مرتب‌سازی و مرتبه‌ی آماری

n عنصر با کلیدهای $a_1, a_2, \ldots, a_n$ و رابطه‌ی ترتیب کامل (total order) $\leq$ داده شده‌اند جای‌گشت π از عناصر داده شده را پیدا کنید به‌طوری‌که:

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

یک رابطه ترتیب جزیی (partial order) است اگر غیرمتقارن، بازتابی و تراگذاری 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

◁ داخلی (internal) ◁ خارجی (external)

۲) از نظرحفظ ترتیب نسبی عناصر

◁ پایدار (stable) ◁ ناپایدار (unstable)

۳) از نظر نحوه‌ی مرتب‌سازی داده‌ها

◁ مبتنی بر مقایسه (comparison sort)

◁ غیر مبتنی بر مقایسه (non-comparison sort)

کران پایین الگوریتم‌های مرتب‌ساز

- هر الگوریتم مرتب‌ساز $\Omega(n)$ است.
- هر الگوریتم مرتب‌ساز مبتنی بر مقایسه در بدترین حالت و حالت متوسط $\Omega(n \lg n)$ است.

ترتیب الفبایی (Lexicographic Ordering)

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

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

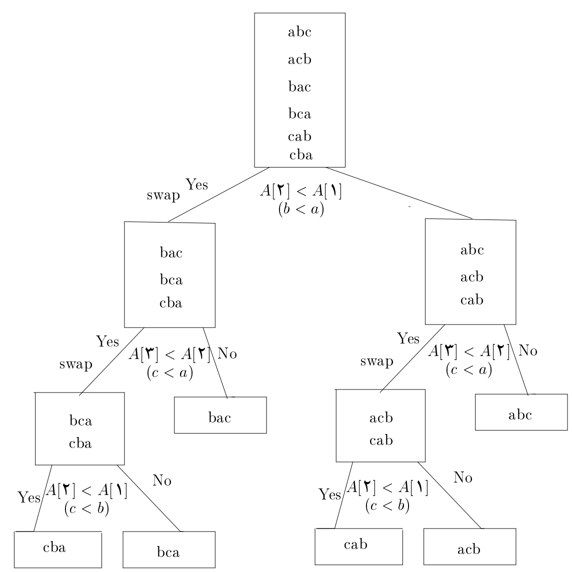
$$ab < abc < adc < adda$$

اثبات کران پایین در بدترین حالت

درخت تصمیم (Decision Tree)

هر الگوریتم مبتنی بر مقایسه را می‌توان با یک درخت مدل کرد.

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



چندتا لم و قضیه

لم: یک درخت دودویی با ارتفاع h حداکثر 2^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).

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).

حداقل چند تا برگ دارد؟ دودویی کامل و دودویی ؟

جواب: ۱ برای درخت عادی و $h+1$ برای درخت دودویی که هر گره دو یا صفر فرزن
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 2^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 مرتب‌ساز مبتنی بر مقایسه، برای مرتب کردن n عنصر در بدترین حالت حداقل $\lceil \lg n! \rceil$ مقایسه بین عناصر ورودی انجام می‌دهد.

اما می‌دانیم که ..

$n! \le n^n$، پس $\lg n! \le n \lg n$. ولی این حد بالای ضعیفی است. تقریب استرلینگ (Stirling's Approximation) حد به‌تری را به‌دست 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:

$$\begin{aligned} n! &= o(n^n) \\ n! &= \omega(2^n) \\ \lg(n!) &= \Theta(n \lg n) \end{aligned}$$

کران پایین در حالت متوسط

یعنی چه؟ چه چیز را باید اثبات کنیم؟

کران پایین در حالت متوسط

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

کران پایین در حالت متوسط

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

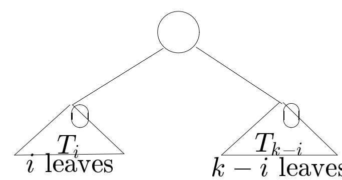
ادامه‌ی اثبات

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

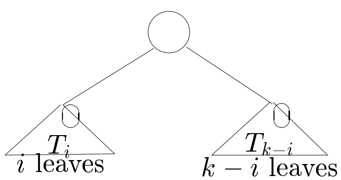
۱) پایه: $m = ۱$ **واضح است.**

۲) فرض: برای $m < k$ **درست است.**

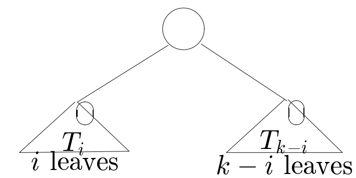
۳) حکم: T **با** k **برگ را در نظر بگیرید.**



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



$$\begin{aligned} D(T) &= i + D(T_i) + (k-i) + D(T_{k-i}) \\ D(k) &= \min_{۱ \leq i \leq k} \{k + D(i) + D(k-i)\} \end{aligned}$$



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

ادامه‌ی اثبات

برای اعداد طبیعی مقدار کمینه‌ی $i\lg i + (k-i)\lg(k-i)$ در $i = \frac{k}{2}$ اتفاق می‌افتد.

