مرتب‌سازی و مرتبه‌ی آماری

n عنصر با کلیدهای $a_1, a_2, \ldots, a_n$ و رابطه‌ی ترتیب کامل (total order) $\leq$ بر روی کلیدها داده شده‌اند.

تعریف مرتب‌سازی جای‌گشت π از عناصر داده شده را پیدا کنید به‌طوری‌که:

$$a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$$

یادآوری

یک رابطه ترتیب جزیی (partial order) **است اگر نامتقارن، بازتابی و تراگذری باشد.**

رابطه‌ی ترتیب جزیی R **بر روی مجموعه‌ی** A **رابطه‌ی ترتیب کامل است، اگر برای هر** $a, b \in A$ **داشته باشیم** aRb **یا** bRa.

دسته‌بندی الگوریتم‌های مرتب‌سازی

۱) از نظر موقعیت داده‌ها در زمان مرتب‌سازی (یعنی داده در کجا ذخیره می شود)

- **داخلی** (internal) ◁ **خارجی** (external) ▷

۲) از نظرحفظ ترتیب نسبی عناصر پس از مرتب‌سازی

- **پایدار** (stable) ◁ **ناپایدار** (unstable) ▷

۳) از نظر نحوه‌ی مرتب‌سازی داده‌ها

- **مقایسه‌ای** (comparison sort) (با مقایسه‌ی عناصر مرتب می‌کند)
- **غیر مقایسه‌ای** (non-comparison sort)

کران پایین الگوریتم‌های مرتب‌سازی

- زمان اجرای هر الگوریتم مرتب‌سازی $\Omega(n)$ است.
- زمان اجرای هر الگوریتم مرتب‌سازی مقایسه‌ای هم در بدترین حالت و هم در حالت میانگین $\Omega(n \lg n)$ است.

یادآوری: تعریف ترتیب الفبایی (Lexicographic Ordering)

فرض:

$$\vec{a} = a_1 a_2 \cdots a_i \cdots a_n$$
$$\vec{b} = b_1 b_2 \cdots b_i \cdots b_m$$

می‌گوییم $\vec{a} \leq \vec{b}$ **اگر برای** $i < k \leq n$، $a_i = b_i$ **و** $a_k < b_k$،
یا $n < m$ **و برای** $1 \leq i \leq n$ **داشته باشیم** $a_i = b_i$.

مثلاً

$$ab < abc < adc < adda$$

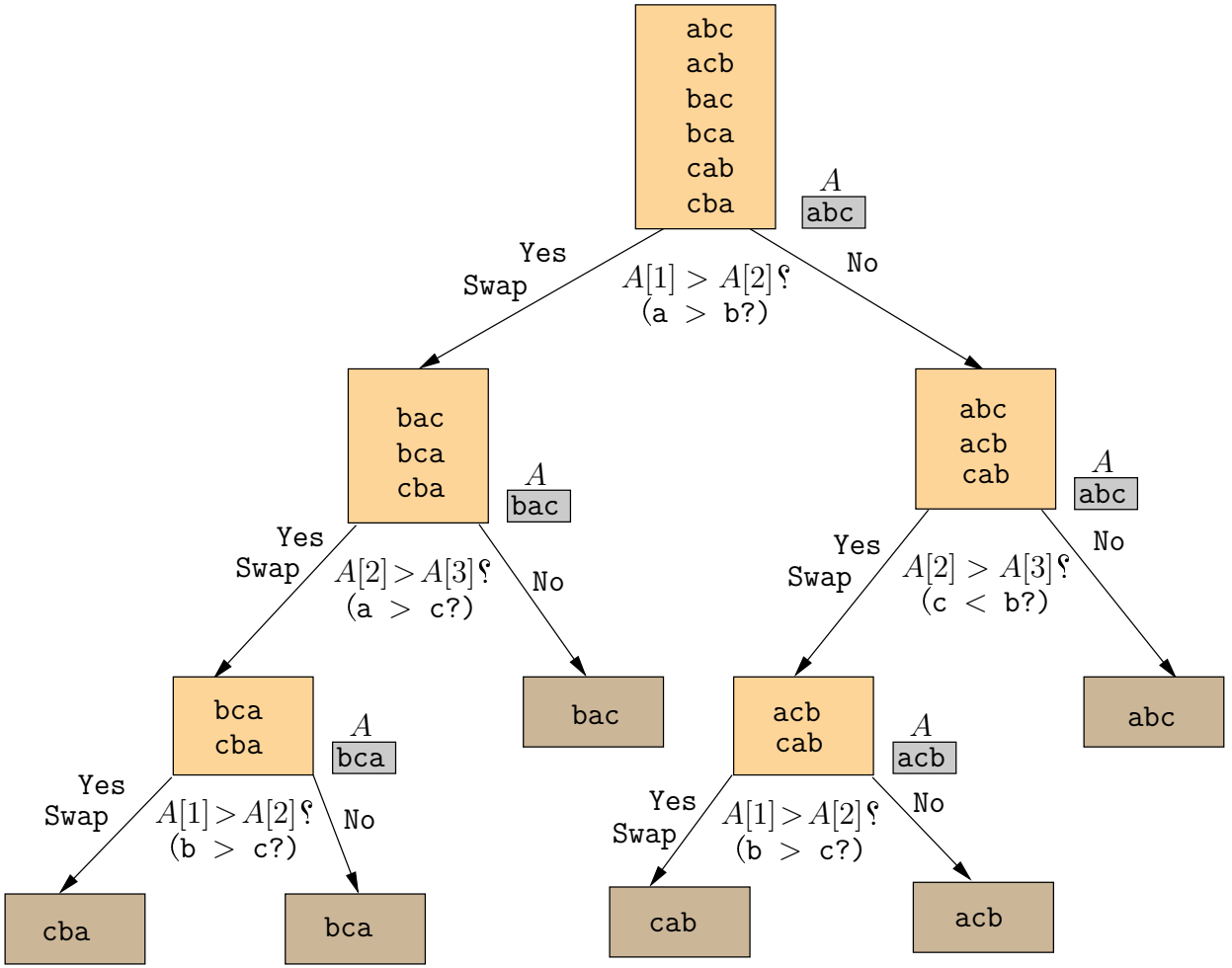
اثبات کران پایین در بدترین حالت

درخت تصمیم (Decision Tree)

هر الگوریتم مقایسه‌ای را می‌توان با یک درخت با ویژگی‌های زیر مدل کرد.

- «حالت مسئله»
- درخت دودویی کامل است (هر مقایسه یک انشعاب)
- برگ‌ها حالت نهایی

مثال: درخت تصمیم برای مرتب‌سازی درجی بر روی $A = [a, b, c]$. هر حالت شامل همه‌ی جای‌گشت‌های ممکن عناصر مرتب است.


abc
acb
bac
bca
cab
cba
A
abc
Yes
Swap
A[1] > A[2]؟
(a > b?)
No
bac
bca
cba
A
bac
abc
acb
cab
A
abc
Yes
Swap
A[2] > A[3]؟
(a > c?)
No
Yes
Swap
A[2] > A[3]؟
(c < b?)
No
bca
cba
A
bca
bac
acb
cab
A
acb
abc
Yes
Swap
A[1] > A[2]؟
(b > c?)
No
Yes
Swap
A[1] > A[2]؟
(b > c?)
No
cba
bca
cab
acb

چندتا لم و قضیه

لم: یک درخت دودویی با ارتفاع h حداکثر ۲^h برگ دارد.

این قضیه با استقرا روی h اثبات می‌شود (امتحان کنید).

چندتا لم و قضیه

لم: یک درخت دودویی با ارتفاع h حداکثر ۲^h برگ دارد.

این قضیه با استقرا روی h اثبات می‌شود (امتحان کنید).

نکته: حداقل چند تا برگ دارد؟ دودویی کامل و دودویی ؟

چندتا لم و قضیه

لم: یک درخت دودویی با ارتفاع h حداکثر ۲^h برگ دارد.

این قضیه با استقرا روی h اثبات می‌شود (امتحان کنید).

نکته: حداقل چند تا برگ دارد؟ دودویی کامل و دودویی ؟

جواب: ۱ برای درخت عادی و $h+۱$ برای درخت دودویی که هر گره دو یا صفر فرزند داشته باشد.

چندتا لم و قضیه

لم: یک درخت دودویی با ارتفاع h حداکثر ۲^h برگ دارد.

این قضیه با استقرا روی h اثبات می‌شود (امتحان کنید).

نتیجه: ارتفاع یک درخت تصمیم که n عنصر را مرتب می‌کند حداقل $\lceil \log n! \rceil$ است.

چندتا لم و قضیه

لم: یک درخت دودویی با ارتفاع h حداکثر ۲^h برگ دارد.

این قضیه با استقرا روی h اثبات می‌شود (امتحان کنید).

نتیجه: ارتفاع یک درخت تصمیم که n عنصر را مرتب می‌کند حداقل $\lceil \log n! \rceil$ است.

اثبات: این درخت تصمیم حداقل $n!$ برگ دارد، بنابراین ارتفاعش حداقل $\lceil \log n! \rceil$ است.

چندتا لم و قضیه

لم: یک درخت دودویی با ارتفاع h حداکثر ۲^h برگ دارد.

این قضیه با استقرا روی h اثبات می‌شود (امتحان کنید).

نتیجه: ارتفاع یک درخت تصمیم که n عنصر را مرتب می‌کند حداقل $\lceil \log n! \rceil$ است.

اثبات: این درخت تصمیم حداقل $n!$ برگ دارد، بنابراین ارتفاعش حداقل $\lceil \log n! \rceil$ است.

نتیجه: هر الگوریتم مقایسه‌ای که n عنصر را مرتب می‌کند در بدترین حالت حداقل $\lceil \lg n! \rceil$ مقایسه بین عناصر ورودی انجام می‌دهد.

اما می‌دانیم که ..

$n! \leq n^n$، پس $\lg n! \leq n \lg n$. ولی این کران بالای ضعیفی است. تقریب استرلینگ (Stirling's Approximation) کران به‌تری را می‌دهد. براساس این تقریب داریم:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

با استفاده از این فرمول می‌توان اثبات کرد که:

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\lg(n!) = \Theta(n \lg n)$$

کران پایین در حالت میانگین

یعنی چه؟ چه چیز را باید اثبات کنیم؟

کران پایین در حالت میانگین

قضیه: اگر کلیه‌ی جای‌گشت‌های یک ترتیب n تایی با احتمال یک‌سان در ورودی ظاهر شوند، آن‌گاه میانگین عمق برگ‌های درخت تصمیم حداقل $\lg n!$ خواهد بود.

