الگوریتم های مرتب سازی

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)

جایگاه الگوریتم هایی که می شناسیم

- مرتب سازی حبابی (داخلی، پایدار، مقایسه ای) $\theta(n^2)$
- مرتب سازی درجی (داخلی، پایدار، مقایسه ای) $O(n^2)$
- مرتب سازی انتخابی (داخلی، پایدار، مقایسه ای) $\theta(n^2)$
- مرتب سازی ادغامی (داخلی، پایدار، مقایسه ای) $\theta(n\log n)$
- مرتب سازی هرمی (داخلی، ناپایدار، مقایسه ای) $\theta(n\log n)$

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.

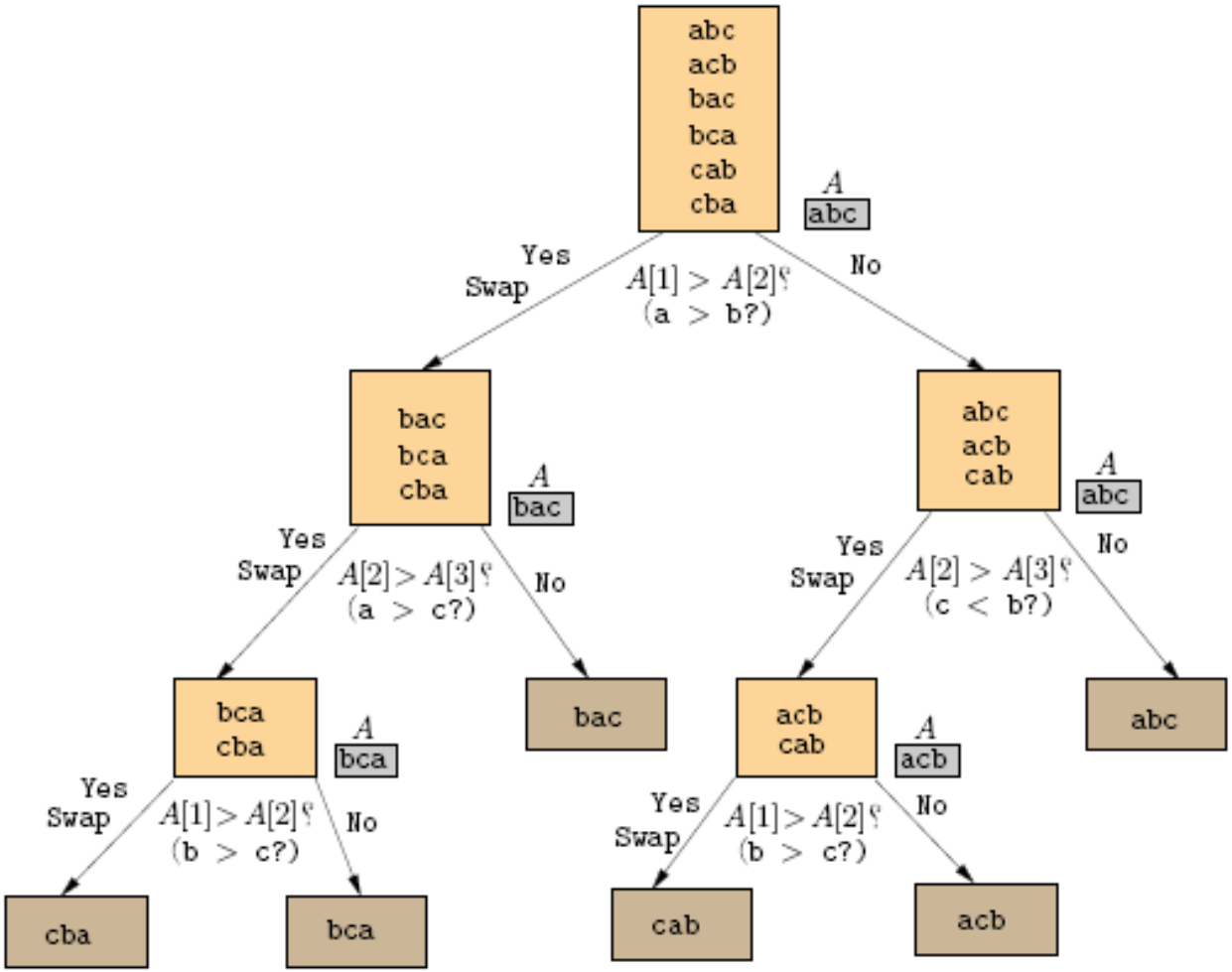
اثبات کران پایین در بدترین حالت

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

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 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:

$$n! = \sqrt{۲\pi n}\left(\frac{n}{e}\right)^n \left(۱ + \Theta\left(\frac{۱}{n}\right)\right)$$

