ساختمان داده

گردآورنده: عالیه همتی

آدرس وب سایت:

www.ahemmati.com

آدرس پست الکترونیکی:

Aalia.hemmati92@gmail.com

مهر ۱۳۹۳

منابع پیشنهادی:

- ساختمان داده ها و الگوریتم ها– مهندس جعفر تنها و مهندس سید ناصر آیت
- ساختمان داده ها – حمیدرضا مقسمي
- ساختمان داده ها- نیک محمد بلوچ زهی
- ساختمان داده ها درC - جعفرنژاد قمی
- ساختمان داده ها به زبان C++ - الیس هورویتز – سارتج ساهنی – دینش مهتا
- اصول ساختمان داده ها- سیمورلیپ شوتز

ریز نمره درس ساختمان داده ها

- ❖۱۰ نمره پایان ترم
- ❖۸ نمره میان ترم
- ❖۲ نمره پروژه(کارعملی- حل تمرین)

رئوس مطالب:

۱. روش های تحلیل الگوریتم
۲. آرایه ها و رشته
۳. پشته و صف
۴. لیست های پیوندی
۵. درخت ها
۶. گراف ها
۷. مرتب سازی
۸. درهم سازی

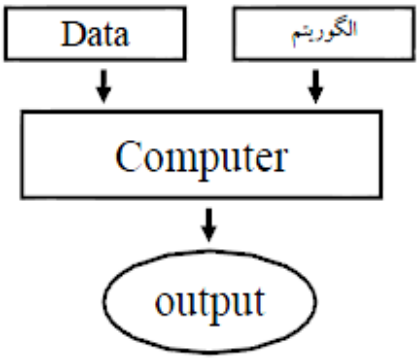
فصل اول

روش‌های تحلیل الگوریتم

مراحل حل مساله

- مراحل حل مسأله:

۱. ایجاد یک مدل انتزاعی از مسأله -- مدل
۲. یافتن الگوریتم برای حل مدل -- الگوریتم
۳. تحلیل الگوریتم برای حل کارآ – الگوریتم صحیح و کارآ
۴. تعریف ساختمان داده ها
۵. پیاده سازی – برنامه قابل اجرا روی ماشین
۶. ...



برای انجام هر عملی در کامپیوتر نیاز به دو عنصر مهم داریم:

- ✓ الگوریتم: که باید مناسب و کارا باشد.
- ✓ ساختمان داده مناسب: تا داده ها به درستی در آن سازماندهی شوند.

ساختمان داده:

- ✓ به مدل ریاضی سازماندهی داده ها ، ساختمان داده گفته میشود.
- ✓ به ساختارهایی که جهت ذخیره سازی ، بازیابی و ... اطلاعات بکار می روند ساختمان داده گفته میشود.

انواع ساختمان داده:

- در درس ساختمان داده ها مدلهای منطقی و ریاضی سازماندهی داده ها به شکلهای مختلف را بررسی می کنیم.
- **ساختمان داده ها به طور کلی به سه دسته تقسیم می شوند:**

۱)ایستا(استاتیک یا ثابت) **مانند** آرایه

۲)نیمه ایستا **مانند** پشته(STAK) و صف(QUEUE)

۳)پویا **مانند** لیست پیوندی ،درخت و گراف

تعریف الگوریتم:

الگوریتم مجموعه ای از دستورالعمل ها است که اگر دنبال شوند، موجب انجام کار خاصی می گردد.

ویژگی های الگوریتم:

- **ورودی**
- **خروجی: الگوریتم بایستی حداقل یک کمیت به عنوان خروجی داشته باشد.**
- **قطعیت: هر دستورالعمل باید واضح و بدون ابهام باشد.**
- **محدودیت: اگر دستورالعمل های یک الگوریتم را دنبال کنیم ،برای تمام حالات ، الگوریتم باید پس از طی مراحل محدودی خاتمه یابد.**
- **کارایی: هر دستورالعمل باید به گونه ای باشد که با استفاده از قلم و کاغذ بتوان آن را با دست نیز اجرا نمود به عبارتی هر دستورالعمل باید انجام پذیر باشد.**

به نظر شما چگونه می توان فهمید یک برنامه(الگوریتم) نوشته شده از برنامه (الگوریتم)مشابه دیگر بهتر عمل می کند ؟

- **بررسی کارایی الگوریتم:**

دو فاکتور مهمی که باید مورد توجه قرار گیرد:

۱- حافظه مصرفی

۲- زمان مصرفی الگوریتم

الگوریتمی بهتر است که فضا و زمان کمتری بخواهد.

- **پیچیدگی فضا و زمانی یک برنامه:**

میزان حافظه یا پیچیدگی فضای یک برنامه: مقدارحافظه مورد نیازبرای اجرای کامل یک برنامه است.
میزان یا پیچیدگی زمان یک برنامه: مقدار زمانی که کامپیوتربرای اجرای کامل برنامه لازم دارد.

- **زمان اجرای الگوریتم- عوامل دخیل در زمان اجرا:**

۱- سرعت سخت افزار

۲- نوع کامپایلر

۳- اندازه داده ورودی

۴- ترکیب داده های ورودی

۵- پیچیدگی زمانی الگوریتم

۶- پارامترهای دیگر که تاثیر ثابت در زمان اجرا دارند.

از این عوامل، سرعت سخت‌افزار و نوع کامپایلر به صورت ثابت در زمان اجرای برنامه‌ها دخیل هستند. پارامتر مهم، پیچیدگی زمانی الگوریتم است که خود تابعی از اندازه مسئله می‌باشد. ترکیب داده‌های ورودی نیز با بررسی الگوریتم در شرایط مختلف قابل اندازه‌گیری می‌باشد (در متوسط و بدترین حالات).

ادامه-بررسی زمان اجرای الگوریتم

۱- محاسبه کارایی بر حسب زمان

۲- مستقل از کامپیوتر ، زبان برنامه نویسی ،برنامه نویس و تمامی جزئیات الگوریتم
۳- محاسبه تعداد دفعات اجرای عملیات اصلی (مقایسه، جمع ، ضرب ..) بر حسب اندازه ورودی

تعیین عمل اصلی در یک الگوریتم:

به دستور یا دستوراتی که کل کار انجام شده توسط الگوریتم، تقریباً متناسب با تعداد دفعاتی باشد که توسط این دستور یا دستورات انجام می شود، عمل اصلی الگوریتم گویند.

مرتبه زمانی:

در اغلب مسائل تابعی از اندازه مساله می باشدو عبارت است از تعداد دفعاتی که عمل اصلی به ازای هر مقدار از اندازهٔ ورودی انجام می شود.

- برای تحلیل زمان اجرای یک الگوریتم، تابعی به نام $T(n)$ در نظر می گیریم که در آن n اندازه ورودی می باشد.

- اندازه ورودی:

در بسیاری از الگوریتم ها یافتن میزان منطقی از اندازه ورودی آسان است . مثال:

- ✓ الگوریتم جستجوی ترتیبی
- ✓ الگوریتم محاسبه مجموع عناصرآرایه اندازه ورودی: n ، تعداد عناصر آرایه
- ✓ الگوریتم مرتب سازی تعویضی
- ✓ الگوریتم جستجوی دودویی
- ✓ الگوریتم ضرب ماتریس ها اندازه ورودی : nتعداد سطرها و ستون ها
- ✓ در برخی از الگوریتم ها بهتر است اندازه ورودی را بر حسب دو عدد بسنجیم:

اگر ورودی الگوریتم یک گراف باشد، اندازه ورودی را بر حسب تعداد رئوس(n) و تعداد یال ها (m) می سنجیم.

- تابع زمانی$T(n)$:

برای محاسبه تابع زمانی $T(n)$ برای یک الگوریتم موارد زیر را باید در محاسبات در نظر بگیریم:

- **زمان مربوط به اعمال جایگزینی که مقدار ثابت می‌باشند**
- **زمان مربوط به انجام اعمال محاسبات که مقدار ثابتی دارند.**
- **زمان مربوط به تکرار تعدادی دستور یا دستورالعمل (حلقه‌ها)**

• زمان مربوط به توابع بازگشتی

از موارد ذکر شده در محاسبه زمان $T(n)$ یک الگوریتم محاسبه تعداد تکرار عملیات و توابع بازگشتی، اهمیت ویژه‌ای دارند. و در حقیقت در کل پیچیدگی زمانی مربوط به این دو می‌باشد.

