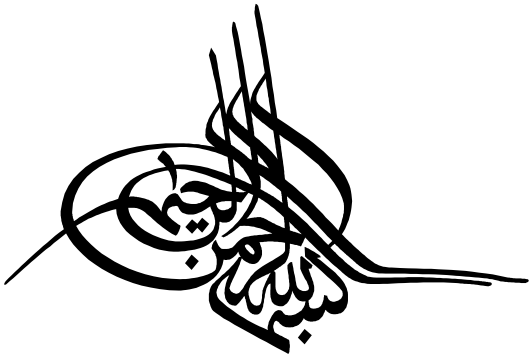


ساختمان داده‌ها

سری کتاب‌های کمک آموزشی کارشناسی ارشد

مجموعه مهندسی کامپیوتر و فناوری اطلاعات
گروه مولفان

ویراستاران علمی:
داوود حیدری‌کانی ـ علی‌اصغر فضل‌اللهی آقاملکی

سرشناسه	: گروه مولفان
عنوان	: ساختمان داده‌ها
مشخصات نشر	: تهران : مشاوران صعود ماهان، ۱۴۰۱
مشخصات ظاهری	: ۲۲۶ ص
فروست	: سری کتاب‌های کمک آموزشی کارشناسی ارشد
شابک	: ۹۷۸-۶۰۰-۴۵۸-۸۰۷-۲
وضعیت فهرست نویسی	: فیپای مختصر
یادداشت	: این مدرک در آدرس http://opac.nlai.ir قابل دسترسی است.
شناسه افزوده	: حیدری‌کانی، داوود، ویراستار علمی
شناسه افزوده	: فضل‌اللهی آقا ملکی، علی‌اصغر، ویراستار علمی
شماره کتابشناسی ملی	: ۳۷۶۸۱۳۷

نام کتاب: ..... ساختمان داده‌ها

مولف: ..... گروه مولفان

ویراستاران علمی: داوود حیدری کانی، علی اصغر فضل‌اللهی آقا ملکی

ناشر: ..... مشاوران صعود ماهان

مدیر تولید محتوی: ..... سمیه بیگی

نوبت و تاریخ چاپ: ..... ۱ / ۱۴۰۱

تیراژ: ..... ۱۰۰۰ نسخه

قیمت: ..... ۲/۲۹۰/۰۰۰ ریال

شابک: ..... ۹۷۸- ۶۰۰-۴۵۸-۸۰۷-۲ :ISBN

انتشـــارات مشاوران صعود ماهان: خیابان ولیعصر، بالاتر از تقـاطع مطهری،

روبروی قنادی هتل بزرگ تهـــران، جنب بانک ملی، پلاک ۲۰۵۰

تلفن: ۴- ۸۸۱۰۰۱۱۳

سخن ناشر

ن والقلم و ما یسطرون»

کلمه نزد خدا بود و خدا آن را با قلم بر ما نازل کرد.

به پاس تشکر از چنین موهبت الهی، موسسه ماهان درصدد برآمده است تا در راستای انتقال دانش و مفاهیم با کمک اساتید مجرب و مجموعه کتب آموزشی خود برای شما داوطلبان ادامه تحصیل در مقطع کارشناسی ارشد گام موثری بردارد.امید است تلاش‌های خدمتگزاران شما در این موسسه پایه‌گذار گام‌های بلند فردای شما باشد.

مجموعه کتاب‌های کمک آموزشی ماهان به‌منظور استفاده داوطلبان کنکور کارشناسی ارشد سراسری و آزاد تالیف شده‌اند. در این کتاب‌ها سعی کرده‌ایم با بهره‌گیری از تجربه اساتید بزرگ و کتب معتبر داوطلبان را از مطالعه کتاب‌های متعدد در هر درس بی‌نیاز کنیم.

دیگر تالیفات ماهان برای سایر دانشجویان به‌صورت ذیل می‌باشد.

- **مجموعه کتاب‌های ۸ آزمون:** شامل ۵ مرحله کنکور کارشناسی ارشد ۵ سال اخیر به همراه ۳ مرحله آزمون تالیفی ماهان همراه با پاسخ تشریحی می‌باشد که برای آشنایی با نمونه سوالات کنکور طراحی شده است. این مجموعه کتاب‌ها با توجه به تحلیل ۳ ساله اخیر کنکور و بودجه‌بندی مباحث در هریک از دروس، اطلاعات مناسبی جهت برنامه‌ریزی درسی در اختیار دانشجو قرار می‌دهد.
- **مجموعه کتاب‌های کوچک:** شامل کلیه نکات کاربردی در گرایش‌های مختلف کنکور کارشناسی ارشد می‌باشد که برای دانشجویان جهت جمع‌بندی مباحث در ۲ ماهه آخر قبل از کنکور مفید می‌باشد.

بدین‌وسیله از مجموعه اساتید، مولفان و همکاران محترم خانواده بزرگ ماهان که در تولید و به‌روزرسانی تالیفات ماهان نقش موثری داشته‌اند، صمیمانه تقدیر و تشکر می‌نماییم.

دانشجویان عزیز و اساتید محترم می‌توانند هرگونه انتقاد و پیشنهاد درخصوص تالیفات ماهان را از طریق سایت ماهان به آدرس mahan.ac.ir با ما در میان بگذارند.

موسسه آموزش عالی آزاد ماهان

سخن مولف

کتاب ساختمان داده که پیش روی شماست، براساس منابع اصلی و همچنین منابعی که در سال‌های قبل از آنها سوالاتی مطرح شده (که در انتهای این کتاب نیز آورده شده است)، همچنین با توجه به جدیدترین تغییراتی که در سوالات سراسری و آزاد صورت گرفته، طراحی شده است که پس از چندین بار ویرایش می‌توان از آن به‌عنوان یک کتاب کمک‌آموزشی و یک منبع اصلی برای داوطلبین کارشناسی ارشد در رشته‌های مهندسی کامپیوتر، مهندسی فناوری اطلاعات و همچنین علوم کامپیوتر یاد کرد. این کتاب شامل پنج فصل می‌باشد که در انتهای هر فصل تست‌های سراسری و آزاد مربوط به هر فصل را با توجه به زیرعنوان‌های آن فصل طبقه‌بندی کرده‌ایم و برای هریک از سوالات جواب‌های کاملا تشریحی درنظر گرفته‌ایم که بتوانیم داوطلبین را به درک بهتری از مساله برسانیم.

سعی شده است که مطالب بدون اشکال باشد ولی ممکن است اشکالاتی را در کتاب مشاهده بفرمایید. از این رو از شما داوطلبین گرامی تقاضا داریم تا درصورت مشاهده اشکالات احتمالی ما را مطلع بسازید تا بتوانیم در ویرایش‌های بعدی این اشکالات را برطرف سازیم.

