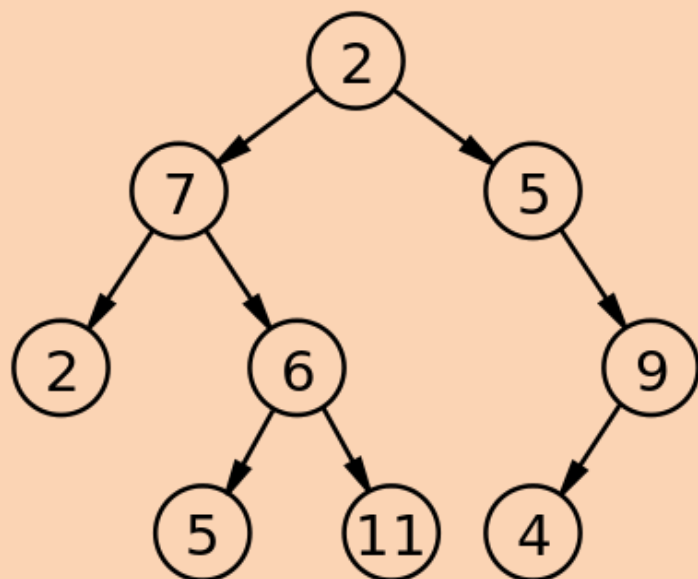




ساختارهای داده

مرتب‌ساز سریع

Quick Sort



مدرس:

سید کمال الدین غیاثی شیرازی

ایده اصلی

partition pivot

- اگر عناصر آرایه را حول یک محور افراز کنیم ، به دو مساله کوچک تر می رسیم.

6	11	7	1	10	6	2	8	6	4
---	----	---	---	----	---	---	---	---	---

~~$< \text{pivot}$~~ vs ~~$> \text{pivot}$~~

- اگر عناصر آرایه را حول یک محور افراز کنیم، به دو مساله کوچک تر می رسیم.

1	6	2	4	6	11	7	10	8	6
---	---	---	---	---	----	---	----	---	---

$\leq \text{pivot}$
 \downarrow
 $\geq \text{pivot}$

۹ = انیس عفر محمد
یس از انرا

الگوریتم مرتب‌سازی سریع معمولی (با افراز دوراها)

~~$P(n) = P(n)$~~

$$\text{طول} = r - p + 1$$

```
void QuickSort(int A[], int p, int r) {
```

```
    if (r > p) {
```

```
        int pivot = ChoosePivot(A, p, r)
```

```
        q = Partition (A, p, r, pivot)
```

```
        QuickSort (A, p, q-1);
```

```
        QuickSort (A, q+1, r);
```

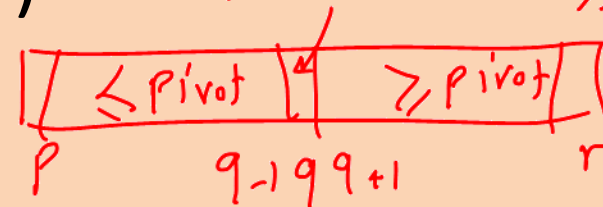
```
    } else {
```

```
}
```



عنصر محدد

پس از افراز



$$\text{طول} = q - p$$

$$\text{طول} = r - q$$

استقرای قوی

که نشان می‌دهد هر زمان که q

تقسیم می‌شود که طول آرایه $A[p \dots q-1]$

و $A[q+1 \dots r]$ از آرایه اولیه کمتر است.

گام بسته این اثبات استقرایی

Quick Sort

بازگشتی برای

QuickSort(A, p, r)

if $p < r$

{ ChoosePivot(A, p, r)

$q \leftarrow \text{Partition}(A, p, r)$

QuickSort($A, p, q-1$)

QuickSort($A, q+1, r$)

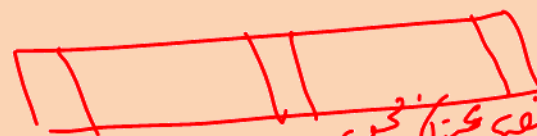
محور، اجزای آرایه می آید

انتخاب محور (ChoosePivot)

• ایده آل این است که اندازه آرایه چپ و راست مساوی باشد.



○ روش اول: انتخاب سمت چپ‌ترین داده $A[p]$
اشکال، انتخاب عنصر محور وابسته به ترتیب داده است