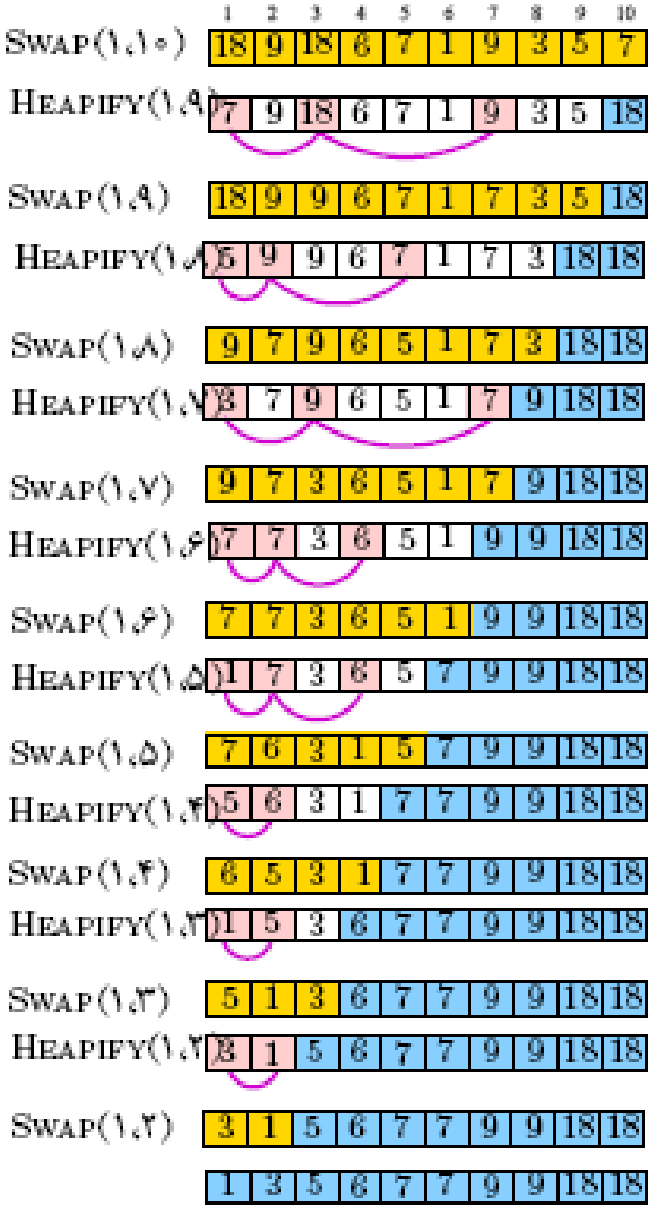
با استفاده از این فرمول می‌توان اثبات کرد که:

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

heapsort

```
BUILD-HEAP(A)
1  for i ← ⌊length[A]/2⌋ downto 1
2      do HEAPIFY(A, i)
```

```
HEAPSORT(A)
1  BUILD-HEAP(A)
2  for i ← length[A] downto 2
3      do swap(A[1], A[length[A]])
4         length[A] ← length[A] − 1
5         HEAPIFY (A, 1)
```