داود حیدری‌کانی و علی‌اصغر فضل‌اللهی آقاملکی

فهرست

عنوان | صفحه

فصل اول: الگوریتم‌ها و مرتبه اجرایی آنها ۹
خصوصیات الگوریتم ۱۰
تعریف ساختمان داده ۱۱
نمادهای تحلیل الگوریتم از دیدگاه ریاضی ۱۶
نماد O بزرگ ۱۶
خواص عملگر O ۱۷
نماد امگا Ω ۱۷
نماد تتا θ ۱۸
تحلیل غیرتفصیلی ۲۳
تحلیل الگوریتم‌های بازگشتی ۲۳
مشخصات الگوریتم‌های بازگشتی ۲۳
اصول برنامه‌نویسی بازگشتی ۲۴
قضیه استقرای برنامه‌نویسی ۲۴
قضیه اصلی (Master theorem) ۲۵
تحلیل برنامه‌های بازگشتی ۲۶
مسائل بازگشتی کلاسیک ۲۷
تعیین مقدار یک تابع بازگشتی به‌ازای یک ورودی خاص ۲۹
تحلیل برنامه‌های غیربازگشتی به روش غیرتفصیلی ۳۲
دستورات تکرار منطقی ۳۴
سوالات و پاسخنامه چهارگزینه‌ای سراسری فصل اول ۳۶
سوالات و پاسخنامه چهارگزینه‌ای آزاد فصل اول ۵۳
فصل دوم: آرایه‌ها و لیست‌های پیوندی ۵۷
آرایه ۵۸
آرایه دو بعدی ۵۸
چندجمله‌ای‌ها ۶۰
ماتریس اسپارس ۶۲
ترانهاده ماتریس اسپارس ۶۳
الگوریتم ترانهاده سریع برای ماتریس‌های اسپارس ۶۴
ضرب ماتریس‌ها ۶۵
الگوریتم ضرب سریع برای ماتریس‌های اسپارس ۶۶
جست‌وجوی خطی ۶۷
جست‌وجوی دودویی (Binary search) ۶۸
ساختمان‌های داده‌ای کلاسیک ۶۸

ساختمان داده لیست ۶۸
لیست از دیدگاه ADT ۶۹
رشته (string) ۷۱
تطابق الگو (Pattern Matching) ۷۱
الگوریتم اول تطبیق الگو ۷۱
الگوریتم دوم تطبیق الگو ۷۱
پیاده‌سازی لیست ۷۳
الگوریتم‌های لیست ۷۴
خصوصیات لیست آرایه‌ای ۷۵
پیاده‌سازی لیست با اشاره‌گر ۷۵
تخصیص حافظه پویا ۷۵
الگوریتم‌های لیست پیوندی یک طرفه ۷۶
برنامه‌نویسی بازگشتی در لیست یک طرفه ۷۷
خصوصیات لیست یک طرفه ۷۹
لیست چرخشی ۷۹
لیست پیوندی دو طرفه (Doubly linked list) ۸۰
خواص لیست دو طرفه ۸۱
لیست پیوندی حلقوی (Circular Linked List) ۸۴
سوالات و پاسخنامه چهارگزینه‌ای سراسری فصل دوم ۸۷
سوالات و پاسخنامه چهارگزینه‌ای آزاد فصل دوم ۹۶
فصل سوم: صف و پشته ۹۹
صف ۱۰۰
پیاده‌سازی آرایه‌ای صف ۱۰۱
الگوریتم‌های حذف و درج صف ۱۰۲
صف حلقوی (Cyclic Queue) ۱۰۲
صف پیوندی (Linked Queue) ۱۰۳
درج در صف پیوندی ۱۰۴
ساختمان داده پشته (Stack) ۱۰۴
پیاده‌سازی پشته ۱۰۵
پشته چندگانه ۱۰۶
مساله مسیر پر پیچ و خم (MAZING) ۱۰۷
تبدیل فرم‌های مختلف عبارات ریاضی ۱۰۸
الگوریتم ارزیابی یک عبارت Postfix توسط پشته ۱۰۹
دنباله بازگشت (Tail recursion) ۱۱۱
سوالات و پاسخنامه چهارگزینه‌ای سراسری فصل سوم ۱۱۴
سوالات و پاسخنامه چهارگزینه‌ای آزاد فصل سوم ۱۲۰
فصل چهارم: درخت‌ها و گراف‌ها ۱۲۳
درخت ریشه‌دار ۱۲۴
درخت بدون ریشه ۱۲۴

گراف همبند ۱۲۴
اصطلاحات درخت‌ها ۱۲۴
تعریف استقرایی درخت ریشه‌دار ۱۲۵
انواع درخت‌های متعارف ۱۲۶
درخت از دیدگاه ADT ۱۲۷
پیاده‌سازی درخت با استفاده از آرایه ۱۲۸
درخت‌های دودویی ۱۲۹
پیاده‌سازی درخت دودویی به وسیله لیست پیوندی ۱۲۹
الگوریتم تبدیل درخت عمومی به درخت دودویی ۱۳۱
درخت عبارت ۱۳۱
نخ‌کشی درخت (Tree threading) ۱۳۲
پیمایش درخت (Tree Traversal) ۱۳۳
الگوریتم پیمایش عمقی ۱۳۶
بازسازی درخت از روی پیمایش آن ۱۳۶
بازسازی منحصر به فرد درخت‌های دودویی با استفاده از پیمایش‌های آن ۱۳۷
نخ‌کشی درخت دودویی بر مبنای پیمایش‌های آن ۱۳۸
پیمایشinorder یک درخت نخی دودویی به‌صورت غیربازگشتی ۱۳۹
درج یک گره در درخت نخی دودویی ۱۴۰
فرهنگ داده‌ای جست‌وجو (Data Dictionary) ۱۴۱
درخت جست‌وجوی دودویی ۱۴۱
جست‌وجوی یک درخت دودویی ۱۴۲
الگوریتم درج در درخت جست‌وجوی دودویی ۱۴۲
الگوریتم حذف از BST ۱۴۳
درخت AVL ۱۴۳
چرخش در درخت دودویی جست‌وجو ۱۴۴
درختان انتخابی (selection Tree) ۱۴۶
جداول درهم (Hash table) ۱۴۷
روش‌های برخورد یا تصادم در جدول درهم ۱۴۸
وارسی خطی ۱۴۸
جست‌وجو درهم ۱۴۹
ساختمان داده Heap (توده) ۱۵۱
پیاده‌سازی heap ۱۵۱
درج در یک Max Heap ۱۵۲
حذف از یک Max Heap ۱۵۳
تبدیل آرایه به Heap ۱۵۳
صف اولویت (Priority Queue) ۱۵۶
پیاده‌سازی گراف ۱۵۷
لیست مجاورت ۱۵۷
لیست‌های مجاورتی چندگانه ۱۵۷

یال‌های وزن‌دار ۱۵۷
ماتریس برخورد (Incidence Matrix) ۱۵۷
اعمال ابتدایی گراف ۱۵۸
جست‌وجوی اول عمق گراف (DFS) ۱۵۸
جست‌وجوی اول سطح گراف (BFS) ۱۵۹
جنگل‌ها (Forest) ۱۵۹
تبدیل جنگل به یک درخت دودویی ۱۶۰
پیمایش جنگل ۱۶۰
درخت‌های پوشا (Spanning Tree) ۱۶۲
الگوریتم کروسکال ۱۶۴
الگوریتم پریم ۱۶۴
الگوریتم سولین ۱۶۵
کدگذاری هافمن ۱۶۶
انواع کدگذاری هافمن ۱۶۶
مساله کوتاه‌ترین مسیر از یک رأس به همه رئوس ۱۶۸
مساله کوتاه‌ترین مسیر بین هر جفت از رئوس ۱۶۸
سوالات و پاسخنامه چهارگزینه‌ای سراسری فصل چهارم ۱۷۰
سوالات و پاسخنامه چهارگزینه‌ای آزاد فصل چهارم ۱۹۶
فصل پنجم: الگوریتم‌های مرتب‌سازی ۲۰۱
الگوریتم‌های مرتب‌سازی ۲۰۲
مرتب‌سازی انتخابی (Selection Sort) ۲۰۳
مرتب‌سازی حبابی (Bubble sort) ۲۰۳
مرتب‌سازی درجی (Insertion sort) ۲۰۵
مرتب‌سازی هرمی (Heap Sort) ۲۰۶
مرتب‌سازی ادغام (Marge sort) ۲۰۷
مرتب‌سازی سریع (Quick sort) ۲۰۹
مرتب‌سازی سطحی (Bucket Sort) ۲۱۲
مرتب‌سازی مبنا (Radix Sort) ۲۱۳
مرتب‌سازی شمارشی (Counting Sort) ۲۱۴
ویژگی‌های مرتب‌سازی شمارشی ۲۱۴
مرتب‌سازی درخت دودویی (Tree Sort) ۲۱۵
سوالات و پاسخنامه چهارگزینه‌ای سراسری فصل پنجم ۲۱۷
سوالات و پاسخنامه چهارگزینه‌ای آزاد فصل پنجم ۲۲۱
سوالات و پاسخ کنکور سراسری ۹۵ ۲۲۵
منابع ۲۲۶