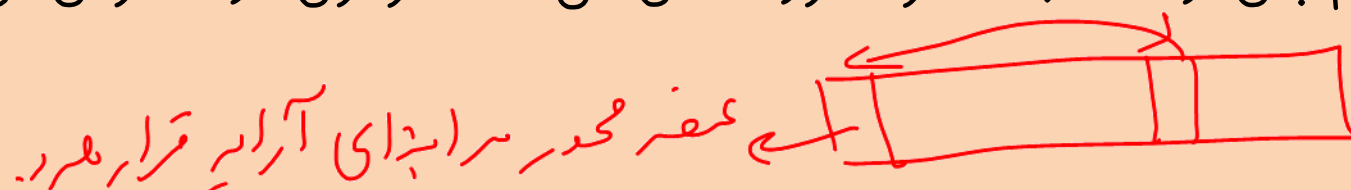


○ روش دوم: انتخاب میانه عناصر چپ، وسط و آخر

میان $6 = 4, 12, 6$

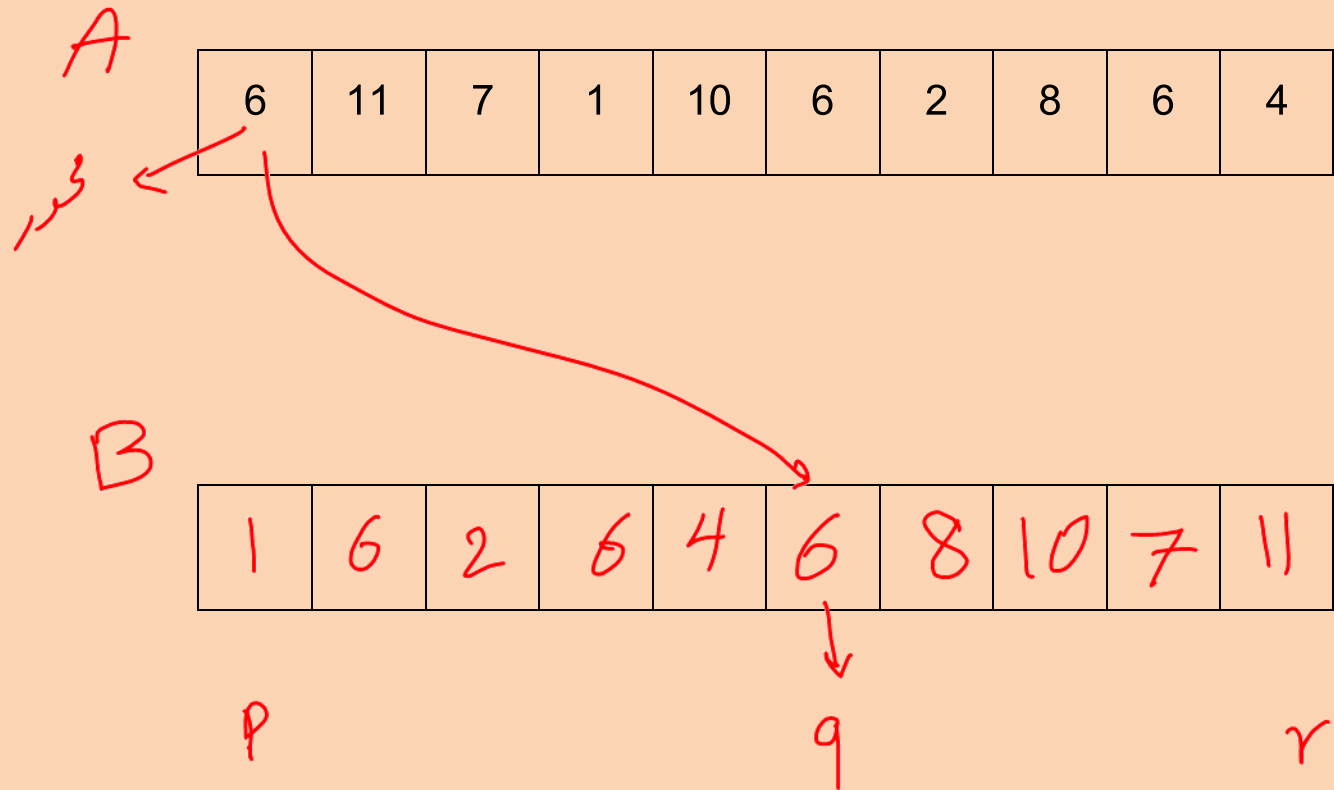
○ روش سوم (برگزیده): انتخاب یک عنصر به صورت تصادفی با تابع $rand()$ که وابسته به ترتیب داده نیست
احتمال انتخاب یک عنصر محور

• فرض می‌کنیم پس از انتخاب عنصر محور، محل آن با عنصر اول آرایه عوض می‌شود.



• زمان اجرای تابع ChoosePivot از مرتبه $\Theta(1)$ است.

الگوریتم افراز با استفاده از حافظه کمکی



الگوریتم اول افراز درجا

استقرای ریاضی

loop invariance

Partition (Array A, int p, int r)

pivot = A[p]; i=p+1; j=r;

while (1)

while (i ≤ j && A[i] ≤ pivot) i++

while (i ≤ j && A[j] ≥ pivot) j--

if (i < j)

swap (A[i], A[j])

continue

else

break

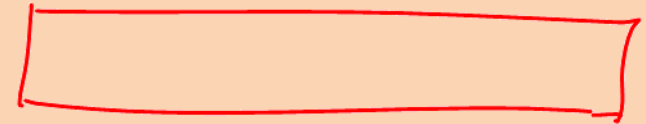
swap (A[i-1], A[p]) // or exchange (A[j], A[p])

