به نام یگانه معبود بخشنده مهربان

طراحی الگوریتم ها

Design and Analysis of Algorithms

گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه اصفهان

ترم دوم سال تحصیلی ۹۰–۹۱

ارائه دهنده : پیمان ادیبی

اهداف درس

- آشنایی با روشهای عمومی حل کارآمد مسائل توسط کامپیوتر، مستقل از زبان برنامه نویسی و نحوه پیاده سازی
- شناخت چارچوبهای مطالعه شده و نیازهای زمانی و مکانی آنها (زمان اجرا و حافظه مصرفی)
- کسب بینش برای انتخاب روش مناسب از میان روشهای مختلف موجود برای حل یک مسأله

کتاب اصلی درس

- **R. Neapolitan and K. Naimipour, *Foundations of Algorithms Using C++ Pseudocode*, Third Ed., Jones and Bartlett Publisher, 2004.**

- **ریچارد نیپولیتان و کیومرث نعیمی پور، *طراحی الگوریتمها با شبه کدهای C++*، ویراست سوم، م: عین ا... جعفرنژاد قمی، ناشر: علوم رایانه، 1387.**

سایر مراجع

- **S. Bass and A. Gelder, *Computer Algorithms*, Third Ed., Addison-Wesley, 2000.**
- **G. Brassard and P. Bartley, *Fundamentals of Algorithmics*, Prentice-Hall, 1996.**
- **T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd Ed., MIT Press, 2001.**
- **J. Kleinberg and E. Tardos, *Algorithm Design*, Addison-Wesley, 2005.**
- **U. Manber, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, 1989.**

سرفصل مطالب

- کارایی الگوریتمها، تحلیل پیچیدگی، و مرتبه
- روش تقسیم و حل (Divide-and-Conquer)
- روش برنامه ریزی پویا (Dynamic Programming)
- رویکرد حریصانه (Greedy Approach)
- روش عقب گرد (Backtracking)
- روش شاخه و حد (Branch-and-Bound)

ارزیابی

- آزمون میان ترم : %25
- آزمون پایان ترم: %55
- تکالیف: %20

کارایی، تحلیل، و مرتبه الگوریتم ها

Efficiency, Analysis, and Order

تعاریف اولیه

- مسأله (**Problem**): پرسشی که به دنبال پاسخ آن هستیم، و میتواند شامل متغیرهای بدون مقدار باشد، که به آنها *پارامتر* گفته میشود.
- نمونه ای از یک مسأله (**An Instance of a Problem**): با قرار دادن مقادیر مشخص بجای پارامترهای یک مسأله حاصل میشود.
- حل (**Solution**): پاسخی به پرسش مطرح شده در نمونه ای از یک مسأله.
- الگوریتم (**Algorithm**): یک روش گام به گام برای حل تمام نمونه های یک مسأله.

یک مثال: یافتن بزرگترین عدد در یک لیست

- مسأله: در لیست S شامل n عدد، بزرگترین عدد را بیابید.
- نمونه ای از مسأله: مثلا با جایگذاری S=[8,7,6,10,12,1] و n=6.
- حل نمونه مسألهٔ فوق: 12 .
- یک الگوریتم برای این مسأله:
  - مقدار عنصر اول را در متغیری بنام "بیشینه" قرار میدهیم. با شروع از عنصر دوم تا آخر لیست، مقدار هریک از عناصر لیست که بزرگتر از "بیشینه" باشد را در "بیشینه" جایگزین میکنیم. در انتها مقدار متغیر "بیشینه" پاسخ مسأله خواهد بود.

بیان یک الگوریتم

- **بیان توصیفی (descriptional):**
  - مانند آنچه که در مثال قبل انجام گرفت.
  - مشکلات:
    1- دشوار بودن بیان و فهم الگوریتمهای پیچیده.
    2- سرراست نبودن تبدیل به یک برنامهٔ کامپیوتری.

- **بیان الگوریتم با شبه کد (pseudocode):**
  - شکل کلی یک برنامه را دارا است؛ اما برای سادگی از برخی جزئیات صرف نظر میشود.

شبه کدهای C++

- در این شبه کدها نوع داده های زیر را تعریف میکنیم:
  - keytype : برای تعریف عناصر یک لیست مستقل از نوع واقعی عناصر.
  - index : برای تعریف متغیری که بعنوان اندیس بکار میرود.
  - number : برای تعریف متغیری که یک کمیت عددی را در بر دارد.
  - bool : برای تعریف متغیری با مقادیر "درست" یا "نادرست".
- در یک زیربرنامه برای تعریف آرگومانی که خروجی را برمیگرداند، از & استفاده میشود.
- بسیاری از جزئیات بصورت آزاد نوشته میشود:
  - مثلا بجای i>=5 && i<=10 نوشته میشود $5 \le i \le 10$.