پس

$$D(k) \ge k + k\lg\frac{k}{2} = k\lg k$$

الگوریتم‌های مرتب‌ساز خطی

الگوریتم مرتب‌ساز شمارشی (Count Sort)

ورودی: n **عنصر با کلیدهای بین ۱ تا** m

```
Count-Sort(A, B, m)
for i ← 1 to m
    do C[i] ← 0
for i ← 1 to length[A]
    do C[A[i]] ← C[A[i]] + 1
for i ← 2 to m
    do C[i] ← C[i] + C[i − 1]
for i ← length[A] downto 1
    do B[C[A[i]]] ← A[i]
       C[A[i]] ← C[A[i]] − 1
```

چرا درست است؟

چرا درست است؟

- یک عنصر در جای مناسب قرار می‌گیرد، چون در هر مرحله عناصر قبلی که در جای خود قرار گرفته‌اند دست‌کاری نمی‌شوند.
- بعد از n مرحله‌همه‌ی عناصر مرتب می‌شوند.

تحلیل

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

حالت خاص

کلیدهای عناصر اعداد ۱ تا n هستند.

```
Count-Sort(A, n)
for i ← 1 to n
    do while key[A[i]] ≠ i
            do Swap(A[i], A[key[A[i]]])
```

چرا درست است؟

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

الگوریتم Radix Sort

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

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


- اگر تعداد رقم‌های ورودی یکسان نباشند؟
- درستی الگوریتم: با استقرا بر روی i
- زمان اجرای Radix Sort به الگوریتم دوم بستگی دارد.
- اگر از Count Sort استفاده کنیم: زمان اجرا $O(dn)$

Radix Sort حالت کلی

- ورودی آرایه‌ای از رکوردها،
- هر رکورد دارای کلیدی با k مؤلفه‌ی $f_1 \ldots f_k$ از داده‌گونه‌های $t_1 \ldots t_k$،
- تعداد مقادیری که هر داده‌گونه می‌تواند داشته باشد محدود و مستقل از n باشد.

مرتب‌ساز سطلی (Bucket Sort)

```
BUCKET-SORT(A)
for i ← 1 to k
    do for each value v of type t_i
           do make B_i[v] empty
       for each record r in list A
           do let v is the value of f_i of the key for r
              move r from A to the end of B_i[v]
       for each value v of type t_i from lowest to highest
           do concat B_i[v] to the end of A
```


- ورودی: لیست A با n رکورد، که هر رکورد دارای کلیدی با k مؤلفه به نام‌های $f_۱ \dots f_k$ از داده‌گونه‌های $t_۱ \dots t_k$
- تعداد حالت‌های t_i برابر s_i
- فرض: f_i بر $f_{i-۱}$ اولویّت دارد.
- برای $(۱ \le i \le k)$ داریم: B_i : `array[t_i] of list-type`.
- اگر هر سطل را به‌صورت یک صف پیاده‌سازی کنیم، عمل concat در زمان ثابت قابل اجرا است.
- زمان اجرا این الگوریتم برابر است با:

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

اگر $s_i = n$ 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، ۱ ... ۱۰۰ و ۱۳۰۰ ... ۱۴۰۰.

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_3	f_2	f_1
a_1	a	۵	۱۳۲۰
a_2	c	۱۲	۱۳۱۰
a_3	b	۱۲	۱۳۰۵
a_4	a	۸	۱۴۰۰
a_5	z	۱۰	۱۳۰۸
a_6	b	۱۲	۱۳۰۴
a_7	a	٦	۱۳۱۰

ورودی	Bucket[1]	خروجی ۱	Bucket[2]	خروجی۲	Bucket[3]	خروجی ۳
a_1	.[1304] a_6	a_6	.[5] a_1	a_1	.['a'] a_1, a_7, a_4	a_1
a_2	.[1305] a_3	a_3	.[6] a_7	a_7	.['b'] a_6, a_3	a_7
a_3	.[1308] a_5	a_5	.[8] a_4	a_4	.['c'] a_2	a_4
a_4	.[1310] a_2, a_7	a_2	.[10] a_5	a_5	.['z'] a_5	a_6
a_5	.[1320] a_1	a_7	.[12]a_6, a_3, a_2	a_6		a_3
a_6	.[1400] a_4	a_1		a_3		a_2
a_7		a_4		a_2		a_5