خاصیت حتم $A[p \dots i-1]$ کوچکتر از پوی می شود

خاصیت $A[j+1 \dots r]$ بزرگتر از پوی می شود

اولین باری که $i > j$ شود دیگر i و j تغییر نمی کنند

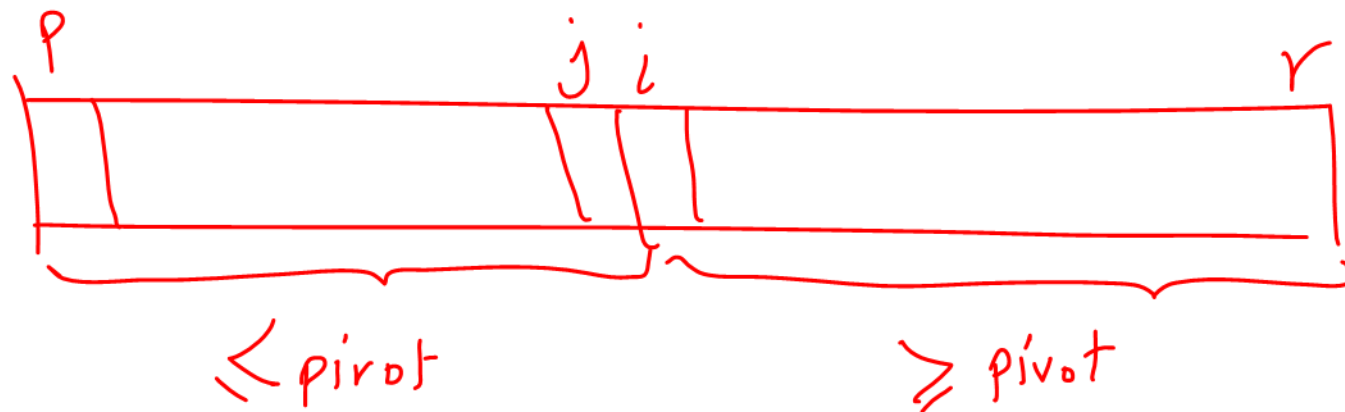
$i \leq j$
 $i = j+1$
 $i = j+1$
 $j = i-1$



حالت $i = j$ در خط $*$ امکان ندارد.
فرض خلاف: در خط $*$ $i = j$ است
 \Rightarrow در دو حلقه while همواره $i \leq j$
 \Rightarrow دلیل خروج از دو حلقه while داخلی بهتر از سبذی شرط دوم بوده است. \Rightarrow

$A[i] > \text{pivot}$
 $A[j] < \text{pivot}$

پس از آنکه i و j به هم برسند $\text{pivot} < A[i] < \text{pivot}$ می شود که این فرض خلاف باطل و در خط $*$ $i \neq j$ است



بالبرای دستور
 $A[p] \leftrightarrow A[j]$

$$q = j$$

عناصر بعد از خانه q و i بزرگتر مساوی مقدار
 خانه q مساوی مقدار
 و عناصر قبل از خانه q کوچکتر مساوی مقدار عناصر بعد از

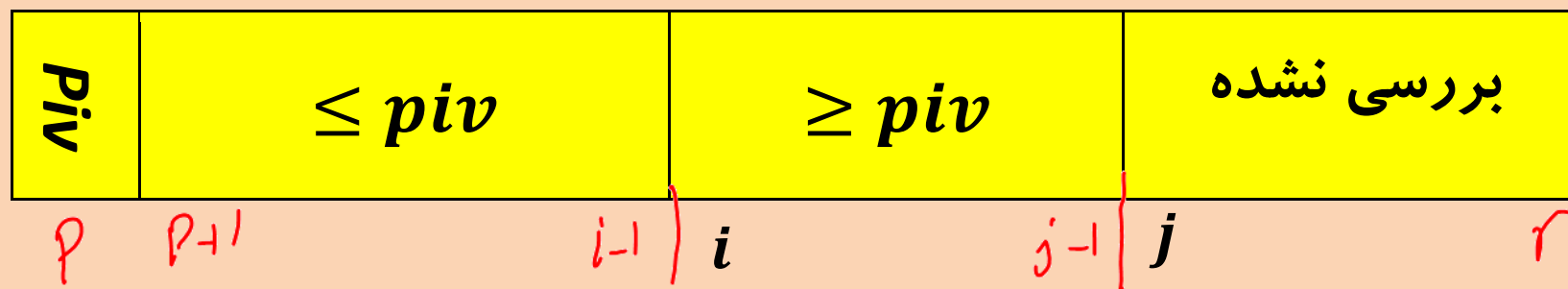
اجرای الگوریتم اول افراز درجا روی يك مثال

p		$p+1$												r
10	4	7	18	15	4	8	5	19	5	13	5	1	2	
			$\uparrow i$									$\uparrow j$		
10	4	7	2	15	4	8	5	19	5	13	5	1	18	
				$\uparrow i$								$\uparrow j$		
10	4	7	2	1	4	8	5	19	5	13	5	15	18	
								$\uparrow i$				$\uparrow j$		
10	4	7	2	1	4	8	5	5	5	13	19	15	18	
									$\uparrow j$	$\uparrow i$				

الگوریتم CLRS برای افراز درجا

● به نحوی عمل می کنیم که همواره داشته باشیم:

- عنصر اول آرایه محور است
- از عنصر دوم تا ^{اولی} عنصر $i - 1$ کوچک تر مساوی محور هستند.
- از عنصر i تا $j - 1$ بزرگ تر و یا مساوی محور هستند.
- عناصر j تا انتهای آرایه هنوز بررسی نشده اند.