شبه کد C++ برای مثال قبل

- مساله: بزرگترین عنصر یک لیست S حاوی n عنصر چیست؟
- ورودی‌ها: عدد مثبت n و آرایه S حاوی کلیدها با اندیس 1 تا n
- خروجی‌ها: متغیر max حاوی بزرگترین کلید داخل آرایهٔ S

```
void find_max (int n, Const keytype S[], keytype& max )
{
    index i;
    max = S[1];
    for( i = 2 ; i <= n ; i ++ )
        if ( S[i] > max )
            max = S[i];
}
```

اهمیت کارآمد بودن الگوریتم ها

- با وجود افزایش سرعت پردازش و کاهش قیمت حافظه، کارآمد بودن یک الگوریتم از لحاظ زمان و حافظه همیشه مهم خواهد بود.
- مثال: مسألهٔ محاسبهٔ جملهٔ n ام از دنبالهٔ فیبوناچی :
  - تعریف دنبالهٔ فیبوناچی:

$$f_n = \begin{cases} n & 0 \le n \le 1 \\ f_{n-1} + f_{n-2} & n > 1 \end{cases}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ….

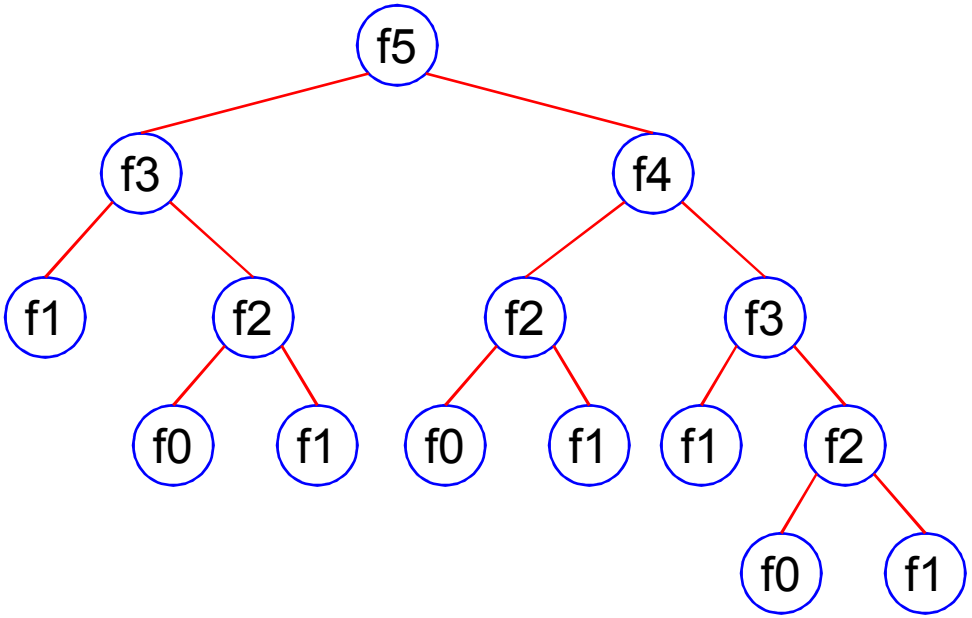
  - دو الگوریتم برای این مثال ارائه میشود:

محاسبهٔ جملهٔ n ام فیبوناچی–الگوریتم بازگشتی (Recursive)

- **مساله**: محاسبهٔ جملهٔ **n** ام از دنبالهٔ فیبوناچی
- **ورودی‌ها**: عدد صحیح نامنفی **n**
- **خروجی‌ها**: **fib1** (جملهٔ **n** ام فیبوناچی)

```
int fib1 (int n)
{
   if (n<=1)
        return  n;
   else
        return fib1(n-1)+fib1(n-2);
}
```

محاسبهٔ جملهٔ n ام فیبوناچی–الگوریتم بازگشتی (Recursive)



- درخت بازگشتی برای محاسبهٔ جملهٔ پنجم فیبوناچی:
- کارایی؟
  - بسیار ناکارآمد است، چون جملات چند بار محاسبه میشوند.
- اگر $T(n)$ تعداد جملات محاسبه شونده برای یافتن جملهٔ n ام باشد، با استقراء نشان داده میشود:

$$T(n) > 2^{n/2}$$

قضیه 1.1 کتاب مطالعه شود.