1 2 3 4 5 6 7 8 9 10
Swap(۱،۱۰)
18 9 18 6 7 1 9 3 5 7
Heapify(۱،۹)
7 9 18 6 7 1 9 3 5 18
Swap(۱،۹)
18 9 9 6 7 1 7 3 5 18
Heapify(۱،۸)
5 9 9 6 7 1 7 3 18 18
Swap(۱،۸)
9 7 9 6 5 1 7 3 18 18
Heapify(۱،۷)
3 7 9 6 5 1 7 9 18 18
Swap(۱،۷)
9 7 3 6 5 1 7 9 18 18
Heapify(۱،۶)
7 7 3 6 5 1 9 9 18 18
Swap(۱،۶)
7 7 3 6 5 1 9 9 18 18
Heapify(۱،۵)
1 7 3 6 5 7 9 9 18 18
Swap(۱،۵)
7 6 3 1 5 7 9 9 18 18
Heapify(۱،۴)
5 6 3 1 7 7 9 9 18 18
Swap(۱،۴)
6 5 3 1 7 7 9 9 18 18
Heapify(۱،۳)
1 5 3 6 7 7 9 9 18 18
Swap(۱،۳)
5 1 3 6 7 7 9 9 18 18
Heapify(۱،۲)
3 1 5 6 7 7 9 9 18 18
Swap(۱،۲)
3 1 5 6 7 7 9 9 18 18
1 3 5 6 7 7 9 9 18 18

- پیچیدگی الگوریتم مرتب سازی هرمی در بهترین حالت $\theta(n)$
- پیچیدگی الگوریتم مرتب سازی هرمی در بدترن حالت $\theta(n\log n)$
- پیچیدگی الگوریتم مرتب سازی هرمی در حالت میانگین $\theta(n\log n)$

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.

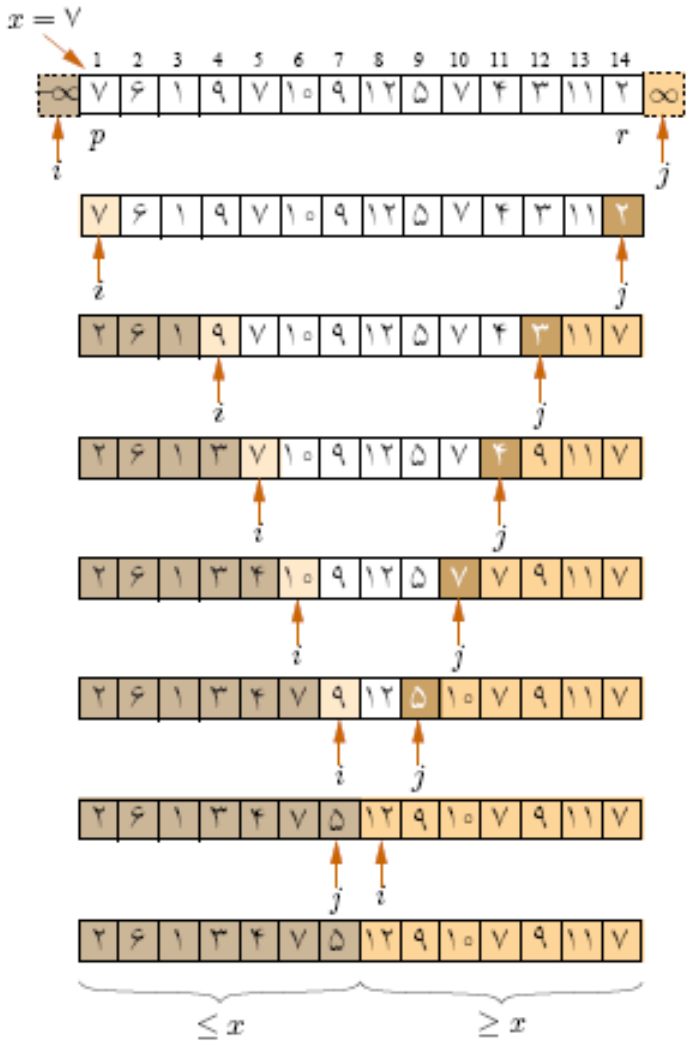
```
QuickSort(A, p, r)
 1  if p < r
 2    then q ← Partition(A, p, r)
 3         QuickSort (A, p, q)
 4         QuickSort(A, q + 1, r)
```