- مثال۱: بررسی محاسبه اجرای الگوریتم ها

قطعه برنامه زیر را در نظر بگیرید:

```
(1)    x = 0 ;
(2)    for ( i = 0 ;  i < n ; i++ )
(3)        x++ ;
```

در قطعه کد بالا عملیات متفاوتی از جمله جایگزینی، مقایسه و غیره انجام می‌گیرد که هر کدام زمانهای متفاوتی را برای اجراشدن نیاز دارند. تابع زمانی قطعه کد بالا را می‌توان بصورت زیر محاسبه کرد:

سطر	زمان	تعداد
1	$C_۱$	۱
2	$C_۲$	$n+۱$
3	$C_۳$	n

با توجه به جدول، $T(n)$ برابر است با:

$$T(n) = C_۱ + C_۲(n+۱) + C_۳n$$

مثال ۲: بررسی محاسبه اجرای الگوریتم ها-محاسبه !n

تابع زیر مربوط به محاسبه فاکتوریل عدد n را در نظر بگیرید:

```
(1)     int    Factorial( int  n )
        {
(2)             int  fact= 1 ;
(3)             for( int  i=2  ; i<= n ; i ++ )
(4)                   fact*= i ;
(5)             return fact ;
        }
```

تابع زمانی، تابع بالا بصورت زیر محاسبه می‌شود:

سطر	هزینه	تعداد
2	$C_۱$	۱
3	$C_۲$	n
4	$C_۳$	n − ۱
5	$C_٤$	۱

بنابراین $T(n)$ برابر است با:

$$T(n) = C_۱ + C_۲n + C_۳(n-۱) + C_٤$$

C را بیشترین مقدار $C_۱$، $C_۲$، $C_۳$ و $C_٤$ در نظر می‌گیریم بنابراین خواهیم داشت:

$$T(n) = C(۲n+۱) .$$

مثال۳: محاسبه تابع زمانی الگوریتم جمع دو ماتریس

تابع زیر مربوط به حاصل‌جمع دو ماتریس می‌باشد:

```
(1)     void    Add( a, b, c, int m,n )
        {
(2)             for( int i=0 ; i<n ; i ++ )
(3)                 for( int j=0 ; j<m ; j ++ )

                        c[i,j]= a[i,j] + b[i,j] ;
    }
```

تابع زمانی، الگوریتم بالا بصورت زیر محاسبه می‌شود:

سطر	هزینه	تعداد
1	C_1	$n+1$
2	C_2	$n(m+1)$
3	C_3	nm

بنابراین $T(n)$ برابر است با:

$$T(n) = C_1(n+1) + C_2n(m+1) + C_3nm$$

C را بیشترین مقدار C_1، C_2، C_3 در نظر می‌گیریم بنابراین خواهیم داشت:

$$T(n) = C(n+1+n(m+1)+nm)$$
$$= C(2nm+2n+1)$$

تمرین ۱:

تعداد کل مراحل برنامه زیر را محاسبه کنید.

```
float  sum (float  list [ ] , int  n)
{
float  s=0 ;
 int  i;
for (i=0;  i<n ;  i++)
        s= s + list [i] ;
         return   s;
}
```

آهنگ(نرخ) رشد:

فرض کنید M یک الگوریتم و n تعداد داده های ورودی آن باشد. واضح است که تابع زمانی T(n) الگوریتم M با زیاد شدن تعداد داده های ورودی n افزایش می یابد.

برای اینکه آهنگ افزایش T(n) را تعیین کنیم این تابع زمانی را با چند تابع استاندارد مقایسه می کنیم نظیر:

آهنگ های رشد این توابع استاندارد، در شکل۲ نشان داده شده است که مقادیر تقریبی آنها را، به ازای چند مقدار معین n به دست می دهد.

n \ $g(n)$	$\log n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

شکل ۶ـ ۲ . آهنگ رشد چندتابع استاندارد

مقایسه

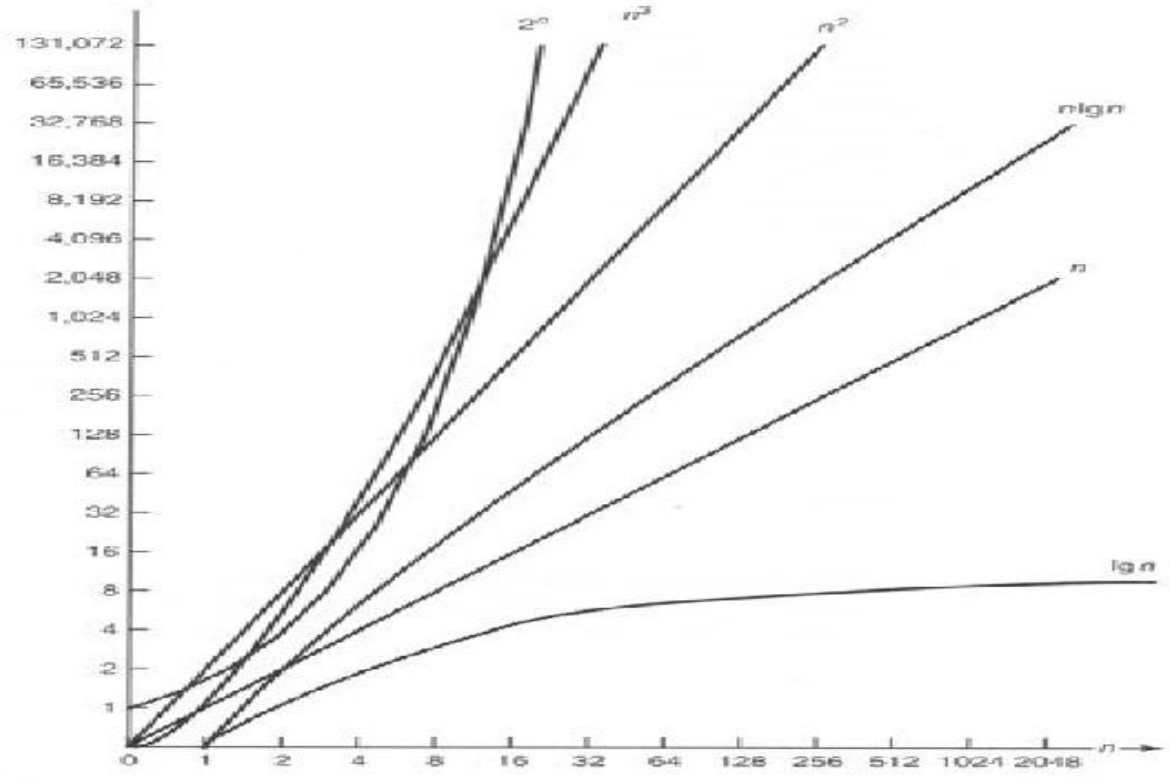


Figure 1.3 ■ Growth rates of some common complexity functions.

ملاحظه می شود که تابع ها به ترتیب آهنگ رشدشان ارائه شده اند. تابع لگاریتمی رشد نسبتا کندی را دارد، تابع نمایی رشد نسبتا تندی را دارد و تابع چند جمله ای با توجه به مقدار توانش رشد می کند. یک راه برای مقایسه تابع T(n) با این تابع های استاندارد، استفاده از نماد تابعی O می باشد که به صورت زیر تعریف می شود:

- مرتبه اجرایی الگوریتم- بررسی میزان رشد توابع زمانی الگوریتم

برای بررسی کارایی الگوریتم‌ها، نمادهایی معرفی شده است کـه در زیـر آنهـا را بررسی می‌کنیم.