کران پایین در حالت میانگین

اثبات: فرض

- $D(T)$ **مجموع عمق برگ‌های یک درخت دودویی** T **و**
- $D(m)$ **کوچک‌ترین مقدار** $D(T)$ **برای کلیه‌ی درخت‌های دودویی** T **با** m **برگ باشد.**

ثابت می‌کنیم $D(m) \geq m \lg m$، $\Longleftarrow$ **میانگین عمق برگ‌های درخت تصمیم** $\Omega(\lg n!)$

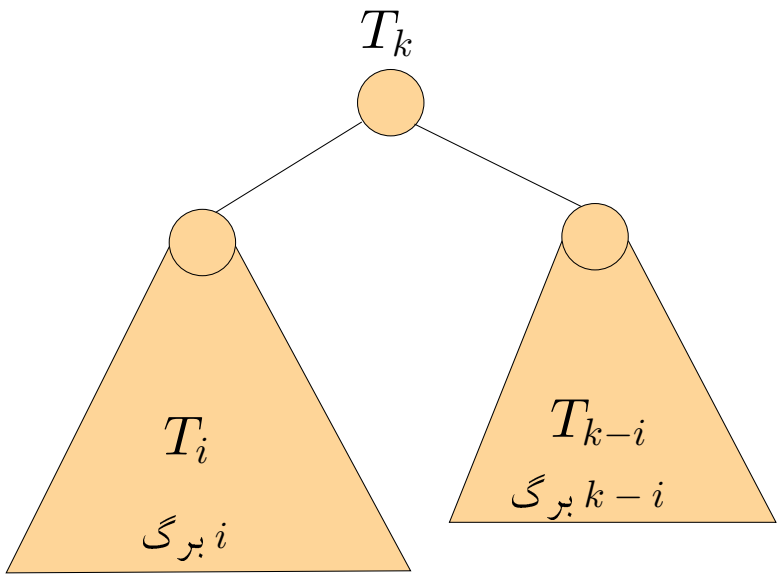
ادامه‌ی اثبات

با استقرا ثابت می‌کنیم که $D(m) \geq m \lg m$.

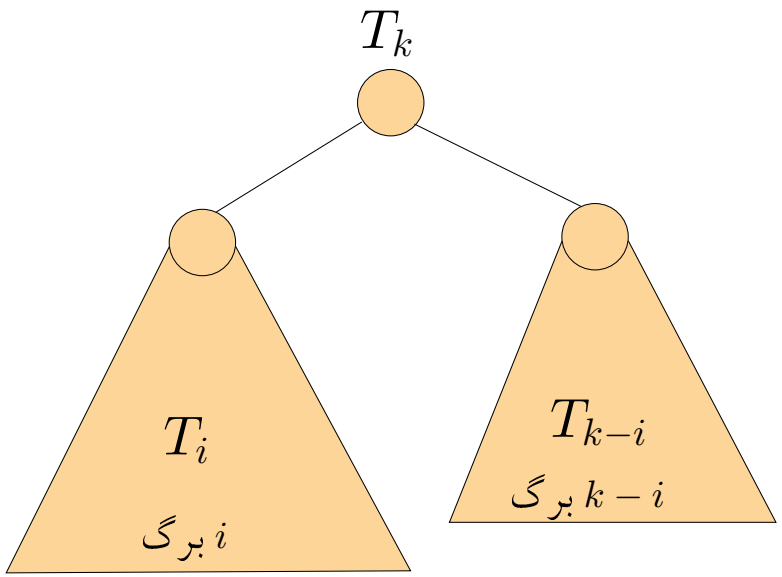
۱) پایه: $m = ۱$ **واضح است.**

۲) فرض: برای $m < k$ **درست است.**

۳) حکم: T **با** k **برگ را در نظر بگیرید (با** T_k **نشان می‌دهیم).**

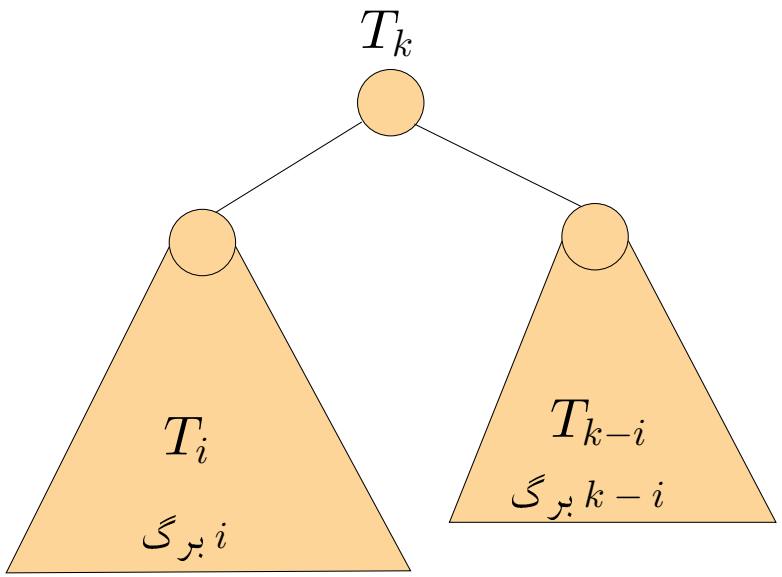


$$D(T) = i + D(T_i) + (k - i) + D(T_{k-i})$$



$$D(T) = i + D(T_i) + (k-i) + D(T_{k-i})$$

$$D(k) = \min_{1 \le i \le k} \{k + D(i) + D(k-i)\}$$



$$
\begin{aligned}
D(T) &= i + D(T_i) + (k-i) + D(T_{k-i}) \\
D(k) &= \min_{1 \le i \le k} \{k + D(i) + D(k-i)\} \\
D(k) &\ge k + \min_{1 \le i \le k} \{i \lg i + (k-i) \lg (k-i)\}
\end{aligned}
$$

ادامه‌ی اثبات

برای اعداد طبیعی مقدار کمینه‌ی $i \lg i + (k-i) \lg (k-i)$ **در** $i = \frac{k}{۲}$ **اتفاق می‌افتد.**

پس

$$D(k) \geq k + k \lg \frac{k}{۲} = k \lg k$$

مرتب‌سازی خطی

مرتب‌سازی شمارشی (Count Sort)

ورودی: n عنصر با کلیدهای بین ۱ تا m

```
COUNT-SORT(A, B, m)
 1  for i ← 1 to m
 2      do C[i] ← 0
 3  for i ← 1 to length[A]
 4      do C[A[i]] ← C[A[i]] + 1
 5  for i ← 2 to m
 6      do C[i] ← C[i] + C[i − 1]
 7  for i ← length[A] downto 1
 8      do B[C[A[i]] ← A[i]
 9         C[A[i]] ← C[A[i]] − 1
```

چرا درست است؟

چرا درست است؟

- در سطر ۴ تعداد عناصر با کلید یک‌سان را می‌شماریم.
- در پایان حلقه‌ی ۵، $C[i]$ آخرین اندیس آرایه‌ی B است که عناصر با کلید $A[i]$ در آن‌جا قرار می‌گیرند.
- در حلقه‌ی ۷ عناصر در جای خودشان قرار می‌گیرند. این کار به‌صورت پایدار انجام می‌شود.

تحلیل

- زمان اجرا $\mathcal{O}(n)$.
- دلیل آن‌که این حلقه از انتها به ابتدای A تکرار می‌شود آن است که الگوریتم پایدار شود.
- زمان اجرای کل الگوریتم $\mathcal{O}(n+m)$ می‌باشد که اگر m از $\mathcal{O}(n)$ باشد زمان اجرای کل $\mathcal{O}(n)$ خواهد بود.

حالت خاص

کلیدهای عناصر اعداد ۱ تا n هستند.

```
COUNT-SORT(A, n)
  1  for i ← 1 to n
  2      do while key[A[i]] ≠ i
  3             do SWAP(A[i], A[key[A[i]]])
```

چرا درست است؟

- الگوریتم به‌صورت «درجا» مرتب می‌کند.
- هر بار تعویض ← یک یا دو عنصر در جای نهایی خود.
- از هر کلید بیش‌از یک عدد ← الگوریتم ممکن است در حلقه بیفتد.

مرتب‌سازی مبنایی (Radix Sort)

d تعداد رقم‌های اعداد ورودی (رقم i ام بر $1-i$ ام اولویت دارد)

```
RADIX-SORT(A, d)
  1  for i ← 1 to d
  2      do sort array A on digit i by a stable sort
```

- اگر تعداد رقم‌های ورودی یکسان نباشند؟
- درستی الگوریتم: با استقرا بر روی i
  - در انتهای مرحله‌ی i عناصر برحسب بیت‌ها ۱ تا i کلیدشان مرتب‌اند.
- زمان اجرا به الگوریتم دوم بستگی دارد.
- اگر از مرتب‌سازی شمارشی استفاده کنیم: زمان اجرا $\mathcal{O}(dn)$

مرتب‌سازی مبنایی در حالت کلی

- ورودی آرایه‌ای از رکوردها، (یک دسته از کارت‌ها)
- هر رکورد دارای کلیدی با k مؤلفه‌ی $f_1 \dots f_k$ به‌ترتیب از داده‌گونه‌های $t_1 \dots t_k$،
- تعداد مقادیری که هر داده‌گونه t_i می‌تواند داشته باشد محدود و مستقل از n است.

مرتب‌سازی سطلی (Bucket Sort)

```
BUCKET-SORT(A)
    ▷ مرتب‌سازی سطلی که لیست A را مرتب می‌کند
 1  for i ← 1 to k
 2      do for each value v of type t_i
 3          do make B_i[v] empty
 4      for each record r in list A in order
 5          do let v be the value of f_i of the key for r
 6              move r from A to the end of B_i[v]
 7      for each value v of type t_i from lowest to highest
 8          do concatenate B_i[v] to the end of A
```

- **ورودی: لیست A با n رکورد، که هر رکورد دارای کلیدی با k مؤلفه به نام‌های $f_1 \dots f_k$ از داده‌گونه‌های $t_1 \dots t_k$**
- **تعداد حالت‌های t_i برابر s_i**
- **فرض: f_i بر f_{i-1} اولویّت دارد.**
- **برای $(1 \le i \le k)$ داریم:** B_i : array[t_i] of list-type.

- **اگر هر سطل را به‌صورت یک صف پیاده‌سازی کنیم، عمل الحاق (concat) در زمان ثابت قابل اجرا است.**
- **زمان اجرا این الگوریتم برابر است با:**

$$\sum_{i=۱}^{k} \Theta(s_i + n) = \Theta(kn + \sum_{i=۱}^{k} s_i)$$

اگر $s_i = \Theta(n)$ **آن گاه**

$$T(n) = \Theta(kn + \sum_{i=۱}^{k} n) = \Theta(n + kn) = \Theta(n)$$

مثال: کلیدها شامل سه مؤلفه با مقادیر a..z، ۱۰۰ . . . ۱ **و** ۱۴۰۰ . . . ۱۳۰۰.