به‌صورت $\text{QuickSort}(A, ۱, length[A])$ 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
```


$x = ۷$
۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹ ۱۰ ۱۱ ۱۲ ۱۳ ۱۴
∞ ۷ ۶ ۱ ۹ ۷ ۱۰ ۹ ۱۲ ۵ ۷ ۴ ۳ ۱۱ ۲ ∞
p
r
i
j
≤ x
≥ x

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 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:

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

تحلیل در بدترین حالت

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

k امین عنصر در یک دنباله

- k امین عنصر، عنصری است که بعد از مرتب کردن دنباله در مکان k ام قرار می گیرد.

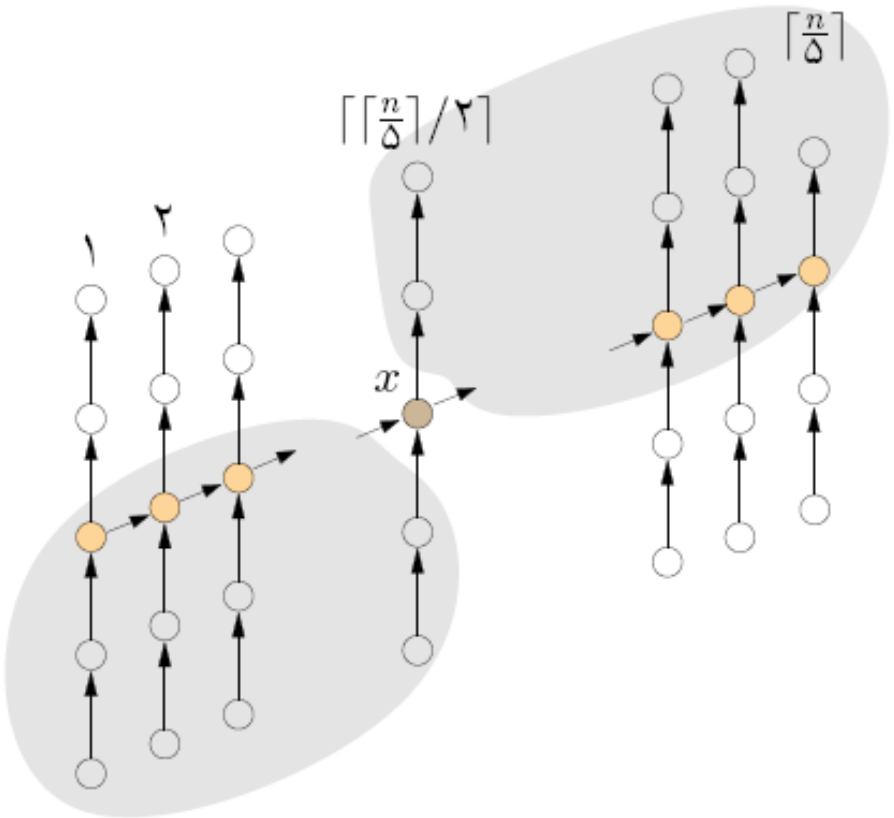
الگوریتم خطی برای یافتن k امین عنصر

۱- دنباله را به زیر مجموعه های ۵ تایی و احتمالا یک مجموعه با کمتر از ۵ عضو افراز می کند و میانه هر دسته را می یابد.

۲- بصورت بازگشتی میانه ، میانه ها را می یابد و در x قرار می دهد.

۳- دنباله اولیه را برحسب x افراز می کند. فرض کنید بعداز افراز بخش اول دارای i عضو باشد.

۴- اگر $(i > k)$ بصورت بازگشتی k امین عنصر بخش اول را می یابد در غیر این صورت بصورت بازگشتی $(k-i)$ امین عنصر بخش دوم را می یابد.


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

- 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.

تحلیل الگوریتم خطی برای یافتن k امین عنصر

اگر T(n) را زمان اجرای الگوریتم روی n عنصر در نظر بگیریم آنگاه

مراحل ۱ و ۳ در زمان O(n) قابل انجام هستند. مرحله ۲ در زمان T([n/5]) و مرحله ۴ در بدترن حالت در زمان T(7n/10+6) انجام می شود.

تحلیل الگوریتم خطی برای یافتن k امین عنصر

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

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

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

نتیجه

اگر در الگوریتم مرتب سازی سریع از عنصر میانه بعنوان عنصر محور استفاده کنیم آنگاه الگوریتم در زمان $\theta(n \log n)$ اجرا میشود.

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؟

- 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
```

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:

$$\sum_{i=1}^{k} \Theta(s_i + n) = \Theta(kn + \sum_{i=1}^{k} s_i)$$

اگر $s_i = \Theta(n)$ آن گاه

$$T(n) = \Theta(kn + \sum_{i=1}^{k} n) = \Theta(n + kn) = \Theta(n)$$

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