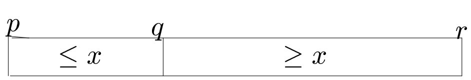
الگوریتم‌های مرتب‌ساز مبتنی بر مقایسه: مرتب‌سازی سریع

- quicksort یک آرایه‌ی n عنصری را در بدترین حالت با $O(n)^2$ و در حالت متوسط د
$O(n \lg n)$ مرتب می‌کند.
- ضریب ثابت $n \lg n$ کاملاً کوچک است.
- این الگوریتم برای محیط‌های حافظه‌ی خارجی نیز کارا می‌باشد.

مرتب‌سازی سریع: توصیف الگوریتم

- مرتب‌سازی سریع مانند مرتب‌سازی ادغامی (merge-sort) 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]`
`A[q+1..r]`،
به‌طوری‌که هر عنصر `A[p..q]` از هر عنصر `A[q+1..r]` بیشتر نباشد.)



۲) حل: `A[p..q]` و `A[q+1..r]` به صورت بازگشتی مرتب می‌شوند.

۳) ترکیب: چون بخش‌ها به صورت درجا مرتب شده‌اند، نیازی به ترکیب آن‌ها نیست
آرایه مرتب است.

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

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


```
Partition(A, p, r)
  x ← A[p]
  i ← p − 1
  j ← r + 1
  while true
       do repeat j ← j − 1
            until A[j] ≤ x
            repeat i ← i + 1
            until A[i] ≥ x
            if i < j
              then Swap(A[i], A[j])
              else return j
```

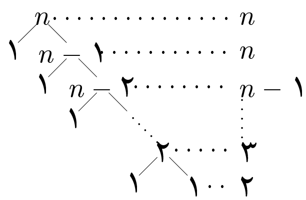

چرا بخش‌بندی درست است؟

مرتب‌سازی سریع (تحلیل الگوریتم)

- بدترین حالت از $\Theta(n^۲)$ و در حالت متوسط از $\Theta(n \log n)$ است و این کارایی وابست
  به نحوه‌ی تقسیم‌بندی است.
- هزینه‌ی عمل بخش‌بندی برابر $O(n)$ با ثابت کوچک است:
  - i و j به‌عقب بر نمی‌گردند
  - تعداد تعویض‌ها حداکثر برابر $n/۲$ است.

تحلیل در بدترین حالت

- هنگامی که بخش‌بندی همیشه n عنصر را به $n-۱$ و ۱ عنصر تقسیم کند
- پیچیدگی الگوریتم: $T(n) = T(n-۱) + \Theta(n) = \Theta(n^۲)$



درخت بازگشت برای $T(n) = T(n-۱) + \Theta(n)$.

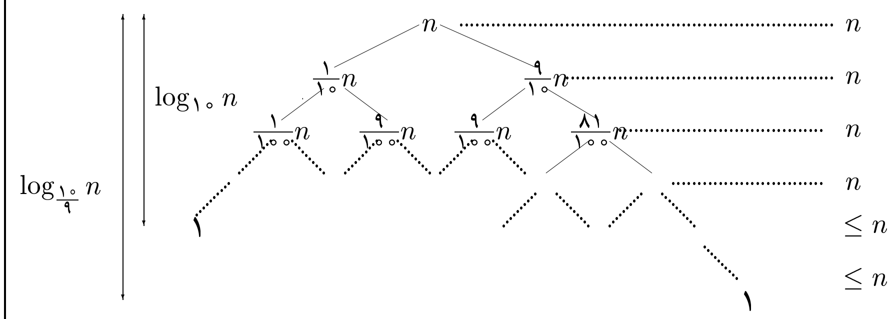
تحلیل در به‌ترین حالت

- Partition درهر مرحله n 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

- $T(n) = T(\frac{۹n}{۱۰}) + T(\frac{n}{۱۰}) + n$
- حل: $T(n) = O(n \lg n)$ (درخت بازگشت)
- در حالت کلی تا وقتی که همواره کسری، از تعداد عناصر در بخش‌ها قرار گیرند نی
الگوریتم $O(n \lg n)$ خواهد بود.



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

فکر کنید و پاسخ دهید!

تحلیل دقیق الگوریتم quicksort چه‌قدر است اگر:

- **همه‌ی عناصر مساوی باشند،**
- **عناصر مرتب باشند،**
- **عناصر برعکس مرتب باشند؟**
- **چه‌گونه می‌توان آن‌را همیشه در $O(n \lg n)$ انجام داد؟**

تحلیل در حالت متوسط

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

گونه‌ی تصادفی «مرتب‌سازی سریع»

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

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

تحلیل Randomized-Quicksort بدترین حالت

- فرض: آرایه n عضوی به دو بخش q عضوی و $n-q$ عضوی تقسیم می‌شود.
- پس

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

- 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 $T(n) = O(n^2)$.

