بسم الله الرحمن الرحیم

ساختمان‌های داده

جلسه ۳

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

تحلیل الگوریتم‌ها

- ما اغلب به روش‌های مختلفی می‌توانیم یک مسئله را حل کنیم.
- یافتن یک راه حل برای مسئله کافی نیست این راه حل باید کارآمد (ترین) باشد.

تحلیل الگوریتم‌ها

IUT-ECE

- محدودیت‌های استفاده از روش تجربی برای تحلیل:

  - باید الگوریتم را پیاده کنیم که ممکن است کار زمانبری باشد.
  - نتیجه ممکن است نشان دهنده زمان اجرای برنامه روی ورودی‌های دیگر خارج از محدوده آزمایش نباشد.
  - برای مقایسه دو الگوریتم نیاز است که از سخت‌افزار و محیط نرم‌افزاری یکسان استفاده شود.

تحلیل الگوریتم‌ها

- تحلیل نظری:

  - به جای اینکه الگوریتم را پیاده کنیم از یک توصیف سطح بالای الگوریتم استفاده می‌کنیم.
  - همه ورودی‌های ممکن را مد نظر قرار می‌دهیم.
  - این کار ما را قادر می‌سازد مستقل از سخت‌افزار و محیط نرم‌افزاری الگوریتم را تحلیل کنیم.

تحلیل الگوریتم‌ها

- تحلیل الگوریتم‌ها با هدف‌های زیر انجام می‌شود:
  - بررسی و پیش‌بینی زمان اجرا و میزان حافظه مصرفی یک الگوریتم قبل از پیاده‌سازی
  - مقایسه الگوریتم‌های مختلف برای حل یک مسئله از نظر میزان کارایی

تحلیل الگوریتم‌ها

- تحلیل زمانی
  - چقدر طول می‌کشد تا الگوریتم اجرا شود.
- تحلیل فضا
  - میزان حافظه مورد نیاز الگوریتم برای اجرای الگوریتم.
  - حافظه مورد نیاز برای ذخیره داده ورودی را حساب نمی‌کنیم.

تحلیل زمانی الگوریتم‌ها

IUT-ECE

- عوامل زیر در زمان اجرای یک برنامه بر روی یک کامپیوتر موثرند:
  - سرعت پردازنده کامپیوتر
  - نوع کامپایلر یا زبان برنامه نویسی استفاده شده
  - اندازه داده ورودی مسئله
  - ترکیب یا ساختار داده‌های ورودی
  - پیچیدگی الگوریتم استفاده شده در برنامه
  - عوامل دیگر که وابسته به ورودی نیستند و تاثیر خطی در زمان اجرای برنامه دارد.

تحلیل زمانی الگوریتم‌ها

IUT-ECE

- اندازه‌گیری زمان اجرای الگوریتم‌ها، مستقل از سخت افزار یا ماشین
- درک رابطه آن با اندازه ورودی
- هر ورودی مسئله دو مشخصه مهم دارد: یکی اندازه یا تعداد داده‌ها و دیگری ترکیب یا ساختار آنها.
  - مثال: مرتب بودن یا نبودن مجموعه داده در یک الگوریتم مرتب سازی (بهترین و بدترین حالت)
- مقایسه کارآمدی الگوریتم‌ها

مسئله مرتب‌سازی

- مسئله مرتب‌سازی:

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

مرتب‌سازی

- کاربردهای عملی زیاد
  - رده‌بندی: شهرهای ایران بر اساس جمعیت، تیم‌ها بر اساس تعداد جام، قیمت‌ها کاهشی
- پایه‌ای برای دیگر الگوریتم‌ها
  - استفاده در جستجوی دودویی
  - اشتراک یا اجتماع روی دو مجموعه وقتی که مرتب شده‌اند ساده‌تر است.
- روش‌های مختلفی با بده بستان‌های متفاوتی وجود دارند.