محاسبهٔ جملهٔ n ام فیبوناچی–الگوریتم تکراری (Iterative)

```
int fib2 (int n)
{
    index i ;
    int f[0..n];
    f[0]=0;
    if (n>0){
        f[1]=1;
        for(i=2; i<=n; i++)
            f[i] = f[i-1] + f[i-2];
    }
    return f[n];
}
```

- مساله: محاسبهٔ جملهٔ n ام از دنبالهٔ فیبوناچی
- ورودی ها: عدد صحیح نامنفی n
- خروجی ها: fib2 (جملهٔ n ام فیبوناچی)
- کارایی؟

  در اینجا $T(n) = n+1$
- تمرین: برای صرفه جویی در حافظه مصرفی، این الگوریتم را بدون استفاده از آرایه بنویسید.

مقایسه الگوریتم بازگشتی و تکراری برای محاسبهٔ جملهٔ n ام فیبوناچی

- فرض: هر جملهٔ فیبوناچی در 1 ns (10^{-9} s) محاسبه میشود.

n	$2^{n/2}$	n+1	حدپایین زمان اجرای الگوریتم بازگشتی	زمان اجرای الگوریتم تکراری
50	$2^{25}=3.4\times10^{7}$	51	34 ms	51 ns
100	$2^{50}=1.1\times10^{15}$	101	13 روز	101 ns
200	$2^{100}=1.3\times10^{30}$	201	4×10^{13} سال	201 ns

- نتیجه: انتخاب الگوریتم مناسب بسیار مهم است (مستقل از سرعت پردازش سخت افزار).

تحلیل پیچیدگی زمانی الگوریتم ها
(Time Complexity Analysis)

- معیارهایی مورد نیاز است که مستقل از جزئیاتی مانند سخت افزار مورد استفاده، زبان برنامه نویسی، و برنامه نویس باشند.
- معمولاً زمان اجرا متناسب است با:
  - اندازهٔ ورودی
  - تعداد تکرار عملیات اصلی
- اندازهٔ ورودی: تعداد عناصر مورد استفاده برای ذخیره ورودی
  - مثال: تعداد عناصر آرایهٔ ورودی
- عمل اصلی: قاعدهٔ مشخصی برای تعیین آن وجود ندارد:
  - مثال: یک مقایسه، یک انتساب، یک عمل ریاضی بین دو متغیر، ...
  - خیلی مواقع دستوری است که در داخلی ترین حلقه برنامه اجرا میشود.

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: الگوریتم یافتن بیشینهٔ در یک لیست

- اندازهٔ ورودی: اندازهٔ لیست (n)
- عمل اصلی: دستور مقایسه (S[i] > max)

```
void find_max (int n, Const keytype S[], keytype& max )
{
    index i;
    max = S[1];
    for( i = 2 ; i <= n ; i ++ )
        if ( S[i] > max )
            max = S[i];
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: الگوریتم تکراری محاسبهٔ جملهٔ n ام فیبوناچی

```
int fib2 (int n)
{
    index i ;
    int f[0..n];
    f[0]=0;
    if (n>0){
        f[1]=1;
        for(i=2; i<=n; i++)
            f[i] = f[i-1] + f[i-2];
    }
    return f[n];
}
```

- اندازه ورودی: تعداد بیت بکار رفته برای ذخیرهٔ ورودی ← $\lfloor \lg n \rfloor + 1$
- عمل اصلی: دستور انتساب (f[i] = f[i-1] + f[i-2])

تحلیل پیچیدگی زمانی الگوریتم ها

- تعریف تحلیل پیچیدگی زمانی (T. C. A.):

تعیین تعداد دفعات اجرای عمل اصلی بر حسب اندازه ورودی

- انواع تحلیل پیچیدگی زمانی:
  - در هر حالت (Every-case T. C. A.) ← محاسبهٔ T(n)
  - در بدترین حالت (Worst-case T. C. A.) ← محاسبهٔ W(n)
  - در حالت متوسط (Average-case T. C. A.) ← محاسبهٔ A(n)
  - در بهترین حالت (Best-case T. C. A.) ← محاسبهٔ B(n)

که در آن:

- T(n) تعداد تکرار عمل اصلی
- W(n) بیشترین تعداد تکرار عمل اصلی
- A(n) میانگین تعداد تکرار عمل اصلی
- B(n) کمترین تعداد تکرار عمل اصلی

برای حالتی که تعداد تکرار عمل اصلی به *مقادیر ورودیها* بستگی دارد.

برای حالتی که تعداد تکرار عمل اصلی به *مقادیر ورودیها* بستگی ندارد.

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: مرتب سازی تعویضی (Exchange Sort)

- مساله: مرتب کردن n کلید به ترتیب غیر نزولی
- ورودی ها: عدد صحیح مثبت n، آرایهٔ S حاوی کلیدها با اندیسهای 1 تا n
- خروجی ها: آرایهٔ S حاوی کلیدها به ترتیب غیر نزولی

