Selection in worst-case linear time

- از یک تقسیم بندی قطعی با Pivot مشخص استفاده می کنیم.
- مراحل کار به صورت زیر است. (SELECT)

۱- n تا عنصر را به $\lceil \frac{n}{5} \rceil$ تا گروه ۵ تایی تقسیم کن. اگر n بر 5 تقسیم پذیر نبود آخری کمتر عضو دارد.

۲- عنصر میانه (median) تا $\lceil n/5 \rceil$ گروه را بدست می آوریم.

- insertion sort را بر روی هر گروه انجام می دهیم، $O(1)$ برای هر گروه
- median هر گروه را در $O(1)$ بدست می آوریم.

۳- با استفاده عملیات بازگشتی میانه x این $\lceil n/5 \rceil$ گروه را می یابیم.

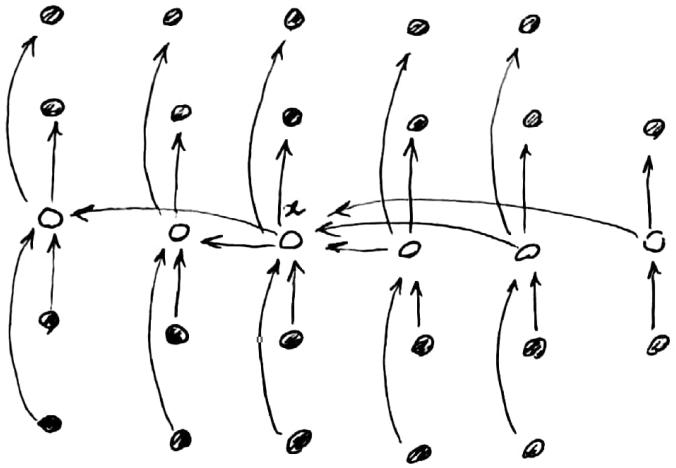
۴- از نسخه ای از PARTITION استفاده می کنیم که عنصر pivot را به عنوان آرگومان دریافت می کند.

آرایه ورودی را با توجه به x، تقسیم بندی می کنیم، فرض کنید x، k امین عنصر آرایه بعد از پارتیشن باشد بنابراین $k-1$ عنصر در یک قسمت و $n-k$ در قسمت دیگر قرار می گیرد.

۵- اکنون ۳ حالت ممکن است:

- اگر $k = i$ باشد، آنگاه x را برگردان
- اگر $k > i$ باشد، i امین کوچکترین را در قسمت کوچکتر به صورت بازگشتی می یابیم.
- اگر $k < i$ باشد، $(i-k)$ امین کوچکترین را در قسمت بزرگتر می یابیم.

تحلیل :



هر گروه ← یک ستون

هر دایره سفید، میانه ی گروه

جهت فلش ها از عنصر بزرگتر به عنصر کوچکتر است

تحلیل را با استفاده از تعداد عناصری که از x بزرگتر هستند انجام می دهیم

- حداقل نیمی از میانه های مرحله دوم بزرگتر یا مساوی x هستند

- در این گروه ها که میانه ایشان از x بزرگتر است، ۳ عنصر بزرگتر از x وجود دارد.
به جز گروهی که شامل x است که دو عنصر بزرگتر دارد و گروهی که کمتر از 5 عضو دارد.

- بزرگتر یا مساوی $\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 2$ گروه با ۳ عنصر بزرگتر یا مساوی x داریم.

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

- به طور مشابه همین تعداد عنصر هم کوچکتر x هستند

- بنابراین در گام ۴ مرحله ۵ الگوریتم SELECT بر روی کمتر یا مساوی $\frac{7n}{10} + 6$ عنصر بازگشت انجام می شود.

- steps 1, 2 and 4 $\longrightarrow$ $\Theta(n)$
  - step 1: $O(n)$
  - step 2: $\lceil \frac{n}{5} \rceil \times O(1)$
  - step 4: $O(n)$
- step 3 takes time $T(\lceil n/5 \rceil)$
- step 5 $\leq T(\frac{7n}{10} + 6)$

با فرض آنکه $T(n)$ بصورت یکنواخت افزایشی است.

$$T(n) = O(1) \qquad n < \text{constant}$$

140 ?

$$T(n) \leq \begin{cases} O(1) & \text{if } n < 140 \\ T(\lceil n/5 \rceil) + T(\frac{7n}{10} + 6) + O(n) & \text{if } n \geq 140 \end{cases}$$

$$T(n) = \Theta(n)$$

برای اثبات به کتاب CLRS مراجعه کنید.

