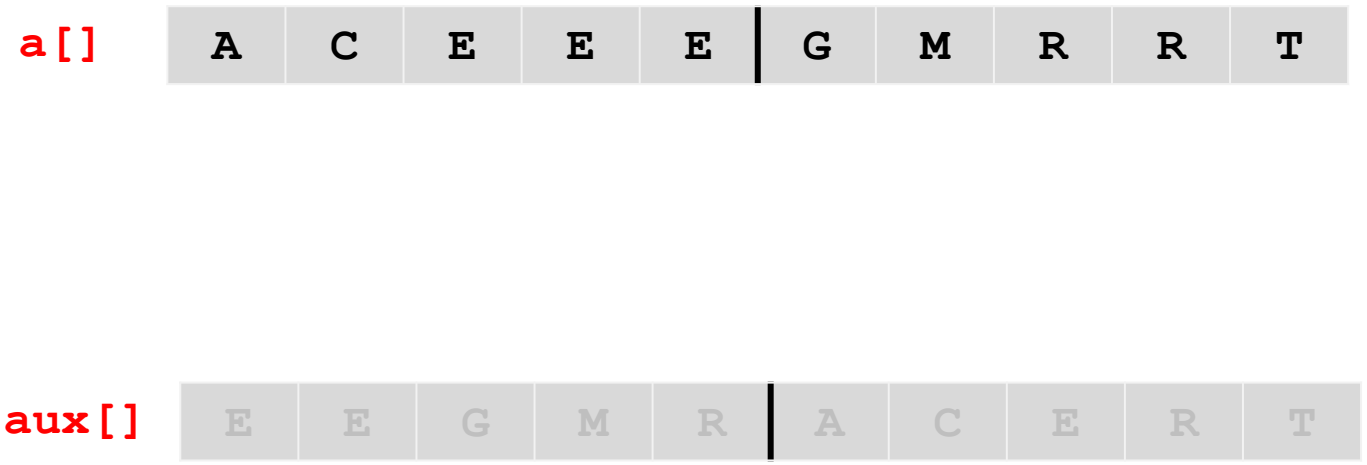
- هدف. با داشتن دو زیرآرایه‌ی مرتب یکی از **a[lo]** تا **a[mid]** و دیگری از **a[mid+1]** تا **a[hi]**، زیرآرایه‌ی **a[lo]** تا **a[hi]** را مرتب کن.

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 27 */  }

  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);
  }
}
```

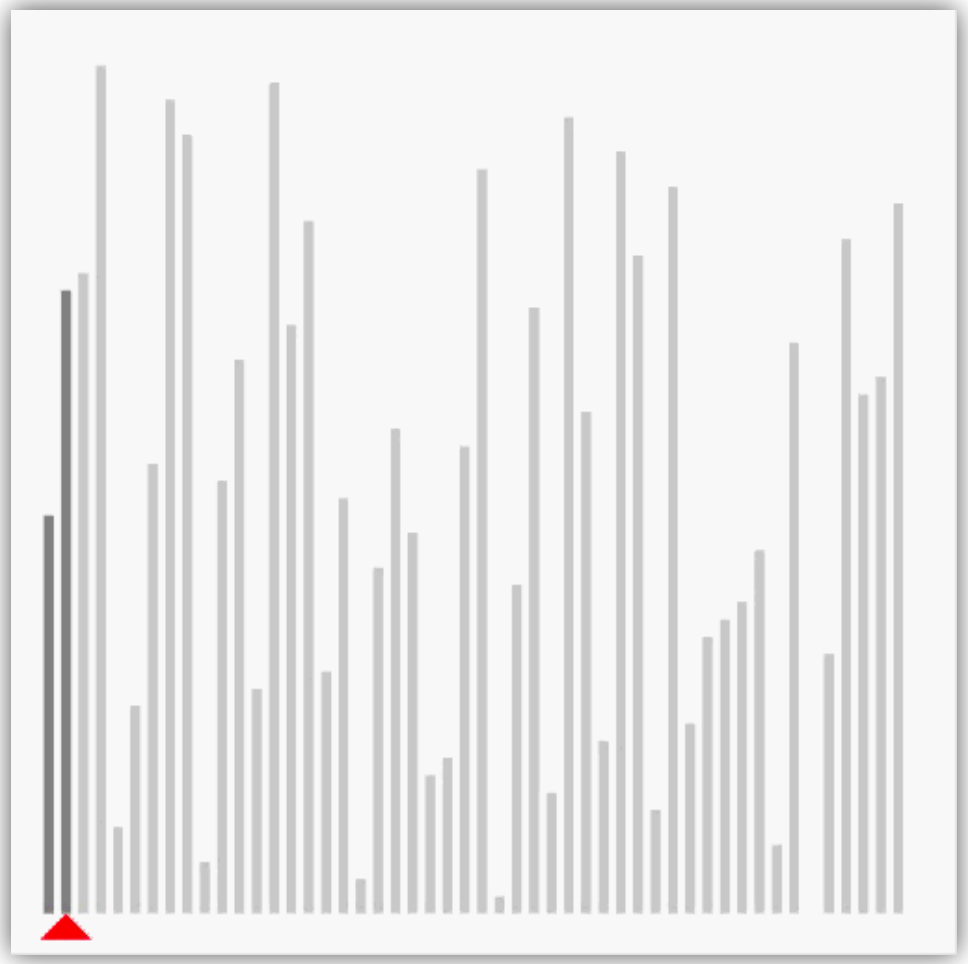
مرتب‌سازی ادغامی: ردیابی اجرا

	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)

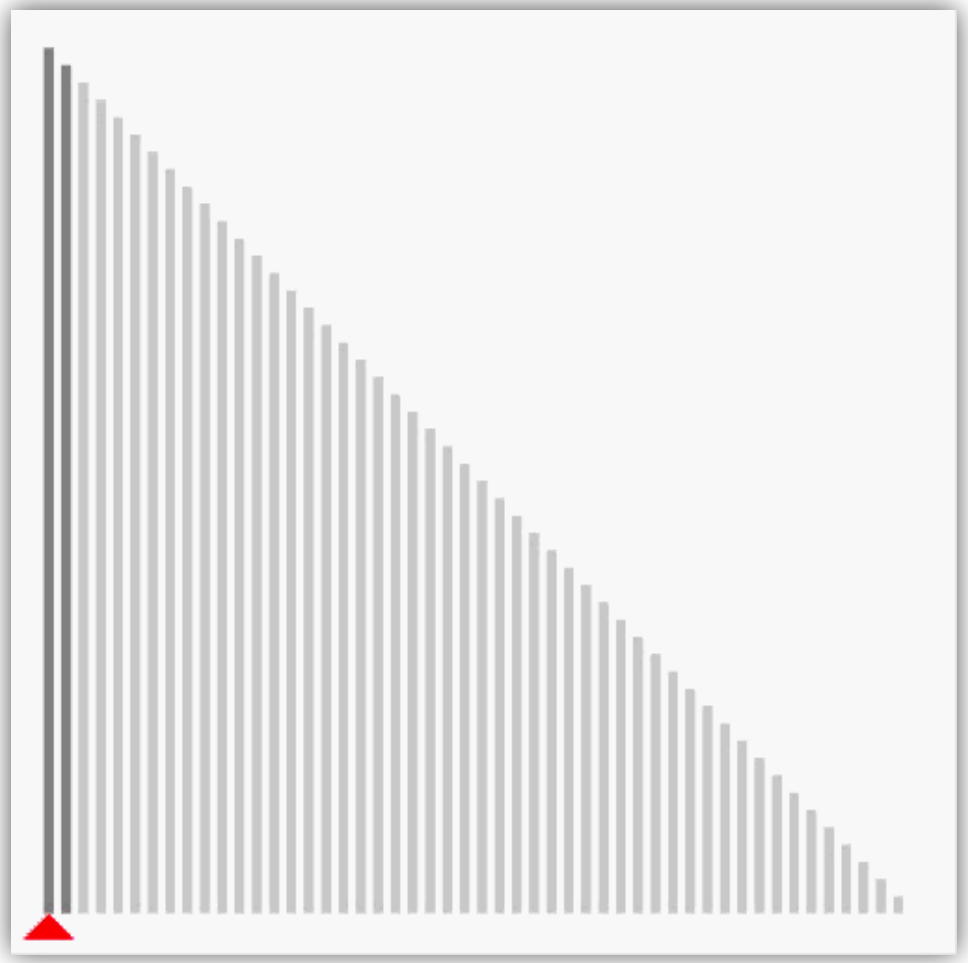
- مرتب‌سازی ۵۰ عنصر تصادفی.



```
% java Merge 50
```

مرتب‌سازی ادغامی (اجرای نمایشی)

- مرتب‌سازی ۵۰ عنصر که به صورت نزولی مرتب‌اند.



```
% 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!

