sorting in Linear time

یک الگوریتم مرتب سازی تا چه اندازه می تواند سریع باشد؟

- در اینجا یک lower bound برای مسئله مرتب سازی ارائه می دهیم.

مرتب سازی مقایسه ای (comparsion sorting)

- تنها عملیاتی که باید انجام شود تا درباره نظم و مرتب سازی دنباله اطلاعاتی کسب کنیم محل مقایسه است
- به تمامی رسد که تمام الگوریتم های مرتب سازی تا اینجا همه comparsion sort هستند.

- insertion sort
- selection sort
- merge sort
- quick sort
- heap sort
- tree sort

حد پایین یا lower bound برای مرتب سازی:

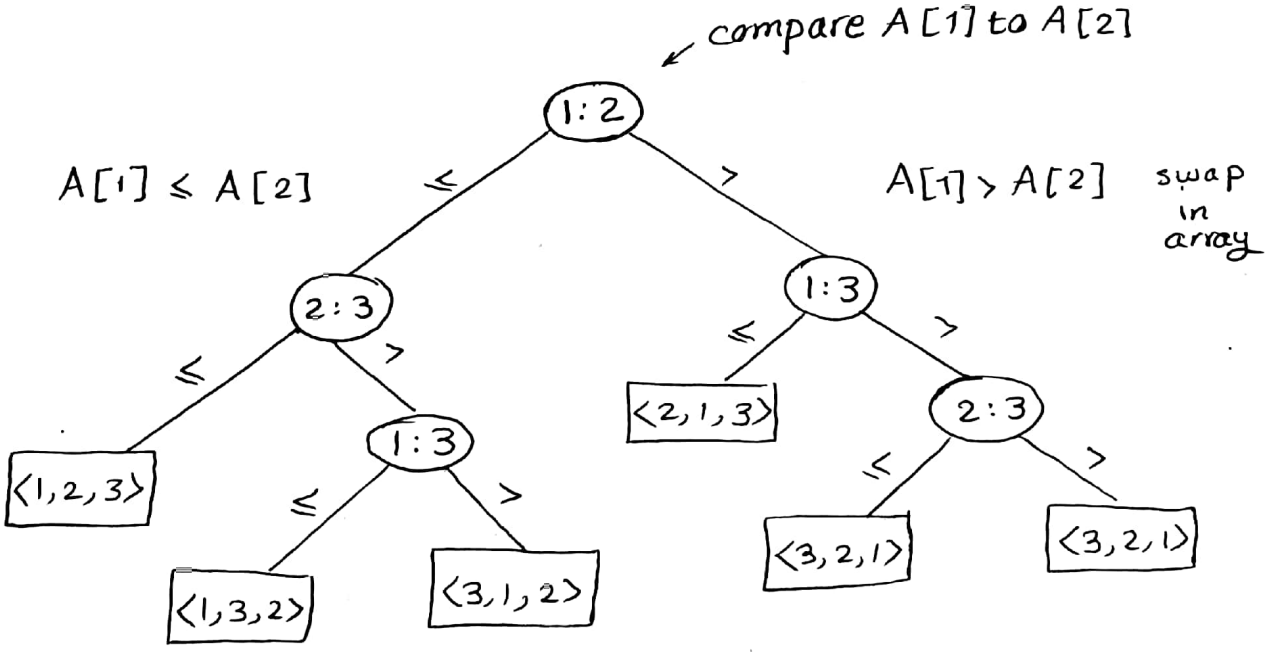
Lower bounds:

- $\Omega(n)$ زمان برای بررسی تمام دنباله ورودی
- تاکنون همه مرتب سازی ها $\Omega(n\lg n)$ بودند
- نشان می دهیم که $\Omega(n\lg n)$ یک حد پایین برای مرتب سازی مقایسه ای است.

Decision tree :

- انتزاعی سازی هر مرتب سازی مقایسه ای
- نشان دادن مقایسه های انجام شده .
- بوسیله یک الگوریتم جستجو
- بر روی ورودی های با سایز مشخص
- کنار گذاشتن هر عمل دیگر مانند کنترل و انتقال داده
- فقط مقایسه ها را می شماریم

برای Insertion sort روی ۳ عنصر



هر node داخلی با برچسبی از عنصر آرایه نامگذاری شده است که از مکان اصلی آن در ورودی آمده است .

