Counting Sort $\to \Theta(n+K)$

Radix Sort $\to \Theta(d(n+K))$

رقم

بازه

طراحی الگوریتم‌ها – جلسه نهم

Introduction to Algorithm

استاد: جوانمردی

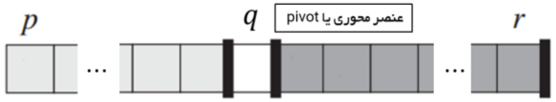
۱۳۹۹/۸/۳

مرور جلسه قبل

Quick Sort

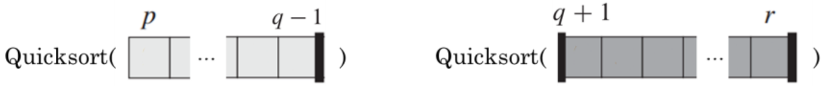
Partition (rearrange) the array $A[p \,..\, r]$

تقسیم



$A[p \,..\, q-1] \;\le\; A[q] \;\le\; A[q+1 \,..\, r]$

حل



ترکیب

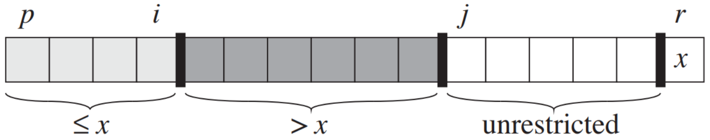
$A[p \,..\, r]$ is now sorted

پیاده سازی Partition

$\text{PARTITION}(A, p, r)$

```
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

- انتخاب آخرین عنصر به عنوان pivot
- تغییر چیدمان عناصر آرایه با رعایت قوانین چهار بخش
- قرارداد pivot در جایگاه مناسب



پیاده‌سازی مرتب‌سازی سریع

$\text{QUICKSORT}(A, p, r)$

```
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

اولین فراخوانی تابع:

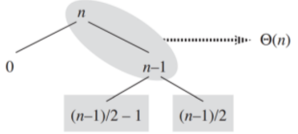
$\boxed{\text{QUICKSORT}(A, 1, A.length).}$

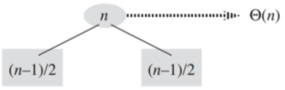
تحلیل زمانی شهودی مرتب‌سازی سریع

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n)\,. \end{aligned} \longrightarrow \begin{aligned} \sum_{k=1}^{n} k &= \frac{1}{2}n(n+1) \\ &= \Theta(n^2)\,. \end{aligned} \longrightarrow \Theta(n^2)$$

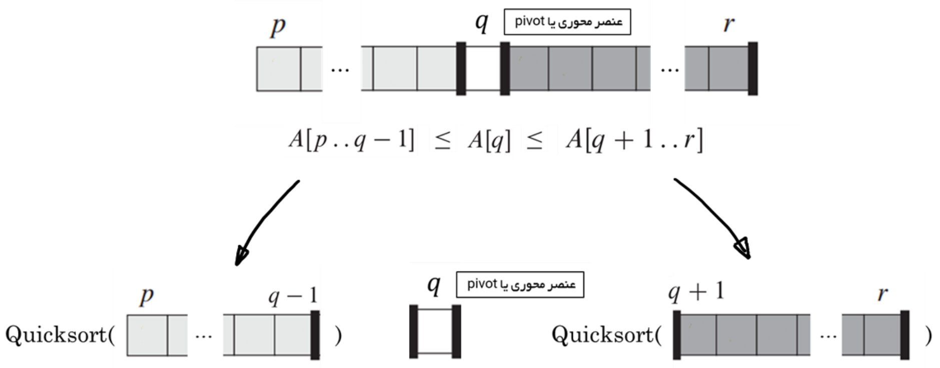
$T(n) = 2T(n/2) + \Theta(n)\,, \longrightarrow$ حالت دوم قضیه اصلی $\longrightarrow T(n) = \Theta(n \lg n)$