عنصر	$f_۳$	$f_۲$	$f_۱$
$a_۱$	a	۵	۱۳۲۰
$a_۲$	c	۱۲	۱۳۱۰
$a_۳$	b	۱۲	۱۳۰۵
$a_۴$	a	۸	۱۴۰۰
$a_۵$	z	۱۰	۱۳۰۸
$a_۶$	b	۱۲	۱۳۰۴
$a_۷$	a	۶	۱۳۱۰

عنصر	$f_۳$	$f_۲$	$f_۱$
$a_۱$	a	۵	۱۳۲۰
$a_۲$	c	۱۲	۱۳۱۰
$a_۳$	b	۱۲	۱۳۰۵
$a_۴$	a	۸	۱۴۰۰
$a_۵$	z	۱۰	۱۳۰۸
$a_۶$	b	۱۲	۱۳۰۴
$a_۷$	a	۶	۱۳۱۰

ورودی	$B_۱[.]$	خروجی ۱	$B_۲[.]$	خروجی۲	$B_۳[.]$	خروجی ۳
$a_۱$	$[۱۳۰۴]\ a_۶$	$a_۶$	$[۵]\ a_۱$	$a_۱$	$[\text{'a'}]\ a_۱, a_۷, a_۴$	$a_۱$
$a_۲$	$[۱۳۰۵]\ a_۳$	$a_۳$	$[۶]\ a_۷$	$a_۷$	$[\text{'b'}]\ a_۶, a_۳$	$a_۷$
$a_۳$	$[۱۳۰۸]\ a_۵$	$a_۵$	$[۸]\ a_۴$	$a_۴$	$[\text{'c'}]\ a_۲$	$a_۴$
$a_۴$	$[۱۳۱۰]\ a_۲, a_۷$	$a_۲$	$[۱۰]\ a_۵$	$a_۵$	$[\text{'z'}]\ a_۵$	$a_۶$
$a_۵$	$[۱۳۲۰]\ a_۱$	$a_۷$	$[۱۲] a_۶, a_۳, a_۲$	$a_۶$		$a_۳$
$a_۶$	$[۱۴۰۰]\ a_۴$	$a_۱$		$a_۳$		$a_۲$
$a_۷$		$a_۴$		$a_۲$		$a_۵$

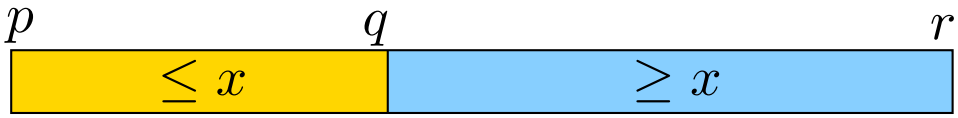
مرتب‌سازی مقایسه‌ای: مرتب‌سازی سریع

- مرتب‌سازی سریع n عنصر را در بدترین حالت با $\mathcal{O}(n)^2$ و در حالت‌میانگین در $\mathcal{O}(n \lg n)$ مرتب می‌کند.
- ضریب ثابت $n \lg n$ کاملاً کوچک است.
- این الگوریتم برای محیط‌های حافظه‌ی خارجی و موازی نیز کارا می‌باشد.

مرتب‌سازی سریع: توصیف الگوریتم

- مرتب‌سازی سریع مانند مرتب‌سازی ادغامی مبتنی بر روش تقسیم‌وحل است.
- می‌خواهیم آرایه‌ی $A[p..r]$ را مرتب کنیم.
- الگوریتم شامل سه مرحله‌ی زیر است.

۱) **تقسیم: بخش‌بندی** (partition) $A[p..r]$ **به دو بخش ناتهی** $A[p..q]$ **و** $A[q+۱..r]$**،**
به‌طوری‌که هر عنصر $A[p..q]$ **از هر عنصر** $A[q+۱..r]$ **بیش‌تر نباشد.**



۲) **حل: دو بخش** $A[p..q]$ **و** $A[q+۱..r]$ **به صورت بازگشتی مرتب می‌شوند.**

۳) **ترکیب: با توجه به ویژگی بخش‌ها، نیازی به ترکیب آن‌ها نیست و کل آرایه مرتب است.**

```
QuickSort (A, p, r)
 1  if p < r
 2    then q ← Partition(A, p, r)
 3         QuickSort (A, p, q)
 4         QuickSort(A, q + 1, r)
```

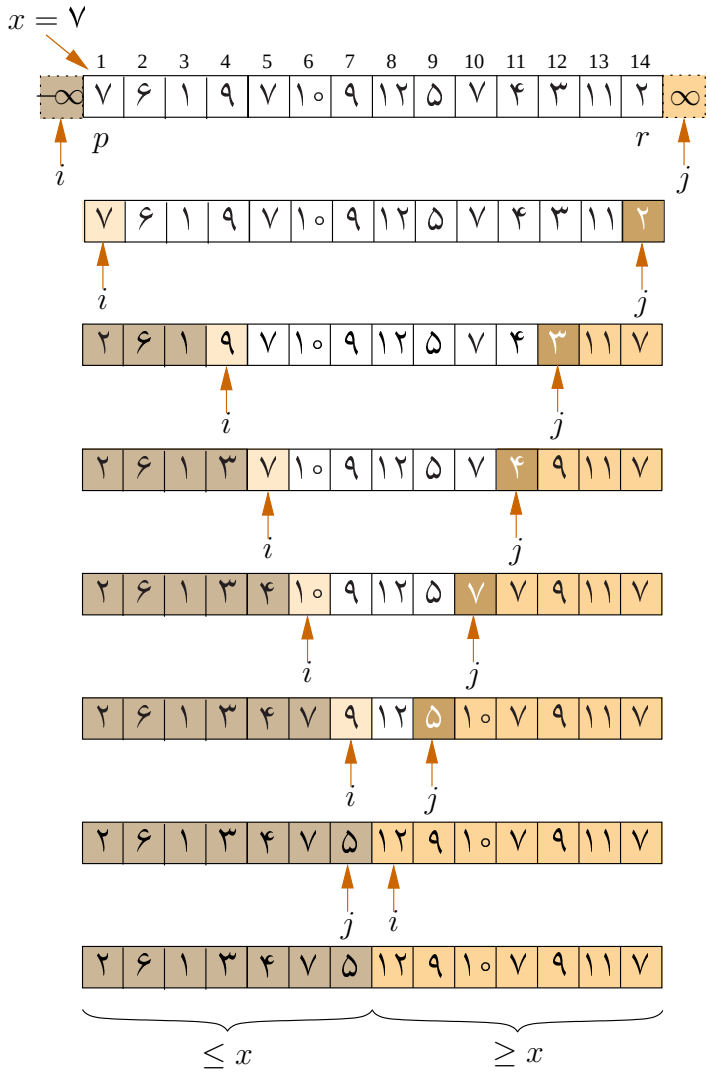
به‌صورت $\textsc{QuickSort}(A, ۱, length[A])$ فراخوانی می‌شود.

بخش‌بندی

بخش اصلی الگوریتم

```
PARTITION(A, p, r)
 1  x ← A[p]
 2  i ← p − 1
 3  j ← r + 1
 4  while true
 5        do repeat j ← j − 1
 6           until A[j] ≤ x
 7           repeat i ← i + 1
 8           until A[i] ≥ x
 9           if i < j
10             then SWAP(A[i], A[j])
11             elsereturn j
```

مثالی از بخش‌بندی


$x = ۷$
1 2 3 4 5 6 7 8 9 10 11 12 13 14
$-\infty$ ۷ ۶ ۱ ۹ ۷ ۱۰ ۹ ۱۲ ۵ ۷ ۴ ۳ ۱۱ ۲ ∞
p
r
i
j
۷ ۶ ۱ ۹ ۷ ۱۰ ۹ ۱۲ ۵ ۷ ۴ ۳ ۱۱ ۲
۲ ۶ ۱ ۹ ۷ ۱۰ ۹ ۱۲ ۵ ۷ ۴ ۳ ۱۱ ۷
۲ ۶ ۱ ۳ ۷ ۱۰ ۹ ۱۲ ۵ ۷ ۴ ۹ ۱۱ ۷
۲ ۶ ۱ ۳ ۴ ۱۰ ۹ ۱۲ ۵ ۷ ۷ ۹ ۱۱ ۷
۲ ۶ ۱ ۳ ۴ ۷ ۹ ۱۲ ۵ ۱۰ ۷ ۹ ۱۱ ۷
۲ ۶ ۱ ۳ ۴ ۷ ۵ ۱۲ ۹ ۱۰ ۷ ۹ ۱۱ ۷
۲ ۶ ۱ ۳ ۴ ۷ ۵ ۱۲ ۹ ۱۰ ۷ ۹ ۱۱ ۷
$\leq x$
$\geq x$

چرا بخش‌بندی درست است؟

ویژگی مستقل از حلقه (loop invariant):

- $i \leq j + ۱$
- **به‌ازای هر مقدار** i **و** j،
  - **بخش** $A[p - ۱, i - ۱]$ **حاوی عناصر کم‌تر یا مساوی** x
  - **بخش** $A[j + ۱, q + ۱]$ **حاوی عناصر بیش‌تر یا مساوی محور** x

اثبات این ویژگی در انتها، اثبات درستی بخش‌بندی است.

اثبات ویژگی مستقل حلقه

- در ابتدا درست است.
- فرض: قبل از حلقه‌ی ۴ درست است.
- j از همه‌ی عناصر بیش‌تر از x عبور می‌کند.
- i از همه‌ی عناصر کم‌تر از x عبور می‌کند.
- i نمی‌تواند بیش از یک عنصر از j عبور کند ($i \leq j + ۱$)
- انتهای حلقه، یا $i < j$، یا $i = i$، یا $i = j + ۱$

که ویژگی برقرار است.

حالت‌های خاص مرتب‌سازی سریع

- آرایه مرتب باشد
- آرایه برعکس مرتب باشد
- همه‌ی عناصر آرایه برابر باشد

تعداد مقایسه‌های کلیدها و تعداد تعویض‌ها را در هر حالت می‌توان دقیقاً شمرد.

مرتب‌سازی سریع (تحلیل الگوریتم)

- بدترین حالت $\Theta(n^2)$ و در حالت میانگین $\Theta(n \log n)$. این کارایی وابسته به نحوه‌ی بخش‌بندی است.
- هزینه‌ی بخش‌بندی برابر $\mathcal{O}(n)$ با ثابت کوچک است:
  - i و j به‌عقب بر نمی‌گردند
  - تعداد تعویض‌ها حداکثر برابر $n/2$ است.

تحلیل در بدترین حالت

- هنگامی که بخش‌بندی همیشه n عنصر را به $n-1$ و ۱ عنصر تقسیم کند
- پیچیدگی الگوریتم: $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$

تحلیل در به‌ترین حالت

شهود

- **بخش‌بندی درهر مرحله n عنصر را به دو آرایه با تعداد عناصر $\lceil \frac{n}{۲} \rceil$ و $\lfloor \frac{n}{۲} \rfloor$ تقسیم کند.**
- **پیچیدگی الگوریتم:**

