دانشگاه صنعتی شریف
دانشکدهٔ ریاضی

ساختمان‌داده‌ها
بهار ۱۴۰۰
استاد: حسین بومری
گردآورنده:علی سرائر

جلسهٔ سیزدهم

ادامهٔ سورت‌ها

QuickSort

توضیح: در مرحلهٔ اول، یک عدد به عنوان محور[1] انتخاب می‌شود. حال دو پوینتر تعریف می‌کنیم. پوینتر قرمز جای محور قرار می‌گیرد و پوینتر آبی از آخر آرایه، عدد سمت چپ خود را هر دفعه با عدد محور مقایسه می‌کند، و اگر عدد سمت چپ پوینتر آبی از عدد محور کوچک‌تر باشد، آن را به سمت چپ پوینتر قرمز منتقل می‌کند.(swapping) این کار تا زمانی انجام می‌شود که پوینتر آبی کنار پوینتر قرمز قرار بگیرد. حال برای کامل کردن سورت، این کار را یک بار برای همهٔ اعداد سمت چپ محور و یک بار برای اعداد سمت راست آن تکرار می‌کنیم.
مثال:
مرحل اول سورت کردن اعداد زیر، به این صورت است:

```
   ۲ ۳ ۱ ۴ ۵ ۷ ۲
   | ۳ ۱ ۴ ۵ ۷ ۲|
   | ۳ ۱ ۴ ۵ ۷| ۲
   | ۳ ۱ ۴ ۵| ۷ ۲
   | ۳ ۱ ۴| ۵ ۷ ۲
   | ۳ ۱| ۴ ۵ ۷ ۲
    ۱| ۳| ۴ ۵ ۷ ۲
    ۱|| ۳ ۴ ۵ ۷ ۲
  ۱ |۲| ۳ ۴ ۵ ۷ ۲
```

اگر i امین عدد پس از سورت را به عنوان محور انتخاب کنیم(هنوز نمیدانیم این i چند است، پس از سورت کردن مشخص می‌شود)، مرتبهٔ زمانی به شکل زیر می‌شود.

$$T(n) = T(n - i - ۱) + T(i) + O(n);$$

که $T(n - i - ۱)$ اعمال انجام شده روی اعداد سمت چپ محور است و $T(i)$ روی اعداد سمت راست محور است.
همچنین واضح است که در پیمایش اعداد با پوینتر آبی، در هر مرحله از مرتبه n تا کار انجام می‌دهیم.
دقت کنید که بدترین حالت برای این سورت زمانی اتفاق می‌افتد که اعداد از همان اول سورت شده باشند. به عنوان مثال، اگر اعداد ورودی به شکل زیر باشند

۱ ۲ ۳ ۴ ۵ ۶

خواهیم داشت

[1] pivot

$$T(n) = T(n-۱) + T(۰) + O(n)$$
$$\rightarrow T(n) \in O(n^۲)$$

همچنین بهترین حالت این است که در هر مرحلهٔ سورت کردن، میانه را به عنوان محور انتخاب کنیم.[۲] اگر هر دفعه میانه را انتخاب کنیم، همیشه نصف اعداد سمت راست محور قرار می‌گیرند و نصف اعداد سمت چپ آن؛ بنابراین خواهیم داشت:

$$T(n) = ۲T(\frac{n}{۲}) + O(n)$$
$$\rightarrow T(n) \in O(n\, log\, n)$$

یادآوری: به مثال زیر دقت کنید. در الگوریتم insetion sort از دومین عدد سمت چپ شروع می‌کنیم، اگر از عدد سمت چپ خود کوچک تر بود، جای این دو تا را با هم عوض می‌کنیم. در این مثال ۷ از ۱ بزرگتر است، بنابراین جای آن‌ها تغییر نمی‌کند. حال یک خانه جلوتر می‌رویم. این عدد را با عدد سمت چپش که ۷ است مقایسه می‌کنیم، و چون کوچک تر است آن‌ها را جابجا می‌کنیم. در مرحلهٔ بعد ۴ را با ۱ مقایسه می‌کنیم،اما چون ۴ از ۱ بزرگتر است، جای ۴ مشخص شده است. حال به خانه بعدی می‌رویم و همین کار را آنقدر تکرار می‌کنیم تا جای هر عنصر مشخص شود. به این صورت اعداد ما سورت خواهند شد.