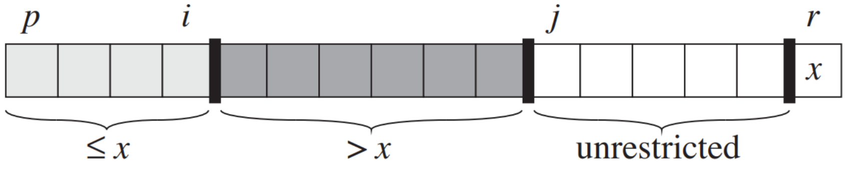
$T(n) = T(9n/10) + T(n/10) + cn$





مدل تصادفی Quicksort





~~فرض: تمام جایگشت‌های اعداد ورودی با احتمال یکسان رخ میدهند~~

↓

random sampling

Pivot selection: $A[r] \longrightarrow$ Randomly choose from $A[p \mathinner{.\,.} r]$

RANDOMIZED-PARTITION(A, p, r)

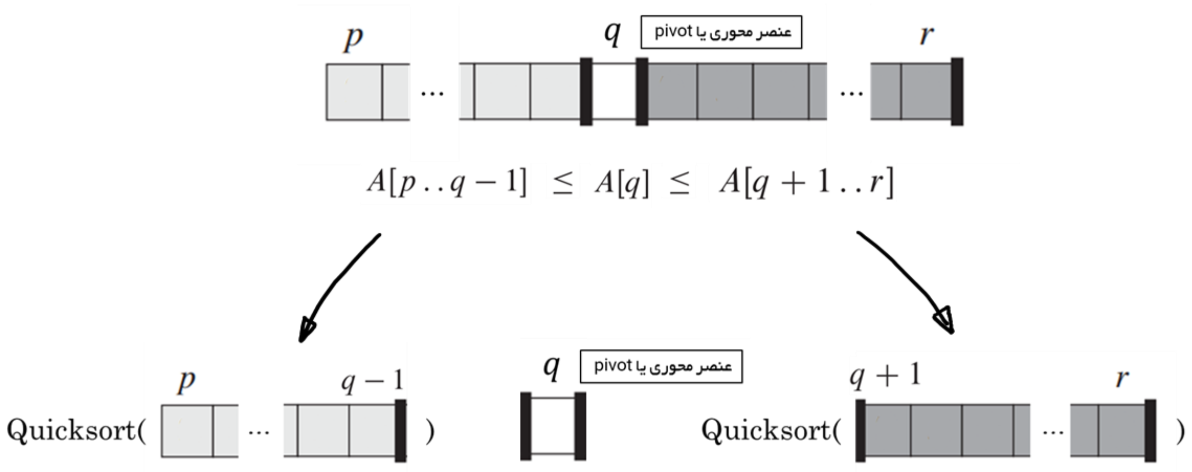
1 $i = $ RANDOM(p, r)
2 exchange $A[r]$ with $A[i]$
3 **return** PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)

1 **if** $p < r$
2     $q = $ RANDOMIZED-PARTITION(A, p, r)
3   RANDOMIZED-QUICKSORT$(A, p, q - 1)$
4   RANDOMIZED-QUICKSORT$(A, q + 1, r)$

تحلیل ریاضی بدتری زمان اجرای Quicksort

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



دارای $n - q - 1$ عنصر

دارای q عنصر

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

$$\boxed{O(n^2)}$$

$$\boxed{T(n) \le cn^2}$$

$$\begin{aligned} T(n) &\le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n) . \end{aligned}$$

$$\begin{aligned} T(n) &\le cn^2 - c(2n-1) + \Theta(n) \\ &\le cn^2 , \end{aligned}$$

$$\max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) \le (n-1)^2$$

مرتب‌سازی مقایسه ای

- الگوریتم‌های معرفی شده تا الان ← مرتب‌سازی n عدد در $O(n\lg n)$
- مرتب‌سازی ادغامی و هرمی در بدترین حالت در $O(n\lg n)$
- مرتب‌سازی سریع در حالت متوسط در $O(n\lg n)$
- خاصیت مشترک الگوریتم‌های معرفی شده تا کنون:

مرتب‌سازی اعداد صرفا بر اساس مقایسه المان‌ها با یکدیگر ← مرتب‌سازی مقایسه‌ای ← بدترین زمان اجرا $\Omega(n\lg n)$