```
void exchangesort (int n, keytype S[]) {
   index i, j ;
   for(i=1; i<= n-1; i++)
        for(j=i+1; j<=n; j++)
             if (S[j]<S[i])
                exchange S[i] and S[j];
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: مرتب سازی تعویضی (Exchange Sort)

- عمل اصلی: مقایسه S[i]<S[j]
- اندازه ورودی: n
- تحلیل در هر حالت ؟ وجود دارد چون تعداد تکرار عمل اصلی وابسته به مقادیر ورودیها نیست

```
void exchangesort (int n, keytype S[]) {
   index i, j ;
   for(i=1; i<= n-1; i++)
        for(j=i+1; j<=n; j++)
             if (S[j]<S[i])
                exchange S[i] and S[j];
}
```

...

$T(n)=n(n-1)/2$

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: مرتب سازی تعویضی (Exchange Sort)

$$T(n)=\sum_{i=1}^{n-1}(n-i)=n(n-1)-\sum_{i=1}^{n-1}i=n(n-1)-\frac{n(n-1)}{2}=\frac{n(n-1)}{2}$$

```
void exchangesort (int n, keytype S[]) {
   index i, j ;
   for(i=1; i<= n-1; i++)
        for(j=i+1; j<=n; j++)
             if (S[j]<S[i])
                exchange S[i] and S[j];
}
```

از پیوست اول :

$$\sum_{i=1}^{n} i=\frac{n(n+1)}{2}$$

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی ترتیبی (Sequential Search)

- مساله: آیا کلید x در آرایهٔ S شامل n کلید وجود دارد؟
- ورودی ها: عدد صحیح مثبت n، آرایهٔ S حاوی کلیدها با اندیس 1 تا n، کلید x
- خروجی ها: اندیس کلید x در آرایه (در صورت وجود) یا 0 (در صورت عدم وجود)

```
index seqsearch (int n, const keytype S[], keytype x) {
    index ind=1;
    while(ind<=n && S[ind]!=x)
        ind++;
    if (ind>n)
        ind=0;
    return ind;
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی ترتیبی (Sequential Search)

- عمل اصلی: مقایسه S[ind]!=x در شرط while
- اندازه ورودی: n
- تحلیل در هر حالت ؟ وجود ندارد چون تعداد تکرار عمل اصلی وابسته به مقادیر ورودیها (x و عناصر S) است.

```
index seqsearch (int n, const keytype S[], keytype x) {
    index ind=1;
    while(ind<=n && S[ind]!=x)
        ind++;
    if (ind>n)
        ind=0;
    return ind;
}
```

- تحلیل در بدترین حالت ← W(n)
- تحلیل در بهترین حالت ← B(n)
- تحلیل در حالت متوسط ← A(n)

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی ترتیبی (Sequential Search)

- بدترین حالت: بیشترین تعداد تکرار مربوط به زمانی است که x در S نباشد ← $W(n) = n$