مرتب‌سازی ادغامی: تحلیل زمان

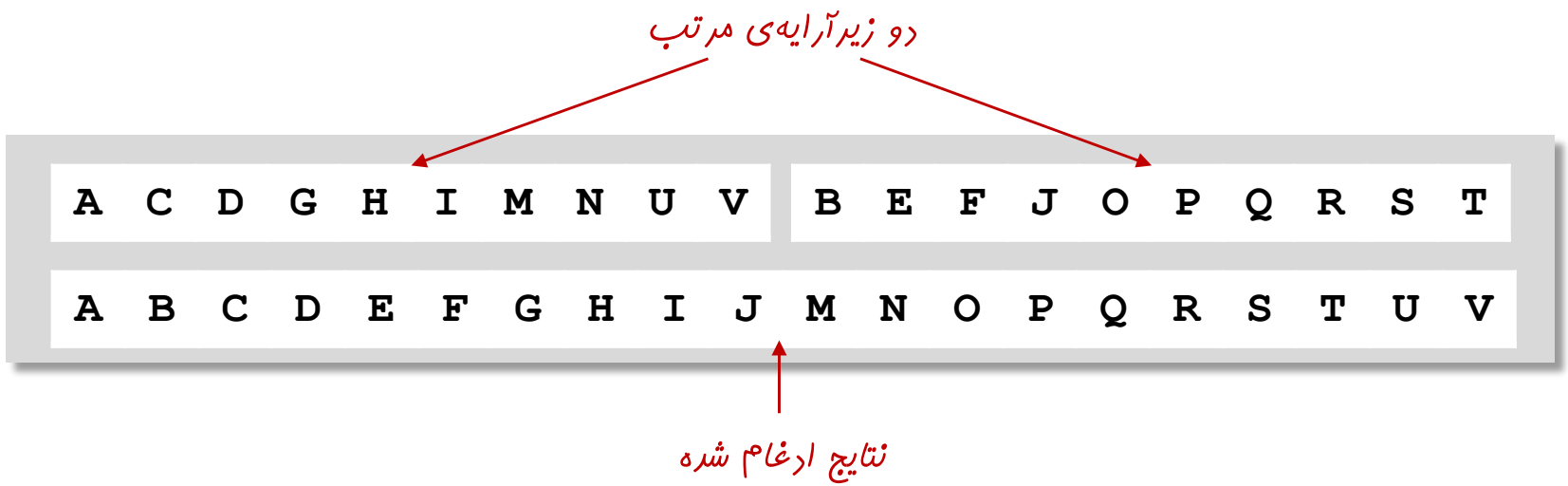
- گزاره. تعداد مقایسه‌های لازم توسط مرتب‌سازی ادغامی برای مرتب کردن یک آرایه‌ی N عنصری در بدترین حالت برابر است با $\sim N \lg N$.

- اثبات. [فرض می‌کنیم N توانی از دو باشد]
  - ادغام: N - 1 مقایسه
  - مرتب‌سازی هر یک از دو نیمه: T(N/2)

$$T(N) = 2T(N/2) + N - 1 \Rightarrow T(N) \sim N \lg N$$

مرتب‌سازی ادغامی: تحلیل حافظه

- گزاره. مقدار حافظه‌ی کمکی در مرتب‌سازی ادغامی متناسب با N است.
- اثبات. اندازه‌ی آرایه‌ی aux[] در آخرین ادغام برابر با N است.



- تعریف. یک الگوریتم مرتب‌سازی درجا است اگر مقدار حافظه‌ی کمکی آن $\leq c \lg N$ باشد.
  - مرتب‌سازی درجی و انتخابی درجا هستند؛ اما مرتب‌سازی ادغامی یک الگوریتم غیردرجا است.

مرتب‌سازی ادغامی: بهبود عملی

- استفاده از مرتب‌سازی درجی برای زیرآرایه‌های کوچک.
  - سربار مرتب‌سازی ادغامی برای آرایه‌های کوچک زیاد است.
  - استفاده از مرتب‌سازی درجی برای آرایه‌های ۷ عنصری و کمتر.

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

مرتب‌سازی ادغامی: بهبود عملی

- توقف مرتب‌سازی بر روی آرایه‌های مرتب.
  - بزرگ‌ترین عنصر نیمه‌ی چپ از کوچک‌ترین عنصر نیمه‌ی راست کوچک‌تر است.
  - بهبود زمان اجرا برای آرایه‌های مرتب و تقریباً مرتب.

A	B	C	D	E	F	G	H	I	J	M	N	O	P	Q	R	S	T	U	V

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

مرتب‌سازی ادغامی: بهبود عملی

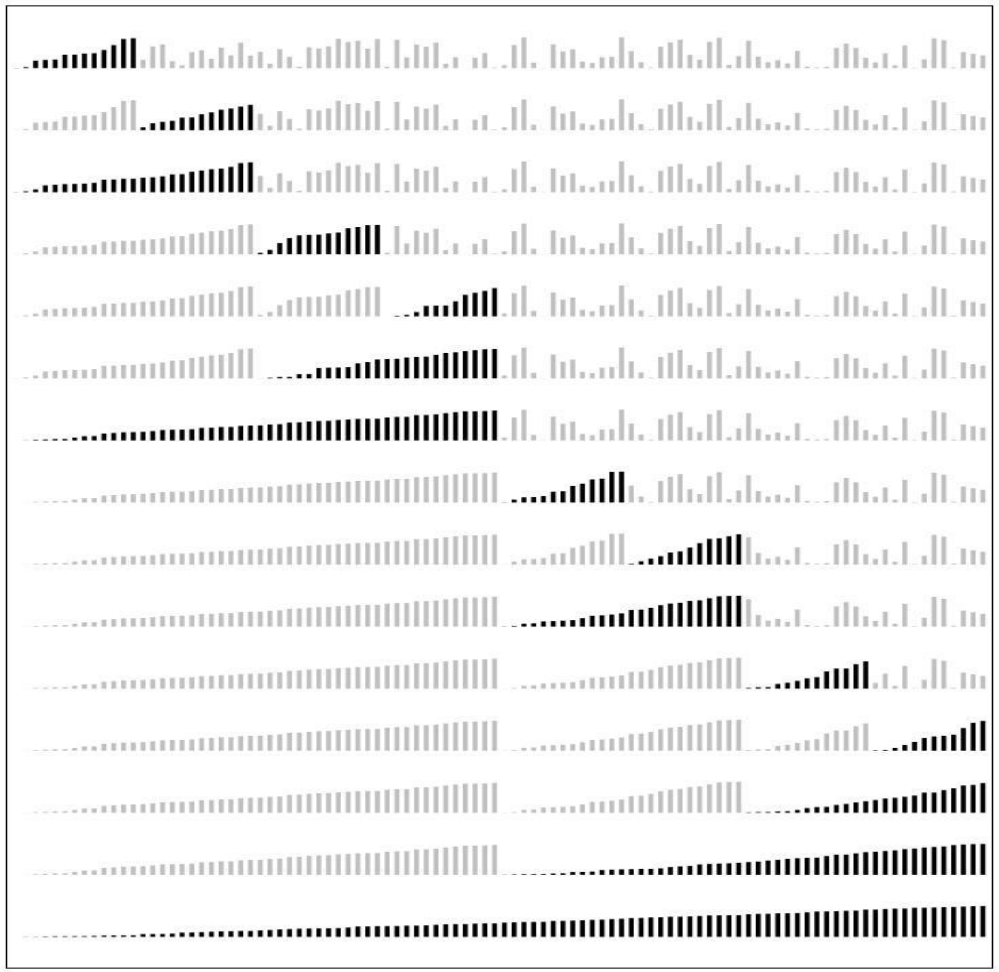
- حذف مرحله‌ی کپی کردن عناصر در آرایه‌ی کمکی.
  - صرفه‌جویی در زمان با تعویض نقش آرایه‌ی ورودی و کمکی در هر فراخوانی بازگشتی.

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

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

ادغام از a[] به aux[]

تعویض نقش a[] و aux[]