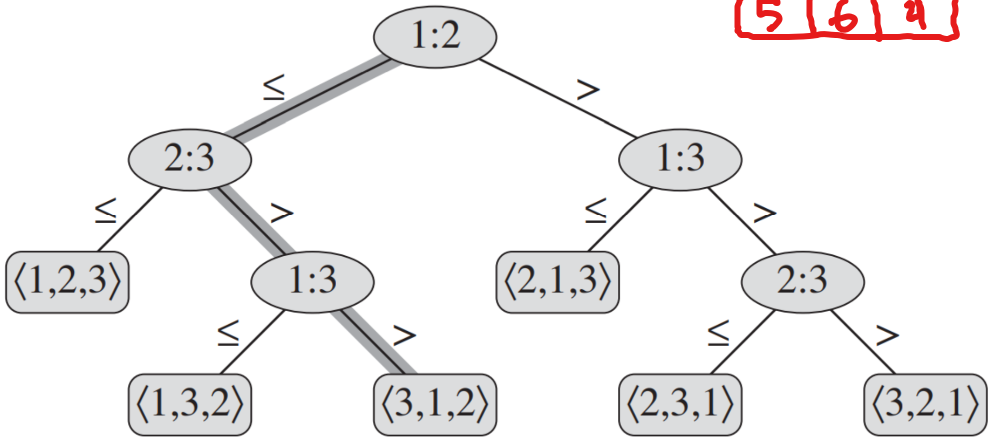
عدم دسترسی به هرگونه اطلاعات مازاد در مورد خود اعداد

تحلیل رفتار مرتب‌سازی مقایسه با استفاده از درخت تصمیم‌گیری دودویی

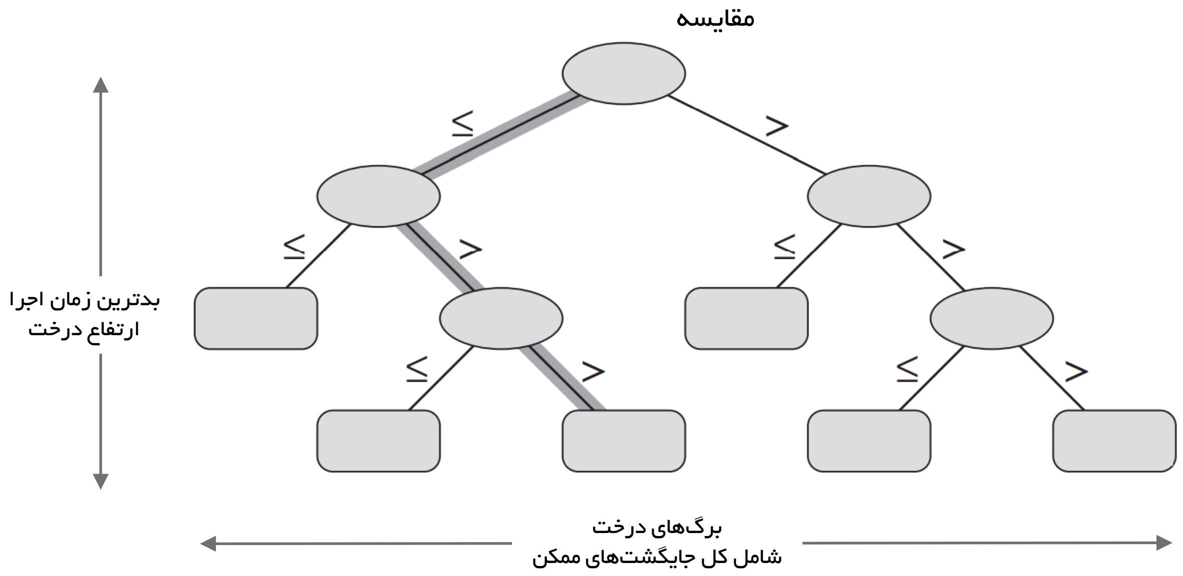
بهترین بدترین زمان اجرای مرتب‌سازی مقایسه‌ای

درخت تصمیم‌گیری برای مرتب‌سازی درجی با ۳ عدد

1 2 3
| 5 | 6 | 9 |



درخت تصمیم‌گیری برای الگوریتم‌های مرتب‌سازی



≤ 5:6
6:9
>
5:9
>
<9,5,6>

Theorem 8.1
Any comparison sort algorithm requires $\Omega(n \lg n)$ comparisons in the worst case.