برگها در واقع جایگشتی هستند که مرتب شده ورودی است که این ترتیب توسط الگوریتم تشخیص داده شده است .

چه تعداد leaf در decision tree است؟

در کل بزرگتر یا مساوی $n!$ حالت می‌تواند در برگها باشد، چون هر جایگشت حداقل یکبار رخ می‌دهد.

- برای هر مرتب سازی مقایسه‌ای:
  - یک درخت برای هر n
  - در هر مرحله درخت با مقایسه split یا تقسیم به دو زیرشاخه می‌شود که این با توجه به عمل مقایسه‌ای که رخ می‌دهد انجام می‌شود.
  - درخت تمام گامهای اجرای آن را نشان می‌دهد.
- طول بلندترین مسیر از ریشه به برگ کدام است؟
  - به الگوریتم بستگی دارد.
  - برای Insertion sort برابر است با $\Theta(n^2)$
  - برای merge sort برابر است با $\Theta(n \lg n)$

لم: هر درخت باینری با ارتفاع h کمتر یا مساوی 2^h برگ دارد.

به عبارت دیگر اگر

l = # of leaves

h = height

Then $l \leq 2^h$

این لم در ادامه به کار برده می‌شود. (اثبات با استقرا)

قضیه: هر درخت تصمیم که n عنصر را مرتب می‌کند ارتفاع $\Omega(n \lg n)$ دارد.

اثبات:

- $l \geq n!$
- by lemma, $n! \leq l \leq 2^h$ or $2^h \geq n!$
- Take logs, $h \geq \lg(n!)$
- Use Stirling's approximation: $n! > (n/e)^n$

$$h \geq \lg (n/e)^n$$
$$= n \lg (n/e)$$
$$= n \lg n - n \lg e$$
$$= \Omega(n \lg n)$$

جمع‌بندی:

هم heapsort و هم merge sort هر دو asymptotically الگوریتم مرتب‌سازی مقایسه‌ای بهینه هستند.

Sorting in linear time:

مرتب‌سازی غیرمقایسه‌ای (Non-comparsion sorts)

Counting sort:

وابسته به Key assumption: اعدادی که می‌خواهند مرتب شوند عدد صحیح در مجموعه $\{0, 1, \ldots, k\}$ هستند.

Input : $A[1..n]$, where $A[j] \in \{0,1,\ldots,k\}$ for $j = 1,2,\ldots,n$. Array A and values n and k are given as parameters

Output: $B[1..n]$, sorted. B is assumed to be already allocated and given as a parameter.

Auxilliary storage: $C[0..k]$

- n عنصر به عنوان ورودی داده شده است که هر عنصر یک عدد صحیح در بازه‌ی ۰ تا k فرض شده است.
- خروجی یک آرایه B است که مقدار آن از قبل برروشده است.

```
Counting-SORT (A, B, n, k)
    for i ← 0 to k
        do C[i] ← 0
    for j ← 1 to n
        do C[A[j]] ← C[A[j]] + 1
    for i ← 1 to k
        do C[i] ← C[i] + C[i-1]
    for j ← n downto 1
        do B[C[A[j]]] ← A[j]
           C[A[j]] ← C[A[j]] - 1
```

Example:

A: 5 4 12 2 0 3 6 7

B:

$n = 8$

$k = 12$

	0	1	2	3	4	5	6	7	8	9	10	11	12
C:	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	1	1	1	1	1	1	0	0	0	0	1

↓

	1	2	3	4	5	6	7	8
C:	1	1	2	3	4	5	6	7	7	7	7	7	8

↓

	1	2	3	4	5	6	7	8
B:	0	2	3	4	5	6	7	12

counting sort یک روش stable است، (keyهای با مقادیر یکسان در همان ترتیب و مکان ورودی قرار می‌گیرند) این به دلیل عملکرد loop آخر است.

Analysis: $\Theta(n+k)$ که اگر $k = O(n)$ باشد می‌شود $\Theta(n)$

توضیح:

practically، مقدار k چقدر بزرگ می‌تواند باشد؟

چه میزان از بزرگی k عملیاتی است در این حالت؟

- برای مرتب سازی اعداد 32-bit ی مناسب است؟ نه
- 16-bit ؟ نه، احتمالاً
- 8-bit ؟ شاید، بستگی دارد به مقدار n
- 4-bit ؟ احتمالاً، مگر آنکه n خیلی کوچک باشد

radix sort در counting sort استفاده می‌شود.