```
index seqsearch (int n, const keytype S[], keytype x) {
    index ind=1;
    while(ind<=n && S[ind]!=x)
        ind++;
    if (ind>n)
        ind=0;
    return ind;
}
```

- بهترین حالت: کمترین تعداد تکرار مربوط به زمانی است که x اولین عنصر S باشد ← $B(n) = 1$
- حالت متوسط: میانگین تعداد دفعات تکرار ← $A(n)=?$

تحلیل حالت متوسط

- ابتدا به هر وضعیت ممکن بر اساس اطلاعات موجود، یک احتمال وقوع نسبت میدهیم ← p_i ها $(i=1,\ldots,m)$
- سپس تعداد تکرار عمل اصلی را در هر وضعیت تعیین میکنیم ← t_i ها $(i=1,\ldots,m)$
- آنگاه تحلیل حالت متوسط را بصورت زیر خواهیم داشت:

$$A(n) = \sum_{i=1}^{m} p_i t_i$$

که m تعداد وضعیتهای مختلف است.

تحلیل حالت متوسط جستجوی ترتیبی

- تعداد وضعیتهای گوناگون ← m=n+1 (در صورت وجود x در S تعداد n وضعیت و در صورت عدم وجود هم یک وضعیت داریم)
- فرض: با احتمال p کلید x در آرایهٔ S موجود است و با احتمال 1-p موجود نیست.
- در صورت وجود x در S، احتمال بودن آن در هر خانه از S برابر است (چون اطلاعات خاصی در این زمینه نداریم):

$$p_1 = p_2 = \cdots = p_n = \frac{p}{n}, \qquad p_{n+1} = 1 - p$$

$$t_k = k \quad (k = 1, \cdots, n), \quad t_{n+1} = n$$

تحلیل حالت متوسط جستجوی ترتیبی

$$A(n)=\sum_{k=1}^{n+1} p_k t_k = \sum_{k=1}^{n} \frac{p}{n}\times k+(1-p)n =$$

$$\frac{p}{n}\times\frac{n(n+1)}{2}+n(1-p)=n(1-\frac{p}{2})+\frac{p}{2}$$

- هنگامی که میدانیم x قطعاً در S موجود است ← **p=1** :

(تقریباً نیمی از آرایه بطور متوسط جستجو میشود)

$$A(n)=(n+1)/2$$

- هنگامی که اطلاعی در مورد وجود یا عدم وجود x در S موجود نیست ← **p=0.5** :

(تقریباً 3/4 از آرایه بطور متوسط جستجو میشود)

$$A(n)=\frac{3n}{4}+\frac{1}{4}$$

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

- مساله: آیا کلید x در آرایهٔ S شامل n کلید وجود دارد؟
- ورودی ها: عدد صحیح مثبت n، آرایهٔ مرتب شده (به ترتیب غیر نزولی) S حاوی کلیدها با اندیس 1 تا n، کلید x
- خروجی ها: اندیس کلید x در آرایه (در صورت وجود) یا 0 (در صورت عدم وجود)

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

- اجرای دستی (trace):

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low + high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

low high

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low + high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

low mid high

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

low high

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

low mid high

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

low high

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

low high

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

mid

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

low high

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

low high

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

mid

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

high low

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

high low

S=[5,7,11,12,15,16,18,19,22,25,28,31,32,35,39,41], n=16, x=38

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

ind=0 → not found

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);      // mid = ⌊(low+high)/2⌋;
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی دودویی (Binary Search)

- عمل اصلی: مقایسه x با S[mid] (در زبان اسمبلی دو مقایسه با یک دستور انجام میشوند)
- اندازه ورودی: n
- تحلیل در هر حالت ؟ وجود ندارد چون تعداد تکرار عمل اصلی وابسته به مقادیر ورودیها (x و عناصر S) است.

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

- تحلیل در بدترین حالت ← W(n)
- تحلیل در بهترین حالت ← B(n)
- تحلیل در حالت متوسط ← A(n)

(با فرض اینکه n توان صحیحی از 2 باشد)

تحلیل پیچیدگی زمانی الگوریتم ها – مثال: جستجوی ترتیبی (Sequential Search)

- بدترین حالت: بیشترین تعداد تکرار مربوط به زمانی است که

  x بزرگتر از عنصر آخر باشد ← $W(n) = 1+\lg(n)$

- بهترین حالت: کمترین تعداد تکرار مربوط به زمانی است که

  x عنصر میانی S باشد ← $B(n) = 1$