$$T(n) = ۲T(\frac{n}{۲}) + \Theta(n) \rightarrow T(n) = \Theta(n \log n)$$

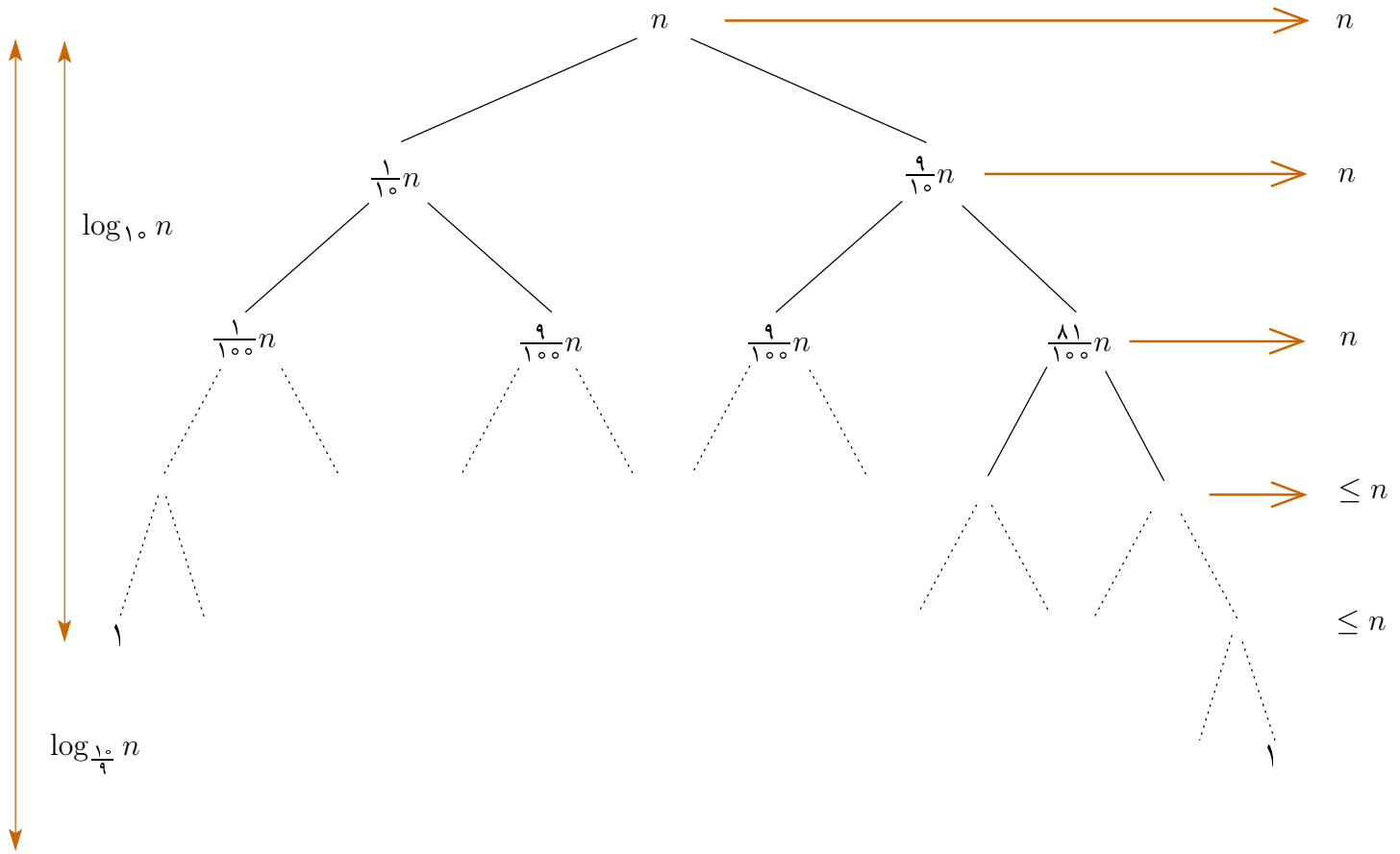
بخش‌بندی متوازن

- **برای هر ثابت** $0 < \alpha < 1$

$$T(n) = T(\alpha n) + T((1 - \alpha)n) + n$$

- **مثلاً** $T(n) = T(\frac{9n}{10}) + T(\frac{n}{10}) + n$
- **حل** $T(n) = \Theta(n \lg n)$ **(درخت بازگشت)**

درخت بازگشت برای $T(n) = T(\frac{۹n}{۱۰}) + T(\frac{n}{۱۰}) + n$



تحلیل در حالت میانگین

برای تحلیل الگوریتم در حالت میانگین، گونه‌ی تصادفی آن را در نظر می‌گیریم.

گونه‌ی تصادفی مرتب‌سازی سریع

به‌صورت تصادفی یک عنصر دل‌خواه را با عنصر اول تعویض می‌کنیم تا محور شود.

```
RANDOMIZED-PARTITION(A, p, r)
1  i ← RANDOM(p, r)
2  SWAP(A[p], A[i])
3  return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT (A, p, r)
 1  if p < r
 2    then q ← RANDOMIZED-PARTITION (A, p, r)
 3         RANDOMIZED-QUICKSORT(A, p, q)
 4         RANDOMIZED-QUICKSORT(A, q + 1, r)
```

تحلیل Randomized-Quicksort در بدترین حالت

- فرض: محور طوری انتخاب شود که آرایه n عضوی به دو بخش q عضوی و $n-q$ عضوی تقسیم می‌شود ($1 \le q \le n-1$).
- در بدترین حالت،

$$T(n) = \max_{1 \le q \le n-1} \{T(q) + T(n-q)\} + \Theta(n)$$

- اثبات می‌کنیم که $T(n) = \Theta(n^2)$

ادامه‌ی تحلیل در بدترین حالت

$$T(n) = \max_{1 \le q \le n-1} \{T(q) + T(n-q)\} + \Theta(n)$$

ابتدا با استقرا اثبات می‌کنیم که $T(n) = \mathcal{O}(n^2)$.

- **پایه: برای مقدار ثابت بدیهی است**
- **فرض: برای** $m < n$، $T(m) \le cm^2$
- **پس**

$$T(n) \le \max_{1 \le q \le n-1} \{cq^2 + c(n-q)^2\} + \Theta(n)$$

$$T(n) \leq \max_{1 \leq q \leq n-1} \{cq^2 + c(n-q)^2\} + \Theta(n)$$

- می‌دانیم که بیشینه‌ی $q^2 + (n-q)^2$ روی هر دو مرز ($q = 1$ و $q = n-1$) اتفاق می‌افتد (با رسم نمودار تابع و با درنظر گرفتن $1 \leq q \leq n-1$)
- پس در حالت $q = 1$ داریم:

$$T(n) \leq c(1 + (n-1)^2) + \Theta(n) = cn^2 - 2c(n-1) + \Theta(n)$$
$$T(n) \leq cn^2$$

به شرطی که c را آن‌قدر بزرگ بگیریم که $\Theta(n)$ را بپوشاند.

ادامه‌ی تحلیل در بدترین حالت

$$T(n) = \max_{۱ \le q \le n-۱} \{T(q) + T(n-q)\} + \Theta(n)$$

نشان می‌دهیم که $T(n) = \Omega(n^۲)$.

- **فرض:** $T(m) \ge cm^۲$ **برای** $m < n$
- **پس** $T(n) \ge \max_{۱ \le q \le n-۱} \{cq^۲ + c(n-q)^۲\} + \Theta(n)$
- **بنابراین،** $T(n) \ge cn^۲ - ۲c(n-۱) + \Theta(n) \ge cn^۲$

به‌شرطی که c **آن قدر کوچک باشد که** $\Theta(n) - ۲c(n-۱)$ **مثبت شود.**

- **پس** $T(n) = \Theta(n^۲) \Longleftarrow T(n) = \Omega(n^۲) = O(n^۲)$

تحلیل مرتب‌سازی سریع تصادفی در حالت میانگین

- فرض می‌کنیم محور با احتمال یکسان از بین n عنصر انتخاب می‌شود.
- احتمال این‌که مرتبه‌ی محور k باشد برابر $\frac{۱}{n}$ است.
- طبق این الگوریتم بخش‌بندی، در هر دو حالت $k = ۱$ و $k = ۲$، آرایه‌ی A به دو بخش با اندازه‌های ۱ و $n - ۱$ تقسیم می‌شود.

- **تعریف:** $\bar{T}(n)$: **میانگین زمان اجرا**

$$\bar{T}(n) = \frac{1}{n}\left[\bar{T}(1) + \bar{T}(n-1) + \sum_{q=1}^{n-1}[\bar{T}(q) + \bar{T}(n-q)]\right] + \Theta(n)$$

- **می‌دانیم که در بدترین حالت** $T(n-1) = \Theta(n^2)$، $T(1) = \Theta(1)$ **و** $\bar{T}(n) \leq T(n)$.
- **پس:**

$$\frac{1}{n}[\bar{T}(1) + \bar{T}(n-1)] \leq \frac{1}{n}[\Theta(1) + \Theta(n^2)] = \Theta(n)$$

- **بنابراین**

$$\bar{T}(n) = \frac{1}{n}\sum_{q=1}^{n-1}[\bar{T}(q) + \bar{T}(n-q)] + \Theta(n) = \frac{2}{n}\sum_{k=1}^{n-1}\bar{T}(k) + \Theta(n)$$

حل $\bar{T}(n) = \frac{۲}{n}\sum_{k=۱}^{n-۱}\bar{T}(k) + \Theta(n)$

- **حدس می‌زنیم** $\bar{T}(n) \le an\lg n + b$ **برای** $a, b > ۰$

$$\begin{aligned}
\bar{T}(n) &= \frac{۲}{n}\sum_{k=۱}^{n-۱}\bar{T}(k) + \Theta(n) \\
&\le \frac{۲}{n}\sum_{k=۱}^{n-۱}[ak\lg k + b] + \Theta(n) \\
&= \frac{۲}{n}\sum_{k=۱}^{n-۱} ak\lg k + \frac{۲}{n}\sum_{k=۱}^{n-۱} b + \Theta(n) \\
&= \frac{۲a}{n}\sum_{k=۱}^{n-۱} k\lg k + \frac{۲b}{n}(n-۱) + \Theta(n)
\end{aligned}$$

و از تقریب انتگرال ثابت می‌شود که:

$$\sum_{k=۱}^{n-۱} k \lg k \leq \int_{x=۱}^{n-۱} x \lg x = \frac{۱}{۲} n^۲ \lg n - \frac{۱}{۸} n^۲$$

پس

$$\begin{aligned} \bar{T}(n) &\leq \frac{۲a}{n}(\frac{n^۲}{۲} \lg n - \frac{n^۲}{۸}) + \frac{۲b}{n}(n-۱) + \Theta(n) \\ &\leq an \lg n + b + [\Theta(n) + b - \frac{an}{۴}] \end{aligned}$$

a و b را طوری انتخاب می‌کنیم تا درون پرانتز منفی شود.

بنابراین $\bar{T}(n) \leq an \lg n + b$.

یا $\bar{T}(n) = O(n \lg n)$

مرتب‌سازی با کم‌ترین تعداد مقایسه برای تعداد عناصر کم

- حداقل تعداد مقایسه‌ی مورد نیاز (بین عناصر) برای مرتب‌سازی n عنصر $\lceil \lg n! \rceil$ است.
- برای nهای کوچک آیا الگوریتمی با کم‌ترین تعداد مقایسه‌ی ممکن وجود دارد؟
- برای n برابر ۱، ۲، ۳ مرتب‌سازی «درج دودویی» (binary insertion sort) تعداد مقایسه‌ی بهینه دارد (برابر ۰، ۱، ۳)
- ولی برای $n > ۳$ بهینه نیست.

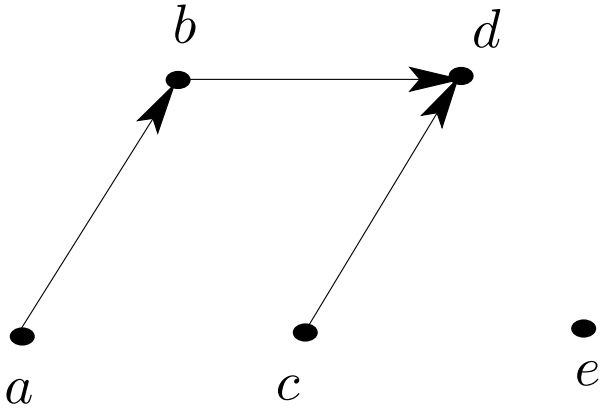
مرتب‌سازی درج دودویی

اگر $B(n)$ تعداد دقیق مقایسه‌های الگویتم درج دودویی (در بدترین حالت ورودی) برای مرتب‌سازی n عنصر باشد، مقادیر $B(n)$ برای nهای کوچک و مقایسه‌ی آن با $\lceil \lg n! \rceil$ در جدول زیر آمده است.