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

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

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

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

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

ادامه‌ی تحلیل در بدترین حالت

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

نشان می‌دهیم که $T(n) = \Omega(n^2)$.

- فرض: $T(m) \ge cm^2$ برای $m < n$
- پس $T(n) \ge \max_{1 \le q \le n-1} \{cq^2 + c(n-q)^2\} + \Theta(n)$
- بنابراین، $T(n) \ge cn^2 - 2c(n-1) + \Theta(n) \ge cn^2$
به‌شرطی که c آن قدر کوچک باشد که $\Theta(n) - 2c(n-1)$ مثبت شود.
- پس $T(n) = \Omega(n^2) = O(n^2) \implies T(n) = \Theta(n^2)$

تحلیل مرتب‌سازی سریع تصادفی در حالت متوسط

- فرض می‌کنیم محور با احتمال یکسان از بین n عنصر انتخاب می‌شود.
- احتمال این‌که مرتبه‌ی محور k باشد برابر $\frac{1}{n}$
- در هر دو حالت $k = 1$ و $k = 2$، آرایه‌ی A به دو قسمت با اندازه‌های ۱ و $n - 1$ تقسیم می‌شود.

- $\bar{T}(n)$: متوسط 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 $\bar{T}(n) = \frac{2}{n}\sum_{k=1}^{n-1}\bar{T}(k) + \Theta(n)$

- حدس می‌زنیم $\bar{T}(n) \leq an\lg n + b$، برای $a, b > 0$

$$\begin{aligned}
\bar{T}(n) &= \frac{2}{n}\sum_{k=1}^{n-1}T(k) + \Theta(n) \\
&\leq \frac{2}{n}\sum_{k=1}^{n-1}[ak\lg k + b] + \Theta(n) \\
&= \frac{2}{n}\sum_{k=1}^{n-1}ak\lg k + \frac{2}{n}\sum_{k=1}^{n-1}b + \Theta(n) \\
&= \frac{2a}{n}\sum_{k=1}^{n-1}k\lg k + \frac{2b}{n}(n-1) + \Theta(n)
\end{aligned}$$

و از تقریب انتگرال ثابت می‌شود که:

$$\sum_{k=1}^{n-1}k\lg k \leq \int_{x=1}^{n-1}x\lg x = \frac{1}{2}n^2\lg n - \frac{1}{8}n^2$$

پس

$$\begin{aligned}
\bar{T}(n) &\leq \frac{2a}{n}\left(\frac{n^2}{2}\lg n - \frac{n^2}{8}\right) + \frac{2b}{n}(n-1) + \Theta(n) \\
&\leq an\lg n + b + [\Theta(n) + b - \frac{an}{4}]
\end{aligned}$$

a و b را طوری انتخاب می‌کنیم تا درون پرانتز منفی شود. بنابرای
$\bar{T}(n) \leq an\lg n + b$.

یا $\bar{T}(n) = O(n\lg n)$

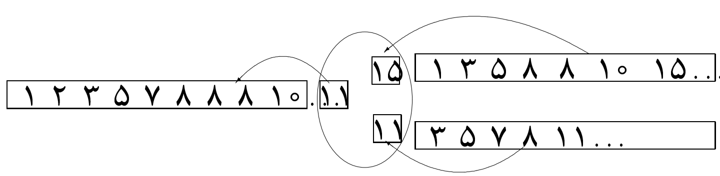
مرتب‌سازی خارجی (فرض)

- اطلاعات بر روی فایل‌ها به‌صورت ترتیبی ذخیره شده‌است.
- 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 دسترسی به‌دیسک

مرتب‌سازی خارجی ادغامی (External Merge)

ادغام دو قطعه‌ی مرتب



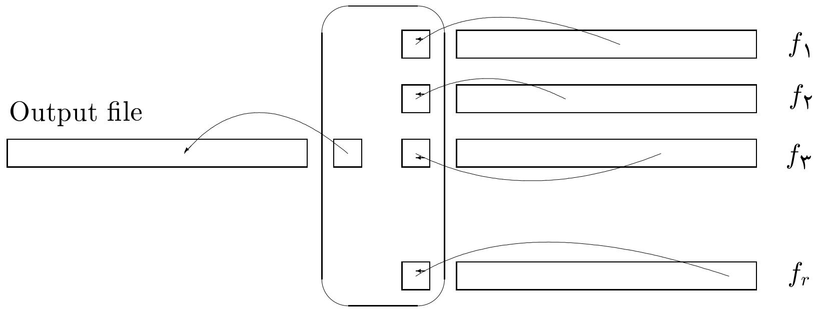
قطعه‌ی اول n_1 رکورد و قطعه‌ی دوم n_2 رکورد