مرتب‌سازی ادغامی



% java MergeBars 1000 100

اولین زیرآرایه

دومین زیرآرایه

اولین ادغام

مرتب‌سازی نیمه‌ی اول

مرتب‌سازی نیمه‌ی دوم

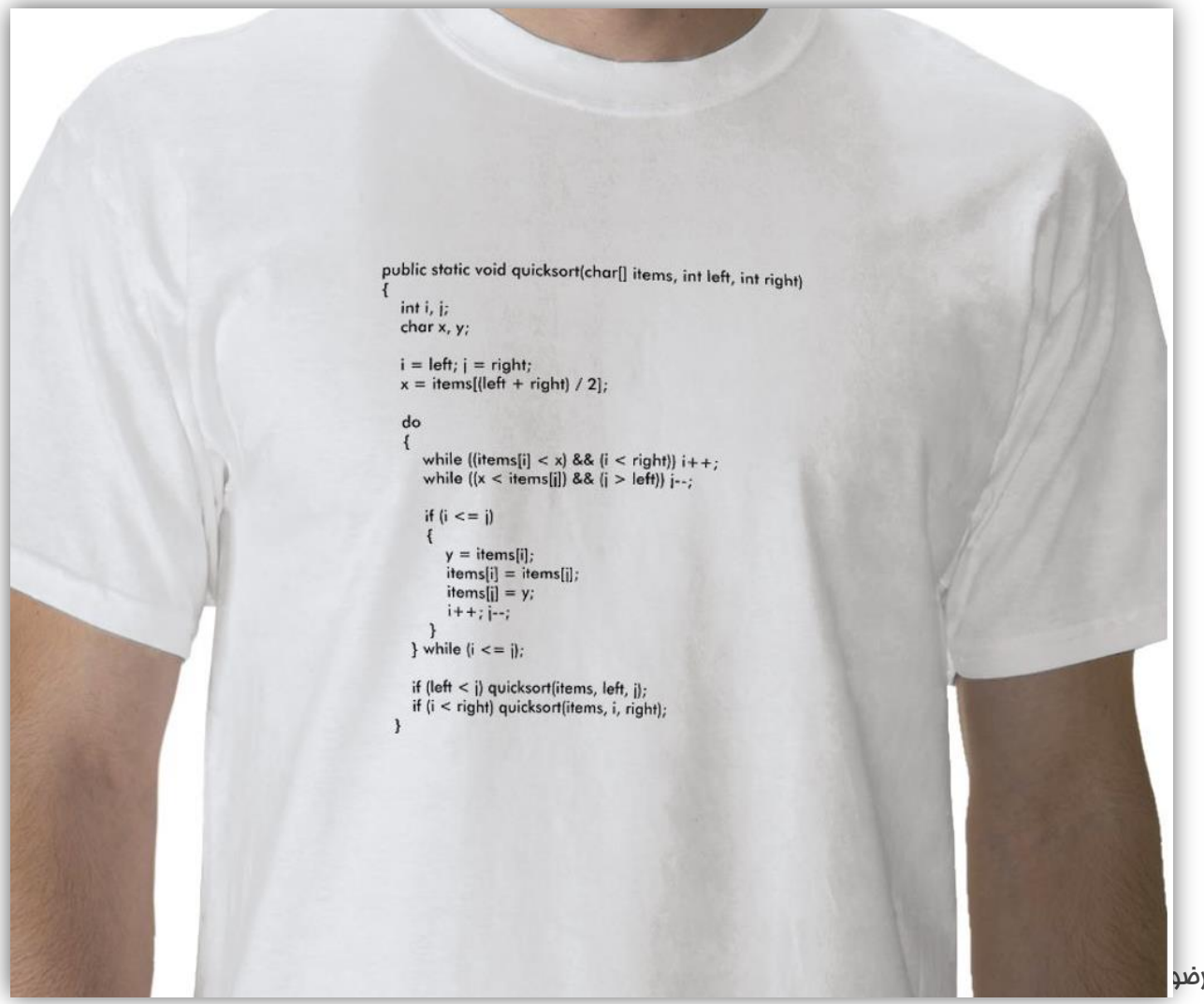
نتیجه

مرتب‌سازی سریع

مرتب‌سازی سریع

- یک مؤلفه‌ی حیاتی در سیستم‌های محاسباتی.
  - یکی از ۱۰ الگوریتم برتر قرن بیستم در علوم و مهندسی
- مرتب‌سازی ادغامی.
  - مرتب‌سازی اشیا در جاوا
  - روش مرتب‌سازی در Perl، C++، Python ، فایرفاکس و ...
- مرتب‌سازی سریع.
  - مرتب‌سازی انواع اولیه در جاوا
  - روش مرتب‌سازی در C، Visual C++، Python، Matlab، گوگل کروم و ...

تی‌شرت مرتب‌سازی سریع



مرتب‌سازی سریع

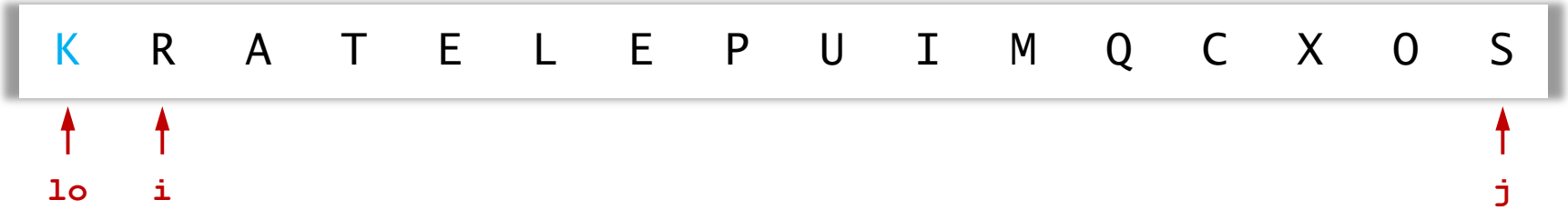
آنتونی ریچارد هوآر
جایزه تورینگ ۱۹۸۰

- طرح اصلی. ابتدا آرایه را بر بزن.
  - افراز آرایه، به طوری که به ازای یک اندیس مانند j
    - عنصر a[j] در مکان درست قرار گرفته باشد
    - در سمت چپ a[j] هیچ عنصری بزرگ‌تر از آن وجود نداشته باشد
    - در سمت راست a[j] هیچ عنصری کوچک‌تر از آن وجود نداشته باشد
  - مرتب‌سازی عناصر سمت چپ و سمت راست a[j] به صورت بازگشتی

input	Q	U	I	C	K	S	O	R	T	E	X	A	M	P	L	E
shuffle	K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
partition	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
	کوچک‌تر یا مساوی						بزرگ‌تر یا مساوی									
sort left	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
sort right	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
result	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X

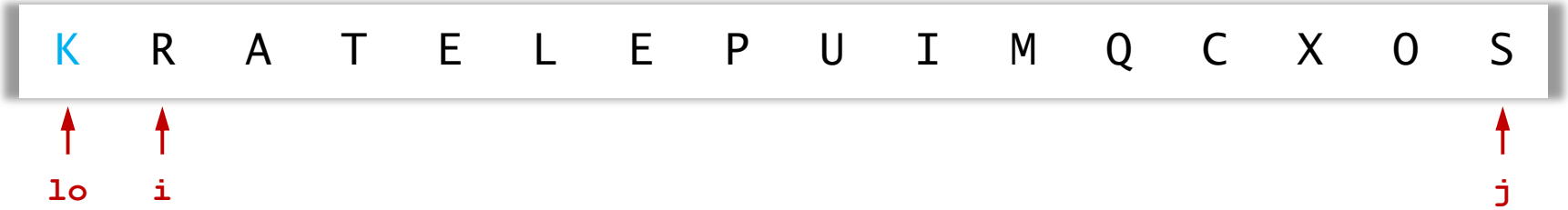
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



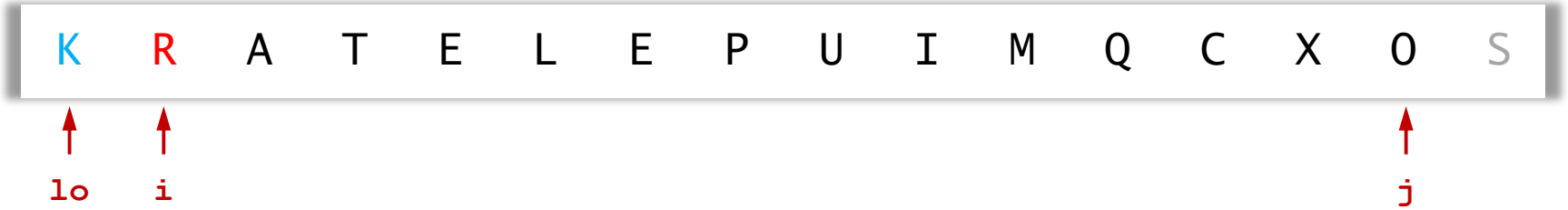
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



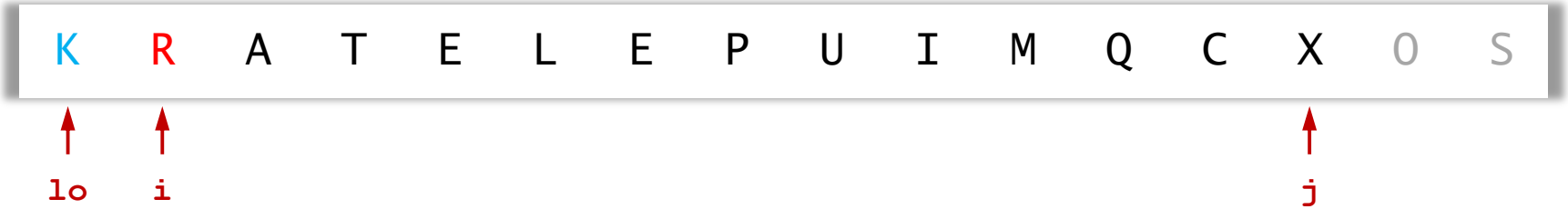
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