$n =$	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵	۱۶	۱۷
$\lceil \lg n! \rceil =$	۰	۱	۳	۵	۷	۱۰	۱۳	۱۶	۱۹	۲۲	۲۶	۲۹	۳۳	۳۷	۴۱	۴۵	۴۹
$B(n) =$	۰	۱	۳	۵	۸	۱۱	۱۴	۱۷	۲۱	۲۵	۲۹	۳۳	۳۷	۴۱	۴۵	۴۹	۵۴

مرتب‌سازی بهینه‌ی ۵ عنصر

آیا الگوریتمی با ۷ مقایسه برای مرتب‌سازی ۵ عنصر (در بدترین حالت ورودی) وجود دارد؟



پس از ۳ مقایسه

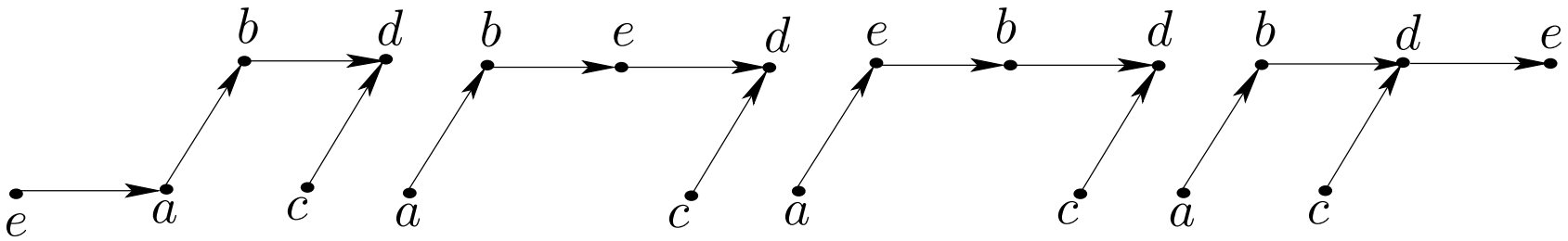
a با b (فرض b بزرگ‌تر است)

d با c (فرض c بزرگ‌تر است)

b با c (فرض c بزرگ‌تر است)

ابتدا e را در زنجیره‌ی $a \to b \to c$ (با ۲ مقایسه در بدترین حالت) درج می‌کنیم.

حالات مختلف پس از درج e



با حداکثر ۳ مقایسه‌ی دیگر d در زنجیره درج می‌شود.

اگر این عناصر K_1 تا K_5 باشند، روش این کار مطابق زیر است:

۱) K_1 را با K_2 و K_3 را با K_4 مقایسه کن و عناصر کوچک‌تر و بزرگ‌تر را پیدا کن.

۲) با یک مقایسه‌ی دیگر دو عنصر کوچک بند فوق را با هم مقایسه کن. در انتهای این مرحله ۵ عنصر به‌صورت شکل در می‌آید. اگر این عناصر a تا e باشند، داریم: $a \leq b \leq d$ و $c \leq d$.

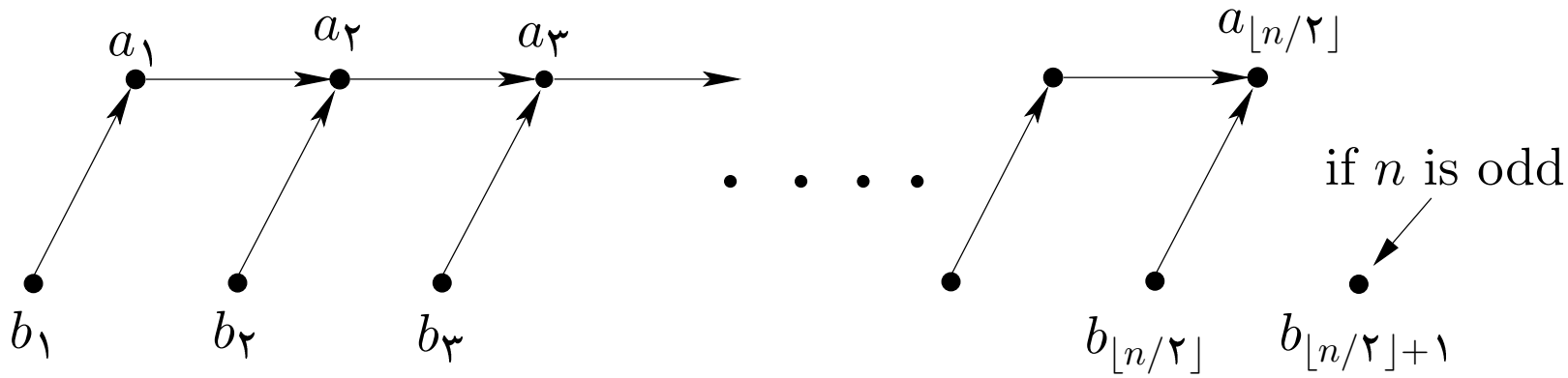
۳) e را در زنجیره‌ی $a < b < d$ به روش دودویی درج کن، این کار حداکثر ۲ مقایسه نیاز دارد و یکی از حالات شکل قبل حاصل می‌شود.

۴) با توجه به‌این که $c < d$، درج c در زنجیره‌ی چهارتایی از عناصری که در مراحل بالا مرتب شده‌اند، به‌روش دودویی حداکثر ۲ مقایسه دیگر نیاز دارد.

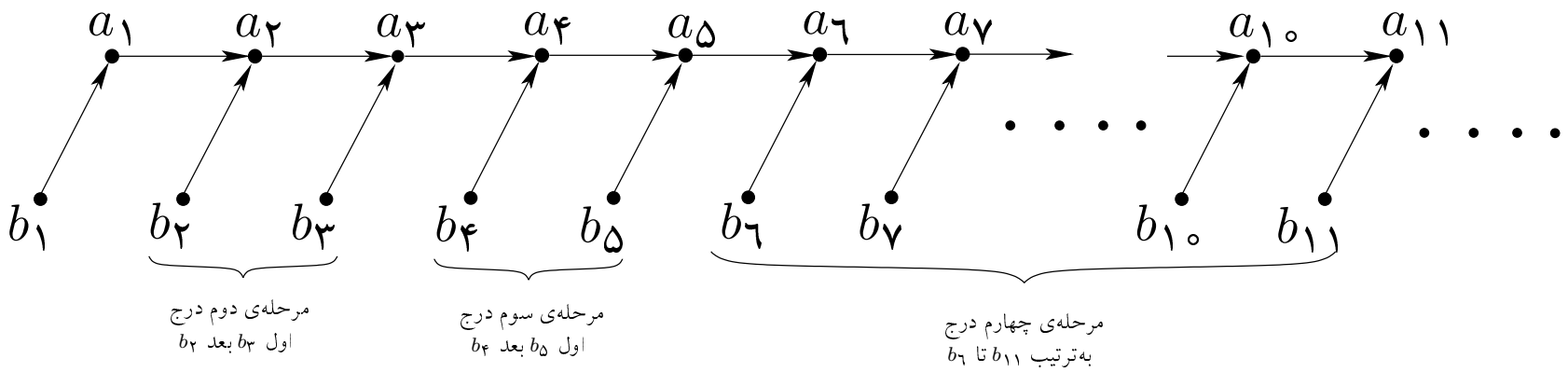
بنابراین این کار حداکثر ۷ مقایسه نیاز دارد.

تعمیم الگوریتم (الگوریتم فورد-جانسون)

الگوریتم بالا به‌طرز جالبی توسط آقایان فورد (Lester Ford Jr.) و جانسون (Slemer Johnson) در سال ۱۹۵۹ تعمیم داده شد. ایده‌ی اصلی این الگوریتم تشکیل یک «زنجیره‌ی اصلی» از عناصری است که مرتب هستند و تعیین یک ترتیب مشخص برای درج دیگر عناصر در این زنجیره به‌طوری‌که در انتها زنجیره‌ی اصلی شامل همه‌ی عناصر شود.



نحوه‌ی درج در زنجیره‌ی اصلی در مرحله‌ی k ام



نحوه‌ی درج در زنجیره‌ی اصلی در مرحله‌ی kام.

$$2t_{k-1} + (t_k - t_{k-1}) - 1 = t_k + t_{k-1} - 1 = 2^k - 1$$

الگوریتم برای مرتب‌سازی n عنصر به‌صورت زیر است:

۱) عناصر را به $\lceil \frac{n}{۲} \rfloor$ زوج‌عنصر و احتمالاً یک عنصر اضافی (اگر n فر باشد) تقسیم کن.

۲) هر زوج‌عنصر را با هم مقایسه کن و آن‌ها را مرتب‌کن.

۳) به‌صورت بازگشتی $\lceil \frac{n}{۲} \rfloor$ عنصر کوچک‌تر (از زوج‌عناصر) را مرتب کن و پس از آن عناصر را مطابق شکل نام‌گذاری کن. عناصر $a_{\lceil \frac{n}{۲} \rfloor} \leq \ldots \leq a_۲ \leq a_۱$ را «زنجیره‌ی اصلی» می‌نامیم.

۴) عناصر b در شکل را با ترتیب زیر به بخش اولیه‌ی زنجیره‌ی اصلی به‌روش دودویی درج کن. این ترتیب طوری انتخاب شده است که تعداد عناصر بخش اول زنجیره‌ی اصلی $۲^k - ۱$ باشد تا با k مقایسه بتوان زنجیره‌ی کامل را ایجاد کرد:

a) ابتدا $b_۳$ را به زنجیره‌ی $b_۱ \leq a_۱ \leq a_۲$ و سپس $b_۲$ را به زنجیره‌ی $b_۱ \leq a_۱$ و احتمالاً $b_۳$ درج کن. تعداد عناصر زنجیره حداکثر ۳ و درج با ۲ مقایسه امکان‌پذیر است.

b) ابتدا b_5 را به زنجیره‌ی شامل ۷ عنصر a_1 تا a_4 و b_1 تا b_3 درج کن. سپس b_4 را به زنجیره‌ی a_1 تا a_3، b_1 تا b_3 و احتمالاً b_4 درج کن. تعداد عناصر زنجیره حداکثر ۷ و هر درج با ۳ مقایسه امکان‌پذیر است.

c) این روند را ادامه بده. در حالت کلی برای $t_1, t_2, \ldots = 1, 3, 5, 11, \ldots$، در مرحله‌ی kام هر عنصر b_{t_k} تا $b_{t_{k-1}+1}$ را به‌ترتیب در زنجیره‌ی عناصری که حتماً از آن عنصر کوچک‌ترند درج کن. آخرین مرحله شامل عنصر $b_{\lfloor \frac{n}{2} \rfloor + 1}$ (در صورت وجود) می‌شود.

محاسبات و تحلیل الگوریتم

t_kها را طوری تعیین کنیم که

$$۲t_{k-۱} + (t_k - t_{k-۱}) - ۱ = t_k + t_{k-۱} - ۱ = ۲^k - ۱$$

می‌دانیم که $t_۰ = ۱$

پس کافی است رابطه‌ی بازگشتی $t_k = ۲^k - t_{k-۱}$ را برای $k > ۰$ حل کنیم.