Piv	$\leq piv$	$\geq piv$	بررسی نشده
p	p+1	i-1 i	j-1 j
			r

$$i = p+1$$

$$j = p+1$$

for $j = p+1$ to r

if $A[j] < pivot$

$A[i] \leftrightarrow A[j]$

$i++$

$A[p] \leftrightarrow A[i-1]$

الگوریتم CLRS برای افراز درجا

```
Partition (A, p, r)
    pivot = A[p];
    i = p+1;
    for j=p+1 to r
        if (A[j] < pivot)
            swap A[j] and A[i]
            i = i + 1
    swap A[p] and A[i-1]
```

اجرای الگوریتم دوم افراز درجا روی يك مثال

محرر ←

10	4	7	18	15	4	8	5	19	5	13	5	1	2
----	---	---	----	----	---	---	---	----	---	----	---	---	---

i j

10	4	7											
----	---	---	--	--	--	--	--	--	--	--	--	--	--

i j

10	4	7	18										
----	---	---	----	--	--	--	--	--	--	--	--	--	--

i j

10	4	7	18	15	4	8	5	19	5	13	5	1	2
----	---	---	----	----	---	---	---	----	---	----	---	---	---

i j

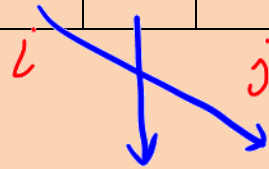
(a)

10	4	7	18	15	4	8	5	19	5	13	5	1	2
----	---	---	----	----	---	---	---	----	---	----	---	---	---

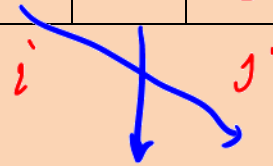
 اجر

i j

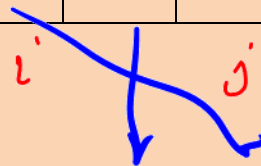
10	4	7	18	15	4	8	5	19	5	13	5	1	2
----	---	---	----	----	---	---	---	----	---	----	---	---	---



10	4	7	4	15	18	8	5	19	5	13	5	1	2
----	---	---	---	----	----	---	---	----	---	----	---	---	---



10	4	7	4	8	18	15	5	19	5	13	5	1	2
----	---	---	---	---	----	----	---	----	---	----	---	---	---



10	4	7	4	8	5	15	18	19	5	13	5	1	2
----	---	---	---	---	---	----	----	----	---	----	---	---	---

i j

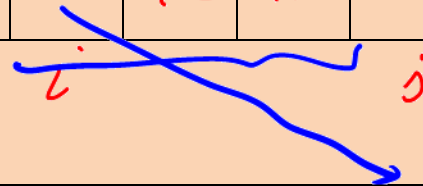
(مه)

10	4	7	4	8	5	15	18	19	5	13	5	1	2
----	---	---	---	---	---	----	----	----	---	----	---	---	---

 اج

i j

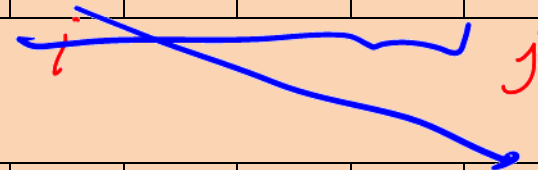
10	4	7	4	8	5	15	18	19	5	13	5	1	2
----	---	---	---	---	---	----	----	----	---	----	---	---	---



10	4	7	4	8	5	5	18	19	15	13	5	1	2
----	---	---	---	---	---	---	----	----	----	----	---	---	---

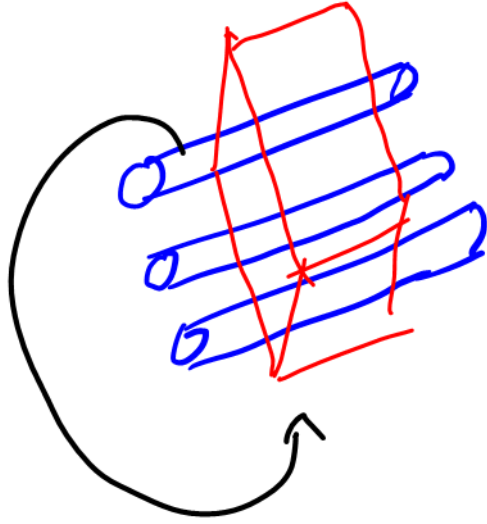
i j

							18	19	15	13	5		
--	--	--	--	--	--	--	----	----	----	----	---	--	--



10	4	7	4	8	5	5	5	19	15	13	18	1	2
----	---	---	---	---	---	---	---	----	----	----	----	---	---

i j



پیچیدگی زمانی الگوریتم افراز اول

Partition (Array A, int p, int r)

pivot = A[p]; i=p+1; j=r; $\Theta(1)$

while (1)

while (i ≤ j && A[i] ≤ pivot) i++

while (i ≤ j && A[j] > pivot) j--

if (i < j)

swap (A[i], A[j])

continue

else

break

swap (A[i-1], A[p]) // or exchange (A[j], A[p])

تعداد عملیات = تعداد دفعات افزایش i
+ تعداد دفعات کاهش j
= $\Theta(r - p) = \Theta(\text{اندازه آرایه})$

$\Theta(1)$

پیچیدگی زمانی الگوریتم افراز دوم

Partition (A, p, r)

pivot = A[p];

i = p+1;

for j=p+1 to r

$\theta(1)$ { if (A[j] < pivot)
swap A[j] and A[i]

i = i + 1

swap A[p] and A[i-1]

