دانشکده مهندسی کامپیوتر

جزوه درس

ساختمان‌های داده

استاد درس: سید صالح اعتمادی

پاییز ۱۳۹۸

جلسه ۶

قضیه اصلی و مرتب سازی

ملیکا نوبختیان - ۱۳۹۸/۷/۱۸

جزوه جلسه ۶ام مورخ ۱۳۹۸/۷/۱۸ درس ساختمان‌های داده تهیه شده توسط ملیکا نوبختیان. در جهت مستند کردن مطالب درس ساختمان های داده، بر آن شدیم که از دانشجویان جهت مکتوب کردن مطالب کمک بگیریم. هر دانشجو می‌تواند برای مکتوب کردن یک جلسه داوطلب شده و با توجه به کیفیت جزوه از لحاظ کامل بودن مطالب، کیفیت نوشتار و استفاده از اشکال و منابع کمک آموزشی، حداکثر یک نمره مثبت از بیست نمره دریافت کند. خواهش‌مند است نام و نام خانوادگی خود، عنوان درس، شماره و تاریخ جلسه در ابتدای این فایل را با دقت پر کنید.

۱.۶ معیار ارزیابی جزوه

معیارهای مورد استفاده برای ارزشیابی کیفیت جزوه به شرح زیر است:

- پوشش مطالب
- رعایت قواعد نگارشی دستور زبان فارسی
- استفاده از اشکال مناسب
- اشاره به منابع کمک آموزشی

۲.۶ Master Theorem

در تحلیل و واکاوی الگوریتم ها،قضیه اصلی یک راه حل سر راست برای بسیاری از الگوریتم های تقسیم و حل که بازگشتی هستند ارائه می کند.الگوریتم هایی را که می توان به شکل :

$$T(n) = aT([\frac{n}{b}]) + O(n^d)$$

نوشت،با بسط دادن می توان کار کل انجام شده آن را بدست آورد اما قضیه اصلی به ما اجازه می دهد به آسانی پیچیدگی زمانی این الگوریتم های بازگشتی را بدون بسط دادن حساب کنیم.

۳.۶ شکل کلی قضیه اصلی

اگر الگوریتمی را بتوان به شکل زیر نوشت،پیچیدگی زمانی آن به راحتی حساب می شود:

$$T(n) = aT([\frac{n}{b}]) + O(n^d)$$

$$a > 0, b > 1, d >= 0$$

در اینجا aتعداد زیر مسئله ها، n/bاندازه هر زیر مسئله و n^dمقدار کاری است که باید انجام دهیم تا این زیر مسئله ها را با هم جمع کنیم. شکل کلی:

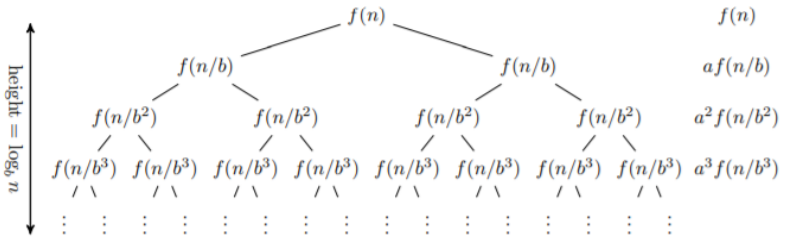
$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

شکل ۱.۶: Master Theorem

۴.۶ اثبات قضیه اصلی

ابتدا درخت الگوریتم مورد نظرمان را که به دلیل رابطه بازگشتی ساخته می شود را می کشیم.درخت ما عمق$log_b n$ را دارد. a تعداد فرزندان هرگره درخت را نشان می دهد و در عمق i از درخت a^i گره داریم. پیچیدگی زمانی

هر گره n/b^i است. پیچیدگی زمانی کلی الگوریتم برابرجمع پیچیدگی های زمانی همه ی گره ها است. بنابراین در این درخت $n^{\log_b a}$ برگ داریم پس پیچیدگی زمانی کلی برگ ها $O(n.\log_b a)$ می شود.حالا می توانیم با توجه به پیچیدگی زمانی $O(n^d)$ پیچیدگی زمانی کلی الگوریتم را حساب کنیم