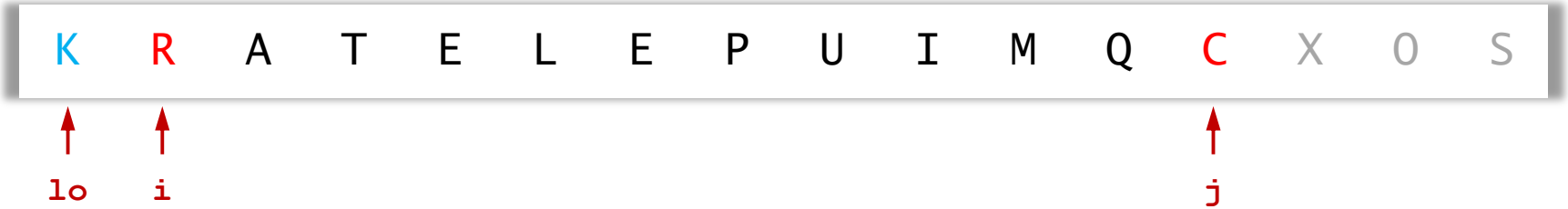
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



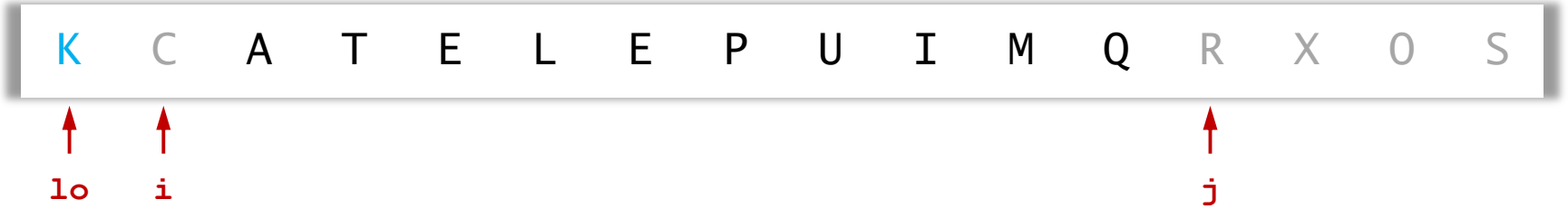
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



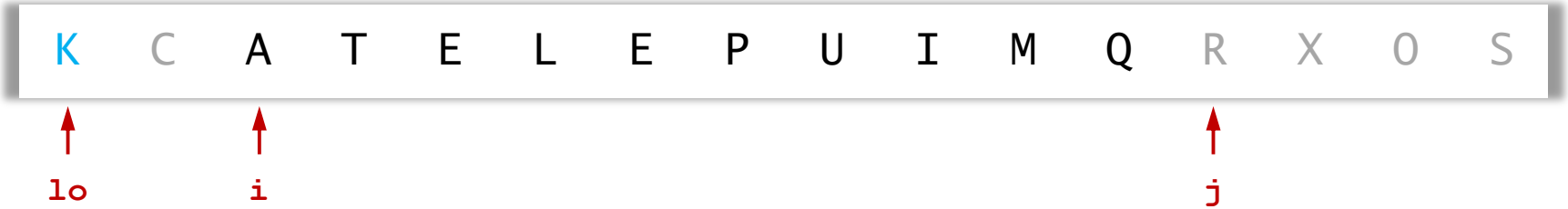
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



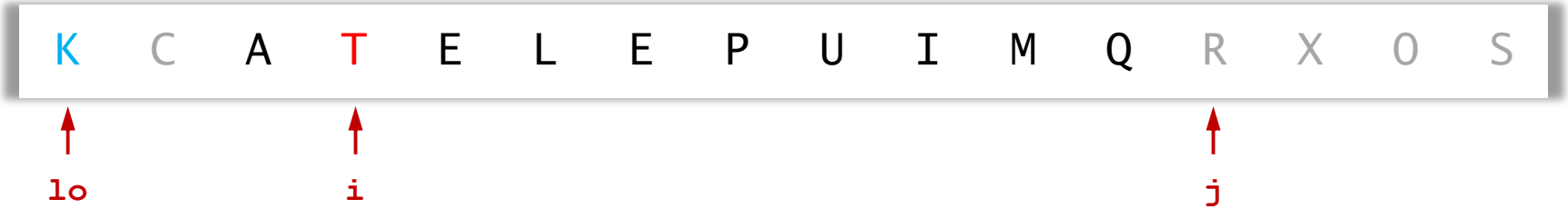
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo] است**، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



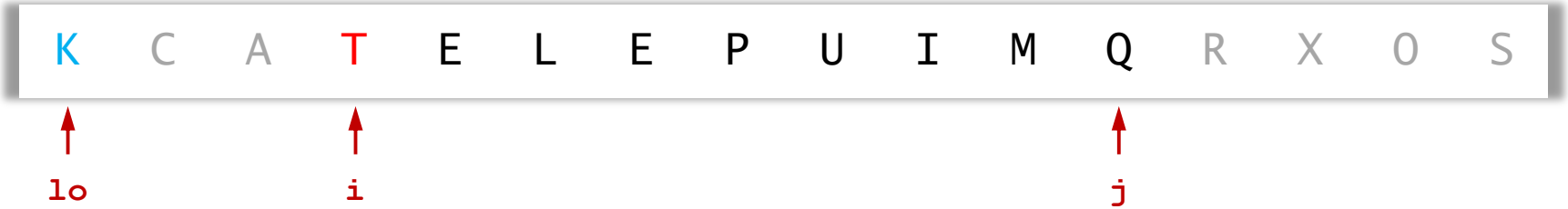
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



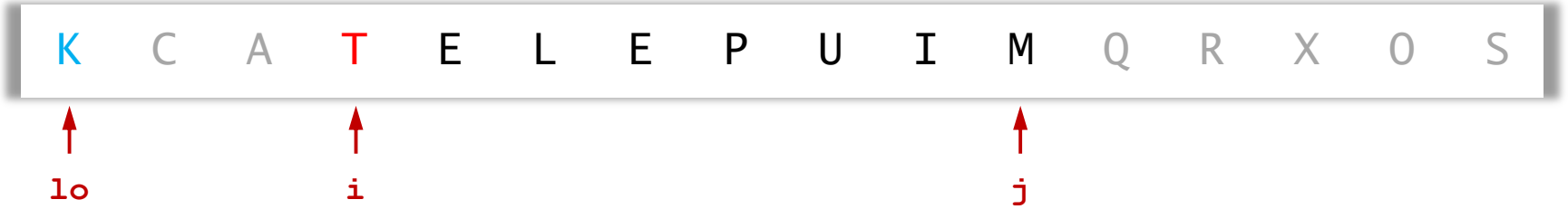
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



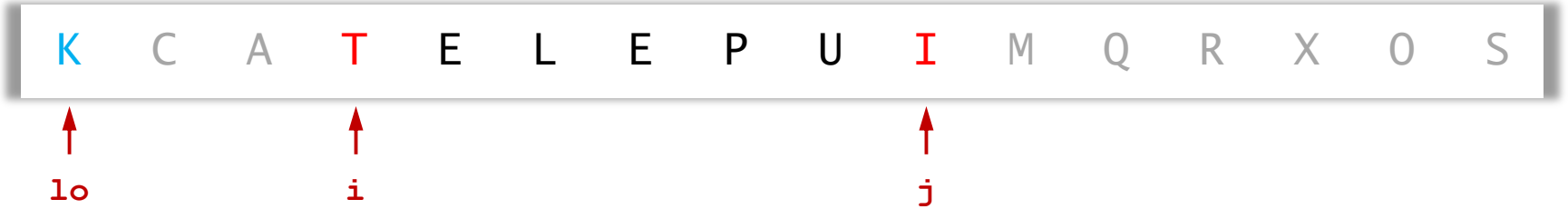
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



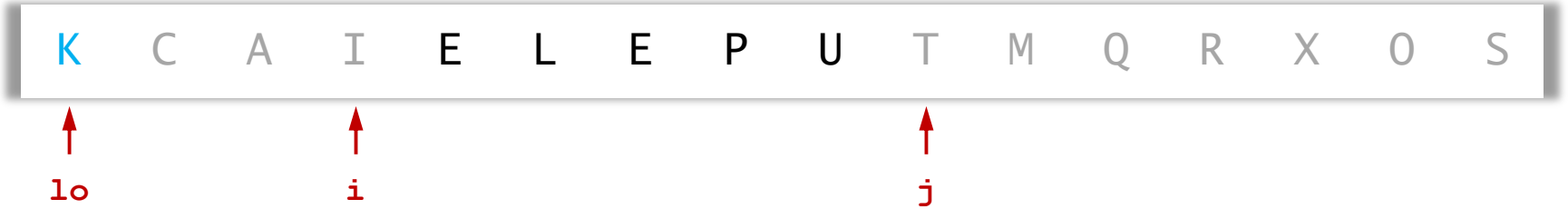
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