} $\theta(r-p)$
(انزاله آرایه) $= \theta(n)$

کدام روش افراز سریع تر است؟ نتایج برای ۱۰۰ میلیون داده

```
Number of exchanges in TwoWayPartition_Classic is: 841333 in 0.13325 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 841795 in 0.1375 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 138571612 in 0.68075 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 403618897 in 0.6825 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 77341450 in 0.817625 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 123163873 in 0.8195 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 166895617 in 1.09387 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 212783122 in 1.16983 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 131735908 in 1.41138 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 164367931 in 1.47508 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 25954414 in 1.58571 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 27329941 in 1.64275 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 159295162 in 2.29171 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 360034576 in 2.27675 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 195602116 in 2.77971 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 300533908 in 2.88725 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 125342662 in 3.43971 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 300833719 in 5.96825 seconds.  
Number of exchanges in TwoWayPartition_Classic is: 146102869 in 3.75021 seconds.  
Number of exchanges in TwoWayPartition_CLRS is: 192412408 in 6.2385 seconds.
```

```
Total Number of exchanges in TwoWayPartition_classic is: 464439328  
Total Number of exchanges in TwoWayPartition_CLRS is: 1494261673  
Total time of TwoWayPartition_classic in seconds is: 6.2385  
Total time of TwoWayPartition_CLRS in seconds is: 6.2385
```

تقریباً CLRS
3 برابر جا به جا
شود.

کدام روش افراز سریع تر است؟ نتایج برای یک میلیون داده

```
Number of exchanges in TwoWayPartition_Classic is: 827413 in 0.00110375 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 970132 in 0.000941675 seconds.
Number of exchanges in TwoWayPartition_Classic is: 79568401 in 0.00264613 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 80963758 in 0.00248167 seconds.
Number of exchanges in TwoWayPartition_Classic is: 84057124 in 0.00453648 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 91445716 in 0.00422383 seconds.
Number of exchanges in TwoWayPartition_Classic is: 35681668 in 0.00570725 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 32743474 in 0.00549934 seconds.
Number of exchanges in TwoWayPartition_Classic is: 115473745 in 0.00876841 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 115454239 in 0.00857934 seconds.
Number of exchanges in TwoWayPartition_Classic is: 125928451 in 0.0125852 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 121425397 in 0.0125833 seconds.
Number of exchanges in TwoWayPartition_Classic is: 64025059 in 0.0141237 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 65155684 in 0.0140962 seconds.
Number of exchanges in TwoWayPartition_Classic is: 21221044 in 0.0152523 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 20152330 in 0.0152853 seconds.
Number of exchanges in TwoWayPartition_Classic is: 41234206 in 0.0164689 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 43807750 in 0.016503 seconds.
Number of exchanges in TwoWayPartition_Classic is: 108347008 in 0.0187647 seconds.
Number of exchanges in TwoWayPartition_CLRS is: 117391507 in 0.0186629 seconds.
```

Total Number of exchanges in TwoWayPartition_classic is: 3893599

Total Number of exchanges in TwoWayPartition_CLRS is: 12370477

Total time of TwoWayPartition_classic in seconds is: ~~0.0187647~~

Total time of TwoWayPartition_CLRS in seconds is: ~~0.0186629~~

صورت ۲ برابر
CLRS جایگزینی ندارد.

یادآوری الگوریتم مرتب‌سازی سریع

T

```
void QuickSort(int A[], int p, int r) {
```

```
    if (r > p) {
```

```
        int pivot = ChoosePivot(A, p, r)
```

```
        q = Partition (A, p, r, pivot)
```

```
        QuickSort (A, p, q-1);
```

```
        QuickSort (A, q+1, r);
```

```
    }
```

```
}
```

$$n = r - p + 1$$

$$\left. \begin{array}{l} \mathcal{O}(1) \\ \mathcal{O}(n) \end{array} \right\} \mathcal{O}(n)$$

$$n \leq 1$$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ n + T(q-p) + T(r-q) & n \geq 2 \end{cases}$$

\swarrow \nwarrow
 ← $q - p$ ← $r - (q+1) + 1 = r - q$

پیچیدگی زمانی الگوریتم مرتب‌سازی سریع در بدترین حالت

بهترین حالت برای الگوریتم QuickSort این است که پس از افزایش یک مرتبه
تکلی شود. طرز آرایه فقط یکی کم می‌شود.

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

$$\begin{aligned} T(n) &= n + T(n-1) = n + [(n-1) + T(n-2)] \\ &= n + (n-1) + (n-2) + T(n-3) = \dots = \sum_{i=1}^n i = \frac{n(n+1)}{2} \\ &= \Theta(n^2) \end{aligned}$$

پیچیدگی زمانی الگوریتم مرتب‌سازی سریع در صورت نصف شدن داده‌ها در هر افراز

$$T(n) = \begin{cases} 1 & n \leq 1 \\ n + \underbrace{T(n/2) + T(n/2)}_{2 \cdot T(n/2)} & n \geq 2 \end{cases}$$

Diagram illustrating the recurrence relation for the time complexity of a sorting algorithm. The recurrence is defined as $T(n) = \begin{cases} 1 & n \leq 1 \\ n + T(n/2) + T(n/2) & n \geq 2 \end{cases}$. The term $2 \cdot T(n/2)$ is circled in blue, with a blue arrow pointing to the coefficient 2, labeled 'd'. Below the circled term, the letters 'a' and 'b' are written, corresponding to the general form $a \log_b n$.