فصل اول

الگوریتم‌ها و مرتبه اجرایی آنها

- تعریف ساختمان داده
- نمادهای مجانبی
- تحلیل غیرتفضیلی
- الگوریتم‌های بازگشتی
- اصول برنامه‌نویسی بازگشتی
- قضیه اصلی
- مسائل بازگشتی کلاسیک

الگوریتم‌ها و مرتبه اجرایی آنها

➢ **تعریف**: مجموعه متناهی از دستورالعمل‌ها که با دنبال کردن آنها هدف خاصی دنبال می‌شود، الگوریتم گفته می‌شود.
➢ **تعریف**: روش دقیق حل یک مساله به‌صورت گام گام را الگوریتم گویند.
به‌عنوان مثال الگوریتم زیر با استفاده از یک متغیر کمکی محتوای دو متغیر x و y را تعویض می‌کند:

```
read ln(a, b);
temp := a;
a := b;
b := temp;
```

همچنین الگوریتم زیر همین کار را بدون استفاده از متغیر کمکی انجام می‌دهد.

```
read ln(a, b);
a := a + b;
b := a – b;
a := a – b;
```

همان‌طورکه ملاحظه می‌کنید، یک الگوریتم را می‌توان به طرق مختلف نوشت.

خصوصیات الگوریتم

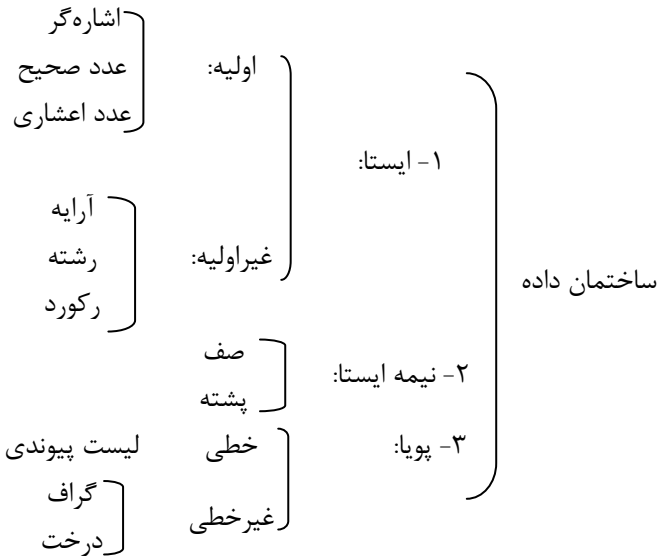
۱) ورودی: الگوریتم لزوماً ورودی ندارد. یعنی هم می‌تواند ورودی داشته باشد و هم می‌تواند نداشته باشد.
۲) خروجی: هر الگوریتم حداقل یک خروجی دارد.
۳) قطعیت (عدم‌ابهام): هر یک از دستورات الگوریتم باید دقیق و بدون ابهام باشد.
۴) محدودیت (پایان‌پذیری): الگوریتم بایستی بعد از طی تعدادی مرحله بالاخره پایان پذیرد.
۵) کارایی (انجام‌پذیری): بایستی امکان پیاده‌سازی و اجرای الگوریتم روی کاغذ وجود داشته باشد. به‌عبارت دیگر بایستی الگوریتم انجام‌پذیر باشد.
برای این‌که کارایی یک الگوریتم را مورد بررسی قرار دهیم باید به عوامل زیر توجه کنیم:
۱) زمان اجرای الگوریتم ۲) حافظه مورد نیاز الگوریتم
که از میان این ۲، عامل اوّل از اهمیت بیشتری برخوردار است.

برنـامـه

به مجموعه‌ای از دستورالعمل‌های یک زبان برنامه‌نویسی که الگوریتمی را پیاده‌سازی می‌کند، برنامه گویند. تنها تفاوت برنامه الگوریتم در این است که برنامه می‌تواند پایان‌پذیر نباشد. به‌عنوان مثال، سیستم‌عامل برنامه‌ای است که هیچ‌گاه پایان نمی‌پذیرد و همواره در یک سیکل انتظار برای دریافت فرمان بعدی قرار می‌گیرد.
هدف از تحلیل الگوریتم‌ها بررسی تغییر سرعت یا زمان اجرای یک الگوریتم در برابر تغییر اندازه ورودی است و معمولاً خروجی این تحلیل، تابعی بر حسب اندازه ورودی آن می‌باشد.

تعریف ساختمان داده (Data structure)

به ساختارهایی که جهت دریافت داده‌های خام به‌صورتی مناسب توسط کامپیوتر برای پیاده‌سازی و اجرای الگوریتم‌های مختلف روی آن‌ها مورد استفاده قرار می‌گیرند، ساختمان داده گفته می‌شود.
نمونه‌هایی از تقسیم‌بندی ساختمان داده‌های مختلف به‌صورت زیر است:



نکته: ساختمان داده‌های پویا با استفاده از اشاره‌گرها تغییراتی نامحدود و پویا متناسب با داده‌ها را فراهم می‌کنند ولی ساختمان داده‌های ایستا در طول اجرای برنامه تغییر نمی‌کنند.

عوامل موثر در سرعت (زمان) اجرای برنامه

۱- نوع سخت‌افزار
۲- نوع برنامه‌نویسی: به‌طورکلی ۲ نوع روش برنامه‌نویسی وجود دارد:
۱) شیوه از پایین به بالا (Down Top)
۲) روش از بالا به پایین (Top Down) که در این روش ابتدا برنامه به بلوک‌هایی تقسیم شده؛ سپس هر بلوک نوشته می‌شود. الگوریتم‌های تقسیم و غلبه یک روش بالا به پایین است و معمولاً الگوریتم‌های آن بازگشتی است.
۳) نوع کامپایلر: به‌عنوان مثال کدهای تولید شده توسط کامپایلر پاسکال، کندتر از کدهای تولید شده توسط کامپایلر C است.
۴) اندازه ورودی
۵) ترکیب ورودی: نحوه چیدمان مجموعه داده‌های ورودی و طرز نمایش آن‌ها
۶) پیچیدگی الگوریتم: در واقع ممکن است برای یک مساله الگوریتم‌های مختلف با زمان‌های مختلف وجود داشته باشد.

زمان اجرای الگوریتم برحسب n را معمولاً با $T(n)$ نشان می‌دهند که n اندازه ورودی می‌باشد. ممکن است یک الگوریتم بیش از یک ورودی داشته باشد. مثلاً با اندازه ورودی‌های m, n، زمان اجرای این برنامه که ۲ ورودی دارد به‌صورت $T(n,m)$ نشان داده می‌شود.
با توجه به حالت‌های مختلف ورودی تحلیل زمانی می‌تواند به سه دسته زیر تقسیم شود که عبارتند از:

۱- بهترین حالت (Best Case): $T_B(N)$

۲- بدترین حالت (Wrost Case): $T_W(N)$

۳- حالت متوسط (Average Case): $T_A(N)$

نکته: ممکن است بهترین حالت ورودی برای یک الگوریتم، بدترین حالت برای الگوریتم دیگری باشد.