$k=1$، 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

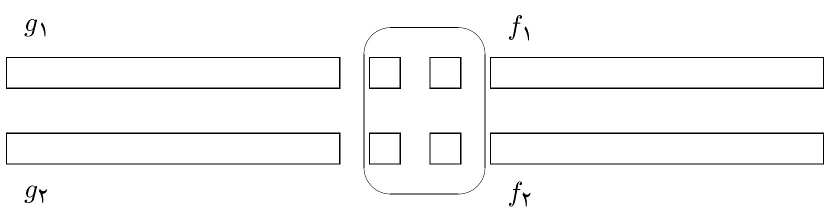


f_i به‌اندازه‌ی n_i رکورد دارد.

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

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

مرتب‌سازی خارجی ادغامی



مرتب‌سازی خارجی ادغامی (مثال)

```
f1   28 3 93 10 54 65 30  90 10  69  8  22
f2   31 5 96 40 85  9 39  13  8  77 10

g1   28 31 | 93 96 | 54 85 | 30 39 |  8 10 |  8 10
g2    3  5 | 10 40 |  9 65 | 13 90 | 69 77 | 22

f1   3  5 28 31 | 9 54 65 85 | 8 10 69 77
f2  10 40 93 06 |13 30 39 90 | 8 10 22

g1   3  5 10 28 31 40 93 96 | 8 8 10 10 22 69 77
g2   9 13 30 39 54 65 85 90 |

f1   3 5  9 10 13 28 39 31 39 40 54 65 85 90 93 96
f2   8 8 10 10 22 69 77

g1   3 5 8 8 9 10 10 10 13 22 28 30 31 39 40 54 ..
```


مرتب‌سازی خارجی ادغامی (الگوریتم کلی)

برای $k = 1$ بیان می‌شود. چهار فایل f_1، f_2، g_1 و g_2 احتیاج است.

۱) فایل ورودی را به دو فایل f_1 و f_2 با حداکثر تعداد یک رکورد اختلاف تقسیم کن.

۲) برای $i = 1, \ldots, M$ مراحل زیر را تکرار کن:

در این مرحله فرض می‌کنیم که f_1 و f_2 (یا g_1 و g_2) شامل قطعاتی به طول $^{i-1}$ هستند و هر قطعه مرتب است و تعداد قطعات دو فایل ورودی حداکثر یک واح 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 (بجز 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.

درستی و تحلیل

- با استقرا می‌توان نشان داد که در انتهای مرحله‌ی i هر فایل خروجی دارای قطعه‌هایی مرتب و به طول 2^i است، بجز حداکثر یک قطعه که طولش از 2^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 نوشت می‌شوند، تعداد دسترسی به دیسک در مجموع برابر $2n(\lceil \log n \rceil + 1)$ است ($2n$ با خواندن و نوشتن برای تقسیم فایل اصلی).
- برای حالت $k > 1$، این تعداد برابر $2\lceil \frac{n}{k} \rceil (\lceil \log n \rceil + 1)$ خواهد بود.

- با تقسیم فایل ورودی 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 Multiway Merge می‌گوییم.

مرتب‌سازی خارجی Polyphase

مثال: $n = 34$

after pass	f1	f2	f3
initial	13(1)	21(1)	-
1	-	8(1)	13(2)
2	8(3)	-	5(2)
3	3(3)	5(5)	-
4	-	2(5)	3(8)
5	2(13)	-	1(8)
6	1(13)	1(21)	-
7	-	-	1(34)

مرتب‌سازی خارجی Polyphase

- سه (در حالت کلی به $r+1$) فایل لازم دارد.
- به فایل ورودی کم‌ترین تعداد رکورد با کلید ∞ اضافه کن تا n برابر F_i (i امین عددد 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 ه
    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 ه
    ادغام کن و حاصل را در f_3 بنویس.
  - حالا 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 به تناسب تغییر بده.

مقدار M چه‌قدر است؟

$M = i$

مرتب‌سازی با کم‌ترین تعداد مقایسه برای تعداد عناصر کم

- حداقل تعداد مقایسه‌ی مورد نیاز (بین عناصر) برای مرتب‌سازی n عنصر $\lceil \lg n!$ 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) تعدا مقایسه‌ی بهینه دارد (برابر ۰، ۱، ۳)
- ولی برای $n > ۳$ بهینه نیست.