```
void binsearch (int n, const keytype S[], keytype x, index& ind) {
    index low, high, mid;
    low=1; high=n; ind=0;
    while(low<=high && ind==0){
        mid = floor((low+high)/2);
        if(x==S[mid])
            ind = mid;
        else if(x<S[mid])
            high=mid-1;
        else
            low=mid+1;
    }
}
```

- حالت متوسط: میانگین تعداد دفعات تکرار

  ← $A(n)=?$

تحلیل حالت متوسط جستجوی دودویی

- انواع وضعیتهای ممکن:

(الف) $x \in S$ ، احتمال ← p_1 ، (احتمال بودن در هر خانه: $\frac{p_1}{n}$)

(ب) $x < S[1], x \notin S$ ، احتمال ← p_2

(ج) $S[1] < x < S[n], x \notin S$ ، احتمال ← p_3 ، (احتمال هر بازه: $\frac{p_3}{n-1}$)

(د) $x > S[n], x \notin S$ ، احتمال ← p_4

- تعداد وضعیتهای فوق: $\mathbf{m = n + 1 + (n-1) + 1}$

(الف) (ب) (ج) (د)

$$A(n) = \sum_{k=1}^{n} \frac{p_1}{n} t_k + p_2 \lg n + \sum_{k=n+2}^{2n} \frac{p_3}{n-1} t_k + p_4 (\lg n + 1)$$

(الف) (ب) (ج) (د)

تحلیل حالت متوسط جستجوی دودویی

(الف) : $$\frac{p_1}{n}\sum_{k=1}^{n} t_k = \frac{p_1}{n}\left(1\times(\lg n+1)+\sum_{j=1}^{\lg n} 2^{j-1}\times j\right)$$

(ج) : $$\frac{p_3}{n-1}\sum_{k=n+2}^{2n} t_k = \frac{p_3}{n-1}\left((n-2)\times\lg n+1\times(\lg n+1)\right)$$

با فرض

$p_1 = p_2 = p_3 = p_4 = 1/4$

از پیوست اول (مثال A.5) :

$$\sum_{i=1}^{n} 2^i \times i = (n-1)2^{n+1}+2$$

$$A(n) = \cdots \longrightarrow A(n) \in \theta(\lg n)$$

نکاتی درباره تحلیل پیچیدگی زمانی حالت متوسط

- تحلیل حالت متوسط زمانی ارزش دارد که انحراف معیار رخدادهای واقعی از میانگین، کوچک باشد. بعبارت دیگر حالت میانگین (average) با حالت متداول (typical) مشابه باشند.
  - مثال: میزان درآمد در یک محله متشکل از دو گروه فقیر و غنی

توزیع دو قله ای ← میانگین بی ارزش ×

  - مثال: دمای هوای یک روز از سال برای یک شهر در صد سال گذشته

توزیع تک قله ای ← میانگین ارزشمند √

  - مثال: مسأله ای با توزیع یکنواخت

توزیع یکنواخت ← میانگین بی ارزش ×

نکاتی درباره تحلیل پیچیدگی زمانی حالت متوسط

- در کاربردهایی که زمان پاسخ از اهمیت بالایی برخوردار است تنها در صورتی میتوان به تحلیل حالت متوسط تکیه کرد که انحراف معیار توزیع باندازه کافی کوچک باشد.
  - مثال: الگوریتم کنترل زمانبندی در یک نیروگاه هسته ای:

    زمان اجرای متوسط = 3 sec

    زمان اجرا در اکثر مواقع = 1 sec

    زمان اجرا در برخی مواقع = 60 sec ×
- در کاربردهایی که الگوریتم به تعداد دفعات زیاد با ورودیهای گوناگون اجرا میشود، تحلیل حالت متوسط ارزشمند است.
  - مثال: یک الگوریتم مرتب سازی عمومی √
- تمام مباحث فوق برای تحلیل پیچیدگی حافظه نیز مطرح است.

تحلیل پیچیدگی زمانی الگوریتم ها – تعاریف

- تابع پیچیدگی: تابعی از اعداد صحیح مثبت به اعداد حقیقی نامنفی.
- انواع دستورات در یک برنامه:
  - دستورات اصلی: بار اصلی محاسبات را دارا هستند — زمان اجرا با اندازه ورودی رشد می کند
  - دستورات سربار: مانند مقداردهی اولیه پیش از حلقه — زمان اجرا با اندازه ورودی رشد نمی کند
  - دستورات کنترلی: مانند افزایش اندیس در حلقه — زمان اجرا با اندازه ورودی رشد می کند
- در تحلیل پیچیدگی زمانی معمولاً از زمان اجرای دستورات سربار و کنترلی صرف نظر میشود، اما گاهی زمان این دستورات قابل توجه بوده و باید لحاظ شود.
  - مثال: دو الگوریتم برای یک منظور با پیچیدگیهای n و n^2 موجود است، اما بواسطه دستورات کنترلی زمان اجرای هر عمل اصلی در اولی هزار برابر دومی است. در چه صورت اولی بهتر از دومی است؟