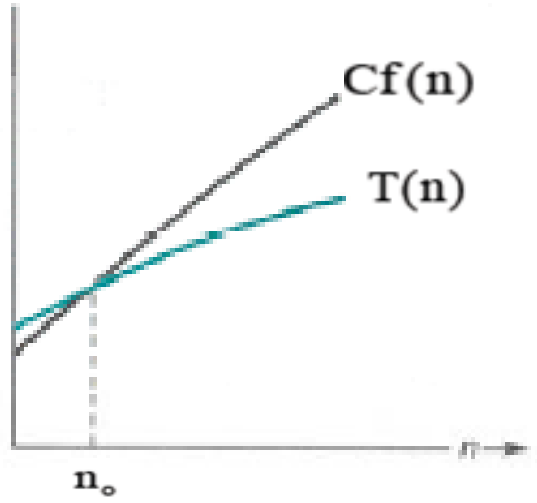
۱۰۲۰۱ نماد Big-oh

برای بررسی میزان رشد توابع زمانی الگوریتم‌ها، نماد Big-oh را بکار مـی‌گیرنـد و آنرا با علامت O نمایش می‌دهند. حال در زیر تعریف این علامت را ارائه می‌دهیم:

تعریف: گوئیم $T(n) \in O(f(n))$ اگر و فقط اگر ثابت C و ثابت صحیح n_0 وجود داشته باشند که برای همه مقادیر $n \geq n_0$ ، داشته باشیم $T(n) \leq Cf(n)$ (رابطه $T(n) \in O(f(n))$ را بخوانید $T(n)$ متعلق به اوی بزرگ $f(n)$).

تعریف بالا به‌صورت زیر نیز بیان می‌شود :

$T(n) \in O(f(n)) \Leftrightarrow \exists C, n_0 > 0$ به‌طوریکه $\forall n \geq n_0 \quad T(n) \leq Cf(n)$



در تعریف بالا $T(n)$ زمان اجرای الگوریتم را مشخص می‌کند و تابعی از اندازه داده‌ها می‌باشد.

در حالت کلی $f(n)$ مرتبه زمانی اجرای الگوریتم نامیده می‌شود (اصطلاحاً پیچیدگی زمانی الگوریتم هم گفته می‌شود) و با $O(f(n))$ نمایش داده می‌شود.

نکته: وقتی $T(n) \in O(F(n))$ هست می‌گوییم $F(n)$ یک کران بالا برای $T(n)$ می‌باشد.

قضیه ۱۰۱: اگر $T(n)=a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ زمان اجرای یک الگوریتم باشد آنگاه $T(n) \in O(n^m)$.

مثال۱۰۳: زمان اجرای $T(n)$ مربوط به تعدادی الگوریتم موجود است مرتبه یا پیچیدگی زمانی این الگوریتم‌ها را محاسبه نمایید:

i) $T_1(n) = 2n^2 + 4n$

ii) $T_2(n) = 3n^3 + 3n$

طبق قضیه ۱۰۱:

در تابع زمانی باید جمله با بیشترین مرتبه را در نظر بگیریم و ضرایب جملات عملا تاثیری در مرتبه زمانی الگوریتم ندارند.

حل:

i) $T_1(n) = 2n^2 + 4n \le 3n^2$

که در آن اگر $C=3$ و $n_0=4$ باشد آنگاه $T_1(n) \in O(n^2)$ می‌باشد.

ii) $T_2(n) = 3n^3 + 3n$

$\le 4n^3$

که در آن اگر $C=4$ و $n_0=2$ باشد، آنگاه $T_2(n) \in O(n^3)$.

- **در جدول زیر مرتبه اجرایی چند تابع به ترتیب صعودی از چپ به راست نوشته شده است:**

نام تابع	ثابت	لگاریتمی	خطی	----	مرتبه۲	توانی	فاکتوریل
مرتبه اجرا	O(1)	O(log n)	O(n)	O(n log n)	$O(n^2)$	$O(2^n)$	O(n!)

تمرین ۲:

درستی یا نادرستی عبارات زیر را ثابت کنید:

i) $T(n) = (2n + 1) \in O(n^2)$

ii) $T(n) = (5n^2 + n + 1) \in O(n)$

iii) $T(n) = (4 * 2^n + n^2) \in O(2^n)$

راه حل:

i) $T(n) = (2n + 1)$

$\leq 2n^2$

که در آن اگر $C = 2$ و $n_0 = 2$ باشد آنگاه $T(n) \in O(n^2)$ خواهد بود. بنابراین رابطه بالا یک رابطه صحیح می‌باشد.

ii) $T(n) = (5n^2 + n + 1)$

$\leq 6n^2$

همانطور که ملاحظه می‌کنید $T(n)$ کمتر از $6n^2$ می‌باشد و به هیچ وجه در حالت کلی نمی‌تواند کمتر از Cn باشد. بنابراین رابطه بالا یک رابطه نادرست می‌باشد.

iii) $T(n) = (4 * 2^n + n^2)$

$\leq 5 * 2^n$

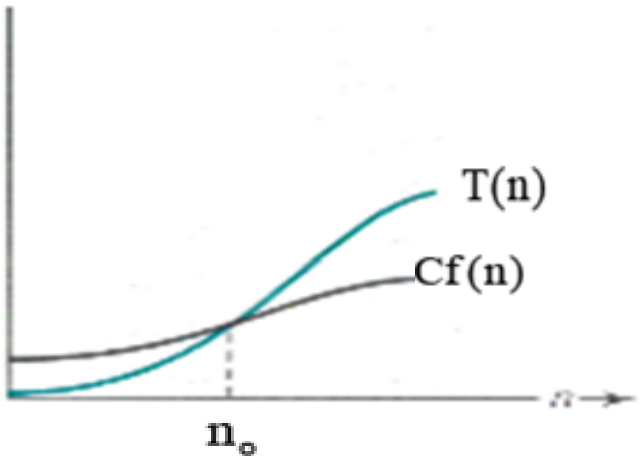
که در آن اگر $C = 5$ و $n_0 = 4$ باشد آنگاه $T(n) \in O(n^2)$ خواهد بود. بنابراین رابطه بالا یک رابطه صحیح می‌باشد.

۱۰۲۰۲ نماد Big-Omega

تعریف: گوئیم $T(n)\in\Omega(f(n))$ اگر و فقط اگـر ثابـت صـحیح C و n_0 وجـود داشـته باشـد کـه بـه ازای همـه مقـادیر $n\geq n_0$ داشـته باشـیم $T(n)\geq Cf(n)$ (رابطـه $T(n)\in\Omega(f(n))$ را بخوانید $T(n)$ امگای بزرگ $f(n)$).

تعریف بالا را بصورت زیر نیز ارائه می‌دهند:

$$T(n)\in\Omega(f(n))\Leftrightarrow\exists C,n_0>0\quad\forall n\geq n_0\qquad Cf(n)\leq T(n)$$



اگر دقت کنید ملاحظه می‌کنید که تعریف بالا یک کران پایین زمـان اجـرا بـرای $T(n)$ ارائه می‌دهد. بنابراین، در حالت کلی می‌توان گفت که $\Omega(f(n))$ بهترین حالـت اجرا برای یک الگوریتم می‌باشد.

برای درک بهتر نماد بالا در زیر چند مثال ارائه می‌دهیم.

قضیه **102** : اگـر $T(n)=a_mn^m+a_{m-1}n^{m-1}+\ldots+a_1n+a_0$ زمـان اجـرای یک الگوریتم بوده و $a_m>0$ باشد آنگاه $T(n)\in\Omega(n^m)$.

مثال۱۰۸:زمان اجرای $T(n)$ الگـوریتمی محاسـبه شـده، $\Omega(f(n))$ آنـرا بدسـت آورید.

$$T(n) = n^{٤} + ٥n^{٢}$$

حل:

$$T(n) = n^{٤} + ٥n^{٢} \geq n^{٤}$$

بنابراین اگر $n_{٠} = ١$ و $C = ١$ باشد، آنگاه $T(n) \in \Omega(n^{٤})$ خواهد بود.

مثال ۱۰۹: زمان اجرای $T(n)$ مربوط به تعدادی الگوریتم موجود است. درسـتی عبارات زیر را ثابت کنید.

i) $T(n) = ٦n + ٤ \in \Omega(n)$

ii) $T(n) = ٣n + ٢ \notin \Omega(n^{٢})$

iii) $T(n) = ٥^{n} + n^{٢} \in \Omega(٢^{n})$

حل:

i) $T(n) = ٦n + ٤ \geq ٦n$

بنابراین اگر $n_{٠} = ١$ و $C = ٦$ باشد، آنگاه $T(n) \in \Omega(n)$ خواهد بود.

ii) $T(n) = ٣n + ٢ \notin \Omega(n^{٢})$

فرض کنید C و $n_{٠}$ موجود است بطوریکه به ازای هر $n \geq n_{٠}$ داشته باشیم:

$$T(n) = ٣n + ٢ \geq Cn^{٢}$$

$$\Rightarrow Cn^{٢} - ٣n - ٢ \leq ٠$$

همانطور که ملاحظه می‌کنید در نامعادله بالا C با مقدار معین وجود ندارد بنابراین $٣n + ٢ \notin \Omega(n^{٢})$ خواهد بود.