تعداد کل جایگشت‌ها n عدد

ماکسیمم برگ برای ارتفاع h

$$\underbrace{n! \le l}_{\text{تعداد برگ‌ها}} \le 2^h \xrightarrow[\text{لگاریتم از طرفین}]{} h \ge \lg(n!) = \Omega(n \lg n)$$

Stirling's approximation

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

مرتب‌سازی خطی

- درصورت وجود برخی اطلاعات در مورد اعداد می‌توان از حد $\Omega(n \lg n)$ عبور کرد

- الگوریتم Counting sort

the input numbers are in the set $\{0, 1, \ldots, k\}$

مرتبه زمانی

worst-case running time $\Theta(k + n)$

expected running time $\Theta(k + n)$

- الگوریتم Radix sort

there are n integers to sort

integer has d digits

digit can take on up to k possible values

مرتبه زمانی

worst-case running time $\Theta(d(n + k))$

expected running time $\Theta(d(n + k))$

- الگوریتم Bucket sort

requires knowledge of the probabilistic distribution of numbers in the input array

real numbers uniformly distributed in the half-open interval $[0, 1)$

مرتبه زمانی

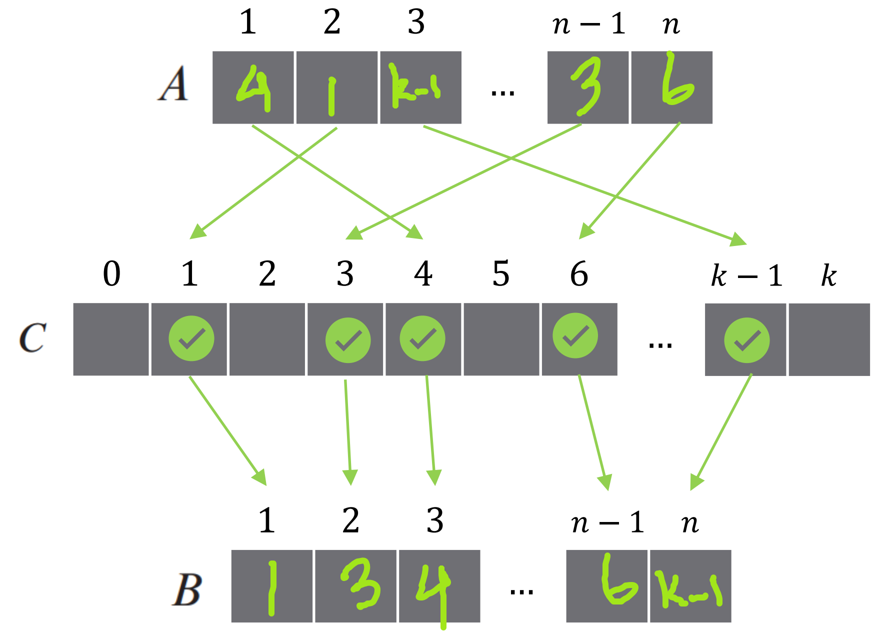
worst-case running time $\Theta(n^2)$

average-case running time $\Theta(n)$

مرتب‌سازی شمارشی counting sort

- اطلاعات مازاد: اعداد داخل آرایه اعداد حسابی ۰ تا k
- زمان اجرا: در صورتی که $k = O(n)$ باشد زمان اجرای counting sort $\theta(n)$ خواهد بود

با فرض تکراری نبودن اعداد



$k = O(n)$

در حال کلی و وجود اعداد تکراری؟

- ایده کار: شمارش تعداد اعداد کوچکتر برای هر عدد و قراردادن آن مستقیماً در جایگاه خود