$$n^2 \times t > n \times 1000t \rightarrow n > 1000$$

مرتبه الگوریتم (Order)

- در تحلیل نظری الگوریتمها معمولا *رفتار نهایی* مورد توجه است .
- رفتار نهایی (eventual behavior): پیچیدگی زمانی هنگامی که اندازه ورودی به میزان کافی بزرگ میشود.
- در توابع پیچیدگی به فرم چندجمله ای، جملهٔ با بالاترین درجه در رفتار نهایی بر سایر جملات غالب میشود.
  - مثال: $f(n)=0.1n^2+n+100$

n	10	20	100	1000
$0.1n^2$	10	40	1000	100000
f(n)	120	160	1200	101100

- گروه های پیچیدگی: شامل توابع پیچیدگی با رفتار نهایی مشابه

مرتبه الگوریتم (Order)

- برخی از گروههای پیچیدگی متداول:

$$\theta(\lg n) \quad \theta(n) \quad \theta(n \lg n) \quad \theta(n^2) \quad \theta(n^3) \quad \theta(2^n)$$

- اگر توابع پیچیدگی **f(n)** و **g(n)** متعلق به دو گروه متفاوت باشند، و گروه حاوی **f(n)** در لیست فوق در سمت چپ گروه حاوی **g(n)** قرار داشته باشد، دراینصورت نمودار **f(n)** در نهایت زیر نمودار **g(n)** قرار میگیرد.
  - مثال: $f(n) \in \theta(n) \quad g(n) \in \theta(n \lg n)$
- شکل **1.3** و جدول **1.4** کتاب مشاهده شود.
- برای مطالعه دقیقتر رفتار نهایی، نمادهای مجانبی (**asymptotic notations**) را تعریف میکنیم: $O \quad \Omega \quad \theta \quad o \quad \omega$

نماد مجانبی θ

- تعریف θ : اگر f(n) و g(n) توابع پیچیدگی باشند:

$$\theta(g(n))=\{f(n)\,|\,\exists \underbrace{n_0, C_1, C_2}_{\text{ثابت، مثبت، متناهی}} > 0,\ \ 0 \le C_1 g(n) \le f(n) \le C_2 g(n),\ \ \forall n \ge n_0\}$$

- در اینصورت مینویسیم: $f(n) \in \theta(g(n))$ یا $f(n) = \theta(g(n))$

- مثال: آیا $4n^2 - 11n \in \theta(n^2)$ ؟ بله
- مثال: آیا $5n^2 + 10n \in \theta(n^2)$ ؟ بله
- مثال: آیا $5n^2 + 10n \in \theta(n)$ ؟ خیر
- مثال: آیا $4n^2 - 11n \in \theta(n)$ ؟ خیر

نماد مجانبی O

- تعریف O : اگر f(n) و g(n) توابع پیچیدگی باشند:

$$O(g(n))=\{f(n)\,|\,\exists \underbrace{n_0, C}_{\text{ثابت، مثبت، متناهی}}>0,\ \ 0\le f(n)\le Cg(n),\ \ \forall n\ge n_0\}$$

- در اینصورت مینویسیم: $f(n)\in O(g(n))$ یا $f(n)=O(g(n))$
- یعنی f(n) در حالت حدی بدتر از g(n) نیست.
- می‌گوییم O یک مرز بالایی مجانبی (asymptotic upper bound) روی تابع قرار میدهد.

- مثال: آیا $4n^2-44n\in O(n)$ ؟ خیر
- مثال: آیا $4n^2-44n\in O(n^3)$ ؟ بله

نماد مجانبی Ω

- تعریف Ω : اگر f(n) و g(n) توابع پیچیدگی باشند:

$$\Omega(g(n))=\{f(n)\,|\,\exists \underbrace{n_0, C}_{\text{ثابت، مثبت، متناهی}}>0,\ \ 0\le Cg(n)\le f(n),\ \ \forall n\ge n_0\}$$

- در اینصورت مینویسیم: $f(n)\in\Omega(g(n))$ یا $f(n)=\Omega(g(n))$
- یعنی f(n) در حالت حدی بهتر از g(n) نیست.
- می‌گوییم Ω یک مرز پایینی مجانبی (asymptotic lower bound) روی تابع قرار میدهد.

- مثال: آیا $n^3-n\in\Omega(n^4)$ ؟ خیر
- مثال: آیا $n^2+n\in\Omega(n)$ ؟ بله

نماد مجانبی o و ω

- تعریف o : اگر f(n) و g(n) توابع پیچیدگی باشند:

$$o(g(n))=\{f(n)\,|\,\forall C>0,\ \exists n_0>0,\ 0\le f(n)\le Cg(n),\ \ \forall n\ge n_0\}$$

  - در اینصورت مینویسیم: $f(n)\in o(g(n))$ یا $f(n)=o(g(n))$
  - یعنی f(n) در حالت حدی قطعاً بهتر از g(n) است.
  - در واقع o یک مرز *اکیداً* بالایی مجانبی روی تابع قرار میدهد.
- مثال: آیا $n^4/3\in o(n^4)$ ؟ خیر
- مثال: آیا $n^2/5\in o(n^3)$ ؟ بله
- تعریف ω : اگر f(n) و g(n) توابع پیچیدگی باشند:

$$\omega(g(n))=\{f(n)\,|\,\forall C>0,\ \exists n_0>0,\ 0\le Cg(n)\le f(n),\ \ \forall n\ge n_0\}$$