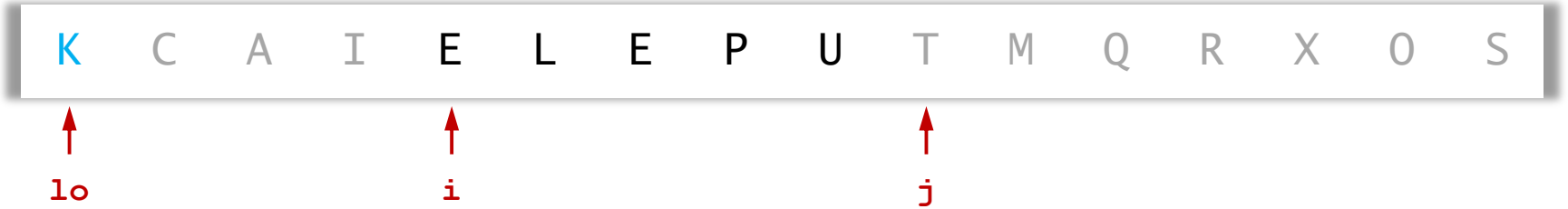
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



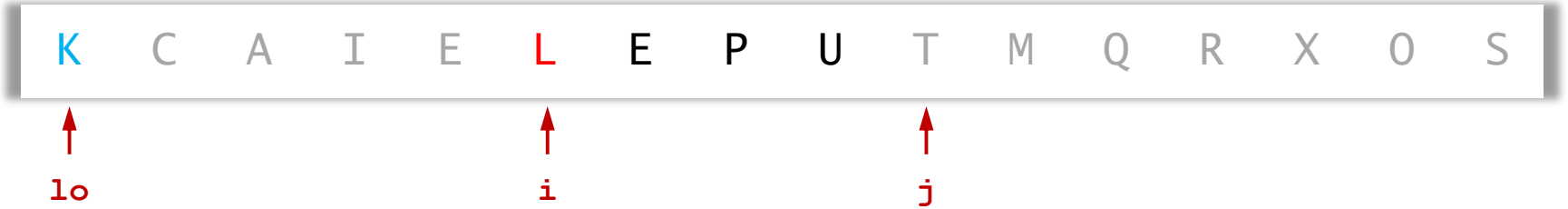
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.



افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.

افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.

افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.

افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.

افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.

افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.

افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.
- فاز ۲. جابجایی **a[lo]** و **a[j]**

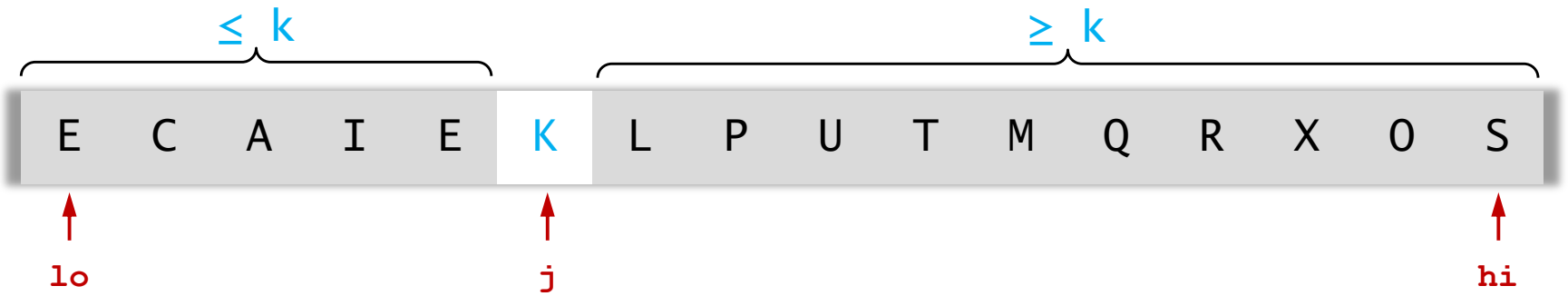
افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.
- فاز ۲. جابجایی **a[lo]** و **a[j]**

افراز در مرتب‌سازی سریع

- فاز ۱. تا زمانی که اندیس‌های **i** و **j** از یکدیگر عبور نکرده‌اند، مراحل زیر را تکرار کن:
  - تا زمانی که **a[i] < a[lo]** است، اندیس **i** را به سمت راست حرکت بده.
  - تا زمانی که **a[j] > a[lo]** است، اندیس **j** را به سمت چپ حرکت بده.
  - عناصر **a[i]** و **a[j]** را با یکدیگر تعویض کن.

- فاز ۲. جابجایی **a[lo]** و **a[j]**



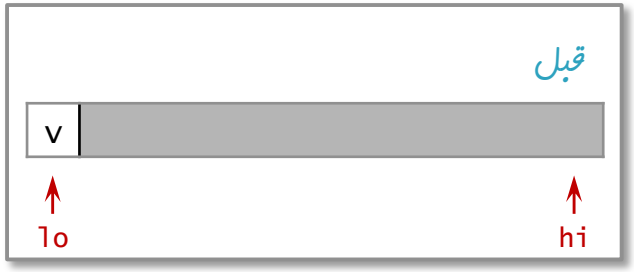
افراز: ردیابی اجرا

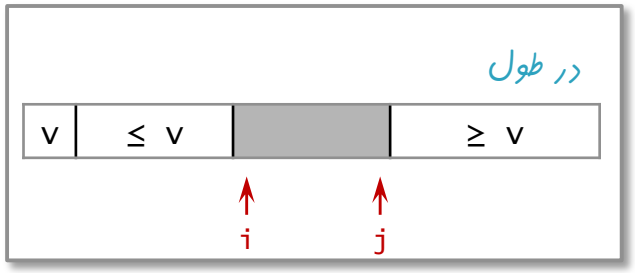
		a[ ]															
i	j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
1	12	K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
1	12	K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S
3	9	K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S
3	9	K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S
5	6	K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S
5	6	K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S
0	5	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
	5	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S

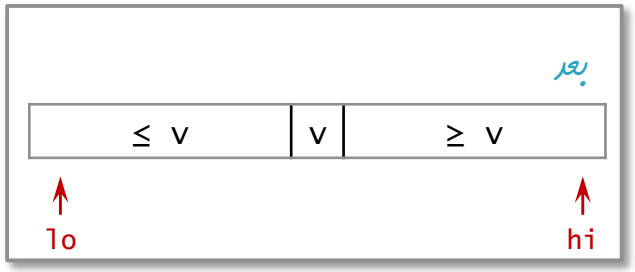
```
% java TracePartition KRATELEPUIMQCXOS
```

مرتب‌سازی سریع: کد جاوا برای افراز


```
private static int partition(Comparable[] a, int lo, int hi)
{
   int i = lo, j = hi + 1;
   while (true)
   {
      while (less(a[++i], a[lo]))
          if (i == hi) break;

      while (less(a[lo], a[--j]))
          if (j == lo) break;

      if (i >= j) break;
      exch(a, i, j);
   }

   exch(a, lo, j);
   return j;
}
```







مرتب‌سازی سریع: پیاده‌سازی در جاوا


```
public class Quick {

   private static int partition(Comparable[] a, int lo, int hi)
   {  /* see previous slide */  }

   public static void sort(Comparable[] a) {
      StdRandom.shuffle(a);
      sort(a, 0, a.length - 1);
   }

   private static void sort(Comparable[] a, int lo, int hi) {
      if (hi <= lo) return;
      int j = partition(a, lo, hi);
      sort(a, lo, j - 1);
      sort(a, j + 1, hi);
   }
}
```

بر زدن آرایه به منظور
تضمین کارایی ضروری است!