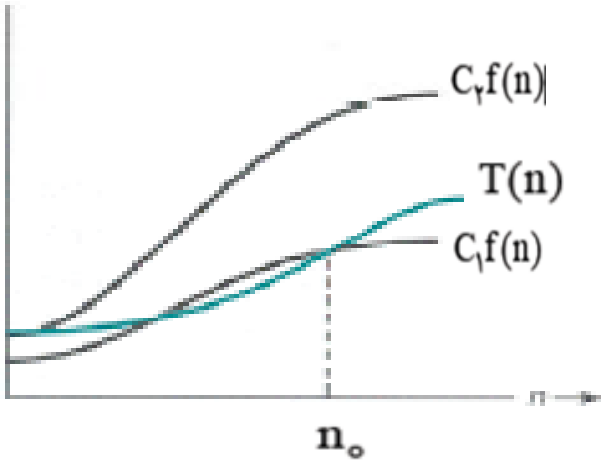
iii) $T(n) = ٥^{n} + n^{٢} \geq ٥^{n} \geq ٢^{n}$

بنابراین اگر $n_{٠} = ١$ و $C = ١$ باشد، آنگاه $T(n) \in \Omega(٢^{n})$ خواهد بود.

نماد θ

تعریف: گوئیم $T(n) \in \theta(f(n))$ اگر و فقط اگر ثابتهای C_1، C_2 و ثابت صحیح n_0 وجود داشته باشد بطوریکه برای همه مقادیر $n \geq n_0$:

$$C_1 f(n) \leq T(n) \leq C_2 f(n)$$



تعریف بالا بصورت زیر نیز ارائه می‌شود:

$$\exists C_1, C_2, n_0 > 0 \text{ بطوریکه } \forall n \geq n_0 \quad C_1 f(n) \leq T(n) \leq C_2 f(n)$$

$$\Leftrightarrow T(n) \in \theta(f(n))$$

توسط نماد بالا تابع $T(n)$ هم از بالا و هم از پایین محدود می‌شود. توجه داشته باشید که، درجه رشد تابع $f(n)$ و $T(n)$ یکسان است.

قضیه ۱۰۳: اگر $T(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ زمان اجرای یک الگوریتم بوده و $a_m > 0$ باشد آنگاه $T(n) \in \theta(n^m)$.

مثال ۱۰۱۱: فرض کنید $T(n)=\forall n^{\mathbf{r}}$ باشد . آیا $T(n)\in\theta(n^{\mathbf{r}})$ می‌تواند باشد.

حل: طبق تعریف θ باید داشته باشیم:

$\exists C_{\mathbf{1}},C_{\mathbf{r}},n_{\circ}>\circ$ بطوریکه $\forall n\geq n_{\circ}\quad C_{\mathbf{1}}n^{\mathbf{r}}\leq \forall n^{\mathbf{r}}\leq C_{\mathbf{r}}n^{\mathbf{r}}$

طرف چپ رابطه بالا همیشه برقرار است اما طرف راست رابطه بالا در صورتی برقرار است که $n\leq C_{\mathbf{r}}/\forall$ باشد و این با تعریف θ که در آن رابطه برای هر n بزرگتر از $n_{\circ}$ برقرار است منافات دارد، لذا $T(n)$ نمی‌تواند متعلق به $\theta(n^{\mathbf{r}})$ باشد.

درستی عبارات زیر را ثابت کنید.

i) $T(n)=\mathbf{\mathfrak{F}}n+\mathbf{\mathfrak{E}}\in\theta(n)$

حل: طبق تعریف θ باید داشته باشیم:

$\exists C_{\mathbf{1}},C_{\mathbf{r}},n_{\circ}>\circ$ بطوریکه $\forall n\geq n_{\circ}\quad C_{\mathbf{1}}n\leq \mathbf{\mathfrak{F}}n+\mathbf{\mathfrak{E}}\leq C_{\mathbf{r}}n$

با توجه به رابطه بالا، طرف راست بازای مقادیر $C_{\mathbf{r}}=\forall$ و $n_{\circ}=\mathbf{r}$ برقرار می‌باشد و طرف چپ رابطه بالا، بازای $C_{\mathbf{1}}=\mathbf{\mathfrak{F}}$ و $n_{\circ}=\mathbf{1}$ برقرار خواهد بود.

بنابراین به ازای $C_{\mathbf{1}}=\mathbf{\mathfrak{F}}$ و $C_{\mathbf{r}}=\forall$ و $n_{,}\geq\mathbf{r}$ عبارت $T(n)\in\theta(n)$ خواهد بود.

- **روش های تحلیل الگوریتم ها(روش های محاسبه زمان):**

معمولا الگوریتم هایی که برای حل مسائل بکار می بریم به دو دسته اصلی تقسیم می شوند :

- ✓ الگوریتم های ترتیبی
- ✓ الگوریتم های بازگشتی

- الگوریتم های ترتیبی:

برای محاسبه زمان اجرای یک تکه برنامه زمانهای زیر را محاسبه می کنیم:

۱) اعمال انتساب، عملگرهای محاسباتی، شرط های if (ساده) و غیره زمان ثابت دارند.

۲) اگر تعدادی دستور تکرار شوند زمان اجرا حاصلضرب تعداد تکرار در زمان اجرای دستورات خواهد بود. که معمولا این قسمت از برنامه ها توسط حلقه ها نمایش داده می شوند.

۳) اگر برنامه شامل ساختار if و else باشد.که هر کدام زمانهای T1,T2 خواهد بود. بنابراین زمان اجرای این تکه برنامه برابر بیشترین مقدار می باشد.
۴) زمان کل برنامه برابر حاصل جمع تکه برنامه ها می باشد.
بطور شهودی، معمولا مرتبه زمان اجرای یک الگوریتم، مرتبه تکه ای از برنامه است که بیشترین زمان را دارا می باشد. برای اینکه ما همیشه برای تابع رشد کران بالا یا بدترین حالت را در نظر می گیریم.

در مثال‌های بعدی برای سادگی محاسبه مقدار ثابت زمان اجراء را برابر یک در نظر خواهیم گرفت.

- **الگوریتم۱۰۲ جستجوی ترتیبی**

مساله : پیچیدگی زمانی الگوریتم جستجوی ترتیبی را تحلیل نمایید.

ورودی: A آرایه‌ای از عناصر، n تعداد عناصر ، x عنصر مورد جستجو.

خروجی: اندیس عنصر مورد جستجو در صورت وجود.

```
int   Seq_Search ( elementtype a[ ] ,  int    n, elementtype   x )
{
      int   i
      for (  i=0 ;  i < n  ;  i++ )
          if (  a[ i ] == x )
              return ( i )
       return (-1)
}
```

بهترین حالت الگوریتم زمانی اتفاق می‌افتد که عنصر مورد جستجو با اولین عنصر آرایه برابر باشد در اینصورت $T(n) \in O(1)$ می‌باشد.

اما در حالت متوسط وضع متفاوت است.

در این حالت احتمال اینکه عنصر مورد جستجو در خانـه اول، دوم، ... و یـا nام آرایه باشد یکسان می‌باشد و مقدار آن برای هر یک از خانـه‌هـا برابـر $\frac{۱}{n}$ مـی‌باشـد. از طرف دیگـر اگـر عنـصر مـورد جـستجو در خانـه اول، دوم، ... و یـا nام باشـد تعـداد مقایسه‌ها به‌ترتیب برابر ۱، ۲، ... یا n خواهد بـود بنـابراین زمـان متوسـط اجـراء برابـر خواهد بود با:

$$T(n) = \frac{۱}{n} \times ۱ + \frac{۱}{n} \times ۲ + ... \frac{۱}{n} \times n = \sum_{i=۱}^{n} i/n$$

$$T(n) = \frac{۱}{n} \sum_{i=۱}^{n} i = \frac{۱}{n} \frac{n(n+۱)}{۲} = \frac{n+۱}{۲}$$

بنابراین:

$$T(n) = \frac{n+۱}{۲} \le \frac{۱}{۲}(n+n) = n$$

لذا به ازای $C = ۱$ و $n_۰ = ۱$ خواهیم داشت:

$$T(n) \in O(n)$$

الگوریتم بالا در بدترین حالت، که عنصر مورد جستجو با عنصر nام برابـر باشـد دارای پیچیدگی زمانی $O(n)$ خواهد بود.