قفسه اصلی

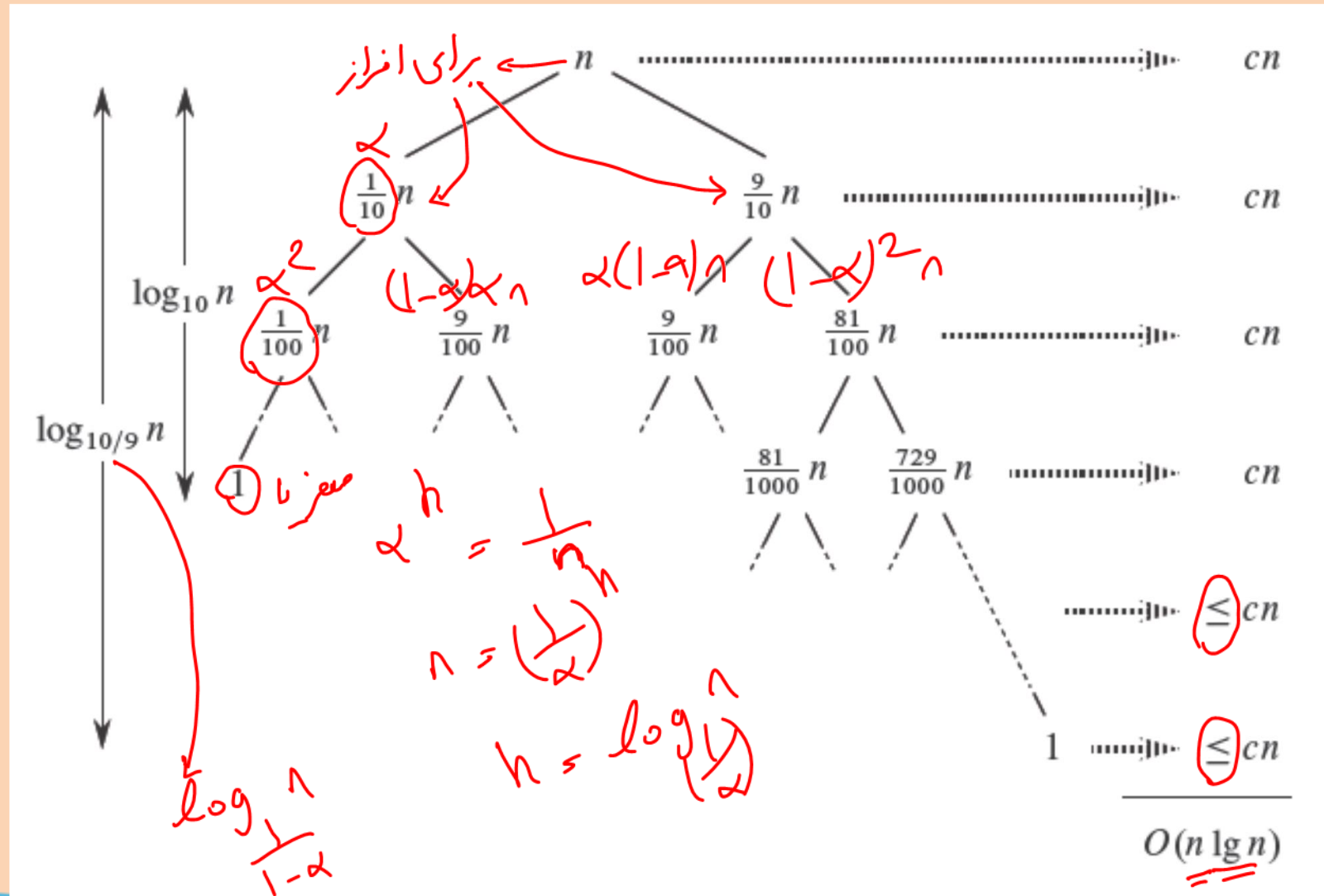
$$\log_a b = d$$

$$T(n) \in \Theta(n \log n)$$

$$\Leftrightarrow \log_2 2 = 1 = d$$

پیچیدگی زمانی الگوریتم مرتب‌سازی سریع در صورتی که افراز کوچک‌تر داده‌ها حداقل α قسمت از داده‌ها باشد

$$0.5 < \alpha < 1$$



زمان اجرای متوسط الگوریتم مرتب سازی سریع

$$O(n \log n)$$

پاسخ:

اشکال‌های الگوریتم مرتب‌سازی سریع

● مشکل اول: مشکل اندازه دو برابر

○ داده‌های مساوی همه یک سمت قرار می‌گیرند و زمان مرتب‌سازی آنها حتماً $\theta(n^2)$ است.

● مشکل دوم: در بدترین حالت به $\theta(n)$ حافظه در پشته نیاز است. هم برای اندازه دو برابر و هم برای اندازه سه برابر مشکل

○ از آنجا که حافظه پشته معمولاً بسیار محدود است (مثلاً حدود ۱ مگابایت)، امکان آنکه اجرای آن

برنامه بخاطر پر شدن پشته با شکست مواجه شود، وجود دارد.

○ نتیجه: هرگز از این پیاده‌سازی الگوریتم مرتب‌سازی سریع استفاده نکنید. در صورت لزوم حتماً یاد

بگیرید که چگونه می‌توان الگوریتم را اصلاح کرد که اندازه پشته $\theta(\log n)$ باشد.