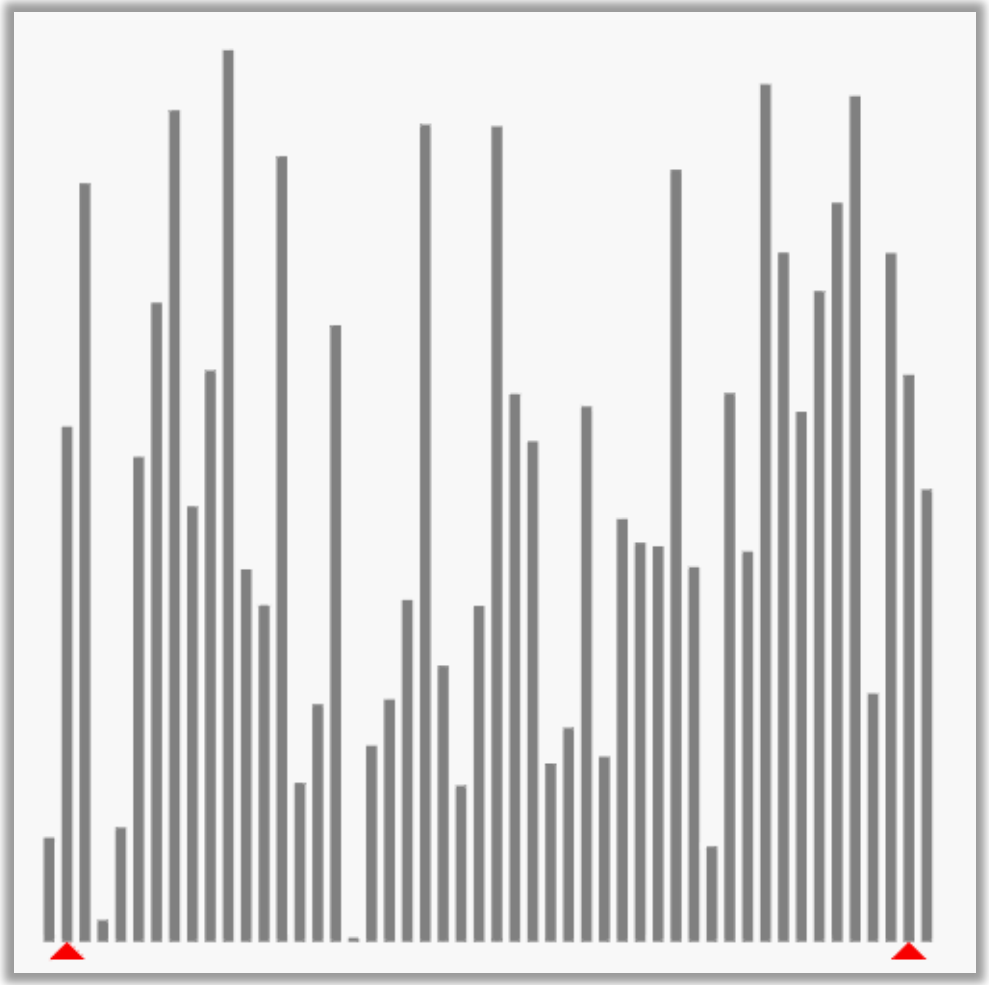
مرتب‌سازی سریع: ردیابی اجرا

lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			Q	U	I	C	K	S	O	R	T	E	X	A	M	P	L	E
			K	M	C	L	X	S	A	U	P	E	T	I	Q	E	O	R
0	5	15	A	E	C	I	E	K	S	U	P	X	T	L	Q	M	O	R
0	0	4	A	E	C	I	E	K	S	U	P	X	T	L	Q	M	O	R
1	3	4	A	E	C	E	I	K	S	U	P	X	T	L	Q	M	O	R
1	2	2	A	C	E	E	I	K	S	U	P	X	T	L	Q	M	O	R
1		1	A	C	E	E	I	K	S	U	P	X	T	L	Q	M	O	R
4		4	A	C	E	E	I	K	S	U	P	X	T	L	Q	M	O	R
6	12	15	A	C	E	E	I	K	Q	R	P	O	M	L	S	T	X	U
6	10	11	A	C	E	E	I	K	M	L	P	O	Q	R	S	T	X	U
6	7	9	A	C	E	E	I	K	L	M	P	O	Q	R	S	T	X	U
6		6	A	C	E	E	I	K	L	M	P	O	Q	R	S	T	X	U
8	9	9	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	X	U
8		8	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	X	U
11		11	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	X	U
13	13	15	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	X	U
14	15	15	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
14		14	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
			A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X

```
% java TraceQuick QUICKSORTEXAMPLE
```

مرتب‌سازی سریع: اجرای نمایشی

- مرتب‌سازی ۵۰ عنصر تصادفی.



مرتب‌سازی سریع: 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 ($N \lg N$)			مرتب‌سازی سریع ($N \lg N$)		
	هزار	میلیون	میلیارد	هزار	میلیون	میلیارد	هزار	میلیون	میلیارد
لپ‌تاپ	فوراً	۲/۸ ساعت	۳۱۷ سال	فوراً	۱ ثانیه	۱۸ دقیقه	فوراً	۰/۶ ثانیه	۱۲ دقیقه
ابرکامپیوتر	فوراً	۱ ثانیه	۱ هفته	فوراً	فوراً	فوراً	فوراً	فوراً	فوراً

- نتیجه‌گیری ۱. یک الگوریتم خوب بیشتر از یک ابرکامپیوتر ارزش دارد!
- نتیجه‌گیری ۲. یک الگوریتم عالی بهتر از یک الگوریتم خوب است!

مرتب‌سازی سریع: تحلیل بهترین حالت

- بهترین حالت.
  - تعداد مقایسه‌ها ~N lg N

			a[ ]														
lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			H	A	C	B	F	E	G	D	L	I	K	J	N	M	O
0	7	14	D	A	C	B	F	E	G	H	L	I	K	J	N	M	O
0	3	6	B	A	C	D	F	E	G	H	L	I	K	J	N	M	O
0	1	2	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O
0		0	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O
2		2	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O
4	5	6	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O
4		4	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O
6		6	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O
8	11	14	A	B	C	D	E	F	G	H	J	I	K	L	N	M	O
8	9	10	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O
8		8	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O
10		10	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O
12	13	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
12		12	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
14		14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

مرتب‌سازی سریع: تحلیل بدترین حالت

- بدترین حالت.
  - تعداد مقایسه‌ها $\sim 1/2\ N^2$

			a[ ]														
lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0	0	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	1	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
2	2	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
3	3	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
4	4	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	5	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
6	6	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
7	7	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
8	8	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
9	9	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
10	10	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
11	11	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
12	12	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
13	13	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
14		14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

مرتب‌سازی سریع: تحلیل حالت متوسط

- گزاره. تعداد متوسط مقایسه‌های انجام شده در مرتب‌سازی سریع برای مرتب کردن آرایه‌ای شامل N عنصر متمایز برابر است با $2\ N \ln N$.

احتمال قرار گرفتن محور در مکان i — مرتب‌سازی عناصر قبل از محور — مرتب‌سازی عناصر بعد از محور — هزینه‌ی افراز

$$T(N) = \sum_{i=1}^{N} \frac{1}{N} [T(i-1) + T(N-i) + (N+1)]$$

ساده‌سازی جبری

$$T(N) = (N+1) + \frac{2}{N} \sum_{i=1}^{N} T(i-1)$$

ضرب طرفین در N

$$NT(N) = N(N+1) + 2 \sum_{i=1}^{N} T(i-1)$$

حذف سیگما از رابطه‌ی بازگشتی

$$NT(N) = (N+1)T(N-1) + 2N$$

مرتب‌سازی سریع: تحلیل حالت متوسط

- گزاه. تعداد متوسط مقایسه‌های انجام شده در مرتب‌سازی سریع برای مرتب کردن آرایه‌ای شامل N عنصر متمایز برابر است با $2\ N \ln N$.

$$\frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{2N}{N(N+1)}$$

تقسیم طرفین بر $N(N+1)$

$$A(N) = \frac{T(N)}{N+1} \Rightarrow A(N) = A(N-1) + \frac{2}{N+1}$$

تبدیل به یک رابطه‌ی بازگشتی خطی

$$A(N) \approx 2 \ln N \Rightarrow T(N) = 2(N+1) \ln N \sim 1.39 N \lg N$$

حل رابطه‌ی بازگشتی خطی

مرتب‌سازی سریع: خلاصه‌ی کارایی

- بدترین حالت. تعداد مقایسه‌ها درجه دوم است.
  - تعداد مقایسه‌ها در بدترین حالت $\sim 1/2\ N^2$
  - اما احتمال بدترین حالت از احتمال این که رعد و برق به کامپیوتر شما اصابت کند، کمتر است!!!
- حالت متوسط. تعداد مقایسه‌ها برابر است با $\sim 1.39\ N \lg N$
  - تعداد مقایسه‌ها در حالت متوسط ۳۹ درصد بیشتر از مرتب‌سازی ادغامی است.
  - اما در عمل به دلیل تعداد جابه‌جایی‌های کمتر، از مرتب‌سازی ادغامی سریع‌تر است.
- بر زدن تصادفی.
  - یک تضمین احتمالاتی در برابر بدترین حالت.
  - پایه‌ای برای مدل ریاضی که اعتبار آن با آزمایش قابل بررسی است.

ویژگی‌های مرتب‌سازی سریع

- گزاره. مرتب‌سازی سریع یک الگوریتم مرتب‌سازی درجا است.
- اثبات.
  - افراز: حافظه‌ی کمکی ثابت
  - عمق فراخوانی‌های بازگشتی: حافظه‌ی کمکی لگاریتمی [با احتمال بسیار بالا]

- گزاره. مرتب‌سازی سریع پایدار نیست.
- اثبات.

i	j	0	1	2	3
		B_1	C_1	C_2	A_1
1	3	B_1	C_1	C_2	A_1
1	3	B_1	A_1	C_2	C_1
0	1	A_1	B_1	C_2	C_1

مراحل افراز آرایه‌ی ورودی

مرتب‌سازی سریع: بهبود

- مرتب‌سازی آرایه‌های کوچک با استفاده از مرتب‌سازی درجی.
  - حتی مرتب‌سازی سریع سربار زیادی برای مرتب کردن آرایه‌های بسیار کوچک دارد.
  - اگر اندازه‌ی آرایه کوچک‌تر یا مساوی ۱۰ باشد، بهتر است از مرتب‌سازی درجی استفاده شود.
  - توجه: می‌توان انجام مرتب‌سازی درجی را تا انتهای الگوریتم به تأخیر انداخت.