- **الگوریتم 1.5 یافتن بیشترین مقدار**

مساله : پیچیدگی زمانی پیدا کردن بیشترین مقدار در یک آرایه را تحلیل نمائید.

ورودی: آرایه A ، n تعداد عناصر.

خروجی: بیشترین مقدار آرایه

```
elementtype  Maximum( elementtype  A[ ] , int  n )
{
      Max = A[0] ;
      for( i = 1  ;  i< n  ;  i++  )
          if( A[i] > Max )
                Max = A[i] ;
       return( Max ) ;
}
```

در الگوریتم بالا حلقه for به تعداد $n-1$، بار تکرار می‌شود و به همراه آن شرط if نیز بررسی می‌شود. و الگوریتم ربطی به ترکیب داده‌ها ندارد. بنابراین زمان اجرای الگوریتم به صورت زیر خواهد بود:

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

بنابراین می‌توان نوشت:

$$T(n) = (n-1)d \le d(n+1)$$

$$\le d(2n) = 2d \times n$$

با در نظر گرفتن $C = 2d$ و $n_0 = 2$ رابطه زیر برقرار خواهد بود:

$$T(n) \in O(n)$$

به وضوح می‌توان نشان داد که:

$$T(n) \in \theta(n)$$.

مثال: مرتبه اجرایی برنامه های زیر را بدست آورید:

a) X:=x+1; O(1)

b) For i:=1 to n do
   x:= x+1 ; O(n)

c) For i:=1 to n do
   for j:=1 to n do $O(n^2)$

- ✓ **O(1) ، زمان محاسبه ثابتی را نشان می دهد**
- ✓ **O(n) ، یک تابع خطی نامیده می شود**
- ✓ $O(n^2)$ **تابع درجه دو نامیده می شود**

الگوریتم‌هایی که تا حال بررسی کردیم از نوع الگوریتم‌های ترتیبی بودند، حال می‌خواهیم الگوریتم‌های نوع دوم که به الگوریتم‌های بازگشتی معروفند را بررسی کنیم.

- **الگوریتم های بازگشتی:**

معمولا در الگوریتم های بازگشتی، مسئله را به دو یا چند زیر مسئله کوچکتر تقسیم می کنیم.عمل تقسیم مسئله به زیر مسئله ها را تا زمانی که اندازه زیرمسئله ها به اندازه کافی کوچک شوند ادامه می دهیم. بعد از تقسیم به اندازه کافی، برای حل زیر مسئله ها از خود الگوریتم استفاده می کنیم. سپس حاصل زیر مسئله را با هم ترکیب می کنیم تا راه حل مسئله بزرگتر حاصل شود. اعمال ترکیب حاصل زیر مسئله ها را تا زمانیکه مسئله اصلی حل نشده باشد ادامه می دهیم.

برای محاسبه زمان اجرای الگوریتم‌های بازگشتی به صورت زیر عمل می‌کنیم:

۱) زمان حل زیرمسئله‌ها را محاسبه می‌کنیم (که معمولاً مقدار ثابتی است)

۲) زمان لازم برای شکستن مسئله به زیرمسئله‌ها

۳) زمان لازم برای ادغام جوابهای زیرمسئله ها.

اگر مجموع سه زمان بالا را محاسبه کنیم، زمان اجرای الگوریتم بدست خواهد آمد.

- **محاسبه مقادیرالگوریتم های بازگشتی(recursive algorithm):**

همانطور که قبلا اشاره کردیم، الگوریتمی را بازگشتی می نامند که برای محاسبه مقدار تابع نیاز به فراخوانی خود به تعداد لازم باشد.از خصوصیات الگوریتم های بازگشتی می توان به سادگی پیاده سازی و همچنین سادگی درک الگوریتم و غیره اشاره کرد.

در بسیاری از موارد با توجه به خصوصیات الگوریتم های بازگشتی ممکن است برای بکارگیری در مسائل نسبت به الگوریتم های ترتیبی ترجیح داده شوند ولی همیشه استفاده از آنها مفید نیست. در بعضی از مواقع ممکن

است حافظه یا زمان اجرای زیادی را در مرحله اجرا هدر دهند. لذا غالبا بعد از تحلیل الگوریتم های بازگشتی در مورد بهتر بودن آنها در مرحله اجرا تصمیم می گیرند.

الگوریتم‌های بازگشتی شامل دو مرحله مهم هستند:

- عمل فراخوانی
- بازگشت از یک فراخوانی

با بکارگیری توابع بازگشتی دو مرحله بالا بترتیب انجام می‌گیرد. در مرحله فراخوانی اعمال زیر انجام می‌شود:

۱) کلیه متغیرهای محلی (Local Variable) و مقادیر آنها در پشته (Stack) سیستم قرار می‌گیرند.

۲) آدرس بازگشت به پشته منتقل می‌شود.

۳) عمل انتقال پارامترها (parameter passing) صورت می‌گیرد.

٤) کنترل برنامه (program counter) بعد از انجام مراحل بالا به ابتدای پردازه جدید اشاره می‌کند.

و در مراحل بازگشت عکس عملیات فوق، بصورت زیر انجام می‌شود:

۱) متغیرهای محلی از سرپشته حذف و در خود متغیرها قرار می‌گیرند.

۲) آدرس بازگشت از بالای پشته بدست می‌آید.

۳) آخرین اطلاعات از پشته حذف (pop) می‌شود.

٤) کنترل برنامه از آدرس بازگشت بند۲ ادامه می‌یابد.

نکته: پشته(Stack) ساختار دادهای است که آخرین ورودی اولین خروجی است(اطلاعات بیشتر را در فصول بعدی بحث خواهیم کرد) در این ساختار داده دو عملگر معروف بنام‌های pop و push وجود دارد که بترتیب اولی برای حذف از بالای پشته و دومی برای اضافه کردن به بالای پشته بکار می‌روند.

همانطور که در بالا اشاره کردیم برای محاسبه مقادیر الگوریتم های بازگشتی دو عمل فراخوانی و بازگشت از فراخوانی را نیاز داریم . که در بعضی مواقع ممکن است محاسبه مقدار الگوریتم بازگشتی مشکل به نظر برسد. بنابراین ترجیح دادیم که در این بخش مثال هایی را برای روشن شدن مطلب ارائه دهیم.

مثال۱: روش بازگشتی محاسبه فاکتوریل

مساله محاسبه فاکتوریل یک عدد صحیح ساده‌ترین مثال، برای بیان الگوریتم‌های بازگشتی می‌باشد. همانطور که می‌دانیم فاکتوریل یک عدد صحیح n بصورت بازگشتی زیر قابل تعریف است:

$$n! = \begin{cases} ۱ & \text{if } n = ۰ \\ n \times (n-۱)! & \text{if } n > ۰ \end{cases}$$

حال دو مرحله اصلی در محاسبه الگوریتم‌های بازگشتی را در مثال بالا بررسی می‌کنیم.