با بازکردن و جای‌گذاری روشن است که

$$t_k = \sum_{i=۰}^{k} (-۱)^{k-i} ۲^i$$

با استفاده از

$$۴t_k = \sum_{i=۰}^{k} (-۱)^{k-i} ۲^{i+۲} = \sum_{i=۲}^{k+۲} (-۱)^{k-i} ۲^i$$

داریم

$$\begin{aligned} ۴t_k - t_k &= ۲^{k+۲} - ۲^{k+۱} - (-۱)^{k-۱} ۲^۱ - (-۱)^k ۲^۰ \\ &= ۲^{k+۱} - \left[۲(-۱)^{k-۱} + (-۱)^k\right] \\ &= ۲^{k+۱} - (-۱)^{k-۱} \\ &= ۲^{k+۱} + (-۱)^k \end{aligned}$$

و از آن داریم:

$$t_k = \frac{۱}{۳}[۲^{k+۱} + (-۱)^k]$$

تحلیل الگوریتم

تعداد مقایسه‌ها $F(n)$

$$F(n) = \lfloor \frac{n}{۲} \rfloor + F(\lfloor \frac{n}{۲} \rfloor) + G(\lceil \frac{n}{۲} \rceil)$$

$G(\lceil \frac{n}{۲} \rceil)$ = **حداکثر تعداد مقایسه‌های لازم برای درج عناصر** b **در زنجیره‌ی اصلی**

اگر $t_{k-۱} \leq m < t_k$

می‌دانیم که درج هر عنصر b_k **که** $t_{j-۱} \leq k \leq t_j$ **حداکثر** j **مقایسه نیاز دارد. پس**

$$G(m) = \sum_{j=۱}^{k-۱} j(t_j - t_{j-۱}) + k(m - t_{k-۱})$$

و داریم

$$\begin{aligned}\sum_{j=۱}^{k-۱} j(t_j - t_{j-۱}) &= [(k-۱)t_{k-۱} - (k-۱)t_{k-۲}] + \\ &\quad [(k-۲)t_{k-۲} - (k-۲)t_{k-۳}] + \ldots + (t_۱ - t_۰) \\ &= (k-۱)t_{k-۱} - t_{k-۲} - t_{k-۳} - \ldots - t_۱ - t_۰\end{aligned}$$

و بنابراین

$$G(m) = km - (t_{k-۱} + t_{k-۲} + \ldots + t_۱ + t_۰) \leq km$$

برای یک مقدار $k \geq ۱$ داریم

$$m \geq t_{k-۱} = (۲^k + (-۱)^{k-۱})/۳ \geq ۲^{k-۲}$$

پس $k \leq \lg m + ۲$

بنابراین

$$G(m) \leq km \leq m \lg m + ۲m$$

پس

$$F(n) \leq \frac{n}{۲} + F(n/۲) + \lceil \frac{n}{۲} \rceil \lg \lceil \frac{n}{۲} \rceil + ۲\lceil \frac{n}{۲} \rceil \leq F(\frac{n}{۲}) + \frac{n}{۲} \lg n + O(n)$$

که نتیجه می‌گیریم

$$n \lg n \leq F(n) \leq n \lg n + O(n)$$

برای محاسبه‌ی دقیق‌تر، فرض کنید

$$w_k = t_۰ + t_۱ + \cdots + t_{k-۱} = \lfloor \frac{۲^{k+۱}}{۳} \rfloor$$

در این صورت،

$$(w_۰, w_۱, w_۲, w_۳, w_۴, \cdots) = (۰, ۱, ۲, ۵, ۱۰, ۲۱, \cdots)$$

می‌توان اثبات کرد که

$$F(n) - F(n-۱) = k \quad \Longleftrightarrow \quad w_k < n \leq w_{k+۱}$$

و شرط آخر معادل است با

$$\frac{۲^{k+۱}}{۳} < n < \frac{۲^{k+۲}}{۳} \implies k+۱ < \lg(۳n) \leq k+۲ \implies k = \lg(\frac{۳}{۴}n)$$

بنابراین

$$F(n) - F(n-۱) = \lceil \lg(\frac{۳}{۴}n) \rceil$$

و جواب این رابطه‌ی بازگشتی به‌صورت زیر است:

$$F(n) = \sum_{k=۱}^{n} \lceil \lg(\frac{۳}{۴}k) \rceil$$

کارایی الگوریتم فورد–جانسون

$n =$	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴	۱۵	۱۶	۱۷
$\lceil \lg n! \rceil =$	۰	۱	۳	۵	۷	۱۰	۱۳	۱۶	۱۹	۲۲	۲۶	۲۹	۳۳	۳۷	۴۱	۴۵	۴۹
$F(n) =$	۰	۱	۳	۵	۷	۱۰	۱۳	۱۶	۱۹	۲۲	۲۶	۳۰	۳۴	۳۸	۴۲	۴۶	۵۰

$n =$	۱۸	۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵	۲۶	۲۷	۲۸	۲۹	۳۰	۳۱
$\lceil \lg n! \rceil =$	۵۳	۵۷	۶۲	۶۶	۷۰	۷۵	۸۰	۸۴	۸۹	۹۴	۹۸	۱۰۳	۱۰۸	۱۱۳
$F(n) =$	۵۴	۵۸	۶۲	۶۶	۷۱	۷۶	۸۱	۸۶	۹۱	۹۶	۱۰۱	۱۰۶	۱۱۱	۱۱۶

جدول ۱: تعداد مقایسه‌های الگوریتم فورد–جانسون $F(n)$ برای تعداد کم عناصر و مقایسه‌ی آن با کران پایین.

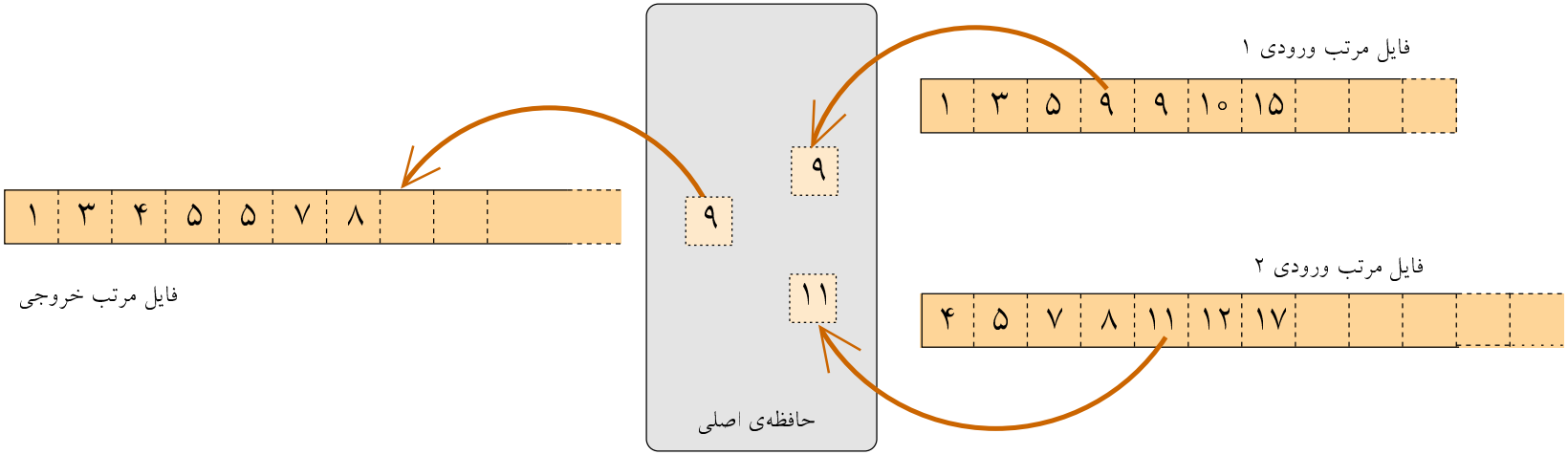
مرتب‌سازی خارجی (فرض)

- اطلاعات بر روی فایل‌ها به‌صورت ترتیبی ذخیره شده است.
- هر فایل شامل n رکورد است. هر رکورد یک کلید دارد.
- می‌خواهیم در فایل خروجی رکوردها براساس کلیدهایشان مرتب باشند.
- با هر دسترسی به دیسک k رکورد خوانده می‌شود.
- تعداد فایل‌هایی که در یک زمان باز هستند r و محدود است.
- تعداد حافظه‌ی اصلی قابل استفاده ثابت است.
- عملیات مقایسه و محاسبات فقط می‌تواند در حافظه‌ی اصلی انجام شود.

معیار کارایی: تعداد دسترسی‌ها به دیسک

مرتب‌سازی خارجی ادغامی (External Merge Sort)

ادغام دو قطعه‌ی مرتب



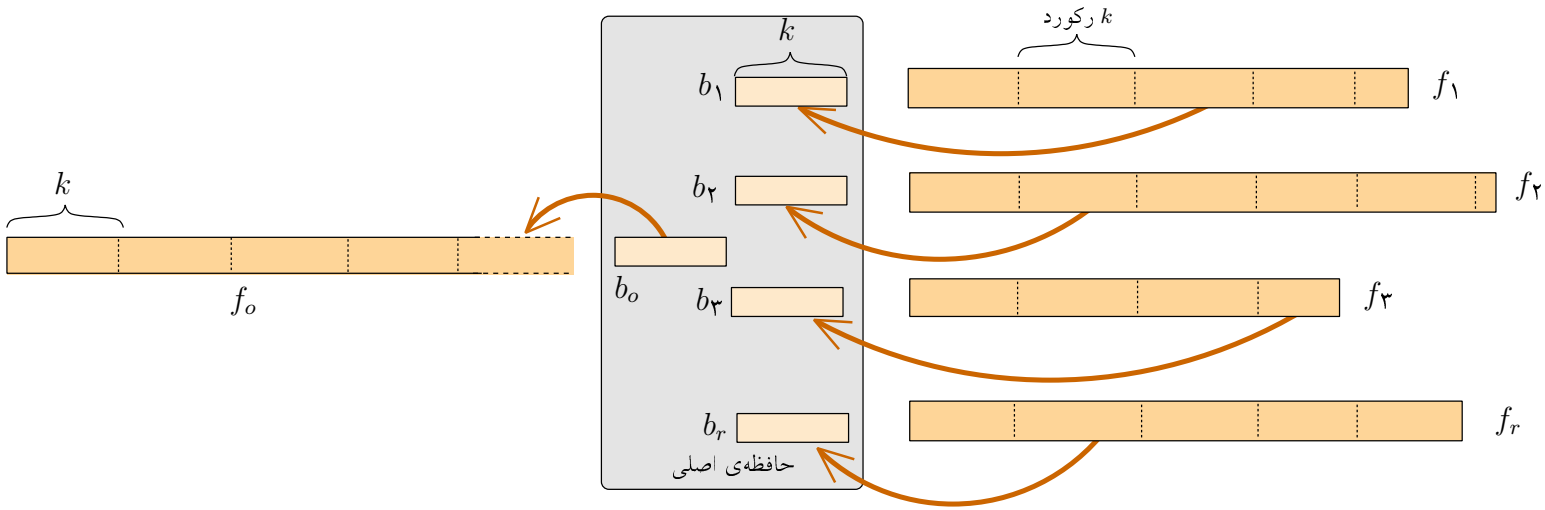
قطعه‌ی اول n_1 رکورد و قطعه‌ی دوم n_2 رکورد

$k = 1$، با $n_1 + n_2$ بار خواندن و همین تعداد نوشتن می‌توان ادغام را انجام داد.

ولی در حالت کلی با $\lceil \frac{n_1}{k} \rceil + \lceil \frac{n_2}{k} \rceil$ بار.

مقدار حافظه‌ی مورد نیاز: به‌اندازه‌ی $3k$ رکورد.