در اغلب اوقات داریم:

$$T_A(N) = \frac{T_B(N) + T_W(N)}{۲}$$

تحلیل تفصیلی

برای انجام تحلیل تفصیلی، مراحل زیر را دنبال می‌کنیم:

۱- تعداد تکرار هر دستور برنامه، به‌ازای اندازه ورودی مشخص می‌شود.

۲- زمان اجرای یک بار هر دستور از برنامه را مشخص می‌کنیم که با توجه به فرض ثابت بودن نوع سخت‌افزار، این تعداد یک مقدار ثابت خواهد بود.

۳- نتایج حاصل از مراحل ۱ و ۲ به‌ازای هر دستور در هم ضرب می‌شوند.

۴- نتایج حاصل از مرحله ۳ به‌ازای هر دستور با هم جمع می‌شوند و حاصل تابع تحلیل الگوریتم را نمایش می‌دهد.

زمان اجرایی یک الگوریتم، با تعداد دفعاتی که عملیات اصلی انجام می‌شود، متناسب است. در تحلیل زمان اجرای یک الگوریتم، تمام دستورات را شمارش نمی‌کنیم، زیرا تعداد دستورات به نوع زبان برنامه‌نویسی بستگی دارد، بلکه تنها عملیات اصلی که مستقل از کامپیوتر و زبان برنامه‌نویسی هستند شمرده می‌شوند.

شمارش گام‌های یک برنامه

اگر هر عمل اصلی در یک برنامه، یک گام باشد، گام‌های هر برنامه، با استفاده از قوانین زیر، قابل شمارش می‌باشد.

۱- تعاریف نوع‌ها، زیربرنامه‌ها و توابع دارای تعداد گام صفر می‌باشند. به‌عنوان مثال دستورات زیر، هیچ تعداد گامی ندارند.

Procedure P(...) → ۰

Function F(...) → ۰

Void V(...) → ۰

Data Type K(...) → ۰

۲- هر بلاک شامل {} یا begin و end دارای گام صفر می‌باشند.

۳- تعاریف متغیرها هنگامی‌که مقداردهی اولیه برای آنها انجام بگیرد دارای گام یک و درغیراین‌صورت دارای گام صفر می‌باشند.

char c; → ۰

char c = "c"; → ۱

۴- هر دستور اجرایی به‌ازای هربار اجرا، دارای گام یک می‌باشد.

x = y + z + k; → ۱

۵- در دستور شرطی if، عبارت شرط دارای گام یک می‌باشد، اما با توجه به درست یا نادرست‌بودن شرط گام به‌صورت مجزا محاسبه می‌شود.

```
if (x < y) →۱
  s = ۲;→۱
else
  s = ۵;→۱
```

تعداد گام‌های شرط درست = ۱+۱ = ۲

تعداد گام‌های شرط نادرست = ۱+۱ = ۲

```
if (x < y)   →۱
  s = ۲;     →۱
else
{
    s = ۵;  }
    t = ۷;  } →۲
}
```

تعداد گام‌های شرط درست = ۱+۱ = ۲

تعداد گام‌های شرط نادرست = ۱+۲ = ۳

۶- دستورات حلقه for

برای شمارش گام‌های دستور for، به‌صورت زیر عمل می‌کنیم:

الف) تعداد تکرار حلقه را محاسبه کنید

ب) تعداد تکرار خود دستور for یکی بیشتر از تعداد تکرار دستورات داخل for است.

ج) جملات تکرارشونده حلقه for، به تعداد تکرار گام دارد.

مثال:

```
for i := ۱ to n do
    s := s + ۱
```

تعداد گام
n + ۱
n
مجموع = ۲n + ۱

```
for i := ۲ to n − ۱ do
    s := s + ۱
```

تعداد گام
n − ۱
n − ۲
مجموع = ۲n − ۳

```
for i := ۲ to n do
    begin
        s := s + ۳;
        r := r − ۳;
    end
```

تعداد گام
n
۰
n − ۱
n − ۱
۰
مجموع = ۳n − ۲

نکته: همان‌طورکه در مثال‌ها ملاحظه کردید، تعداد تکرار در این حلقه‌ها توسط فرمول "۱+ ابتدا - انتها" به‌دست می‌آمد؛ بنابراین داریم:

for i := a to b do ⇒ تعداد تکرار = b − a + ۱

for(i = a; i < b; i++) ⇒ تعداد تکرار = b − a

مثال:

```
for(i = ۲; i < n; i++)
    s = s + ۱
```

تعداد گام
n − ۱
n − ۲
۲n − ۳

```
for i := ۱ to m do
  for j := ۱ to n do
  s := s + ۱;
```

تعداد گام
m + ۱
m × (n + ۱)
m × n
مجموع = (m + ۱) + m(n + ۱) + nm = ۲mn + ۲m + ۱

```
for i := ۱ to m do
  for j := ۲ to n − ۱ do
    for k := ۱ to L do
    s := s + ۱;
```

تعداد گام
m + ۱
m × (n − ۱)
m × (n − ۲)(L + ۱)
m × (n − ۲)(L)
مجموع = ۲mnL − ۴mL + ۲mn − ۲m + ۱

```
procedure  Add(Var a,b,c:matrix; m,n:integer);
var
    i, j :  integer;
begin
   for i:=1 to m do
   for j:=1 to n  do
   c[i,j]:=a[i,j],b[i,j];
end;
```

تعداد گام
۰
۰
۰
۰
m + ۱
m(n + ۱)
mn
۰
مجموع = ۲mn + ۲m + ۱

	تعداد گام‌ها
procedure p(x : integer);	۰
var	۰
s, i, j : integer;	۰
begin	۰
i := ۰;	۱
for i := ۱ to m do	m + ۱
for j := ۱ to n do	m × (n + ۱)
s := s + ۱;	mn
s := s + ۱;	۱
end	۰
	مجموع = ۲mn + ۲m + ۳

نکته: تعداد گام حلقه while مانند for قابل محاسبه است، یعنی گام حلقه، یک واحد از تعداد تکرار حلقه بیشتر است.

	تعداد گام
i = ۱;	۱
while(i <= n)	n + ۱
{	۰
s = s + ۱;	n
i = i + ۱;	n
}	۰
	مجموع = ۳n + ۲

نکته: تعداد گام‌های حلقه‌های repeat...until و do...while برابر تعداد تکرار حلقه می‌باشند، زیرا در این نوع حلقه‌ها، شرط حلقه در انتهای آن کنترل می‌شود.

	تعداد گام
i := ۱	۱
repeat	۰
i := i + ۱;	n − ۱
until i >= n;	n − ۱
	مجموع = ۲n − ۲

تعداد اجرای جمله x = x * ۲ را در برنامه زیر به‌دست آورید:

```
for(i = ۱; i ≤ n − ۱; i + +)
    for (J = ۱; J ≤ i ; J + +)
        x = x * ۲
```

حل: به‌ازای i = ۱، متغیر J دارد از ۱ تا ۱ تغییر می‌کند در نتیجه برنامه اصلی یک بار اجرا می‌شود. به‌ازای i = ۲ متغیر J دارد از ۱ تا ۲ تغییر می‌کند. در نتیجه برنامه اصلی ۲ بار اجرا می‌شود.
به‌ازای i = n − ۱ متغیر J از ۱ تا n − ۱ تغییر می‌کند؛ در نتیجه برنامه اصلی n − ۱ بار اجرا می‌شود. تعداد اجرای برنامه اصلی برابر است با حاصل‌جمع مراحل گفته شده یعنی:

$$۱ + ۲ + \cdots + n - ۱ = \frac{(n-۱)n}{۲}$$

در برنامه بالا مرتبه اجرای برنامه برابر است با: $\theta(n^۲)$ (θ در ادامه درس توضیح داده می‌شود.)