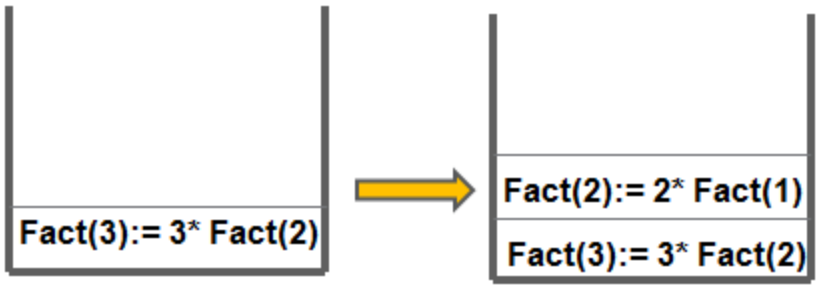
الگوریتم ترتیبی

```
int     Factorial( int  n )
{
        int  fact= 1 ;
        for( int  i=2  ; i<= n ; i ++ )
              fact*= i ;
        return fact ;
}
```

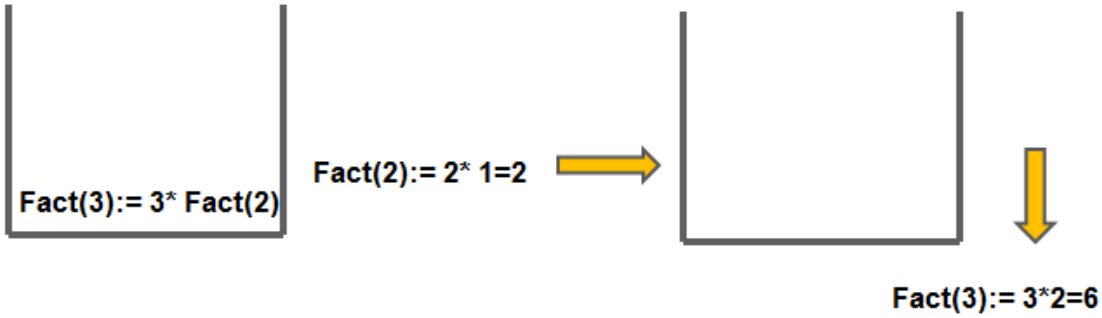
الگوریتم بازگشتی

```
int  factorial(int n)
{
If(n==1)  return 1;
 return ( n* factorial(n-1));
}
```

همانطور که قبلاً اشاره کردیم در مرحله فراخوانی، مقادیر متغیرها در پشته قرار می‌گیرند یا اصطلاحاً در پشته push می‌شوند. بنابراین برای n=3 شکل زیر را خواهیم داشت:



به ازای $n=3$ ابتدا تا هنگامی که شرط برقرار نباشد پشته پر می شود. سپس هنگام رسیدن به شرط if و درست بودن آن، پشته از بالا به پایین خالی می شود.



پس خروجی نهایی عدد ۶ می باشد.(3!=6)

مثال ۲ : روش بازگشتی محاسبه سری فیبوناچی

سری فیبوناچی یکی از مسائلی است که می‌توان آنرا بصورت غیربازگشتی نیز ارائه داد. ولی ذات آنبصورت بازگشتی است. همچنین ارائه آن بصورت بازگشتی بـه نظـر سـاده می‌رسد.

بصورت زیر می‌توان رابطه بازگشتی سری را نمایش داد:

$$fib(n)=\begin{cases}۰ & \text{if} \quad n=۱\\ ۱ & \text{if} \quad n=۲\\ fib(n-۱)+fib(n-۲) & \text{if} \quad n>۲\end{cases}$$

در حالت کلی جملات سری عبارتند از:

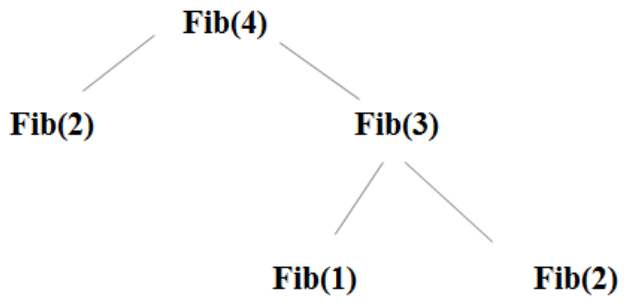
۰ ۱ ۱ ۲ ۳ ۵ ۸ ۱۳ ۰۰۰

حال، تابع بازگشتی زیر را برای تولید جملات سری فیبوناچی بکار می‌بریم:

```
int     fib ( int    n )
{
        if  ( n = = 1 )
              return ( 0 ) ;
        else if ( n = = 2 )
              return (1) ;
        else
              return ( fib ( n – 1 )  +  fib ( n – 2 ) ) ;
}
```

مراحل الگوریتم‌های بازگشتی، برای الگوریتم بازگشتی بالا به ازای n=٤ در شکل (٤. ١) نمایش داده شده است.

در نهایت تابع مقدار **2** را بعنوان خروجی برمی‌گرداند.



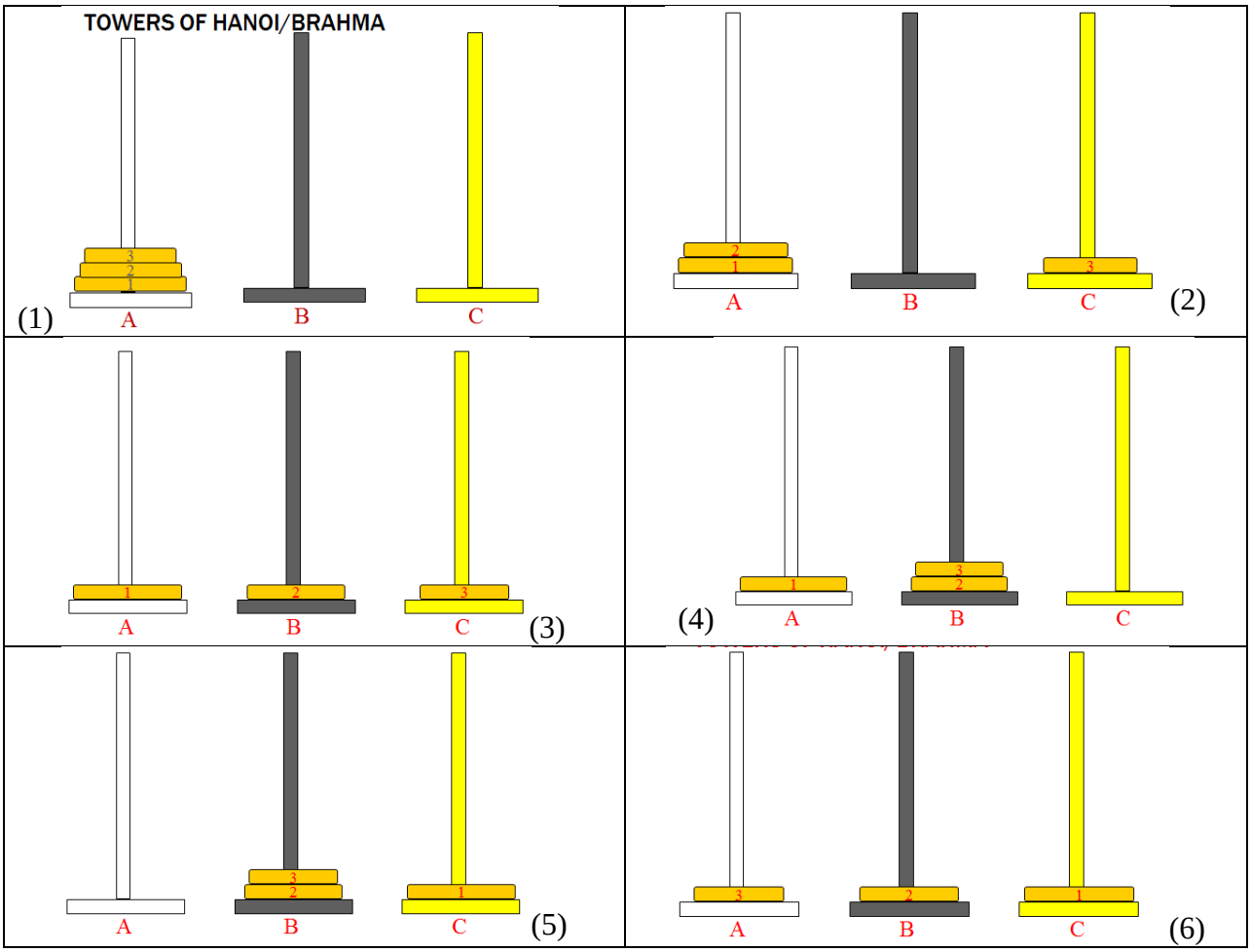
- **مساله کلاسیک برج های هانوی**

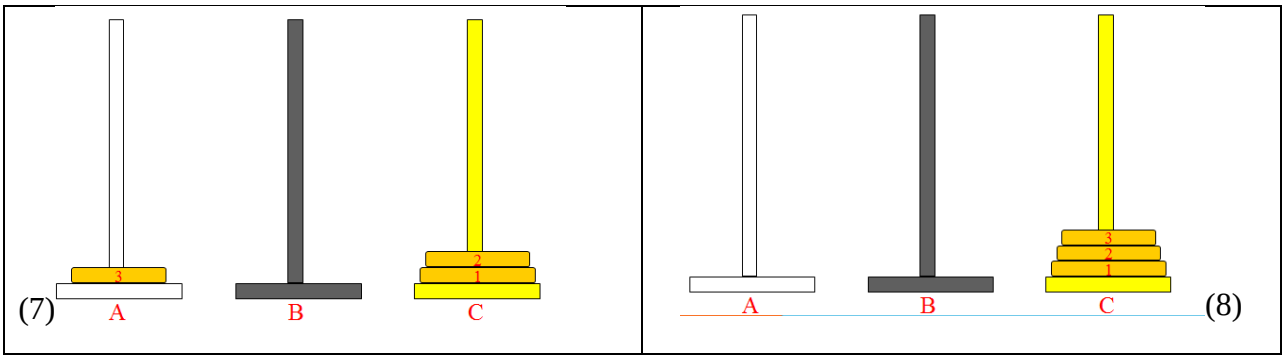
در این مسأله هدف انتقال تمام دیسک‌های روی میله A به میله دیگر مثلاً C می‌باشد، بطوریکه قواعد زیر رعایت شود:

۱) هر بار بالاترین دیسک باید حرکت داده شود.