۱ ۷ ۴ ۱ ۲ ۳ ۵ ۷
۱ ۷ ۴ ۱ ۲ ۳ ۵ ۷
۱ ۴ ۷ ۱ ۲ ۳ ۵ ۷
۱ ۴ ۷ ۱ ۲ ۳ ۵ ۷
۱ ۴ ۱ ۷ ۲ ۳ ۵ ۷
۱ ۱ ۴ ۷ ۲ ۳ ۵ ۷

در جلسات گذشته مطرح شد که این الگوریتم در بهترین حالت از $O(n)$ است و در بدترین حالت از $O(n^۲)$ است.(زمانی بدترین حالتی می‌شد که اعداد برعکس سورت شده باشند و بهترین حالت مربوط به زمانی بود که اعداد از اول سورت شده بودند.)
حال می‌خواهیم مرتبهٔ زمانی میانگین Quick Sort را پیدا کنیم. همه حالت‌های ممکن، برای انتخاب اعداد مختلف به

[۲] در ادامه یک الگوریتم خوب معرفی می‌کنیم که می‌تواند میانه را در زمان $O(n)$ پیدا کند.

عنوان محور را در نظر می‌گیریم. آن‌ها را با هم جمع می‌کنیم و بر تعداد تقسیم می‌کنیم، تا مرتبهٔ میانگین به دست بیاید.

$$
\begin{aligned}
T(n) &= \frac{T(0)+T(n-1)+T(1)+T(n-2)+\dots}{n}\\
&= \frac{\sum_{i=0}^{n-1} T(i)+T(n-i-1)}{n}\\
&= \frac{2\sum_{i=1}^{n-1} T(i)}{n}+O(n)\\
&\rightarrow nT(n) = 2\sum_{0}^{n-1} T(i)+n^2\\
&\rightarrow (n-1)T(n-1) = 2\sum_{0}^{n-2} T(i)+(n-1)^2\\
&\rightarrow nT(n)-(n-1)T(n-1) = 2T(n-1)+O(n)\\
&\rightarrow nT(n) = (n+1)T(n-1)+O(n)\\
&\rightarrow T(n) = \frac{n+1}{n}T(n-1)+O(1)\\
&\rightarrow T(n) = \frac{n+1}{n}(\frac{n}{n-1}T(n-2)+1)+1\\
&\rightarrow T(n) = \prod_{i=0}^{n-1}\frac{n-i+1}{n-i}T(0)+n+1+\dots+\frac{n+1}{n-1}+\frac{n+1}{n}+\frac{n+1}{n+1}\\
&\xrightarrow{T(0)=1} T(n) = \frac{n+1}{1}+(n+1)(1+\frac{1}{2}+\frac{1}{3}+\dots+\frac{1}{n+1}) = n\,log\,n
\end{aligned}
$$

بنابراین Quick Sort به طور میانگین، از مرتبهٔ $n\,log\,n$ است.

راهی برای بهتر کردن Quick Sort

در ادامهٔ یک الگوریتم معرفی خواهیم کرد که با استفاده از آن میانه [3] را با مرتبهٔ زمانی $O(n)$ پیدا می‌کنیم. واضح است که اگر میانه را به عنوان محور در نظر بگیریم خواهیم داشت

$$T(n) = O(n)+2T(n/2)+O(n) = n\,log\,n$$

که $O(n)$ اولی مربوط به مرتبهٔ زمانی پیدا کردن میانه، و $O(n)$ دومی مربوط به پیمایش اعداد و تقسیم بندی آن‌ها به دو آرایهٔ چپ و راست محور است(در بالا توضیح داده شد).
حال کافی است الگوریتم پیدا کردن میانه را توضیح دهیم.