  - در اینصورت مینویسیم: $f(n)\in \omega(g(n))$ یا $f(n)=\omega(g(n))$
  - یعنی f(n) در حالت حدی قطعاً بدتر از g(n) است.

نمادهای مجانبی

- اگر $f(n)=a_k n^k+a_{k-1}n^{k-1}+\ldots+a_1 n+a_0$ باشد، بافرض $a_k>0$ داریم : $f(n)\in\theta(n^k),\quad f(n)\in O(n^k),\quad f(n)\in\Omega(n^k)$
- داریم : $\theta(g(n))=O(g(n))\bigcap\Omega(g(n))$
- مفاهیم نمادهای مجانبی:
  - $f(n)\in O(g(n))$: در حالت حدی f(n) بهتر از یا مشابه با g(n) است.
  - $f(n)\in\Omega(g(n))$: در حالت حدی f(n) بدتر از یا مشابه با g(n) است.
  - $f(n)\in\theta(g(n))$: در حالت حدی f(n) مشابه با g(n) است.
  - $f(n)\in o(g(n))$: در حالت حدی f(n) اکیداً بهتر از g(n) است.
  - $f(n)\in\omega(g(n))$: در حالت حدی f(n) اکیداً بدتر از g(n) است.

ویژگیهای مرتبه

- داریم :

	خاصیت بازتابشی reflexive	خاصیت تقارنی symmetric	خاصیت انتقالی transitive
θ	√	√	√
O	√	×	√
Ω	√	×	√
o	×	×	√
ω	×	×	√

- داریم :
  - اثبات: ...

$$f(n) \in o(g(n)) \Rightarrow f(n) \notin \theta(g(n))$$

$$f(n) \in \theta(g(n)) \Rightarrow f(n) \notin o(g(n))$$

ویژگیهای مرتبه

- اگر g(n) یک تابع پیچیدگی باشد، داریم :

$$o(g(n)) \subseteq O(g(n)) - \Omega(g(n))$$

  - اثبات: قضیه 1.2 کتاب یا با استفاده از خواص قبلی ...

- داریم:

$$f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$$

$$f(n) \in o(g(n)) \Leftrightarrow g(n) \in \omega(f(n))$$

$$f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$$

$$f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$$

$$\log_a n \in \theta(\log_b n), \quad a, b > 1$$

ویژگیهای مرتبه

- اگر $f(n) \in O(g(n))$ ، داریم :

$$f(n) + g(n) \in O(g(n))$$
$$f(n) + g(n) \in \Omega(g(n))$$
$$f(n) \times g(n) \in O(g(n)^2)$$
$$f(n) \times g(n) \in \Omega(f(n)^2)$$

- اگر $g(n) \in O(f(n))$ و $h(n) \in \theta(f(n))$ و $c \geq 0$ و $d > 0$:

$$cg(n) + dh(n) \in \theta(f(n))$$

ویژگیهای مرتبه

- گروه های پیچیدگی با ترتیب زیر را در نظر بگیرید:

$$\theta(\lg n), \theta(n), \theta(n \lg n), \theta(n^j), \theta(n^k), \theta(a^n), \theta(b^n), \theta(n!)$$

با فرض $k > j \geq 2$ و $b > a > 1$ ، اگر توابع پیچیدگی **f(n)** و **g(n)** متعلق به دو گروه متفاوت باشند، و گروه حاوی **f(n)** در لیست فوق در سمت چپ گروه حاوی **g(n)** قرار داشته باشد، دراینصورت:

$$f(n) \in o(g(n))$$

- استفاده از حد:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \begin{cases} k & f(n) \in \theta(g(n)) \\ 0 & f(n) \in o(g(n)) \\ \infty & f(n) \in \omega(g(n)) \end{cases}$$

(***k*** ثابت مثبت)

ویژگیهای مرتبه

- مثال: آیا $\lg n \in o(\sqrt{n})$ ؟ ← $f(n)=\sqrt{n}$ و $g(n)=\lg n$:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{\sqrt{n}}{\lg n}=\lim_{n\to\infty}\frac{1/(2\sqrt{n})}{1/n}=\lim_{n\to\infty}\frac{n}{2\sqrt{n}}=\infty \quad \surd$$

- مثال: برای $a>0$ نشان دهید: $a^n=o(n!)$:

$$\lim_{n\to\infty}\frac{a^n}{n!}\overset{?}{=}0$$

  - برای $a\le 1$ واضح است.
  - با فرض $a>1$ داریم:

$$\lim_{n\to\infty}\frac{a^n}{n!}<\lim_{n\to\infty}\frac{a^n}{(n/e)^n}<\lim_{n\to\infty}\frac{a^n}{a^{2n}}=\lim_{n\to\infty}\left(\frac{1}{a}\right)^n=0 \quad \surd$$

$$\boxed{n!>(n/e)^n} \leftarrow \boxed{n!\approx\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}$$

تقریب استرلینگ:

order of a^2 is lower than (n/e)^n