خواص مرتب‌سازی

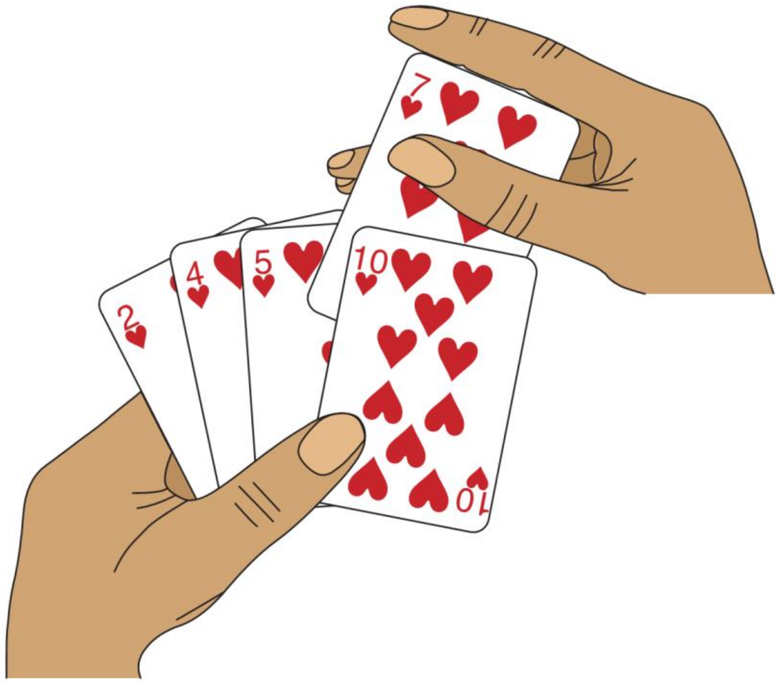
- پایدار: جایگاه نسبی مواردی که مقادیر یکسان دارند تغییر نمی‌کند.
- وفقی: پیچیدگی زمانی وابسته به ورودی خواهد بود.

موارد دیگر درباره مرتب‌سازی

- پیچیدگی زمانی: بهترین/ میانگین/ بدترین
- پیچیدگی فضا:
  - روش‌های "در جا": حافظه مورد نیاز مستقل از اندازه ورودی و مقدار ثابتی است.

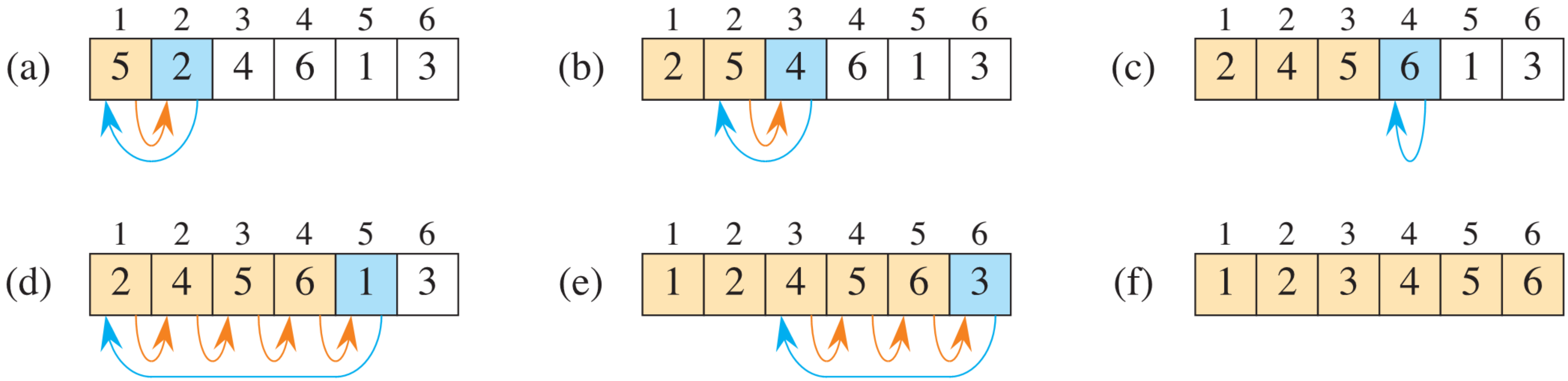
مرتب‌سازی درجی

- یک الگوریتم برای حل این مسئله، مرتب‌سازی درجی است.