پیاده‌سازی counting sort

```
COUNTING-SORT(A, B, k)
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i - 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

از هر عدد چندتا؟

فراوانی تجمعی

A* رو از ته بررسی میکنیم

$A[1..n]$	ورودی
$B[1..n]$	مرتب
$C[0..k]$	موقت

	0	1	2	3	4	5	6		$k-1$	k
C	0	1	1	3	4	4	7	…	12	12

مثال counting sort

از هر کدوم چند تا؟

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

(a)

فراوانی تجمعی

	0	1	2	3	4	5
C	2	2	4	7	7	8

(b)

	1	2	3	4	5	6	7	8
B							3	

	0	1	2	3	4	5
C	2	2	4	6	7	8

(c)

	1	2	3	4	5	6	7	8
B		0					3	

	0	1	2	3	4	5
C	1	2	4	6	7	8

(d)

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

	0	1	2	3	4	5
C	1	2	4	5	7	8

(e)

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

(f)

تحلیل زمان اجرای counting sort

```
COUNTING-SORT(A, B, k)
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i - 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

$\theta(k)$

$\theta(n)$

$\theta(k)$

$\theta(n)$

$\theta(n+k) \xrightarrow{k = O(n)} \theta(n)$

دلیل اینکه زمان اجرای counting sort توانست از حد $\Omega(n\lg n)$ عبور کند اینست که counting sort یک مرتب‌سازی مقایسه ای نیست

در عوض counting sort از مقدار خود اعداد بصورت مستقیم برای تعیین محلشان استفاده میکند

ویژگی پایداری: ترتیب اعداد برابر در آرایه ورودی و آرایه مرتب شده تغییر پیدا نمی کند

مرتب‌سازی مبنایی radix sort

- مرتب‌سازی ۱۰۰۰ عدد که مقدار هرکدام میتواند بین ۰ تا ۹۹۹،۹۹۹ باشد؟
- اطلاعات مازاد: اعداد داخل آرایه دارای d رقم که هر رقم k مقدار متفاوت به خود میگیرد
- ایده اصلی: مرتب‌سازی کل از طریق مرتب‌سازی رقم به رقم

روش شهودی – غلط

329		839		839
457		720		720
657		657		657
839	→	457	→	457
436		436		436
720		329		329
355		355		355

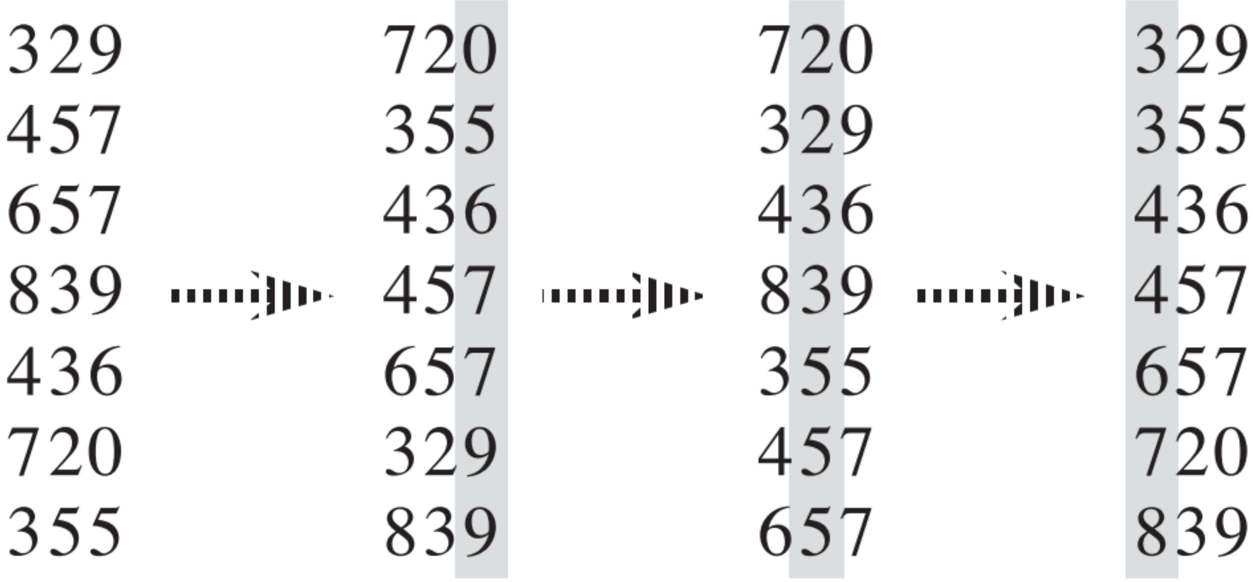
- مرتب‌سازی از رقم با ارزش بیشتر
- در بدترین حالت چندبار تابع sort فراخوانی می‌شود؟