شکل ۲.۶: اثبات قضیه اصلی

$$
\begin{aligned}
&O(n^d) + a.O(n/b)^d + ... + a^i.O(n/b^i)^d + ... + a^{\log_b n} = \\
&O(n^d) + O(n^d)(a/b^d) + ... + O(n^d)(a/b^d)^i + ... + O(n^{\log_b a}) = \\
&\sum_{i=0}^{\log_b n} O(n^d)(a/b^d)^i
\end{aligned}
\tag{۱.۶}
$$

۵.۶ مرتب سازی

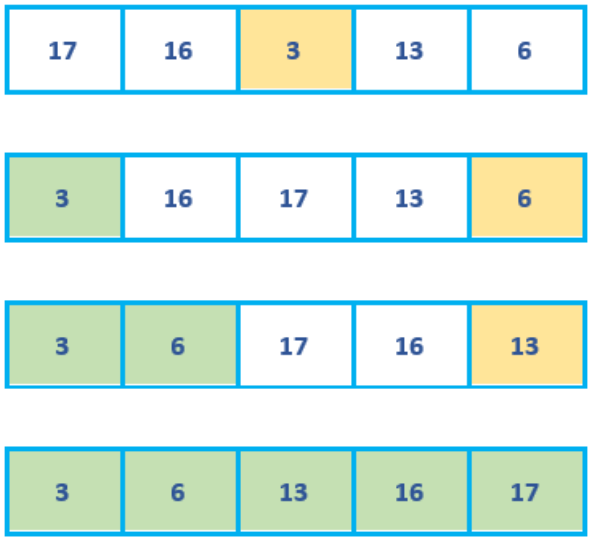
چرا مرتب سازی مهم است؟

- مرتب سازی اطلاعات یک قدم مهم در بسیاری از الگوریتم های کارامد است.
- اطلاعات مرتب شده اجازه جست و جوهای کارآمدتر را به ما می دهد.

۶.۶ Selection Sort

مرتب سازی انتخابی یکی از ساده ترین الگوریتم های مرتب سازی است.پیچیدگی زمانی آن $O(n^2)$است.
مرتب سازی انتخابی شامل سه مرحله است:

- پیدا کردن کوچک ترین عدد با بررسی آرایه
- جا به جا کردن آن عدد با عنصر اول آرایه
- تکرار کردن همین روش با قسمت باقیمانده آرایه



شکل ۳.۶: selection sort

Algorithm 1 Selection Sort

Data: A[1....n]

Result: Sorted Array of A

initialization **for** *i from 1 to n* **do**

    minIndex ← *i* **for** *j from i+1 to n* **do**

        if *A[j]<A[minindex]* **then**

            minIndex ← *j*

        end

        Swap(A[i],A[minindex])