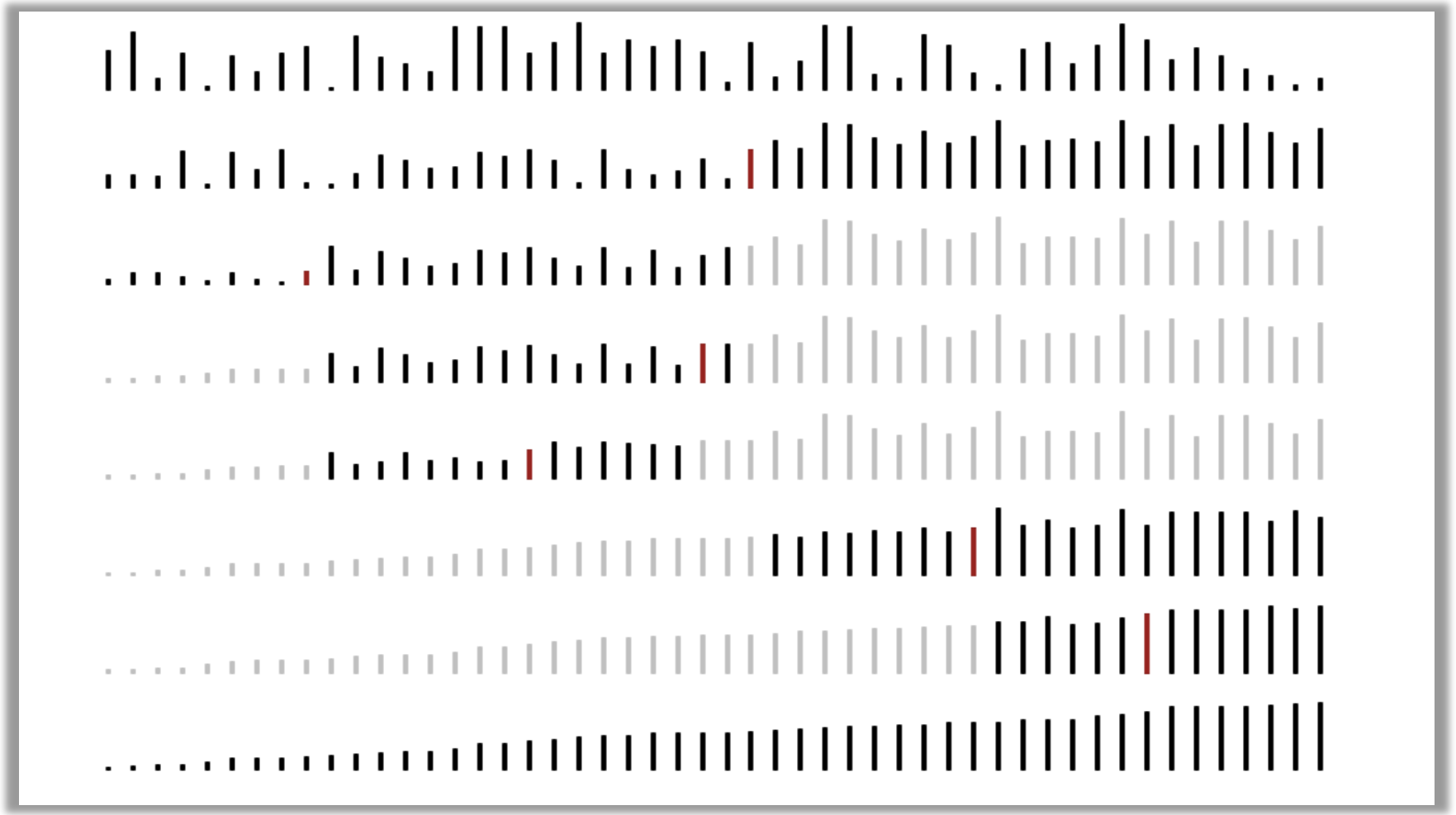
```
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi - lo + 1 <= CUTOFF) {
        Insertion.sort(a);
        return;
    }
    int j = partition(a, lo, hi);
    sort(a, lo, j - 1);
    sort(a, j + 1, hi);
}
```

مرتب‌سازی سریع: بهبود

- انتخاب عنصر محوری.
  - بهترین انتخاب برای عنصر محوری عنصر میانه است.
  - تخمین میانه‌ی واقعی با انتخاب میانه‌ی یک زیرمجموعه‌ی تصادفی
  - میانه‌ی ۳ عنصر (تصادفی)

```
private static void sort(Comparable[] a, int lo, int hi)
{
    if (hi <= low) return;

    int m = medianOf3(lo, lo + (hi - lo)/2, hi);
    ecxh(a, lo, m);

    int j = partition(a, lo, hi);
    sort(a, lo, j - 1);
    sort(a, j + 1, hi);
}
```

مرتب‌سازی سریع بهبود یافته: اجرا



```
% java QuickBars 1000 50
```

انتخاب

انتخاب

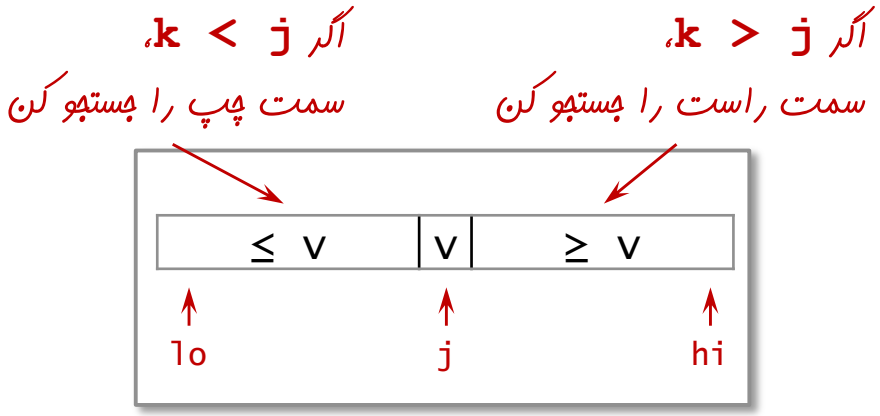
- هدف. آرایه‌ای شامل N عنصر داده شده است، k اُمین عنصر کوچک‌تر را پیدا کن.
- مثال. کوچک‌ترین عنصر (k = 0)؛ بزرگ‌ترین عنصر (K = N - 1)؛ میانه (K = N/2)

- کاربردها.
  - مرتبه‌های آماری
  - یافتن k عنصر برتر

- استفاده از تئوری به عنوان راهنما.
  - حد بالای N lg N. چرا؟
  - حد بالای N برای مقادیر کوچک k مانند ۱، ۲ و ۳
  - حد پایین N. چرا؟

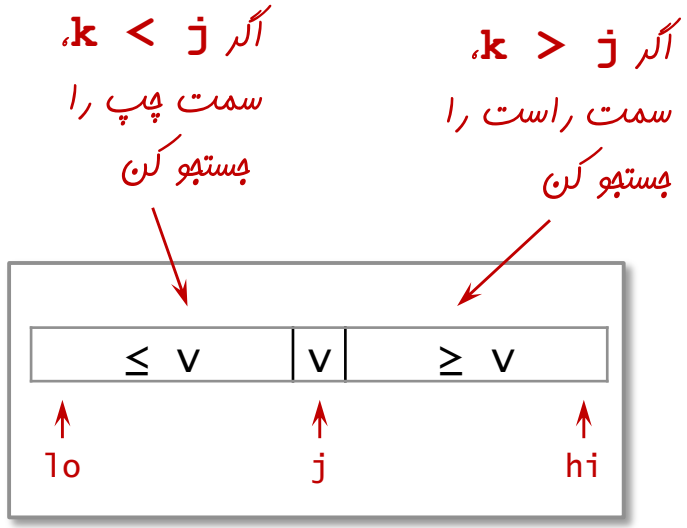
انتخاب مبتنی بر افراز

- آرایه را افراز کن به گونه‌ای که:
  - عنصر a[j] در مکان درست قرار گرفته باشد.
  - در سمت چپ j هیچ عنصری بزرگ‌تر از a[j] وجود نداشته باشد.
  - در سمت راست j هیچ عنصری کوچک‌تر از a[j] وجود نداشته باشد.
- بسته به مقدار j، عمل فوق را در یکی از زیرآرایه‌ها تکرار کن تا زمانی که j با k برابر شود.