الگوریتم پیدا کردن میانه

اعداد زیر را در نظر بگیرید

$$a_1\ a_2\ \dots\ a_n$$

این اعداد را به دسته های ۵ تایی تقسیم می‌کنیم.هر بستهٔ ۵ تایی، میانه را به دست می‌آوریم،که به صورت زیر می‌شوند

$$m_1\ m_2\ \dots\ m_{n/5}$$

[3] median

حال میانهٔ این میانه‌ها را به دست می‌آوریم و آن را p_m می‌نامیم. مرتبهٔ زمانی همهٔ این اعمال، معادلهٔ زیر را به ما می‌دهد.

$$T(n) = O(n) + T(n/۵) + O(n)$$

که $O(n)$ اولی هزینهٔ پیدا کردن median های دسته‌های ۵ تایی. $T(n/۵)$ هم مربوط به تکرار همین الگوریتم به صورت بازگشتی برای پیدا کردن میانهٔ میانه‌ها است.
$O(n)$ دومی هم هزینهٔ این است که ببینیم چند عدد از p_m کوچک‌ترند و چندتا بزرگ‌ترند. با استفاده از الگوریتم partitioning که با دو اشاره‌گر اعداد سمت چپ و راست محور را پیدا کردیم(همان الگوریتم Quick Sort)، با مرتبهٔ زمانی $O(n)$ می‌توان مکان p_m را پیدا کرد.
برای ادامهٔ کار فقط به نصفهٔ راست و یا نصفهٔ چپ p_m نیاز داریم.برای روشن‌تر شدن،این نکته را با یک مثال توضیح می‌دهیم.
مثال: فرض کنیم ۱۰۰ عدد داریم و دنبال عدد ۲۵ام می‌گردیم. واضح است که دستهٔ چپ از p_m کوچک‌تر و دستهٔ راست از آن بزرگ‌تر است. حال اگر بخواهیم دنبال ۲۵امین عدد بگردیم، باید دنبال آن در سمت چپ p_m بگردیم. بنابراین می‌توان دستهٔ سمت راست را دور ریخت.
از آنجا که p_m میانهٔ m_i ها بود، بنابراین نصف آن‌ها از p_m قطعا کوچک‌ترند و نصف آن‌ها قطعا بزرگ‌ترند. از آنجا که $\frac{n}{۵}$ تا دسته داشتیم، بنابراین حداقل $\frac{n}{۱۰}$ میانه‌ها از p_m کوچک‌ترند و $\frac{n}{۱۰}$ از آن بزرگ‌ترند.
به ازای هر m_i که p_m از آن بزرگ‌تر است، خود میانه و ۲ عدد کم‌تر از آن(در دسته‌های ۵ تایی)، از p_m کوچک‌ترند. بنابراین حداقل $\frac{۳}{۱۰}n$ اعداد از p_m کوچک‌ترند. با استدلال مشابه واضح است که حداقل $\frac{۳}{۱۰}n$ از p_m بزرگ‌ترند. بنابراین، تا اینجای کار به دست آوردیم که:

$$\text{تا } \frac{۳n}{۱۰} < p_m < \frac{۳n}{۱۰}\text{تا}$$

بنابراین سایز هر کدام از دسته‌های راست یا چپ حداکثر $\frac{۳n}{۱۰}$ است. می‌دانیم برای پیدا کردن میانهٔ کل اعداد، کافی است دنبال عدد مطلوب داخل یکی از دسته‌های چپ یا راست بگردیم.(بالاخره یا از p_m کوچک‌تر است یا بزرگ‌تر،در نتیجه داخل یکی از دسته‌های راست یا چپ قرار دارد) از آن‌جا که هر دسته حداقل $\frac{۳n}{۱۰}$ عضو دارد، بنابراین حداکثر تعداد اعضای هر دسته $\frac{۷n}{۱۰}$ است. پس حالا باید همین الگوریتمی که توضیح دادیم را حداکثر روی $\frac{۷n}{۱۰}$ اعداد پیاده‌سازی کنیم تا میانهٔ کل را پیدا کنیم. پس تا اینجا داریم:

$$T(n) = O(n) + T(\frac{n}{۵}) + O(n) + T(\frac{۷}{۱۰}n) \qquad (۱)$$

قبل از ادامه،لازم است یک نکته را یادآوری کنیم.