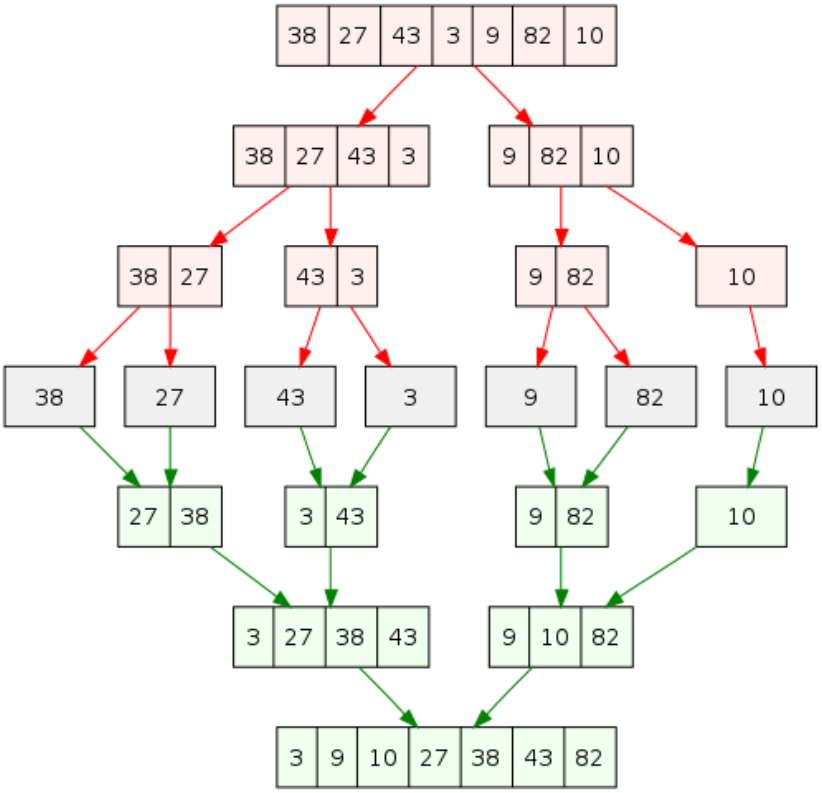
    end

end

Merge Sort ۷.۶

مرتب سازی ادغامی یکی از روش های مرتب سازی و نمونه ای از الگوریتم های تقسیم و حل است.پیچیدگی زمانی این الگوریتم$O(nlogn)$است. مرتب سازی ادغامی شامل مراحل زیر است:

- تقسیم کردن آرایه به دو نیمه
- مرتب کردن نیمه ها به طور بازگشتی
- ادغام کردن نیمه های مرتب شده در یک آرایه



شکل ۴.۶: merge sort

پیچیدگی زمانی مرحله ادغام کردن $O(n)$است.پس پیچیدگی زمانی کلی الگوریتم از رابطه بازگشتی زیر به دست می آید:

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

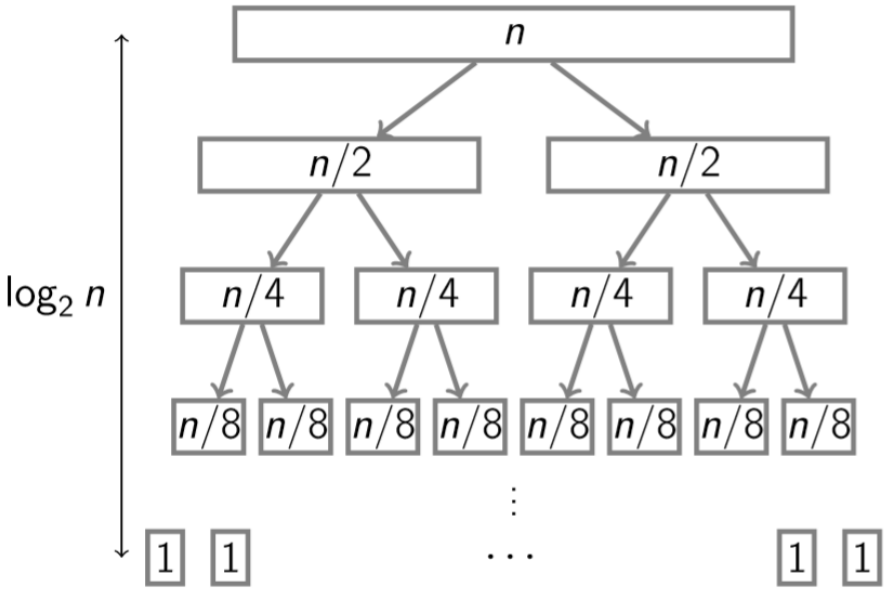
چون پیچیدگی زمانی الگوریتم ما از رابطه بازگشتی به دست می آید پس می توانیم با استفاده از قضیه اصلی پیچیدگی زمانی آن را بدست آوریم. شبه کدهای دو مرحله ادغام کردن و مرتب سازی به صورت زیر است:

```
MergeSort(A[1...n])
if n = 1:
  return A
m ← ⌊n/2⌋
B ← MergeSort(A[1...m])
C ← MergeSort(A[m+1...n])
A' ← Merge(B, C)
return A'
```

شکل ۵.۶: Merge Sort

```
Merge(B[1...p], C[1...q])
{B and C are sorted}
D ← empty array of size p + q
while B and C are both non-empty:
  b ← the first element of B
  c ← the first element of C
  if b ≤ c:
    move b from B to the end of D
  else:
    move c from C to the end of D
move the rest of B and C to the end of D
return D
```

شکل ۶.۶: Merge



شکل ۷.۶: merge sort tree

Bibliography