جلسه هفدهم

13:11 Tuesday, November 24, 2020

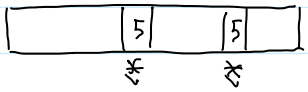
موضوع: مرتب سازی

مسأله: آرایه A شامل n عدد داده شده است. A را مرتب کنید.

برای مقایسه روش‌ها، چند پارامتر تعریف می‌کنیم.

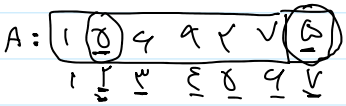
درجا یا inplace بودن: برای مرتب سازی A، نیاز به حافظه اضافی نباشد یا حداکثر $O(1)$

پایدار یا stable بودن: ترتیب عناصر برابر عوض نشود.



	زمان بدترین	زمان متوسط	درجا	پایدار
مرتب سازی حبابی:	$O(n^2)$	$O(n^2)$	✓	✓
" انتخابی	$O(n^2)$	$O(n^2)$	✓	✗
" درجی	$O(n^2)$	$O(n^2)$	✓	✓
" ادغامی	$O(n\log n)$	$O(n\log n)$	✗*	✓
" هرمی	$O(n\log n)$	$O(n\log n)$	✓	✗
" ددج	$O(n^2)$	$O(n\log n)$	✗	✓

* تمرین



* نکته: هر روش غیر پایداری را می‌توان پایدار کرد. با $O(n)$ حافظه اضافه

* چرا پایدار بودن مهم است.

مرتب‌سازی سریع Quick sort (choare 1959):

زیر مسأله Partition:

* Partition(l,h,x): آرایه شامل عناصر صدر اندیس‌های l تا h را طوری بگیرید. هدف

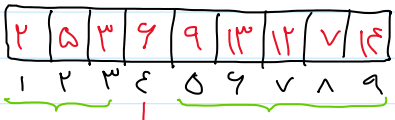
* قرارداده عنصر A[x] در محل مناسب [با توجه به محل آن در حالت مرتب شده]

* قرارداده عنصر $A[x]$ در محل مناسب [باتوجه به محل آن درحالت مرتب شده] و برگرداندن اندیس محل

* عناصر کوچکتر از $A[x]$ در سمت چپ آن و عناصر بزرگتر از $A[x]$ در سمت راست آن قرار گیرند.

مثال:

Partition(l, h, h)



عدد ۴ را برمی گرداند

$h-l=n$

* عمل Partition را در زمان $O(n)$ ... $O(h-l)$ می‌توان انجام داد.

```
Partition(l, h){
    Pivot = A[h]
    i = l
    for(j: l -> h-1){
        if A[j] < Pivot {
            swap(A[i], A[j])
            i++
        }
    }
    swap(A[i], A[h])
}
```

```
Partition(l, h, x){
    swap(A[h], A[x])   *
    Partition(l, h)
}
```

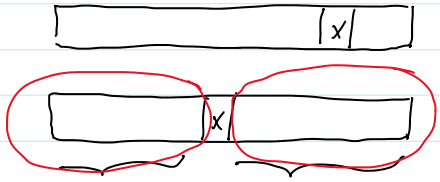
زمان: * $O(n)$

مرتب سازی سریع:

اندیس دلخواه بین l تا h

۱- آرایه را برحسب عنصر دلخواه x ، $A[l..h]$ partition کنید [فرض کنید که P محل عنصر x باشد]

۲- دو زیر آرایه $A[l..P-1]$ و $A[P+1..h]$ را به طور بازگشتی و با استفاده از مرتب‌سازی سریع مرتب کنید.

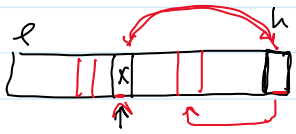


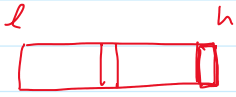

```
quick_sort(l, h){
```

```
if (l >= h)
    return;
  ↳ P = Partition(l, h)
  ↳ quick_sort(l, P-1)
  ↳ quick_sort(P+1, h)
}
```

$x = l \rightarrow \frac{l+h}{2}$, rand(l, h)

Partition(l, h, x)





تحلیل مرتب‌سازی سریع :

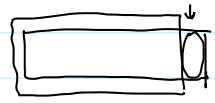
درجا : ✓

X : Stable

زمان اجرا :

بدترین حالت : ✓ محور همیشه آخری افتد

$T(n) = O(n) + T(n-1) = O(n^2)$ ✓



بهترین حالت :

$T(n) = O(n) + 2T(\frac{n}{2})$

$= O(n \log n)$

حالت متوسط : $O(n \log n)$

* به ازای هر انتخاب محور، یک ورودی وجود دارد که زمان مرتب‌سازی $O(n^2)$ می‌شود.

برای حل این مشکل : انتخاب عنصر محور به صورت رندم

$x = rand(l, h)$ ✓

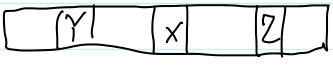
مرتب‌سازی تصادفی.

تحلیل مرتب‌سازی تصادفی سریع :

* فرض : اعداد آرایه از 1 تا n هستند. ✓

مشاهده ۱ : در الگوریتم Q.S ، عناصر صرفا با محورها مقایسه می‌شوند

مشاهده ۲ : اگر دو عنصر توسط یک محور از هم جدا شوند دیگر مقایسه نخواهند شد.



دیگر Y و Z مقایسه نخواهند شد.

100 n

i j ?

P_{ij} : احتمال مقایسه i و j = احتمال مقایسه شدن i و j توسط اولین محور بین اعداد i تا j ($i < k < j$)

$\frac{2}{j-i+1}$

کور بین اعداد $(i, \dots, j)$ باشد $\quad i < k < j \rightarrow$ ...

$$\frac{2}{|j-i|+1}$$

$$x_{ij} = \begin{cases} 0 & \text{اگر با هم مقایسه نشوند} \\ 1 & \text{بشوند} \end{cases}$$

هدف پیدا کردن $\sum_{i, j} x_{ij}$ است!

$$E[x_{ij}] = 1 \times P_{ij} + 0 \times (1 - P_{ij}) = P_{ij}$$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[x_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \underbrace{\frac{2}{|j-i|+1}}_{\to P_{ij}} = O(n \log n)$$

$\downarrow$ شبیه BST

$O(\log n)$

$O(n \log n)$