مرتب‌سازی درجی

1	2	3	4	5	6
5	2	4	6	1	3



مرتب‌سازی درجی

- پایدار؟
- وفقی؟
- درجا؟

مرتب‌سازی درجی

- کد مرتب‌سازی درجی:

```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

IUT-ECE

تحلیل مرتب‌سازی درجی

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n-1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$j = i - 1$	c_4	$n-1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$A[j+1] = key$	c_8	$n-1$

- t_i نشان‌دهنده تعداد اجرای حلقه while برای مقدار i متناظر است.

IUT-ECE

تحلیل مرتب‌سازی درجی

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n - 1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n - 1$
$j = i - 1$	c_4	$n - 1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$A[j+1] = key$	c_8	$n - 1$

Running time:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) .$$

تحلیل مرتب‌سازی درجی

IUT-ECE

INSERTION-SORT(A, n)	cost	times
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n-1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$j = i-1$	c_4	$n-1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n}(t_i - 1)$
$j = j-1$	c_7	$\sum_{i=2}^{n}(t_i - 1)$
$A[j+1] = key$	c_8	$n-1$

- تحلیل بهترین حالت:

$$\boxed{t_i = 1 \text{ for } i = 2, 3, \ldots, n,}$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{i=2}^{n} t_i + c_6\sum_{i=2}^{n}(t_i - 1) + c_7\sum_{i=2}^{n}(t_i - 1) + c_8(n-1) .$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) .$$

تحلیل مرتب‌سازی درجی

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$\quad key = A[i]$	c_2	$n-1$
$\quad$ // Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$\quad j = i-1$	c_4	$n-1$
$\quad$ **while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$\qquad A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$\qquad j = j-1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$\quad A[j+1] = key$	c_8	$n-1$

- تحلیل بهترین حالت:

$$\boxed{t_i = 1 \text{ for } i = 2, 3, \ldots, n,}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \, . \end{aligned}$$

تابع خطی از n

$$\boxed{T(n) = an + b}$$

تحلیل مرتب‌سازی درجی

IUT-ECE

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n-1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$j = i - 1$	c_4	$n-1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$A[j+1] = key$	c_8	$n-1$

- تحلیل بدترین حالت:

$$\boxed{t_i = i \text{ for } i = 2, 3, \ldots, n.}$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) .$$

تحلیل مرتب‌سازی درجی

IUT-ECE

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n-1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$j = i - 1$	c_4	$n-1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$A[j+1] = key$	c_8	$n-1$

- تحلیل بدترین حالت:

$$\boxed{t_i = i \text{ for } i = 2, 3, \ldots, n.}$$

$$\boxed{\begin{aligned}\sum_{i=2}^{n} i &= \left(\sum_{i=1}^{n} i\right) - 1 \\ &= \frac{n(n+1)}{2} - 1\end{aligned}}$$

$$\boxed{\begin{aligned}\sum_{i=2}^{n} (i-1) &= \sum_{i=1}^{n-1} i \\ &= \frac{n(n-1)}{2}\end{aligned}}$$

تحلیل مرتب‌سازی درجی

INSERTION-SORT(A, n)	*cost*	*times*
for $i = 2$ **to** n	c_1	n
$key = A[i]$	c_2	$n-1$
// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
$j = i - 1$	c_4	$n-1$
while $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
$A[j+1] = key$	c_8	$n-1$

- تحلیل بدترین حالت:

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\
&\quad - (c_2 + c_4 + c_5 + c_8)\ .
\end{aligned}
$$

تابع مربعی از n

$$\boxed{T(n) = an^2 + bn + c}$$

تحلیل در بدترین حالت

- در این درس ما غالبا الگوریتم‌ها را در بدترین حالت تحلیل می‌کنیم.
  - برای راحتی
  - یافتن کران بالا (محتاطانه)
  - اغلب در عمل مانند حالت میانگین

تحلیل الگوریتم

- مدل محاسباتی RAM
  - فرض می‌شود هر دستورالعمل و دسترسی به داده، میزان یکسانی از زمان (یک واحد) را لازم دارد.
  - شامل دستورالعمل‌های پایه متداول مانند عملیات ریاضی (جمع، ضرب، …)، جابجایی داده (کپی، ذخیره‌سازی و…)، تخصیص و … است.

- هزینه یا زمان اجرای یک برنامه روی یک ورودی خاص برابرست با تعداد کل دستورالعمل‌ها و دسترسی به داده‌ها برای انجام آن

مثال

```
int example1(int n) {
    int x;   +1
    int y = n - 1;   +2
    x = n + 1;   +2
    y = y + (x * 10) +3
    int z = x - y;  +2
    return z;  +1
}
```

مثال

IUT-ECE

```
int example2(int n) {
    int s = 0;          +1
    int i = 0;          +1
    while (i < n) {         +1
        s = s + (i * 5);    +3
        i = i + 1;      +2
    }
    return s;           +1
}
```

مثال

IUT-ECE

```
int example3(int n) {
    int s = 0;     +1
    int i = 0;     +1
    while (i < n) {    +1
        int j = 0;    +1
        while (j < n) {   +1
            if (j % 2 == 0) {  +2
                // nothing to do
            }
            s = s + (i * 3) + j;  +4
            j = j + 1;   +2
        }
        i = i + 1;   +2
    } return s;      +1
}
```

مثال


```
int example4(int n) {
    int sum = 0;
    for (int i = n; i > 1; i /= 2)
         sum += i;
    return sum;
}
```

مقایسه

- کدام الگوریتم سریع‌تر است؟

- $T_1(n) = 12$
- $T_2(n) = 6n + 3$
- $T_3(n) = 9n^2 + 4n + 6$
- $T_4(n) = 4\log n + 3$

درباره نرخ رشدها

- تغییر سخت‌افزار یا محیط نرم‌افزاری:
  - بر روی تابع زمان اجرا تنها به صورت ضرایب ثابت اثر می‌گذارد؛
  - اما نرخ رشد تابع را تغییر نمی‌دهد.