d رقم که هر رقم k مقدار متفاوت

مرتب‌سازی مبنایی radix sort

عکس روش شهودی

- مرتب‌سازی از رقم با ارزش کمتر
- مرتب‌سازی هر رقم باید پایدار باشد! ← استفاده از counting sort



پیاده‌سازی radix sort و زمان اجرا

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

329	→	720	→	720	→	329
457		355		329		355
657		436		436		436
839		457		839		457
436		657		355		657
720		329		457		720
355		839		657		839

\# ارقام

۹

صفر

- زمان اجرا: $\theta(d(n+k))$
- اگر $k = O(n)$ و $d = O(1)$ باشد زمان اجرای radix sort $\theta(n)$ خواهد بود

تحلیل دقیق‌تر مرتبه زمانی radix sort

$\Theta(d(n+k))$

- اگر n عدد b بیتی داشته باشیم، برای هر عدد مثبت دلخواه $r \leq b$ خواهیم داشت:

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

$$\boxed{\text{زمان اجرایی radix sort برای این اعداد } \theta((b/r)(n+2^r)) \text{ خواهد بود}}$$

به شرطی که مرتب‌سازی پایدار استفاده شده $\theta(n+k)$ باشد

- اثبات:

برای $r \leq b$ فرض کنید که هر رقم r بیتی باشد و تعداد ارقام برابر $d = \lceil b/r \rceil$

در این صورت هر رقم یک عدد صحیح بین ۰ تا $2^r - 1$ خواهد بود

و می‌توان از counting sort با $k = 2^r - 1$ استفاده کرد

$b = 32, r = 8, k = 2^r - 1 = 255, \text{ and } d = b/r = 4$

زمان counting sort: $\Theta(n+k) = \Theta(n+2^r)$

زمان اجرای radix sort: $\Theta(d(n+2^r)) = \Theta((b/r)(n+2^r))$

تحلیل دقیق‌تر مرتبه زمانی radix sort

- اگر n عدد b بیتی داشته باشیم، برای هر عدد مثبت دلخواه $r \leq b$ خواهیم داشت:

زمان اجرایی radix sort برای این اعداد $\Theta((b/r)(n+2^r))$ خواهد بود

به شرطی که مرتب‌سازی پایدار استفاده شده $\theta(n+k)$ باشد

- برای اعداد n و b تعیین $r \leq b$ بگونه‌ای که زمان اجرای $(b/r)(n+2^r)$ را کمینه کند

تعداد بیت

حالت اول: $b < \lfloor \lg n \rfloor$ ← $(n+2^r) = \Theta(n)$ ← برای $r = b$ خواهیم داشت: $(b/b)(n+2^b) = \Theta(n)$

حالت دوم: $b \geq \lfloor \lg n \rfloor$ ← برای $r = \lfloor \lg n \rfloor$ خواهیم داشت: $(b/\lfloor \lg n \rfloor)(n+n) = \Theta(bn/\lg n)$

radix sort یا انواع مرتب‌سازی مقایسه‌ای؟

- درصورتی که $b = O(\lg n)$ باشد با انتخاب $r = \lfloor \lg n \rfloor$ زمان اجری radix sort برابر $O(n)$
- در این صورت، تعداد فراخوان‌های radix sort به مراتب کمتر از quick sort خواهد بود درحالی که زمان اجرای هر فراخوان radix sort به مراتب طولانی تر از دیگری است

- اینکه کدام روش مرتب‌سازی بهتر است بستگی به نحوه پیاده‌سازی و سخت‌افزار دارد
- برای مثال quick sort از نسبت به radix از حافظه نهفته بهتر استفاده میکند
- radix sort درجا عمل نمیکند (درصورت استفاده از counting sort)
- در حالی که اکثر مرتب‌سازی‌های مقایسه‌ای درجا عمل میکنند