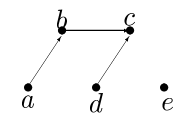
مرتب‌سازی درج دودویی

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) وجو دارد؟



پس از ۳ مقایسه

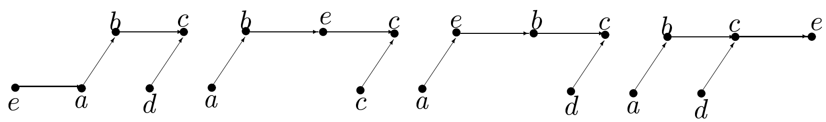
a با b (فرض b بزرگ‌تر است)

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: $a \le b \le d$ و $c \le d$.

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.

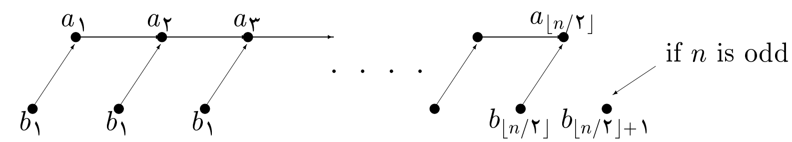
۴) با توجه به‌این که $c < d$، درج c در زنجیره‌ی چهارتایی از عناصری که در مراحل بالا مرتب شده‌اند، به‌روش دودویی حداکثر ۲ مقایسه دیگر نیاز دارد.

بنابراین این‌کار حداکثر ۷ مقایسه نیاز دارد.

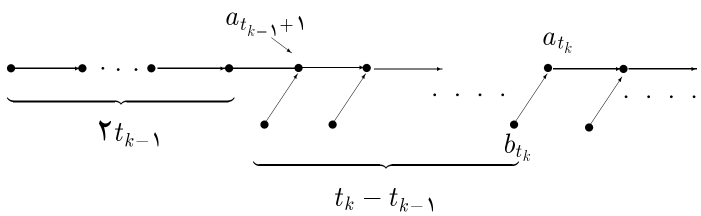
تعمیم الگوریتم (الگوریتم فورد-جانسون)

الگوریتم بالا به‌طرز جالبی توسط آقایان فورد[1] و جانسون[2] در سال ۱۹۵۹ تعمیم داد 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 د انتها زنجیره‌ی اصلی شامل همه‌ی عناصر شود.

[1] Lester Ford Jr.
[2] Slemer Johnson



مراحل اولیه‌ی الگوریتم فورد-جانسون



نحوه‌ی درج در زنجیره‌ی اصلی در مرحله‌ی kام.

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

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

$F(n)$ تعداد مقایسه‌ها

$$F(n) = \lfloor \frac{n}{2} \rfloor + F(\lfloor \frac{n}{2} \rfloor) + G(\lceil \frac{n}{2} \rceil)$$

$G(\lceil \frac{n}{2} \rceil) =$ حداکثر تعداد مقایسه‌های لازم برای درج عناصر b در زنجیره‌ی اصلی

اگر $t_{k-1} \le m < t_k$

می‌دانیم که درج هر عنصر b_k که $t_{j-1} \le k \le t_j$ حداکثر j مقایسه نیاز دارد. پس

$$G(m) = \sum_{j=1}^{k-1} j(t_j - t_{j-1}) + k(m - t_{k-1})$$

و داریم

$$\begin{aligned}\sum_{j=1}^{k-1} j(t_j - t_{j-1}) &= [(k-1)t_{k-1} - (k-1)t_{k-2}] + \\ &\quad [(k-2)t_{k-2} - (k-2)t_{k-3}] + \ldots + (t_1 - t_0) \\ &= (k-1)t_{k-1} - t_{k-2} - t_{k-3} - \ldots - t_1 - t_0\end{aligned}$$

و بنابراین

$$G(m) = km - (t_{k-1} + t_{k-2} + \ldots + t_1 + t_0) \le km$$

برای یک مقدار $k \ge 1$ داریم

$$m \ge t_{k-1} = (2^k + (-1)^{k-1})/3 \ge 2^{k-2}$$

پس $k \le \lg m + 2$

بنابراین

$$G(m) \le km \le m \lg m + 2m$$

پس

$$F(n) \le \frac{n}{2} + F(n/2) + \lceil \frac{n}{2} \rceil \lg \lceil \frac{n}{2} \rceil + 2 \lceil \frac{n}{2} \rceil \le F(\frac{n}{2}) + \frac{n}{2} \lg n + O(n)$$