ادغام چند فایل مرتب

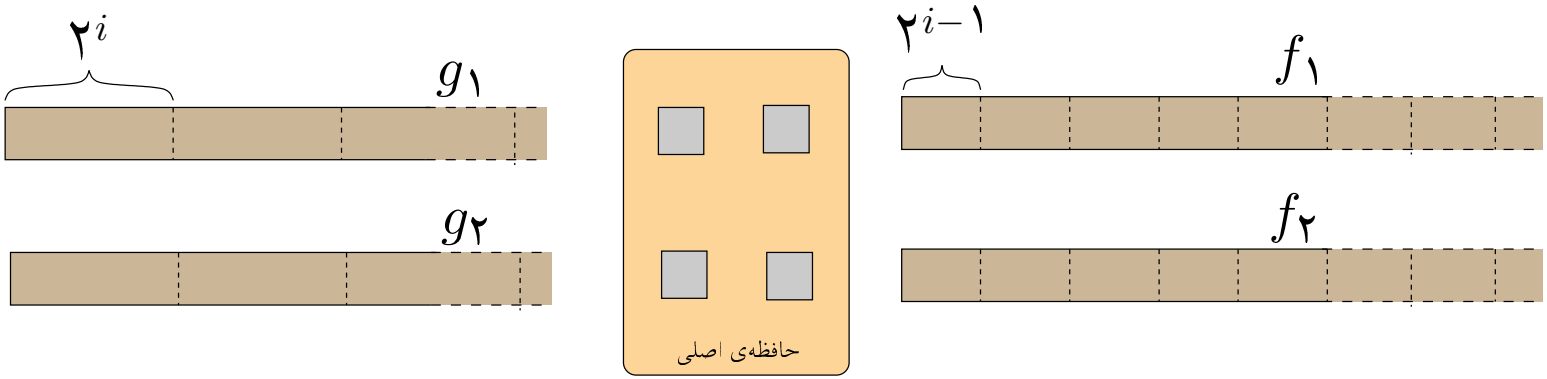


f_i به‌اندازه‌ی n_i رکورد دارد.

تعداد دسترسی‌ها: $\sum_{i=1}^{r} \lceil \frac{n_i}{k} \rceil$ بار.

مقدار حافظه‌ی مورد نیاز: به‌اندازه‌ی $(r+1)k$ رکورد.

مرتب‌سازی خارجی ادغامی



مرتب‌سازی خارجی ادغامی (الگوریتم کلی)

برای $k = ۱$ بیان می‌شود. چهار فایل $f_۱$، $f_۲$، $g_۱$ و $g_۲$ احتیاج است.

۱) فایل ورودی را به دو فایل $f_۱$ و $f_۲$ با حداکثر تعداد یک رکورد اختلاف تقسیم کن.

۲) برای $i = ۱, \ldots, M$ مراحل زیر را تکرار کن:

در این مرحله فرض می‌کنیم که $f_۱$ و $f_۲$ (یا $g_۱$ و $g_۲$) شامل قطعاتی به طول $۲^{i-۱}$ هستند و هر قطعه مرتب است و تعداد قطعات دو فایل ورودی حداکثر یک واحد اختلاف دارد.

۲-۱) f_1 و f_2 را به‌صورت فایل‌های ورودی در نظر می‌گیریم. قطعات با شماره‌های یک‌سان f_1 و f_2 را با یک‌دیگر ادغام کن و قطعه‌ای به طول دو برابر ایجاد کن. حاصل این ادغام قطعاتی مرتب به‌طول 2^i (به‌جز حداکثر یک قطعه به‌طول کم‌تر) است این قطعات را به‌ترتیب یک‌بار در g_1 و بار دیگر در g_2 بنویس.

۲-۲) g_1 و g_2 را به عنوان فایل‌های ورودی و f_1 و f_2 را به عنوان فایل‌های خروجی در نظر بگیر و مرحله‌ی بالا را تکرار کن.

مثال مرتب‌سازی خارجی ادغامی

درستی و تحلیل

- با استقرا می‌توان نشان داد که در انتهای مرحله‌ی i هر فایل خروجی دارای قطعه‌هایی مرتب و به طول ۲^i است، به‌جز حداکثر یک قطعه که طولش از ۲^i کم‌تر است. هم‌چنین تعداد قطعه‌های دو فایل خروجی حداکثر یک واحد اختلاف دارند.
- بنابراین برای $M = \lceil \log n \rceil$ (تعداد تکرار حلقه) یکی از فایل‌های خروجی حاوی یک قطعه‌ی مرتب شامل تمام n رکورد فایل ورودی و دیگری خالی است.
- با توجه به‌این‌که در هر تکرار همه‌ی n رکورد یک‌بار خوانده و یک‌بار نوشته می‌شوند، تعداد دسترسی به دیسک در مجموع برابر $۲n(\lceil \log n \rceil + ۱)$ است ($۲n$ بار خواندن و نوشتن برای تقسیم فایل اصلی).

- **برای حالت $k > ۱$، این تعداد برابر $۲\lceil \frac{n}{k} \rceil (\lceil \log n \rceil + ۱)$ خواهد بود.**

- با تقسیم فایل وردوی به r فایل با اندازه‌هایی یکسان و با استفاده از r حافظه نیز می‌توان فایل را مرتب کرد در آن صورت، تعداد دست‌رسی به دیسک $2n(\lceil \log_r n \rceil + 1)$ می‌شود و در حالت کلی برابر $2\lceil \frac{n}{k} \rceil (\lceil \log_r n \rceil + 1)$.
- در حالت کلی به حافظه‌ای به‌اندازه‌ی $2rk$ نیاز است.
- حالت کلی را مرتب‌ساز ادغامی چندگانه (Multiway Merge) می‌گوییم.

مرتب‌سازی خارجی چندفازه (Polyphase)

مثال: $n = ۳۴$

پس از گام	$f_۱$	$f_۲$	$f_۳$
ابتدا	۱۳(۱)	۲۱(۱)	--
۱	--	۸(۱)	۱۳(۲)
۲	۸(۳)	--	۵(۲)
۳	۳(۳)	۵(۵)	--
۴	--	۲(۵)	۳(۸)
۵	۲(۱۳)	--	۱(۸)
۶	۱(۱۳)	۱(۲۱)	--
۷	--	--	۱(۳۴)

مرتب‌سازی خارجی چندفازه

- سه (در حالت کلی به $r+1$) فایل لازم داریم.
- به فایل ورودی کم‌ترین تعداد رکورد با کلید ∞ اضافه کن تا n برابر F_i (iامین عدد فیبانوچی) شود.
- فایل ورودی را به دو فایل با اندازه‌های F_{i-1} و F_{i-2} تقسیم کن ($F_i = F_{i-1} + F_{i-2}$)

- باندازه‌ی $M =?$ بار تکرار کن

-- از سه‌فایل f_1 دارای F_m قطعه‌ی مرتب (در ابتدا $F_m = F_i$) است که اندازه‌ی هر قطعه‌ی آن F_r است (در ابتدا $F_r = F_0 = 1$).

هم‌چنین f_2 دارای F_{m+1} قطعه‌ی مرتب است که اندازه‌ی هر قطعه‌ی آن F_{r+1} است. و f_3 خالی است.

-- F_m قطعه‌ی مرتب از فایل f_1 را با همین تعداد قطعه‌ی مرتب از فایل f_2 را با هم ادغام کن و حاصل را در f_3 بنویس.

-- حال f_1 خالی، f_2 دارای $F_{m-1} = F_{m+1} - F_m$ قطعه به اندازه‌ی F_{r+1} و f_3 دارای F_m قطعه‌ی مرتب به اندازه‌ی $F_{r+2} = F_r + F_{r+1}$ است.

-- نام‌گذاری فایل‌ها را به‌تناسب تغییر بده.

اثبات درستی و مقدار M

- **رابطه‌ی** $a(b) + c(d) \longrightarrow e(f) + g(h)$

یعنی یک فایل حاوی a قطعه‌ی مرتب هرکدام به‌اندازه‌ی b، با فایلی که c قطعه‌ی مرتب هرکدام به‌اندازه‌ی d دارد ادغام می‌شوند و دو فایل حاوی e و f قطعه‌ی مرتب به‌اندازه‌های f و h ایجاد می‌کنند.

- **ویژگی مستقل از حلقه:**

-- در هر مرحله اعداد a تا h فوق اعداد فیبوناچی هستند.

-- در ابتدا و انتهای هر مرحله یکی از فایل‌ها تهی است.

-- اگر F_i کوچک‌ترین عدد فیبوناچی بزرگ‌تر یا مساوی n باشد، در گام $i-r-1$ام (برای $3 \le r \le i-2$) از الگوریتم، رابطه‌ی زیر برقرار است:

$$F_r(F_{i-r+1}) + F_{r-1}(F_{i-r}) \longrightarrow F_{r-1}(F_{i-r+2}) + F_{r-2}(F_{i-r+1})$$

- **پایه:**

$$F_{i-۱}(F_۲) + F_{i-۲}(F_۱) \longrightarrow F_{i-۲}(F_۳) + F_{i-۳}(F_۲)$$

- **چون $F_{i-۲} < F_{i-۱}$، به‌تعداد $F_{i-۲}$ رکورد از دو فایل ورودی می‌خوانیم و در یکی از فایل‌های خروجی می‌نویسیم.**
- **تعداد قطعات این فایل خروجی برابر $F_{i-۲}$ و اندازه‌ی هر قطعه‌ی آن برابر $F_۱ + F_۲ = F_۳$ خواهد بود.**
- **از فایل بزرگ‌تر به‌تعداد $F_{i-۱} - F_{i-۲} = F_{i-۳}$ قطعه به‌اندازه‌ی $F_۲$ باقی می‌ماند.**
- **با فرض درستی پایه‌ی استقرا، می‌توان فرمول بالا را ثابت کرد.**

- در انتهای گام $i-۴$ام ($r = ۳$ برای) دو فایل خروجی هر کدام یک قطعه دارند و اندازه‌هایشان به‌ترتیب برابر $F_{i-۱}$ و $F_{i-۲}$ است.
- پس، در انتهای گام $i-۳$ام همه‌چیز در یک فایل مرتب می‌شود.
- تعداد گام‌های الگوریتم برابر $i-۴$ است و در گام $i-r-۱$ام الگوریتم به‌تعداد $۲F_{i-۲}F_{i-r+۲}$ رکورد می‌خواند و می‌نویسد.
- با احتساب تقسیم اولیه‌ی فایل که به‌اندازه‌ی $۲F_i$ دست‌رسی به دیسک نیاز دارد، در مجموع الگوریتم به

$$۲F_i + \sum_{r=۳}^{i-۲} ۲F_{i-۲}F_{i-r+۲}$$

حافظه‌ی خارجی دست‌رسی خواهد داشت.

مرتبه‌ی آماری

کوچک‌ترین و بزرگ‌ترین عنصر

با $\lceil ۲ - \frac{۳n}{۲} \rceil$ مقایسه

دو کوچک‌ترین عناصر

راه‌حل بدیهی: با $n-1+n-2$ مقایسه

راه‌حل به‌تر:

- **ایجاد یک درخت مقایسه با n برگ و ارتفاع $\lceil \lg n \rceil$ برای تعیین کوچک‌ترین عنصر در ریشه (که دقیقاً $n-1$ مقایسه نیاز دارد، برابر تعداد گره‌های داخلی این درخت با n برگ)**

- دومین کوچک‌ترین عنصر حتماً در این درخت با اولین عنصر مقایسه شده است.
- پس دومین کوچک‌ترین عنصر، کوچک‌ترین عنصر بین $\lceil \lg n \rceil$ عنصر است
- با $\lceil \lg n \rceil - ۱$ مقایسه دومین کوچک‌ترین عنصر یافت می‌شود.
- در مجموع $n - ۱ + \lceil \lg n \rceil - ۱$ مقایسه.