● راه حل مشکل اول، افراز سه‌راهه است که تکلیف عناصر مساوی را درجا روشن می‌کند.

الگوریتم افراز سه‌راهه

(گرفته شده از درس آقای Sedgwick – دانشگاه پرینستون)

به نحوی عمل می‌کنیم که همواره داشته باشیم:

- عناصر اول تا $i-1$ ام آرایه برابر محور هستند.
- از عنصر $k+1$ تا انتهای آرایه بزرگ‌تر از محور هستند.
- از عنصر $j+1$ تا عنصر k کوچک‌تر از محور هستند.
- عناصر i تا j آرایه هنوز بررسی نشده‌اند.

	i		j		k	
=pivot				<pivot		>pivot

$A[P] = \text{pivot}$

	i		j		k	
=pivot				<pivot		>pivot

$i = P + 1$
 $k = r$
 $j = r$

عبر الیہ اسقرا

while ($i \leq j$)
 if ($A[j] < \text{pivot}$)

$j--$
 elif ($A[j] == \text{pivot}$)
 swap $A[j], A[i]$

$i++$
 else // $A[j] > \text{pivot}$
 swap $A[j], A[k]$
 $k--$
 $j--$

الگوریتم افراز سه‌راهه

(گرفته شده از درس آقای Sedgwick - دانشگاه پرینستون)

Partition (A, p, q)^r

pivot = A[p]; i = p+1; j = q; k = q;

while (j >= i)

if (A[j] > pivot)

swap A[j] and A[k]; j = j - 1; k = k - 1;

else if (A[j] == pivot)

swap A[j] and A[i]

i = i + 1

else // if (A[j] < pivot)

j = j - 1

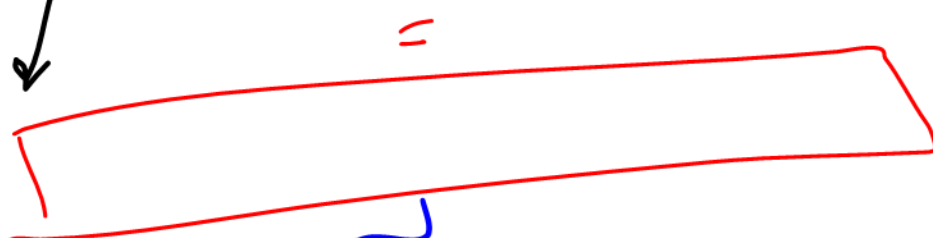
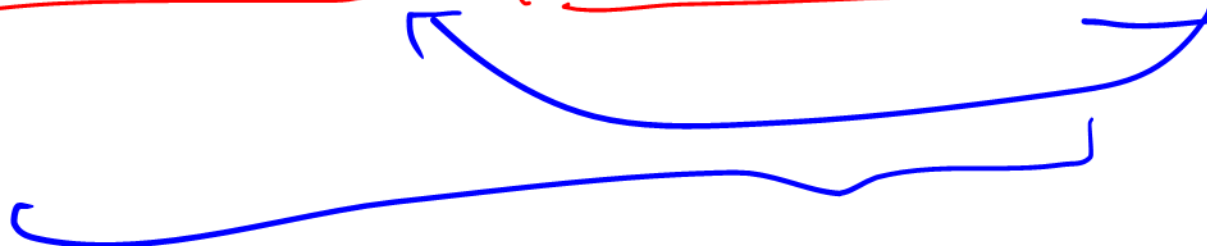
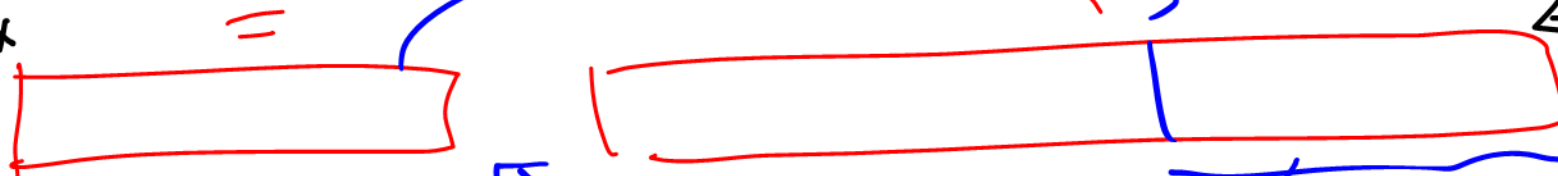
m = min(i - p, k - j)

swap A[p,...,p+m-1] and A[k-m+1,..., k]

تعداد عناصر کوچکتر = i - p
تعداد عناصر بزرگتر = k - j

استاد رضا کریمزاده

ابتدای عناصر



=

مثال از اجرای افراز سه‌راهه به روش اول



6	6	6	6	11	10	4	1	2	8	7
---	---	---	---	----	----	---	---	---	---	---

i

j k

6	6	6	6	11	10	4	1	2	8	7
---	---	---	---	----	----	---	---	---	---	---

i

j

k

6	6	6	6	11	10	4	1	2	8	7
---	---	---	---	----	----	---	---	---	---	---

i

j

k

6	6	6	6	11	2	4	1	10	8	7
---	---	---	---	----	---	---	---	----	---	---

i

j

k

6	6	6	6	1	2	4	11	10	8	7
---	---	---	---	---	---	---	----	----	---	---

i, j

k

6	6	6	6	1	2	4	11	10	8	7
---	---	---	---	---	---	---	----	----	---	---

j

i

k

1	2	4	6	6	6	6	11	10	8	7
---	---	---	---	---	---	---	----	----	---	---

$m = 3$

افراز سه راهه الگوریتم دوم

● به نحوی عمل می کنیم که همواره داشته باشیم:

- عناصر اول تا $i - 1$ ام آرایه کوچکتر از محور هستند.
- از عنصر $k + 1$ تا انتهای آرایه بزرگ تر از محور هستند.
- عناصر i تا $j - 1$ آرایه مساوی محور هستند.
- از عنصر j تا عنصر k هنوز بررسی نشده اند.
- عنصر j ام در حال بررسی است.

	i		j		k	
<pivot	=pivot				>pivot	

$i \leq j$
 $j \leq k$

	i		j		k	
<pivot	=pivot				>pivot	

$i = p$

$k = r$

$j = p + 1$

while ($j \leq k$)

if ($A[j] == \text{pivot}$)

$j++$
 elif ($A[j] < \text{pivot}$)
 $A[i] \leftrightarrow A[j]$

$i++$
 $j++$
 else ($A[j] > \text{pivot}$)
 $A[k] \leftrightarrow A[j]$
 $k--$

الگوریتم دوم برای افراز سه‌راهه

Partition (A, pivot, p, q)

i = p; j = p; k = q;

while (j <= k)

if (A[j] > pivot)

swap A[j] and A[k]; k = k-1;

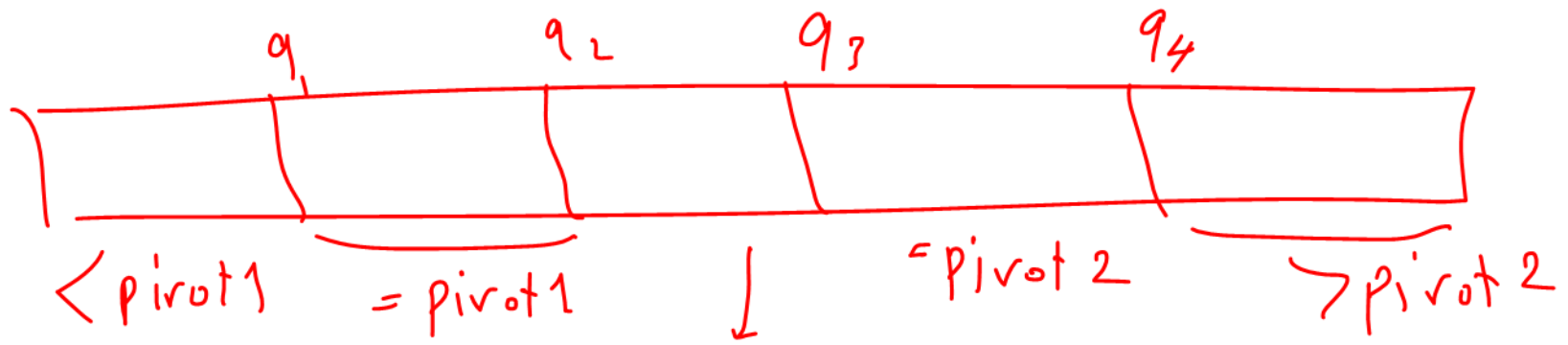
else if (A[j] < pivot)

swap A[j] and A[i]; i = i+1; j=j+1;

else if (A[j] == pivot)

j = j+1

می‌توانست $j = p+1$ هم باشد



$\text{pivot 1} < \text{pivot 2}$

مثال از اجرای افراز سه‌راهه به روش دوم

6	2	8	6	11	10	4	1	6	7	6
---	---	---	---	----	----	---	---	---	---	---

i j k

2	6	8	6	11	10	4	1	6	7	6
---	---	---	---	----	----	---	---	---	---	---

i j k

2	6	6	6	11	10	4	1	6	7	8
---	---	---	---	----	----	---	---	---	---	---

i j k

2	6	6	6	11	10	4	1	6	7	8
---	---	---	---	----	----	---	---	---	---	---

j k

2	6	6	6	7	10	4	1	6	11	8
---	---	---	---	---	----	---	---	---	----	---

i j k

2	6	6	6	6	10	4	1	7	11	8
---	---	---	---	---	----	---	---	---	----	---

i j k

در مرحله اول
تجارت

2	6	6	6	6	10	4	1	7	11	8
	<i>i</i>			<i>j</i>			<i>k</i>			

2	6	6	6	6	10	4	1	7	11	8
	<i>i</i>				<i>j</i>		<i>k</i>			

2	6	6	6	6	1	4	10	7	11	8
	<i>i</i>				<i>j</i>	<i>k</i>				

2	1	6	6	6	6	4	10	7	11	8
	<i>i</i>					<i>j, k</i>				

2	1	4	6	6	6	6	10	7	11	8
		<i>i</i>				<i>k</i>	<i>j</i>			

--	--	--	--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--	--	--	--

C++ STL sort Function

- برای استفاده از مرتب‌سازی سریع در زبان C++ کافی است از تابع sort استفاده کنید.
- در زبان C تابع qsort با الگوریتم مرتب‌سازی سریع داده‌ها را مرتب می‌کند.