که نتیجه می‌گیریم

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

برای محاسبه‌ی دقیق‌تر، فرض کنید

$$w_k = t_0 + t_1 + \cdots + t_{k-1} = \lfloor \frac{2^{k+1}}{3} \rfloor$$

در این صورت،

$$(w_0, w_1, w_2, w_3, w_4, \cdots) = (0, 1, 2, 5, 10, 21, \cdots)$$

می‌توان اثبات کرد که

$$F(n) - F(n-1) = k \iff w_k < n \le w_{k+1}$$

و شرط آخر معادل است با

$$\frac{2^{k+1}}{3} < n < \frac{2^{k+2}}{3} \implies k+1 < \lg(3n) \le k+2 \implies k = \lg(\frac{3}{4}n)$$

بنابراین

$$F(n) - F(n-1) = \lceil \lg(\frac{3}{4}n) \rceil$$

و جواب این رابطه‌ی بازگشتی به‌صورت زیر است:

$$F(n) = \sum_{k=1}^{n} \lceil \lg(\frac{3}{4}k) \rceil$$

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 حد پایین.

الگوریتم برای مرتب‌سازی n عنصر به‌صورت زیر است:

۱) عناصر را به $\lfloor \frac{n}{2} \rfloor$ 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 $\lfloor \frac{n}{2} \rfloor$ عنصر کوچک‌تر (از زوج‌عناصر) را مرتب کن و پس از آن عناصر را مطابق شکل نام‌گذاری کن. عناصر $a_1 \le a_2 \le \ldots \le a_{\lfloor \frac{n}{2} \rfloor}$ 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 $2^k - 1$ باشد تا با k مقایسه بتوان زنجیره‌ی کامل را ایجاد کرد:

a) ابتدا b_2 را به زنجیره‌ی $b_1 \le a_1 \le a_2$ و سپس b_2 را به زنجیره‌ی $b_1 \le a_1$ و احتمالاً b_3 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 ب
زنجیره‌ی a_1 تا a_3، b_1 تا b_3 و احتمالاً b_4 درج کن. تعداد عناصر زنجیره حداکثر
و هر درج با ۳ مقایسه امکان‌پذیر است.

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

مرتبه‌ی آماری

کوچک‌ترین و بزرگ‌ترین عنصر

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

دو کوچک‌ترین عناصر

راه‌حل بدیهی: با $n - ۱ + n - ۲$ مقایسه

راه‌حل به‌تر با $n - ۱ + \lceil \lg n \rceil - ۱$ مقایسه با ایجاد درخت مقایسه

نکته: دومین کوچک‌ترین عنصر حتماً با کوچک‌ترین عنصر مقایسه شده است.

k کوچک‌ترین عناصر

- تعمیم راه‌حل قبل برای ۳ کوچک‌ترین عناصر:

سومین کوچک‌ترین عنصر یا با دومین و یا اولین کوچک‌ترین عنصر مقایسه شده است. پس بین $۲ \lg n$ عنصر باید دنبال کوچک‌ترین عنصر بگردیم.

پس حداکثر $n - ۱ + [\lceil \lg n \rceil - ۱] + [۲\lceil \lg n \rceil - ۱] = n - ۱ + ۳\lceil \lg n \rceil - ۲$ مقایسه.

- تعمیم برای $k > ۳$ با $n - ۱ + \sum_{i=۲}^{k} i \lceil \lg n \rceil - (k - ۱) = n + O(k^{۲} \lg n)$ مقایسه
- با استفاده از min-heap: در $O(n + k \lg n)$
- با استفاده از kامین عنصر و عمل «تقسیم‌بندی» در $O(n + k \lg k)$

پیدا کردن kامین عنصر

۱) با متوسط $O(n)$ براساس quicksort

۲) $O(n)$ در بدترین حالت
براساس quicksort ولی با تضمین این که اندازه‌های دو بخش $O(n)$ هستند