تعمیم راه‌حل قبل برای ۳ کوچک‌ترین عناصر:

- سومین کوچک‌ترین عنصر یا با دومین و یا اولین کوچک‌ترین عنصر مقایسه شده است.
- اولین کوچک‌ترین عنصر حداکثر با $\lceil \lg n \rceil - ۱$ عنصر (غیر از کوچک‌ترین عنصر) مقایسه شده است.
- دومین عنصر حداکثر با $\lceil \lg n \rceil - ۲$ عنصر (غیر از کوچک‌ترین و دومین کوچک‌ترین عنصر) مقایسه شده است.
- پس باید کوچک‌ترین عنصر بین حداکثر $۲\lceil \lg n \rceil - ۳$ عنصر را پیدا کنیم که حداکثر $۲\lceil \lg n \rceil - ۴$ مقایسه نیاز داد

پس در مجموع حداکثر $n + ۳\lceil \lg n \rceil - ۴ = n + ۲\lceil \lg n \rceil - ۳ + (\lceil \lg n \rceil - ۱) + n - ۱$ مقایسه

نیاز است..

تعمیم برای $k > ۳$

- با روش گذشته با $n - ۱ + \sum_{i=۲}^{k} i\lceil \lg n \rceil - (k+۱) = n + \mathcal{O}(k^۲ \lg n)$ مقایسه
- با استفاده از هرم کمینه: در $\mathcal{O}(n + k \lg n)$
- با یافتن k امین عنصر (x) و بخش‌بندی حول x در $\mathcal{O}(n + k \lg k)$

یافتن k امین عنصر

۱) با میانگین $\mathcal{O}(n)$ براساس بخش‌بندی در مرتب‌سازی سریع

۲) $\mathcal{O}(n)$ در بدترین حالت
براساس بخش‌بندی ولی با تضمین این که اندازه‌های دو بخش $\mathcal{O}(n)$ هستند

k امین عنصر با میانگین $\mathcal{O}(n)$

```
RANDOMIZED-SELECT((() A, p, r, i)
      ▷ Find the ith element in A[p..r], assuming 1 ≤ i ≤ r − p + 1
 1  if p = r
 2    then return A[p]
 3  q ← RANDOMIZED-PARTITION (A, p, r)
 4  k ← q − p + 1
 5  if i ≤ k
 6    then return RANDOMIZED-SELECT (A, p, q, i)
 7    elsereturn RANDOMIZED-SELECT (A, q + 1, r, i − k)
```

تحلیل Randomized-Select

- فرض می‌کنیم که عناصر نامساوی هستند (حالت بدتر)
- اگر تعداد عناصر n باشد، محور با احتمال $\frac{۱}{n}$، مرتبه‌اش $k \leq n \leq ۱$ است.
- اگر $k = ۱, ۲$ باشد، آرایه به دو بخش ۱ عضوی و $n - ۱$ عضوی تقسیم می‌شود.
- در حالت کلی n، آرایه به دو بخش k عضوی و $n - k$ عضوی تقسیم می‌شود.

پس

$$T(n) \leq \frac{۱}{n}\left[T(\max(۱, n-۱)) + \sum_{k=۱}^{n-۱} T(\max(k, n-k))\right] + O(n)$$

$$T(n) \leq \frac{۱}{n}\left[T(\max(۱, n-۱)) + \sum_{k=۱}^{n-۱} T(\max(k, n-k))\right] + O(n)$$

داریم

$$\max\{k, n-k\} = \begin{cases} k & \text{if } k \geq \lceil n/۲ \rceil, \\ n-k & \text{if } k < \lceil n/۲ \rceil \end{cases}$$

- **n فرد: جمله‌های** $T($ $lceiln/۲\rceil), T(\lceil n/۲\rceil + ۱), \ldots, T(n-۱)$ **دوبار در Σ ظاهر می‌شوند،**
- **n زوج: جمله‌های** $T(\lceil n/۲\rceil + ۱), T(\lceil n/۲\rceil + ۲), \ldots, T(n-۱)$ **دوبار و جمله‌ی** $T(\lceil n/۲\rceil)$ **یک بار ظاهر می‌شوند.**
- **جمله‌ی** $\frac{۱}{n}T(n-۱)$ **را می‌توان در مقابل** $\mathcal{O}(n)$ **نادیده گرفت.**

پس

$$\begin{aligned} T(n) &\leq \frac{۱}{n}\left[T(\max(۱, n-۱)) + \sum_{k=۱}^{n-۱} T(\max(k, n-k))\right] + O(n) \\ &\leq \frac{۱}{n}\left[T(n-۱) + ۲\sum_{k=\lceil n/۲\rceil}^{n-۱} T(k)\right] + O(n) \\ &= \frac{۲}{n}\sum_{k=\lceil n/۲\rceil}^{n-۱} T(k) + O(n). \end{aligned}$$

رابطه‌ی آخر را با استقرا حل می‌کنیم:

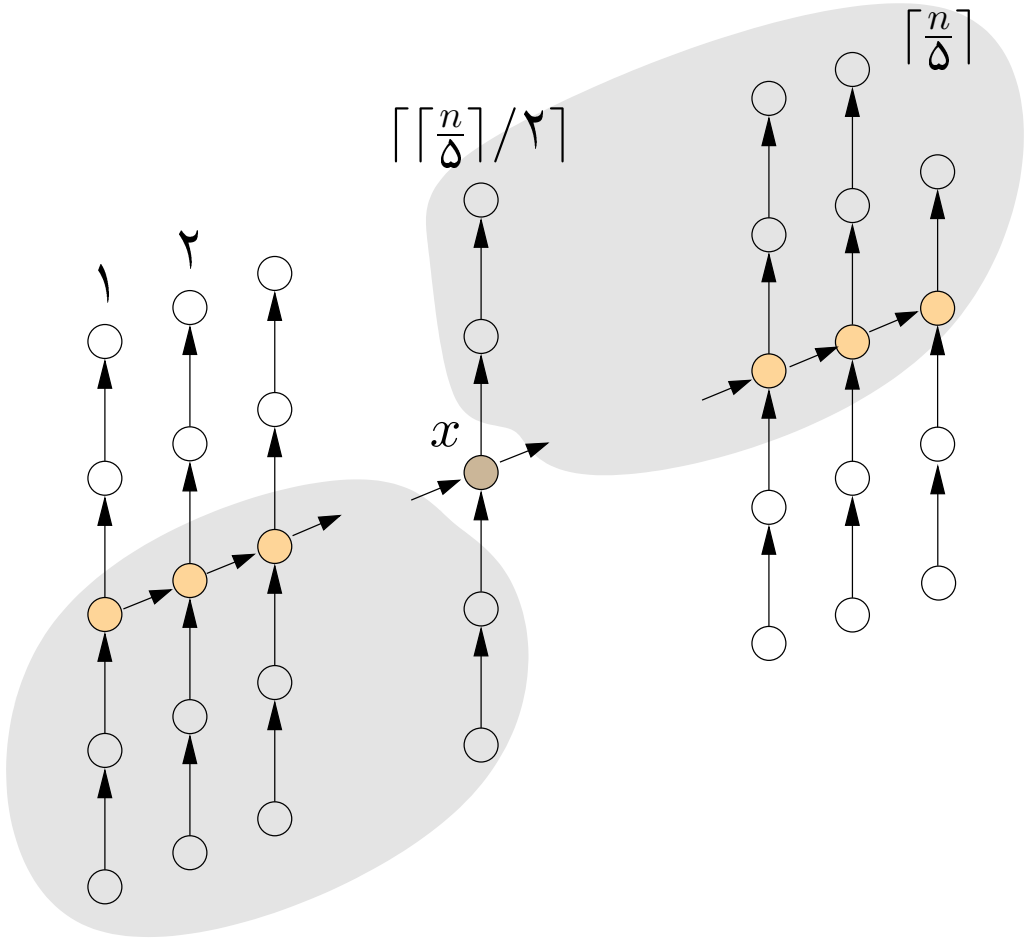
فرض: $T(n) \leq cn$

$$
\begin{aligned}
T(n) &\leq \frac{۲}{n}\sum_{k=\lceil n/۲\rceil}^{n-۱} ck + O(n) \\
&\leq \frac{۲c}{n}\left[\sum_{k=۱}^{n-۱} k - \sum_{k=۱}^{\lceil n/۲\rceil-۱}\right] + O(n) \\
&= \frac{۲c}{n}\left[\frac{۱}{۲}(n-۱)n - \frac{۱}{۲}\left(\left\lceil\frac{n}{۲}\right\rceil - ۱\right)\left\lceil\frac{n}{۲}\right\rceil\right] + O(n) \\
&\leq c(n-۱) - \frac{c}{n}\left(\frac{n}{۲} - ۱\right)\left(\frac{n}{۲}\right) + O(n) \\
&= c\left[\frac{۳}{۴}n - \frac{۱}{۲}\right] + O(n) \\
&\leq cn
\end{aligned}
$$

زیرا می‌توانیم c را به‌قدر کافی بزرگ انتخاب کنیم به‌طوری که $c(n/۴ + ۱/۲) > O(n)$

یافتن k امین عنصر در $\mathcal{O}(n)$

محور را طوری انتخاب می‌کنیم تا تضمین کنیم که اندازه‌ی دوبخش در بدترین حالت $\mathcal{O}(n)$ هستند.


$\lceil \frac{n}{5} \rceil$
$\lceil \lceil \frac{n}{5} \rceil / 2 \rceil$
۲
۱
x

- **حداقل** $۲ - \lceil \frac{\lceil n/۵ \rceil}{۲} \rceil$ **گروه دارای ۳ عنصر کوچک‌تر از** x **هستند.**
- **تعداد عناصر کوچک‌تر از** x **حداقل**

$$۳\left(\left\lceil \frac{۱}{۲}\left\lceil \frac{n}{۵}\right\rceil \right\rceil - ۲\right) \geq \frac{۳n}{۱۰} - ۶$$

است.

- **به‌صورت مشابه، تعداد عناصر بزرگ‌تر از** x **نیز حداقل** $۳n/۱۰ - ۶$ **خواهد بود.**
- **پس، در بدترین حالت، الگوریتم به‌صورت بازگشتی برروی حداکثر** $۷n/۱۰ + ۶$ **عنصر اعمال خواهد شد.**

$$T(n) \leq \begin{cases} \Theta(۱) & n \leq ۸۰ \\ T(\lceil n/۵ \rceil) + T(۷n/۱۰ + ۶) + O(n) & n > ۸۰ \end{cases}$$

فرض: برای یک c مفروض و هر $n \leq ۸۰$ داشته باشیم $T(n) \leq cn$.

$$\begin{aligned} T(n) &\leq c\lceil n/۵ \rceil + c(۷n/۱۰ + ۶) + O(n) \\ &\leq cn/۵ + c + ۷cn/۱۰ + ۶c + O(n) \\ &\leq ۹cn/۱۰ + ۷c + O(n) \\ &\leq cn \end{aligned}$$