نکته: در اکثر موارد محاسبه تعداد اجرای همه خطوط موردنظر نمی‌باشد بلکه فقط مرتبه آن برنامه خواسته می‌شود. بنابراین برای محاسبه مرتبه لازم نیست، تعداد اجرای همه خطوط محاسبه شوند، بلکه تعداد اجرای جمله اصلی مرتبه را مشخص می‌کند.

مرتبه اجرایی برنامه زیر را محاسبه کنید:

```
for(i = ۱ ; i ≤ n ; i++)
    for(J = ۱ ; J ≤ n ; j = j + i)
        x -- ;
```

با توجه به این‌که حلقه دومی به حلقه اولی وابسته است:

i	j	تعداد اجرای دستور اصلی
۱	۱,۲,۳,...,n	n بار
۲	۱,۳,۵,...,n	$\frac{n}{۲}$ بار
۳	۱,۴,۷,...,n	$\frac{n}{۳}$ بار
⋮		
n	۱	$\frac{n}{n}$ = ۱بار

$$x -- \text{ تعداد اجرای جمله } = \text{مجموع حالات } = n + \frac{n}{۲} + \frac{n}{۳} + \cdots + ۱$$

$$= n\left(۱ + \frac{۱}{۲} + \cdots + \frac{۱}{n}\right) = \theta(n \log n)$$

نمادهای تحلیل الگوریتم از دیدگاه ریاضی

نماد O بزرگ

این نماد به‌صورت زیر تعریف می‌گردد:

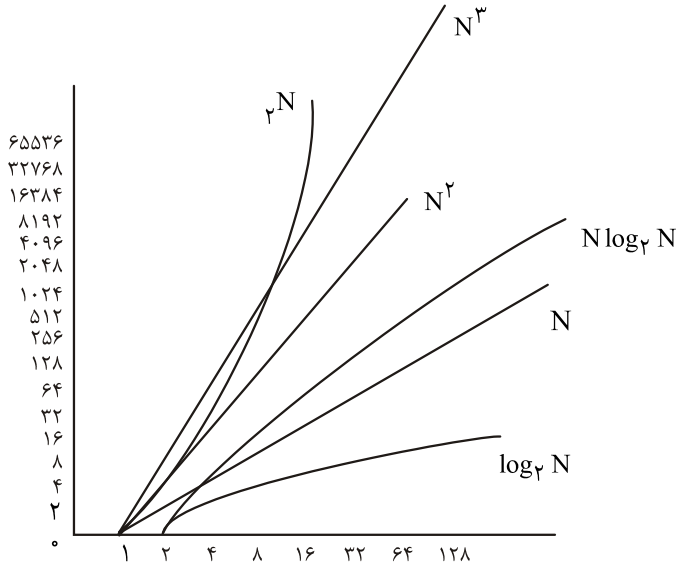
$$f(n) = O(g(n)) \Leftrightarrow \{\exists c > ۰ , n_۰ > ۰, \forall n \geq n_۰ , f(n) \leq cg(n)\}$$

به‌طور شهودی می‌توان گفت که اگر $f(n) = O(g(n))$، آنگاه رشد تابع $f(n)$ در بی‌نهایت از $g(n)$ کمتر است.
یا اینکه حد بالای رشد f(n) به‌ازای nهای به قدر کافی بزرگ g(n) است.

نکته: ترتیب رشد توابع معروف به‌صورت زیر می‌باشد.

$$O(۱) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^۲) < O(n^۳) < \ldots < O(n^k) < O(۲^n) < O(n!) < O(n^n)$$

در شکل زیر نمودار رشد چند تابع مهم آمده است:



خواص عملگر O:

۱- O یک رابطه بازتابی است. زیرا:

$$f(n)=O(f(n))$$

۲- O یک رابطه تقارنی نمی‌باشد. زیرا به‌عنوان مثال نقض داریم:

$$n^۳ \neq O(n^۲) \quad \text{ولی} \quad n^۲ = O(n^۳)$$

۳- O یک رابطه تراگذاری است. زیرا:

$$f(n)=O(g(n)), g(n)=O(k(n)) \Rightarrow f(n)=O(k(n))$$

۴- O یک رابطه پاد تقارنی نیست. زیرا به‌عنوان مثال نقض داریم:

$$f(n)=O(g(n)) \,\&\, g(n)=O(f(n)) \not\Rightarrow f(n)=g(n)$$

$$\begin{matrix} g(n)=n^۲ \\ f(n)=n^۲\log n \end{matrix} \Rightarrow n^۲\log n = O(n^۲) \not\Rightarrow f(n)=g(n)$$

توابع زیر از مرتبه $O(n^۳)$ می‌باشند:

$$f(n)=۵n^۲+۱۰۰۰ \qquad f(n)=n^۳$$

$$f(n)=n^۲+n\log n \qquad f(n)=۴n^۳+n^۲\log n$$

$$f(n)=\frac{n^۳}{\log n+۱۰} \qquad f(n)=۵n^۳-۳n^۲-۴n$$

نماد امگا (Ω)

نماد امگای بزرگ به‌صورت زیر تعریف می‌باشد:

$$f(n)=\Omega(g(n)) \Leftrightarrow \{\exists n_۰, c>۰, \forall n>n_۰, f(n)\geq cg(n)\}$$

به‌طورشهودی اگر $f(n)=\Omega(g(n))$ آنگاه از جایی به بعد مقدار تابع $f(n)$ از ضریبی از $g(n)$ بیشتر است یا به‌عبارت معادل حداقل مقدار تابع $f(n)$ به‌ازای nهای بزرگ برابر $g(n)$ است.

$$f(n)=O(g(n))\Rightarrow g(n)=\Omega(f(n))$$

نکته: رابطه Ω بازتابی و تعدی می‌باشد ولی تقارنی نمی‌باشد.

مثال: ثابت کنید $۵n^۲\in\Omega(n^۲)$

حل:

$$۵n^۲\geq ۱\times n^۲\Rightarrow n_۰=۰,c=۱$$

مثال: ثابت کنید $n^۲+۱۰n\in\Omega(n^۲)$

حل:

$$n^۲+۱۰n\geq n^۲\Rightarrow n_۰=۰,c=۱$$

مثال: ثابت کنید $\frac{n(n-۱)}{۲}\in\Omega(n^۲)$

حل:

$$n\geq ۲\Rightarrow n-۱\geq\frac{n}{۲}\Rightarrow\frac{n(n-۱)}{۲}\geq\frac{n}{۲}\times\frac{n}{۲}=\frac{n^۲}{۴}\Rightarrow n_۰=۲,c=\frac{۱}{۴}$$

توابع زیر از مرتبه $\Omega(n^۳)$ می‌باشند:

$f(n)=n!$ $\quad$ $f(n)=n^۳$

$f(n)=e^n$ $\quad$ $f(n)=۴n^۳+n^۲\log n$

$f(n)=n^۴+۳n^۳$ $\quad$ $f(n)=۵n^۳-۳n^۲-۴n$

نماد تتا (θ)

نماد تتا به‌صورت زیر تعریف می‌شود:

$$f(n)=\theta(g(n))\Leftrightarrow\{\exists c_۱,c_۲,n_۰>۰,\forall n>n_۰,c_۱g(n)\leq f(n)\leq c_۲g(n)\}$$

نکته: اگر $\lim_{n\to\infty}\frac{f(n)}{g(n)}$ برابر عدد ثابتی باشد آن‌گاه رشد تابع $f(n)$ برابر رشد تابع $g(n)$ است و می‌نویسیم:

$$f(n)=\theta(g(n))\quad\text{or}\quad g(n)=\theta(f(n))$$

نکته: رابطه θ خاصیت تقارنی، بازتابی و تراگذاری دارد؛ بنابراین θ یک رابطه هم‌ارزی است.