k امین عنصر با متوسط $O(n)$

```
Randomized-Select(() A, p, r, i)
    ▷ Find the ith element in A[p..r], assuming 1 ≤ i ≤ r − p + 1
  if p = r
    then return A[p]
  q ← Randomized-Partition (A, p, r)
  k ← q − p + 1
  if i ≤ k
    then return Randomized-Select (A, p, q, i)
    else   return 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 $\frac{۱}{n}$، مرتبه‌ی $۱ \le k \le n$ را دارد.
- اگر $k = ۱, ۲$ باشد، آرایه به دو بخش ۱ عضوی و $n - ۱$ عضوی تقسیم می‌شود.
- در حالت کلی k، آرایه به دو بخش k عضوی و $n - k$ عضوی تقسیم می‌شود.

پس

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

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

داریم

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

- n فرد: جمله‌های $T(\lceil n/۲ \rceil), T(\lceil n/۲ \rceil + ۱), \ldots, T(n-۱)$ 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 $O(n)$ نادیده گرفت.

پس

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

رابطه‌ی آخر را با استقرا حل می‌کنیم:

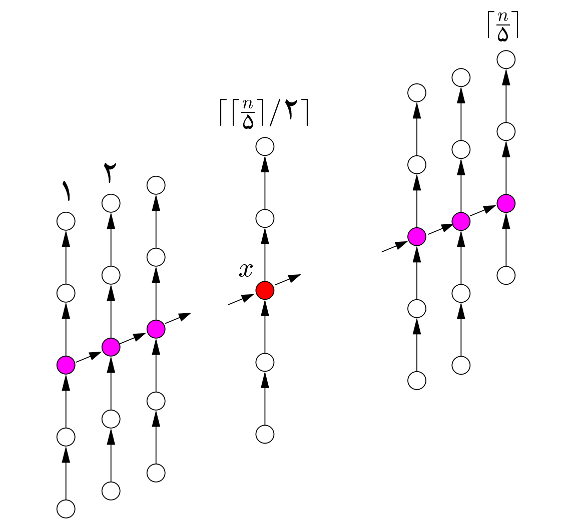
فرض: $T(n) \le cn$

$$
\begin{aligned}
T(n) &\le \frac{2}{n}\sum_{k=\lceil n/2\rceil}^{n-1} ck + O(n)\\
&\le \frac{2c}{n}\left[\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2\rceil-1} k\right] + O(n)\\
&= \frac{2c}{n}\left[\frac{1}{2}(n-1)n - \frac{1}{2}\left(\left\lceil\frac{n}{2}\right\rceil-1\right)\left\lceil\frac{n}{2}\right\rceil\right] + O(n)\\
&\le c(n-1) - \frac{c}{n}\left(\frac{n}{2}-1\right)\left(\frac{n}{2}\right) + O(n)\\
&= c\left[\frac{3}{4}n - \frac{1}{2}\right] + O(n)\\
&\le cn
\end{aligned}
$$

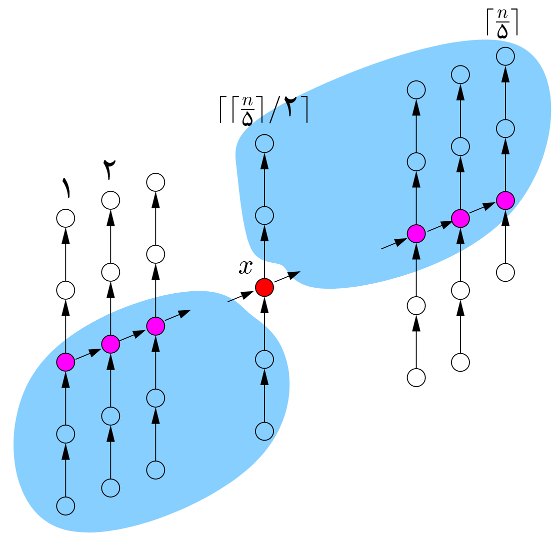
زیرا می‌توانیم c را به‌قدر کافی بزرگ انتخاب کنیم به‌طوری که $c(n/4 + 1/2) > O(n)$

kامین عنصر $O(n)$

می‌خواهیم محور را طوری انتخاب کنیم تا تضمین کنیم که اندازه‌ی دوبخش در بدترین حالت $O(n)$ **هستند.**



تحلیل



- حداقل $\lceil\frac{\lceil n/5\rceil}{2}\rceil - 2$ گروه دارای ۳ عنصر کوچک‌تر از x هستند.
- تعداد عناصر کوچک‌تر از x حداقل

$$3\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

است.

- به‌صورت مشابه، تعداد عناصر بزرگ‌تر از x نیز حداقل $3n/10 - 6$ خواهد بود.
- پس، در بدترین حالت، Select به‌صورت بازگشتی برروی حداکثر $7n/10 + 6$ عنصر اعمال خواهد شد.

$$T(n) \leq \begin{cases} \Theta(1) & n \leq 80 \\ T(\lceil n/5\rceil) + T(7n/10 + 6) + O(n) & n > 80 \end{cases}$$

فرض: برای یک c مفروض و هر $n \leq 80$ داشته باشیم $T(n) \leq cn$.

$$\begin{aligned} T(n) &\leq c\lceil n/5\rceil + c(7n/10 + 6) + O(n) \\ &\leq cn/5 + c + 7cn/10 + 6c + O(n) \\ &\leq 9cn/10 + 7c + O(n) \\ &\leq cn \end{aligned}$$