انتخاب مبتنی بر افراز: پیاده‌سازی


```
public static Comparable select(Comparable[] a, int k)
{
    StdRandom.shuffle(a);
    int lo = 0, hi = a.length - 1;
    while (hi > lo)
    {
        int j = partition(a, lo, hi);
        if      (k < j) hi = j - 1;
        else if (k > j) lo = j + 1;
        else            return a[k];
    }
    return a[k];
}
```



انتخاب مبتنی بر افراز: تحلیل بدترین حالت

- گزاره. پیچیدگی زمانی الگوریتم انتخاب در بدترین حالت درجه دوم است.
- اثبات.

$$
\begin{aligned}
T(N) &= T(N-1) + (N+1) \\
&= T(N-2) + N + (N+1) \\
&= T(N-3) + (N-1) + N + (N+1) \\
&= T(N-4) + (N-2) + (N-1) + N + (N+1) \\
&= \ldots \\
&= T(1) + 1 + 2 + 3 + \ldots + (N+1) \\
&= \tfrac{1}{2}(N+1)(N+2) \\
&\sim \tfrac{1}{2} N^2
\end{aligned}
$$

انتخاب مبتنی بر افراز: تحلیل بهترین حالت

- گزاره. پیچیدگی زمانی الگوریتم انتخاب در بهترین حالت خطی است.
- اثبات.
  - در این حالت، آرایه پس از هر عمل افراز، به دو قسمت مساوی تقسیم می‌شود.
  - در نتیجه، تعداد مقایسه‌ها برابر است با:

$$T(N) = T(N/2) + (N + 1)$$

انتخاب مبتنی بر افراز: تحلیل حالت متوسط

- گزاره. پیچیدگی زمانی الگوریتم انتخاب در حالت متوسط خطی است. [با بر زدن]
- اثبات.
  - با هر عمل افزار، آرایه تقریباً به دو قسمت با اندازه‌های مساوی تقسیم می‌شود.
  - در نتیجه، تعداد مقایسه‌ها تقریباً برابر است با:

$$(N + 1) + (N + 1)/2 + (N + 1)/4 + \ldots + 1 \sim 2(N + 1)$$

  - با یک تحلیل رسمی مشابه با تحلیل مرتب‌سازی سریع نتیجه زیر به دست می‌آید:

$$T(N) = 2N + k \ln(N/k) + (N - k)\ln(N/(N - k))$$

$(2 + \ln 2)\ N$: میانه

انتخاب در زمان خطی

- گزاره. [بلوم، فلوید، پرت، ریوست، تارجان، ۱۹۷۳] یک الگوریتم انتخاب مبتنی بر مقایسه وجود دارد که زمان اجرای آن در بدترین حالت خطی است.

Time Bounds for Selection

by .

Manuel Blum, Robert W. Floyd, Vaughan Pratt,
Ronald L. Rivest, and Robert E. Tarjan

Abstract

The number of comparisons required to select the i-th smallest of n numbers is shown to be at most a linear function of n by analysis of a new selection algorithm -- PICK. Specifically, no more than 5.4305 n comparisons are ever required. This bound is improved for

انتخاب در زمان خطی

- گزاره. [بلوم، فلوید، پرت، ریوست، تارجان، ۱۹۷۳] یک الگوریتم انتخاب مبتنی بر مقایسه وجود دارد که زمان اجرای آن در بدترین حالت خطی است.

- ایده. انتخاب هوشمندانه‌ی عنصر محوری (عنصر محوری: تقریبی از عنصر میانه)
  - تقسیم آرایه به q قسمت با اندازه‌های مساوی.
  - محاسبه‌ی میانه‌ی هر قسمت با مرتب کردن آن.
  - محاسبه‌ی میانه‌ی میانه‌ها به صورت بازگشتی.

ضرب استراسن

ضرب داخلی دو بردار

- ضرب داخلی. دو بردار a و b هر یک به طول N داده شده است. مقدار c = a.b را محاسبه کنید.
- الگوریتم دبیرستان. تعداد ضرب‌ها $\Theta(N)$.

$$a = [.70 \quad .20 \quad .10]$$

$$b = [.30 \quad .40 \quad .30]$$

$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

- ملاحظه. الگوریتم دبیرستان یک الگوریتم بهینه است.

ضرب ماتریس‌ها

- ضرب ماتریسی. دو ماتریس A و B هر یک با ابعاد $N \times N$ داده شده است. ماتریس $C = AB$ را محاسبه کنید.

- الگوریتم دبیرستان. تعداد ضرب‌ها $\Theta(N^3)$.

$$\begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix} = \begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix}$$

الگوریتم ضرب ماتریس‌ها به روش استراسن

- الگوریتم استاندارد.
  - تعداد ضرب‌ها: $T(n) = n^3$
  - تعداد جمع‌ها: $T(n) = n^3 - n^2$

- الگوریتم استراسن برای ضرب ماتریس‌ها (۱۹۶۹).
  - پیچیدگی بهتر از درجه سوم (جمع و ضرب)

روش استراسن: ماتریس ۲ × ۲

- برای محاسبه‌ی:

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- تعریف می‌کنیم:

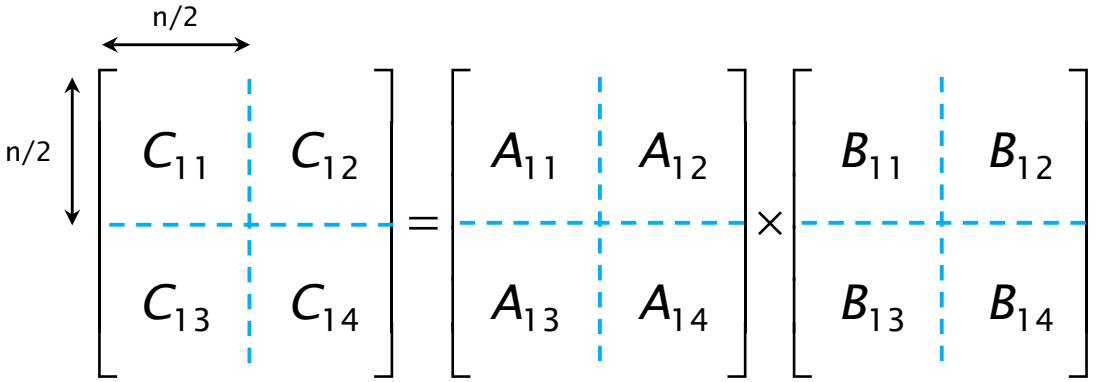
$$\begin{aligned} m_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ m_2 &= (a_{21} + a_{22})b_{11} \\ m_3 &= a_{11}(b_{12} - b_{22}) \\ m_4 &= a_{22}(b_{21} - b_{11}) \\ m_5 &= (a_{11} + a_{12})b_{22} \\ m_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\ m_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \end{aligned}$$

- آنگاه :

$$C = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

روش استراسن: ماتریس N × N

- تقسیم ماتریس‌ها.



- محاسبه‌ی M ها، مثلا:

$$M_1 = (A_{11} + A_{22})\,(B_{11} + B_{22})$$

یک مثال

- وقتی $n = 4$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{13} & C_{14} \end{bmatrix} = \left[\begin{array}{cc:cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hdashline 9 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{array}\right] \times \left[\begin{array}{cc:cc} 8 & 9 & 1 & 2 \\ 3 & 4 & 5 & 6 \\ \hdashline 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \end{array}\right]$$

- محاسبه‌ی M ها، مثلا:

$$M_1 = \begin{bmatrix} 3 & 5 \\ 11 & 13 \end{bmatrix} \times \begin{bmatrix} 17 & 10 \\ 7 & 9 \end{bmatrix} = \begin{bmatrix} 86 & 75 \\ 278 & 227 \end{bmatrix}$$

روش استراسن

$$\overset{n/2}{\longleftrightarrow}\quad \begin{matrix} n/2 \\ \updownarrow \end{matrix} \begin{bmatrix} C_{11} & C_{12} \\ C_{13} & C_{14} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{13} & A_{14} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{13} & B_{14} \end{bmatrix} \quad (1)$$

(2)

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22})B_{11}$$
$$M_3 = A_{11}(B_{12} - B_{22})$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12})B_{22}$$
$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

(3)

$$C = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

پیچیدگی زمانی: همه‌ی حالات (ضرب)

- عمل اصلی. یک ضرب ساده
- اندازه ورودی. **n**، تعداد سطرها و ستون‌های ماتریس‌ها
- پیچیدگی زمانی.

$$\begin{cases} T(n) = 7T\left(\frac{n}{2}\right) & \text{for } n > 1, n \text{ power of } 2 \\ T(1) = 1 \end{cases}$$

- حل.

$$T(n) = n^{\lg n} \approx n^{2.81} \in \Theta(n^{2.81})$$

پیچیدگی زمانی: همه‌ی حالات (جمع و تفریق)

- عمل اصلی. یک جمع یا تفریق ساده
- اندازه ورودی. **n**، تعداد سطرها و ستون های ماتریس ها
- پیچیدگی زمانی.

$$\begin{cases} T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 & \text{for } n > 1, n \text{ power of } 2 \\ T(1) = 1 \end{cases}$$

- حل.

$$T(n) = 6n^{\lg 7} - 6n^2 \quad 6n^{2.81} - 6n^2 \quad (n^{2.81})$$

مقایسه

	الگوریتم استاندارد	الگوریتم استراسن
تعداد ضرب‌ها	n^3	$n^{2.81}$
تعداد جمع و تفریق ها	$n^3 - n^2$	$6n^{2.81} - 6n^2$

تمرین‌ها

- بخش ۲-۱
  - ۴، ۶
- بخش ۲-۲
  - ۱۰، ۱۳
- بخش ۲-۳
  - ۱۴، ۱۵، ۱۶، ۱۷، ۱۸
- بخش ۲-۴
  - ۲۴
- بخش ۲-۵
  - ۲۶، ۲۹
- بخش ۲-۷
  - ۳۶
- بخش ۲-۸
  - ۳۷، ۳۸
- تمرینات اضافی
  - ۳۹ و ۴۰و ۴۱