۲) دیسک بزرگتر بر روی دیسک کوچکتر قرار نگیرد.

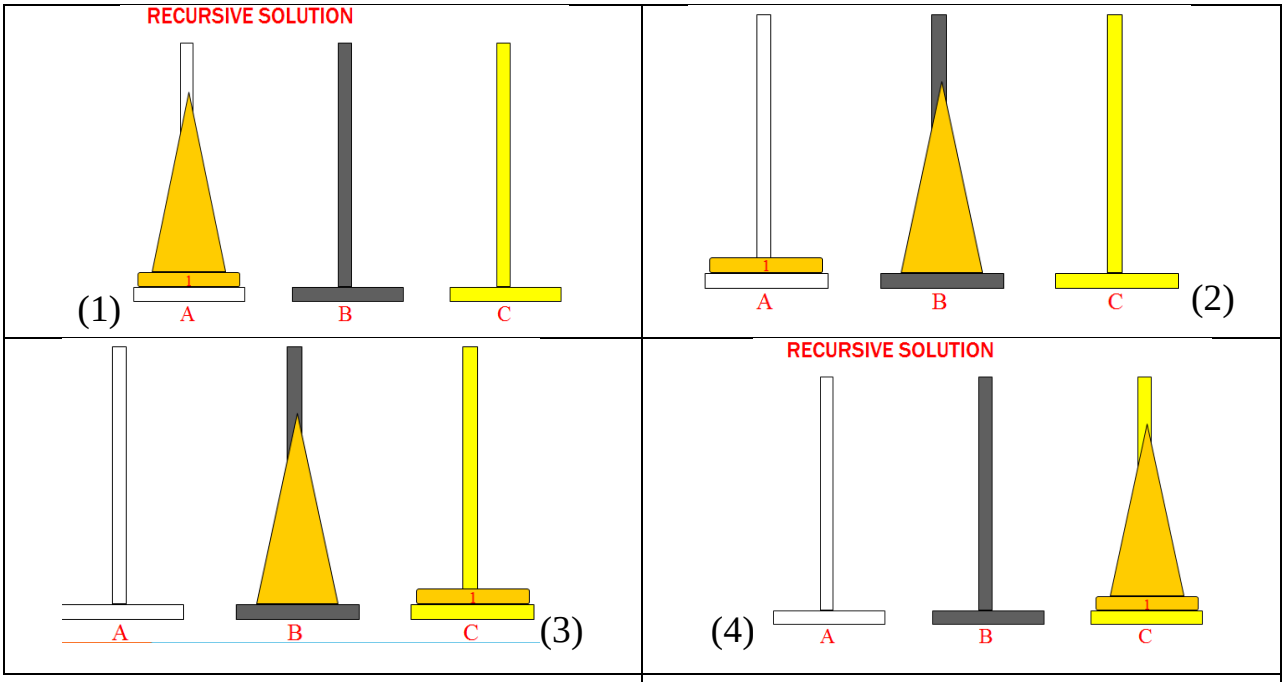
۳) در هر بار حرکت فقط یک دیسک را می‌توان انتقال داد.





اگرn=1 باشد یک جابجایی لازم است اگر n= 2 باشد ۳ جابجایی لازم است اگر n=3 باشد به 7 جابجایی لازم می باشد. در حالت کلی برای n دیسک به 2^n-1جابجایی لازم است.

با استفاده از مساله برج هانوی $n > 0$ حلقه را میتوان از A به C با استفاده از B منتقل کرد.
با استفاده از C تعداد n-1 حلقه را از Aبه B انتقال داد.

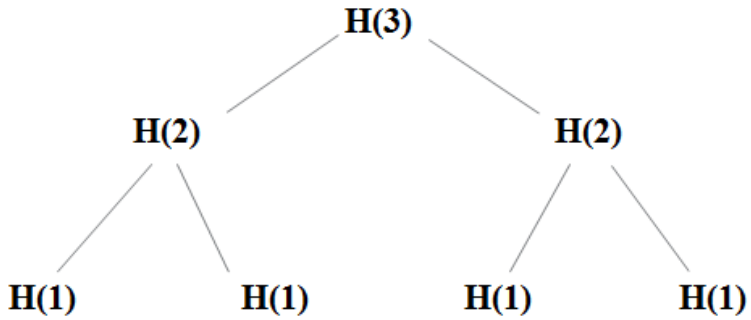


- **moves(n) = 0 when n = 0**
- **moves(n) = 2*moves(n-1) + 1 = 2^n-1 when n > 0**

الگوریتم بازگشتی هانوی:

```
Void Hanoi (int  n, A, B , C)
{
If (n==1) Move a disk from  A to C;
Else{
Hanoi (n-1, A, C, B);
Move a disk from A to C;
Hanoi (n-1, B , A, C);
}
```

الگوریتم بالا را برای n=۳ با توجه به مراحل اجرای یـک الگـوریتم بازگـشتی در شکل٦. ١ نمایش می‌دهیم.



به همین ترتیب اگر n=4 باشد تعداد کل گره های درخت بازگشتی 15 می شود و در حالت کلی برای n تعداد کل گره ها 2^n-1 یعنی از مرتبه $O(2^n)$ است.

۱۰۳۰۵ محاسبه تابع زمانی الگوریتم‌های بازگشتی

در اینجا قصد داریم طریقه محاسبه تابع زمانی الگوریتم‌هـای بازگـشتی را بحـث کنیم. برای روشن شدن مطلب از یک مثال استفاده می‌کنیم.

• **الگوریتم ۱۰۶ محاسبه فاکتوریل**

مساله: الگوریتم بازگشتی برای محاسبه فاکتوریل یک عدد نوشته و زمان اجرای الگوریتم را تحلیل کنید.

ورودی: عدد صحیح n

خروجی: محاسبه فاکتوریل عدد صحیح n

همانطور که می‌دانیم !n می‌تواند به صورت‌های زیر محاسبه شود:

$$(۱) \qquad n! = \begin{cases} ۱ & \text{if } n = ۰ \\ ۱ \times ۲ \times ۳ \times .... \times (n-۱) \times n & \text{if } n > ۰ \end{cases}$$

$$(۲) \qquad n! = \begin{cases} ۱ & \text{if } n = ۰ \\ n \times (n-۱)! & \text{if } n > ۰ \end{cases}$$

شکل (۲) محاسبه فاکتوریل یک عدد در حقیقت شکل بازگشتی مسئله می‌باشد. همانطور که مشاهده می‌کنید برای محاسبه !n نخست باید !(n−۱) محاسبه گردد. همچنین برای محاسبه !(n−۱) باید !(n−۲) محاسبه شود. این تقسیم مسئله به زیرمسئله‌های کوچکتر تا زمانیکه n به صفر نرسیده باشد، ادامه پیدا می‌کند. وقتی n به صفر رسید، با توجه به اینکه !n زمانیکه n به صفر رسیده باشد برابر یک است. زیرمسئله‌ها را حل می‌کنیم. سپس با ادغام جواب زیرمسئله‌ها در مراحل بالاتر، جواب مسئله اصلی حاصل می‌شود.