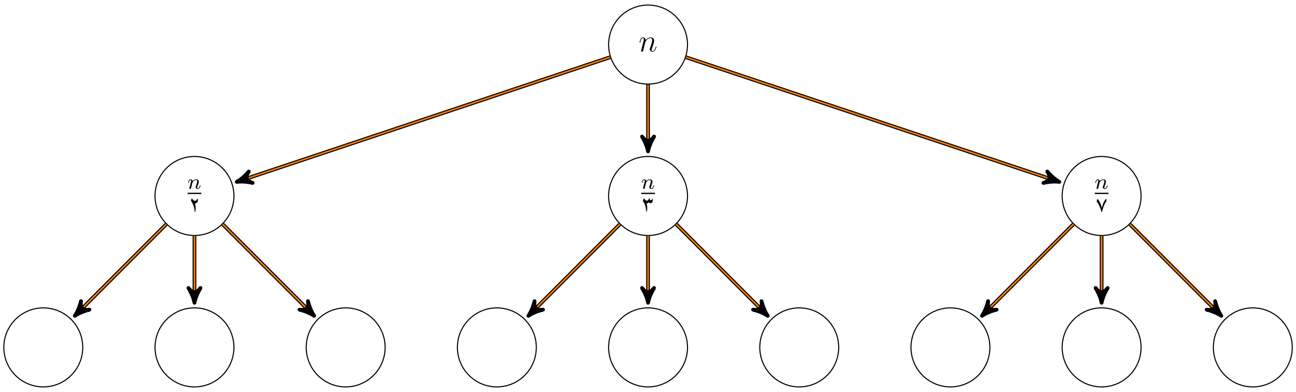
Master Theorem:

$$T(n) = a\,T(\frac{n}{b}) + O(n^{c'})$$
$$if\ \ log(\frac{a}{b}) = c' \Rightarrow T(n) = O(n^{c'} log\, n)$$
$$\Rightarrow T(n) = k\,T(\frac{n}{k}) + O(n) = O(n\, log\, n) + O(n) = O(n\, log\, n)$$

حال مثال[۴] زیر را بررسی می‌کنیم.
مثال: اگر داشته باشیم

$$T(n) = T(\frac{n}{۲}) + T(\frac{n}{۳}) + T(\frac{n}{۷}) + O(n)$$

[۴] که به نکتهٔ فوق ربط خاصی ندارد(:



شکل ۱: درخت مثال

می‌خواهیم نشان دهیم چون

$$\frac{n}{2}+\frac{n}{3}+\frac{n}{7}\leq n$$

داریم

$$T(n)=O(n)$$

درخت هزینه‌ها به صورت فوق می‌شود.
واضح است که هزینهٔ خط سطر دوم برابر

$$\frac{n}{2}+\frac{n}{3}+\frac{n}{7}=\frac{41}{42}n$$

می‌شود. به همین ترتیب در سطر بعدی،هزینه‌ها برابر $n^2(\frac{41}{42})$ است. اگر قرار دهیم $\alpha=\frac{41}{42}<1$ جمع همهٔ هزینه‌ها از سطر اول تا آخر برابر است با:

$$(1+\alpha+\alpha^2+\ldots)n=\frac{1}{1-\alpha}n \quad , \alpha<1$$

بنابراین جمع همهٔ این هزینه‌ها یک ثابت ضربدر n می‌شود که از مرتبهٔ $O(n)$ می‌شود. بنابراین اگر جمع ضریب پشت $T(n)$ ها از یک کم‌تر باشد، و هزینه هر عمل نیز از $O(n)$ باشد، هزینه کل نیز از همان $O(n)$ می‌شود.
طبق معادلهٔ (۱) نیز چون

$$\frac{n}{5}+\frac{7}{10}n=\frac{9}{10}n<n$$

داریم:

$$T(n)=O(n)$$

بنابراین مرتبهٔ پیدا کردن میانه با این الگوریتم از $O(n)$ می‌شود.
بنابراین می‌توان با این الگوریتم میانه را از مرتبهٔ زمانی $O(n)$ پیدا کرد و در Quick Sort قرار داد، تا از مرتبهٔ زمانی $O(n\,log\,n)$ کل اعداد سورت شوند.