جمع بندی کلی:

- عملیات مرتب سازی در مدل مقایسه‌ای $\Omega(n \lg n)$
- عملیات مرتب سازی در زمان خطی نیازمند فرضیات دارد.
- الگوریتم Selection زمان خطی بدون نیاز به فرضیات است.
- الگوریتم Selection آرایه ورودی را مرتب نمی کند.

Amortized Analysis

تحلیل سرشکن :

- تحلیل یک دنباله از operation ها بروی data structure

- انجام operation تنها می‌تواند بسیار پرهزینه‌تر از، میانگین هزینه تعداد زیادی از همان operation ها باشد

- میانگین در اینجا به نحوه تابع توزیع ورودی ارتباطی ندارد. شامل احتمالات نیست بلکه میانگین بروی worstcase لحاظ می‌شود.

سه روش تحلیل سرشکن :

- aggregate analysis
- acounting method
- potential method

با استفاده از ۳ مثال :

- staek with multipop operation
- binary counter
- dynamic tables

Aggregate analysis

stack operations

- PUSH(S, x) : O(1)
- POP(S) = O(1)

MULTIPOP عملیات PUSH و POP هرکدام O(1) زمان می برد. یک operation جدید بنام تعریف می کنیم.

- MULTIPOP(S, k)

```
while S is not empty and k > 0
    do POP(S)
       k ← k − 1
```

زمان اجرای MULTIPOP :

- نسبت به تعداد POP ها خطی است.
- چون هزینه PUSH و POP برابر یک است.
- تعداد گذارهای حلقه while برابر میشیم S و k است که S تعداد عناصر stack است.
- هزینه کل min(S, k) .

یک دنباله n تایی از POP، PUSH، MULTIPOP در بدترین حالت O(n)، O(n) و $O(n^2)$ زمان می برد.

observation

- هر object فقط در ازای یک PUSH می‌تواند یکبار pop شود.
- اگر n تا PUSH داشته باشیم تعداد کل popها شامل pop تنها و MULTIPOP کمتر یا مساوی n است.
- بنابراین اگر n تا operation انجام شود شامل PUSH، POP، MULTIPOP در حالت worst-case، O(n) هزینه صرف می‌شود.
- هزینه سر شکن هر کدام O(1) است.
- به این تحلیل aggregate analysis می‌گویند.

Binary Counter:

یک شمارنده دودویی k بیتی که شامل k بیت A[0 .. k-1] است.

A[0] کمترین ارزش مکانی A[k-1] بیشترین ارزش مکانی

شمارنده رو به بالا upward است که از 0 شروع می‌کند.

- value نشان دهنده روی counter $\sum_{i=0}^{k-1} A[i] \cdot 2^i$

در ابتدا 0 است و هر افزایش یکواحدی است.

```
INCREMENT (A,k)
i ← 0
while i < k and A[i] = 1
    do A[i] ← 0
       i ← i+1
if i < k
    then A[i] ← 1
```

مثال: k-bit counter (k=3)

counter value	A [2 1 0]	cost
0	000	0
1	001	1
2	010	3
3	011	4
4	100	7
5	101	8
6	110	10
7	111	11
0	000	14
⋮		

Cost of INCREMENT $= \theta$(# of bits flipped)

مثال: هر فراخوانی می‌تواند تا k بیت را flip کند پس n تا INCREMENT

$O(nk)$.

همه بیت‌ها در هر افزایش flip نمی‌شوند.

bit	flips how often	times in n INCREMENTs
0	every time	n
1	$1/2$ the time	$\lfloor n/2 \rfloor$
2	$1/4$ the time	$\lfloor n/4 \rfloor$
⋮	⋮	⋮
i	$1/2^i$ the time	$\lfloor n/2^i \rfloor$
⋮	⋮	⋮
$i \geq k$	never	0

$$\#\text{ of flips} = \sum_{i=0}^{k-1} \lfloor n/2^i \rfloor \leq n\left(\frac{1}{1-1/2}\right)$$

$$= 2n$$

n تا INCREMENT با هزینه $O(n)$

هزینه متوسط operation $= O(1)$