$f(n)=n^۳$ $\quad$ $f(n)=۴n^۳+n^۲\log n$ $\quad$ $f(n)=۵n^۳-۳n^۲-۴n$

نماد (o)

نماد o کوچک به‌صورت زیر تعریف می‌شود:

$$f(n)=o(g(n))\Leftrightarrow\{\exists n_۰>۰,\forall c>۰,\forall n>n_۰,f(n)<cg(n)\}$$

مقدار تابع $f(n)$، از جایی به بعد به‌طوری‌که از ضریبی از $g(n)$ کمتر است.

نکته: اگر داشته باشیم:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

آنگاه می‌توان نوشت:

$$f(n)=o(g(n))$$

نکته: o خاصیت بازتابی تقارنی را ندارد ولی پاد تقارنی و تعدی است.

نکته: هر عبارتی که پیچیدگی زمانی آن‌ها از مرتبه o (اُ کوچک) باشد، حتماً از مرتبه O (اُ بزرگ) نیز می‌باشد، ولی عکس آن همیشه درست نمی‌باشد.

مثال: ثابت کنید $5n^2 \in o(n^2)$

حل:

$$5n^2 \le 5n^2 \Rightarrow n_0=0, C=5$$

مثال: ثابت کنید $T(n)=\frac{n(n-1)}{2} \in o(n^2)$

حل:

$$\frac{n(n-1)}{2} \le \frac{n(n)}{2}=\frac{n^2}{2} \Rightarrow n_0=0, C=\frac{1}{2}$$

مثال: ثابت کنید $n^2 \in o(n^2+10n)$

حل:

کافیست قرار دهیم $n_0=1$ و $C=1$

نماد امگای کوچک (ω):

این نماد به‌صورت زیر تعریف می‌شود:

$$f(n)=\omega(g(n)) \Leftrightarrow \{\exists n_0 \ge 0, \forall c>0, \forall n>n_0, f(n)>cg(n)\}$$

نکته: به‌طورشهودی هرگاه $f(n)=\omega(g(n))$ آنگاه به‌ازای nهای بزرگ مقدار تابع $f(n)$ بیشتر از مضربی از $g(n)$ است. در این حالت داریم:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty \quad \text{یا} \quad \lim_{n\to\infty}\frac{g(n)}{f(n)}=0$$

نکته:

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

نکته: ω بازتابی و تقارنی نمی‌باشد ولی تراگذاری و پادتقارنی می‌باشد.

نکته: هر عبارتی که پیچیدگی زمانی آن‌ها از مرتبه ω باشد حتماً از مرتبه Ω نیز می‌باشد، ولی عکس آن همیشه درست نمی‌باشد.

نکته:

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \in \theta(n^k)$$

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

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \in \theta(n^3)$$

$$\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2 \in \theta(n^4)$$

.

.

.

$$\sum_{i=1}^{n} i^k \in \theta(n^k)$$

نکته: تمام توابع لگاریتمی در یک دسته پیچیدگی قرار دارند، بنابراین داریم:

$$\log_a n = \theta(\log_b n)$$

نکته:

$$\log n! = \theta(n \log n)$$

نکته: هر تابع لگاریتمی در نهایت بهتر از هر تابع چند جمله‌ای و هر تابع چند جمله‌ای در نهایت بهتر از هر تابع نمایی و هر تابع نمایی در نهایت بهتر از هر تابع فاکتوریل می‌باشد:

$$\log n \in o(n) \quad , \quad n^{10} \in o(2^n) \quad , \quad 2^n \in o(n!)$$

مثال:

$$10n + 3\log n + 7n \log n + 7n^2 = \theta(n^2)$$

بنابراین در تعیین مرتبه یک عبارت، همواره می‌توان جملاتی از مرتبه پایین‌تر را حذف نمود.

✓ **تست: فرض کنید g,f دو تابع دلخواه به شکل $f,g : N \to R^+$ باشند و $\lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty$**

۱) $f(n) \in O(g(n)), g(n) \notin \Omega(f(n))$

۲) $g(n) \in O(f(n)), f(n) \in O(g(n))$

۳) $f(n) \in \Omega(g(n)), f(n) \notin \theta(g(n))$

۴) $f(n) \in \theta(g(n)), g(n) \notin \Omega(f(n))$

حل: گزینه «۳»

✓ **تست: کدام‌یک از عبارات زیر نادرست است؟**

۱) $10^n + n^{20} \notin \theta(n^n)$

۲) $\log_2 n \in \theta(\log_{10} n)$

۳) $(\log_2 n)! \in \Omega(n!)$

۴) $4n^3 + 5n^2 + 7n \in \Omega(\log_2 n)$

حل: گزینه «۳»

✓ **تست: کدام‌یک از عبارات زیر درست است؟**

الف) $e^{c\sqrt{n}} = O\left(e^{\sqrt{n}}\right)$

ب) $n^{۲} = O(n \log n)$

ج) $n = O(n \log n)$

۱) فقط ب ۲) فقط ج ۳) الف و ب ۴) الف و ج

حل: گزینه «۲»

✓ **تست: کدام‌یک از گزینه‌های زیر صحیح است؟**

۱) $n^{۳} \log n = O\left(n^{۳+\varepsilon}\right) \quad ۰ < \varepsilon < ۰/۱$

۲) $\sqrt{n} = O(\log n)$

۳) $n^{۱+\varepsilon} = O(n \log n) \quad ۰ < \varepsilon < ۰/۱$

۴) $n^{۲} = O\left(\frac{n^{۲}}{\log n}\right)$

حل: گزینه «۳»

✓ **تست: کدام گزینه تعداد مراحل برنامه زیر را نشان می‌دهد؟**

```
void Sum(int m, int n, float S[ ][ ])
{int i, j;
 for(j=۰; j<m; j++)                    m+۱
   {S[n-۱][i]=۰;                       m
    for(i=۰; i<n-۱; i++)               mn
     S[n-۱][j]+=S[i][j];               m(n-۱)
}
```

۱) $۲m + ۲n$

۲) $mn^{۲} + m^{۲}n$

۳) $m(۲n+۱)+۱$

۴) $m(۲n+۱)-۱$

حل: گزینه «۳»

$$m+۱+m+m(۲n-۱) = m(۲n+۱)+۱$$

✓ **تست: کدام‌یک از گزینه‌های زیر صحیح است؟**

۱) $(n+۱)(n^{۲}-۲n+۱) \in O(۲^{n})$

۲) $(n+۱)(n^{۲}-۲n+۱) \in \theta(n)$

۳) $(n+۱)(n^{۲}-۲n+۱) \in \Omega(n^{۴})$

۴) $(n+۱)(n^{۲}-۲n+۱) \in O(n^{۲}\log n)$

حل: گزینه «۱»

✓ **تست: فرض کنید $f(n) = O(g(n))$ کدام‌یک از گزینه‌های زیر برای هر تابع با مقادیر مثبت مانند $f(n), g(n)$ برقرار است؟**

۱) $g(n) = \theta(f(n))$

۲) $۲^{f(n)} = O\left(۲^{g(n)}\right)$

۳) $۲^{f(n)} = \Omega\left(۲^{g(n)}\right)$

۴) $\log(g(n)) = O(\log f(n))$ با فرض: $\log(g(n)) \geq ۱$

حل: گزینه «۲»

مثال: توابع زیر را برحسب رشد مرتب نمایید.