Radix Sort:

کمی تاریخچه: IBM از punch card استفاده می‌کرد و الگوریتمی به نام Card Sorters داشت که هر مرحله روی یک ستون از ارقام کار می‌کرد. بخشی از کار توسط انسان انجام می‌شد.

Key idea: ابتدا ارقام را مرتب کن.

To sort d digits:

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

	sorted ↓	sorted ↓	sorted ↓
326	690	704	326
453	751	608	435
608	453	326	453
835	704	835	608
751	835	435	690
435	435	751	704
704	326	453	751
690	608	690	835

Correctness:

- استقرا بر روی تعداد passها (i در pseudo code)
- فرض کنید ارقام 1، 2، ... ، i-1 مرتب شده‌اند.
- نشان می‌دهیم یک stable sort بر روی رقم i ارقام از i را مرتب شده می‌کند

← اگر دو رقم در موقعیت i متفاوت باشند، مرتب سازی با موقعیت i درست است.
و موقعیت های $i+1, \dots, d$ نامرتبط هستند.

← اگر دو رقم در موقعیت i برابر باشند، اعداد هم اکنون هم در ترتیب درست قرار دارند.
(by inductive hypothesis) stablesort، بر روی رقم i آنها را در موقعیت درست خود حفظ می کند.

- این نشان می دهد که چرا مهم است که از stable sort به عنوان intermediate sort استفاده شود. (مرتب سازی واسطه ای)

: Analysis

فرض کنید که از stable sort به عنوان intermediate sort استفاده می کنیم

- $\Theta(n+k)$ برای هر pass (ارقام در بازه ی 0 تا k)
- d تا pass داریم.
- $\Theta(d(n+k))$ در مجموع.
- اگر $k=O(n)$، زمان $\Theta(dn)$

چگونه هر key را به ارقام تبدیل کنیم؟

- n تا word.
- b bits/word
- تبدیل به ارقام r-bit ای که $d=\lceil b/r \rceil$
- استفاده از Counting Sort، $k=2^r-1$

⋮

مثال: کلمه‌های ۳۲-بیتی و ارقام ۸-بیتی.

$b = 32$

$r = 8$

$d = \lceil 32/8 \rceil = 4$

$k = 2^8 - 1 = 255$

Time o:

$$\Theta\left(\frac{b}{r}(n+2^r)\right)$$

r را چگونه انتخاب کنیم؟

توازن بین b/r و $n+2^r$، انتخاب $r \approx \lg n$ می‌دهد که

$$\Theta\left(\frac{b}{\lg n}(n+n)\right) = \Theta(bn/\lg n)$$

- اگر $r < \lg n$ انتخاب کنیم آن‌گاه $b/r > b/\lg n$ ، $n + 2^r$ بهبود نمی‌یابد.

- اگر $r > \lg n$ انتخاب کنیم آن‌گاه $r > \lg n$ ، $n + 2^r$ بزرگ می‌شود، به‌عنوان مثال $r = \lg n \Leftarrow 2^r = 2^{2\lg n}$

$= n^2$

بنابراین، برای مرتب‌سازی 2^{16} عدد ۳۲-بیتی با استفاده از $r = \lg 2^{16}$ برابر 16 بیت و $2 = \lceil b/r \rceil$ پاس یا گذر.

مقایسه

radix sort ○	merge sort ○	quicksort ○
2^{20} عدد ۳۲ بیتی صحیح	20 passes (lgn)	20 passes (lgn)
2 passes $\lceil 32/20 \rceil$		

در اینجا، radix sort، ۲ pass انجام می‌دهد یکی برای سرشماری و دیگری برای جابجائی دیتا.

چطور radix sort قوانین اصلی دنیای را برای مرتب سازی مقایسه‌ای نقض می‌کند؟

- استفاده از radix sort باعث می‌شود اطلاعات اضافی دارای معنا در مورد اعداد داشته باشیم تا صرفاً آنها را با هم مقایسه کنیم، (تنها مقایسه ۲ عدد)
- استفاده از key به عنوان index ها.