تابع بازگشتی محاسبه !n به صورت زیر می‌باشد:

```
int     fact ( int n )
{
        if ( n == 0 )
                return (1) ;
        else
                return ( n * fact ( n – 1 ) ) ;
}
```

$T(n)$ را زمان اجرای تابع (n)fact در نظر می‌گیریم. زمان اجرای دستور if برابر $O(1)$ می‌باشد و زمان اجرای else دستور if برابر $O(1)+T(n-1)$ که در آن $O(1)$ زمان مربوط به عمل ضرب و فراخوانی تابع می‌باشد. بنابراین:

$$T(n)=\begin{cases}O(1) & \text{if } n=0\\ O(1)+T(n-1) & \text{if } n>0\end{cases}$$

$T(n)$ را به صورت زیر نیز می‌توان نوشت:

$$T(n)=\begin{cases}d & \text{if } n=0\\ T(n-1)+C & \text{if } n>0\end{cases}$$

بنابراین توانستیم تابع زمانی، الگوریتم بازگشتی FACT را محاسبه کنیم . $T(N)$ را یک رابطه بازگشتی می نامند. حال باید بتوانیم رابطه بازگشتی حاصل را حل کنیم.
در اینجا یک روش ساده برای حل روابط بازگشتی ارائه می شود. این روش می تواند برای برخی از مسائل جوابگو باشد.

- **حل روابط بازگشتی:**

برای محاسبه زمان لازم برای اجرای یک الگوریتم بازگشتی و یا حافظه مورد نیاز آن در زمان اجرا، اغلب با رابطه های بازگشتی برخورد می کنیم. روابط بازگشتی معمولا با توجه به اندازه ورودی به یک معادله یا نامعادله تبدیل می شوند.
در اینجا قصد داریم یکی از روش های برای حل روابط بازگشتی ارائه دهیم. در این روش با توجه به خاصیت روابط بازگشتی با ازای n های مختلف و جایگذاری آنها در هم، جواب مسئله حاصل می شود.

٤,١. ١ روش تکرار با جای‌گذاری

این روش با استفاده از جای‌گذاری های متوالی می‌تواند، جواب مناسب را تولید کند.
در این روش با توجه به خاصیت رابطه بازگشتی به ازای nهای مختلف (که در نهایت به یک مقدار ثابت می‌رسد) و جای‌گذاری آنها در هم جواب مسئله حاصل می‌شود.

مثال1: f(10,1) را بدست آورید؟(دولتی ۸۵)

```
int   F (int n, int i)
{
if(i<=n)
return (F(n, i+1)+ i);
else return(0);
}
```

راه حل:

F(10,1) = F(10,2) +1= F(10,3) + (2+1)
= F(10,4) + (3+2+1) = …… = F(10,9) + (8+7+6+……+1)
= F(10,10) + (9+8+7+6+……+1) = F(10 ,11) +(10+9+8+……+1)
=0+ 10+ 9+ 8+7+6+……+1 = 10 *(10+1) / 2= 5* 11 = 55

در حالت کلی تابع مقدار n(n+1)/2 را که جمع اعداد ۱ تا n است، محاسبه می کند.

مثال۲: مقادیر Q(2,3) و Q(14,3) را بدست آورید.(آزاد ۷۷)

$$Q(a,b) = \begin{cases} 0 & if\ a < b \\ Q\left(a-b\ ,\ b\right)+1 & if\ b \le a \end{cases}$$

راه حل :

Q(2,3)= 0
Q(14, 3)= Q(11, 3) +1 = Q(8, 3) +2 = Q(5, 3) +3= Q(2, 3) +4 = 0 +4 =4

مثال۳: حل رابطه بازگشتی محاسبه فاکتوریل

$$T(n) = \begin{cases} d & \text{if } n = ۰ \\ T(n-۱)+C & \text{if } n > ۰ \end{cases}$$

راه حل:

T(n)= T(n-1) + C = T(n-2) + 2C = T(n-3) + 3C = ……. = T(n-(n-1)) + (n-1) C = T(n-n) + (n) C= d+ nC= O(n)

مثال۱۳. ۱: رابطه بازگشتی زیر را در نظر بگیرید:

$$T(n)=\begin{cases}C & \text{if} \quad n=۲\\ T(n-۲)+d & \text{if} \quad n>۲\end{cases}$$

رابطه بالا را به روش تکرار با جایگذاری حل کنید.

طرف راست رابطه بالا را بسط می‌دهیم. بنابراین خواهیم داشت:

$$\begin{aligned}T(n)&=T(n-۲)+d\\ &=T(n-٤)+۲d\\ &=....\\ &=T(n-۲i)+i\times d\end{aligned}$$

رابطه بالا تا زمانیکه به $T(۲)$ نرسیدیم ادامه می‌دهیم. بنابراین اگر $n-۲i$ به عدد ۲ برسد آنگاه $T(۲)$ حاصل می‌شود:

$$n-۲i=۲ \;\Rightarrow\; i=\frac{(n-۲)}{۲}$$

با جایگذاری در رابطه بالا خواهیم داشت:

$$\begin{aligned}T(n)&=T(۲)+\frac{(n-۲)}{۲}\times d\\ &=C+\frac{(n-۲)}{۲}\times d\end{aligned}$$

بنابراین $T(n)\in O(n)$.

مثال ۱.٤. رابطه بازگشتی زیر را در نظر بگیرید:

$$T(n)=۲T\left(\left\lfloor n/۲ \right\rfloor\right)+d \qquad (۱.۱)$$

که در آن d یک ثابت زمانی می‌باشد. روش تکرار با جای‌گذاری را برای رابطه بازگشتی (۱. ۱) بصورت زیر بکار می‌بریم:

$$\begin{aligned}T(n)&=۲T\left(\left\lfloor n/۲ \right\rfloor\right)+d\\ &=٤T\left(\frac{\left\lfloor n/۲ \right\rfloor}{۲}\right)+۳d\end{aligned}$$

$$\le 4T\left(n/4\right)+3d$$
$$=\ldots$$
$$\le 2^i T\left(n/2^i\right)+\left(2^i-1\right)d \qquad (1.2)$$

رابطه بالا را آنقدر ادامه می‌دهیم تا به $T(1)$ برسیم بنابراین:

$$\frac{n}{2^i}=1\Rightarrow i=\text{Log}_2^n$$

حال i را در رابطه (۱. ۲) جای‌گذاری می‌کنیم:

$$T(n)\le 2^{\text{Log}_2^n}T(1)+\left(2^{\text{Log}_2^n}-1\right)d$$

از آنجایی که $T(1)$ یک مقدار ثابت می‌باشد بنابراین خواهیم داشت:

$$T(n)\le Cn+d(n-1)$$

لذا $T(n)\in O(n)$.

تمرین:

مثال ۱. ۱۵: رابطه بازگشتی زیر را در نظر بگیرید:

$$T(n)\le\begin{cases}C_1 & \text{if} \quad n=1\\ 2T\left(n/2\right)+C_2 n & \text{if} \quad n>1\end{cases} \qquad (1.3)$$

روش تکرار با جای‌گذاری را برای حل رابطه بالا بکار ببرید.

در اولین گام برای حل n را با $n/2$ جایگزین می کنیم. تـا $T\left(n/2\right)$ حـال شـود. بنابراین:

$$T\left(n/2\right)\le 2T\left(n/4\right)+C_2\, n/2 \qquad (1.4)$$

با جای‌گذاری (٤. ١) در طرف راست رابطه (٣. ١)، خواهیم داشت:

$$T(n) \le ٤T\left(n/٤\right) + ٢C_٢n$$
$$= \ldots$$
$$\le ٢^i T\left(n/٢^i\right) + iC_٢n \qquad (٥. ١)$$

رابطه بالا را آنقدر ادامه می‌دهیم تا به $T(١)$ برسیم، بنابراین:

(با فرض اینکه n توانی از ٢ می‌باشد)

$$\frac{n}{٢^i} = ١ \Rightarrow \quad i = \text{Log } n$$

حال i را در رابطه (٥. ١) جای‌گذاری می‌کنیم:

$$T(n) \le ٢^{\text{Log } n} T(١) + C_٢n \text{ Log } n$$
$$= C_١n + C_٢n \text{ Log } n$$

بنابراین $T(n) \in O(n \text{ Log } n)$.

روش تکرار با جایگذاری، روش مناسبی برای حل روابط بازگشتی می‌باشد ولی در بعضی از موارد نمی‌توان از بازکردن فرمول، رابطه بازگشتی به جواب رسید.