$(\sqrt{۲})^{\log n}$	$n^{۲}$	$n!$	e^{n}	۲^{n}
$n^{۳}$	$\log(n!)$	$n^{\frac{۱}{\log n}}$	n	$n^{\log\log n}$
$\ln n$	$\ln\ln n$	$۳^{\log n}$	$(\log n)^{\log n}$	$n\log n$
$(\log n)!$	$(۲^{۲})^{n}$	$n۲^{n}$	$۲^{\log n}$	$۲^{\sqrt{۲\log n}}$
$(۲^{۲})^{n+۱}$	$\sqrt{\log n}$	$(n+۱)!$	$۴^{\log n}$	$(\frac{۳}{۲})^{n}$

حل:

نکته: $a^{\log_b^c} = c^{\log_b^a}$ در نتیجه:

$$(\log n)^{\log n} = n^{\log\log n}$$

$$۴^{\log n} = n^{۲}$$

$$۲^{\log n} = n$$

$$n^{\frac{۱}{\log n}} = n^{\frac{\log ۲}{\log n}} = (n^{\log ۲})^{\frac{۱}{\log n}} = (۲^{\log n})^{\frac{۱}{\log n}} = ۲$$

$$(\sqrt{۲})^{\log n} = n^{\log\sqrt{۲}} = n^{\log ۲^{\frac{۱}{۲}}} = n^{\frac{۱}{۲}\log ۲} = n^{\frac{۱}{۲}} = \sqrt{n}$$

$$e^{n} = (۲/۷۱)^{n}$$

$$\log(\sqrt{۲})^{\log n} = \log(۲^{\frac{۱}{۲}\log n}) = \frac{۱}{۲}\log n * \log ۲ = \frac{۱}{۲}\log n$$

$$\log ۲^{\sqrt{۲\log n}} = \sqrt{۲\log n}$$

$$\frac{۱}{۲}\log n = \omega\sqrt{۲\log n}$$

$$\Rightarrow (\sqrt{۲})^{\log n} = \omega(۲^{\sqrt{۲\log n}})$$

$$\log\log^{۲} n = \log(\log n)^{۲} = ۲\log(\log n)$$

$$\log ۲^{\sqrt{۲\log n}} = \sqrt{۲\log n}$$

$$۲^{\sqrt{۲\log n}} \geq \log^{۲} n \Rightarrow \log ۲^{\sqrt{۲\log n}} \geq \log\log^{۲} n$$

$$\sqrt{۲\log n} = \omega(۲\log(\log n))$$

$$\Rightarrow ۲^{\sqrt{۲\log n}} = \omega(۲^{۲\log(\log n)}) = \omega((\log n)^{۲\log ۲}) = \omega(\log^{۲} n)$$

$$n! = \theta(\frac{n^{n}}{e^{n}})$$

$$(\ln n)! = \theta(\frac{(\ln n)^{\ln n}}{e^{\ln n}}) = \theta(\frac{\ln^{\ln n} n}{n})$$

با توجه به توضیحات فوق ترتیب توابع داده شده به‌صورت زیر می‌باشد:

$$(۲^۲)^{n+۱} > (۲^۲)^n > (n+۱)! > n! > e^n > n۲^n > ۲^n > (\frac{۳}{۲})^n > n^{\log\log n} = (\log n)^{\log n} > \log(n!) > n^۳ > ۴^{\log n}$$

$$> \log(n!) = n\log n > ۳^{\log n} > ۲^{\log n} = n > (\sqrt{۲})^{\log n} = \sqrt{n} > ۲^{\sqrt{۲\log n}} > \ln n > \sqrt{\log n} > \ln\ln n > n^{\frac{۱}{\log n}}$$

تحلیل غیرتفصیلی

تحلیل الگوریتم‌های بازگشتی

الگوریتم بازگشتی، الگوریتمی است که برای حل یک مساله بازگشتی نوشته می‌شود و مساله بازگشتی مسأله‌ای است که برای حل آن، به حل همان مساله در ابعاد کوچک‌تر نیاز داریم. به‌عنوان مثال می‌توان تابع فاکتوریل را به‌صورت بازگشتی زیر تعریف کرد:

$$n! = n(n-۱)!$$

$$۰! = ۱$$

پایه بازگشت حالتی است که به‌ازای آن پاسخ مساله دقیقاً و صریحاً مشخص است. (به‌عنوان مثال ۰!=۱) هر مساله بازگشتی، حداقل دارای یک پایه بازگشت است.

مشخصات الگوریتم‌های بازگشتی

۱- کدهای کوتاه: معمولاً الگوریتم بازگشتی یک مساله، از الگوریتم غیربازگشتی آن، کوتاه‌تر است.

۲- اتلاف حافظه بیشتر: به‌علت استفاده از حافظه اضافی پشته، که مراحل بازگشت را در خود نگهداری می‌کند، اتلاف حافظه نسبت به حالت غیربازگشتی بیشتر است.

۳- سرعت اجرای کمتر: سرعت اجرای الگوریتم‌های بازگشتی نسبت به نسخه مشابه غیربازگشتی کمتر است.

۴- راحتی طراحی الگوریتم برخی از مسائل به‌صورت بازگشتی: مسائلی مانند فاکتوریل که روند حل آنها به‌صورت بازگشتی می‌باشد، به راحتی توسط یک الگوریتم بازگشتی، قابل حل می‌باشند. در ادامه الگوریتم بازگشتی برای محاسبه فاکتوریل یک عدد دلخواه، آورده شده است که می‌توانید آنها را با هم مقایسه کنید.

الگوریتم بازگشتی محاسبه فاکتوریل n

```
int Fact (int n)
{
if (n<۱)
return ۱:
else
return n*Fact(n-۱);
}
```

الگوریتم غیربازگشتی محاسبه فاکتوریل n

```
int Fact (int n)
{
int f=۱ , i:
for (i=۱; i<=n;i++)
f = f*i:
return f;
}
```

اصول برنامه‌نویسی بازگشتی

قضیه استقرای ریاضی

از این قضیه برای اثبات درستی یک رابطه به‌ازای ورودی n استفاده می‌شود. برای این کار مراحل زیر را انجام می‌دهیم.

اثبات پایه: درستی رابطه را به‌ازای کوچک‌ترین مقدار ورودی بررسی می‌کنیم.

فرض: فرض می‌کنیم رابطه به‌ازای $(n-۱)$ برقرار باشد.

حکم: با استفاده از فرض استقراء درستی حکم را ثابت می‌کنیم.

قضیه استقرای برنامه‌نویسی

پایه: همواره اولین دستور یک تابع بازگشتی، یک دستور شرطی است که مقدار پایه بازگشت را آزمون می‌کند و به‌ازای آن پاسخ مناسب را تولید می‌کند.

فرض: فرض می‌کنیم زیر برنامه نوشته شده تاکنون چنانچه با مقداری کوچک‌تر از n فراخوانی شود، پاسخ درست را تولید می‌کند.

حکم: با استفاده از فرض استقرا زیر برنامه را به‌گونه‌ای تکمیل می‌کنیم که پاسخ درست برگرداند.

مثال: تابع محاسبه فاکتوریل را می‌توان به‌صورت بازگشتی زیر نوشت:

```
function fact(n:integer) :integer;
{
 if n = ۱ or n = ۰ then return ۱;          {پایه بازگشت}
 x = fact(n - ۱);                          {فرض بازگشت}
 return x * n;                             {حکم بازگشت}
 }
```

مثال: تابع محاسبه جمله nام سری فیبوناتچی را می‌توان به‌صورت بازگشتی زیر نوشت:

```
function fibo(n:integer) :integer;
{
 if  n = ۱ or n = ۲ return ۱;
 x := fibo(n - ۱);                         {پایه بازگشت}
 y := fibo(n - ۲);                         {فرض بازگشت}
 return x + y;
 }                                         {حکم بازگشت}
```

برای تحلیل یک برنامه بازگشتی باید ابتدا یک رابطه بازگشتی متناظر با برنامه را به‌دست آورد؛ سپس آن را حل کرد. به‌عنوان مثال توابع بازگشتی متناظر با برنامه بازگشتی محاسبه فاکتوریل به‌صورت زیر است:

$$T(n)=\begin{cases}۱ & n=۰,۱\\ T(n-۱) & else\end{cases}$$

$$\begin{aligned}T(n)&=T(n-۱)+۱\\&=T(n-۲)+۱+۱\\&=T(n-۳)+۱+۱+۱\\&\vdots\\&=T(n-k)+k\end{aligned}$$

حال با قراردادن $k = n-1$ در رابطه فوق داریم:

$$T(1)+n-1=1+n-1=n$$

نکته: به‌طورکلی، چنانچه یک رابطه بازگشتی به‌صورت زیر موجود باشد، مرتبه آن از رابطه زیر به‌دست می‌آید.

$$T(n)=T(n-m)+O(g(n)) \Rightarrow T(n)=O(n*g(n))$$

نکته: چنانچه رابطه بازگشتی به شکل $T(n)=KT(n-m)+O(g(n))$ داشته باشیم آنگاه:

$$T(n)=O\left(K^{n/m}\right)$$

به‌عنوان مثال داریم:

$$T(n)=2T(n-1)+1=O(2^n)$$

قضیه اصلی (Master theorem)

رابطه بازگشتی زیر را درنظر بگیرید که در آن $\frac{n}{b}$ می‌تواند به فرم‌های $\left\lfloor \frac{n}{b} \right\rfloor$ یا $\left\lceil \frac{n}{b} \right\rceil$ باشد.

$$T(n)=aT\left(\frac{n}{b}\right)+f(n) \quad ,a\geq 1 \ ,b>1$$

در این صورت داریم:

$$1)\text{if } \exists \varepsilon>0;\ f(n)=O\left(n^{\log_b a-\varepsilon}\right)\Rightarrow T(n)=\theta\left(n^{\log_b a}\right)$$

$$2)\text{if} \quad f(n)=\theta\left(n^{\log_b^a}\right) \quad \Rightarrow \quad T(n)=\theta\left(n^{\log_b^a}\times \log n\right)$$

و در حالت کلی‌تر داریم:

$$\text{if} \quad f(n)=\theta\left(n^{\log_b^a}\times(\log n)^k\right) \quad \Rightarrow \quad T(n)=\theta\left(n^{\log_b^a}\times(\log n)^{k+1}\right)$$

$$3)\ \text{if } \exists\ \varepsilon>0; f(n)=\Omega\left(n^{\log_b\ a+\varepsilon}\right)\ ,af\left(\frac{n}{b}\right)<cf(n), c<1, n\geq 0 \Rightarrow T(n)=\theta(f(n))$$

✓ **تست: جواب معادله $T(n)=4T(\frac{n}{2})+n^2$ عبارتست از:**

۱) $\theta(n^2)$ ۲) $\theta(n\log n)$ ۳) $\theta(n^3)$ ۴) $\theta(n^2\log n)$

حل: گزینه «۴»

داریم $n^2=\theta(n^{\log_2 4})$ ؛ بنابراین در حالت دوم از قضیه اصلی هستیم؛ بنابراین $T(n)=\theta(n^2\log n)$

✓ **تست: جواب معادله $T(n)=4T(\frac{n}{2})+n^3$ عبارتست از:**

۱) $\theta(n^3)$ ۲) $\theta(n\log n)$ ۳) $\theta(n^2\log n)$ ۴) هیچ‌کدام

حل: گزینه «۱»

داریم $n^3=\Omega(n^{\log_2 4+\varepsilon})$ ؛ بنابراین در حالت سوم از قضیه اصلی هستیم، بنابراین $T(n)=\theta(n^3)$

✓ **تست: برای معادله $T(n)=4T(\frac{n}{2})+O(n^2)$ کدام گزینه صحیح‌تر است:**

۱) $T(n)=\theta(n^2\log n)$ ۲) $T(n)=O(n^2\lg n)$

۳) $T(n)=\Omega(n^2)$ ۴) موارد (۲) و (۳) صحیح است.

حل: گزینه «۴»

در اینجا در بدترین حالت می‌توان $f(n)$ را برابر تابع $n^۲$ درنظر گرفت که در این حالت داریم $n^۲ = \theta(n^{\log_۲ ۴})$ بنابراین در حالت دوم از قضیه اصلی هستیم، بنابراین $T(n) = O(n^۲ \log n)$ ولی در بهترین حالت می‌توان $f(n)$ را برابر تابع ثابت درنظر گرفت که در این صورت در حالت اول قضیه اصلی هستیم؛ بنابراین $T(n) = \Omega(n^۲)$

✓ **تست: برای معادله** $T(n) = ۴T(\frac{n}{۲}) + \theta(n^۲)$ **کدام گزینه صحیح است:**

۱) $\theta(n^۲ \lg n)$ ۲) $\theta(n^۲)$ ۳) $\theta(n \lg n)$ ۴) هیچکدام

حل: گزینه «۱»

در اینجا در بدترین و بهترین حالت می‌توان $f(n)$ را برابر تابعی با مرتبه $n^۲$ درنظر گرفت، که در این حالت داریم $n^۲ = \theta(n^{\log_۲ ۴})$ که در حالت دوم از قضیه اصلی هستیم؛ بنابراین $T(n) = \theta(n^۲ \log n)$

✓ **تست: جواب معادله** $T(n) = ۹T(\frac{n}{۳}) + n$ **عبارتست از:**

۱) $\theta(n^۴)$ ۲) $\theta(n^۲)$ ۳) $\theta(n \log n)$ ۴) $\theta(n)$

حل: گزینه «۲»

داریم $n = O(n^{\log_۲ ۹ - \varepsilon})$ که در حالت اول از قضیه اصلی هستیم؛ بنابراین $T(n) = \theta(n^{\log_۲ ۹}) = \theta(n^۲)$

✓ **تست: جواب معادله** $T(n) = T(\frac{۲n}{۳}) + ۱$ **عبارت است از:**

۱) $\theta(n \log n)$ ۲) $\theta(n^۲)$ ۳) $\theta(\log n)$ ۴) $\theta(n)$

حل: گزینه «۳»

داریم $۱ = \theta(n^{\log_{۳/۲} ۱}) = \theta(n^۰)$ که در حالت دوم از قضیه اصلی هستیم، بنابراین $T(n) = \theta(۱ \times \log n)$

✓ **تست: جواب معادله** $T(n) = ۳T(\frac{n}{۴}) + n \log n$ **عبارتست از:**

۱) $\theta(n^۲)$ ۲) $\theta(n \log n)$ ۳) $\theta(n^۳)$ ۴) $\theta(n)$

حل: گزینه «۲»

داریم $n \log n = \Omega(n^{\log_۴ ۳ + \varepsilon})$ که در حالت سوم از قضیه اصلی هستیم، بنابراین $T(n) = \theta(n \log n)$

تحلیل برنامه‌های بازگشتی

تحلیل تابع فیبوناتچی

$$T(n) = \begin{cases} ۱ & n = ۱ \text{ or } n = ۲ \\ T(n-۱) + T(n-۲) + ۱ \end{cases}$$

$$T(n-۱) > T(n-۲) \Rightarrow T(n) < T(n-۱) + T(n-۱) + ۱ \Rightarrow T(n) < ۲T(n-۱) + ۱$$

$$\Rightarrow T(n) < O\left(۲^n\right) \Rightarrow T(n) = O\left(۲^n\right)$$

از نظر ریاضی درست است ولی آیا $O\left(۲^n\right)$ کوچک‌ترین Order است؟ می‌توان مساله را به‌صورت زیر نیز حل نمود. $O(۲^n)$

$$T(n) > T(n-۲) + T(n-۲) + ۱ \Rightarrow T(n) > ۲T(n-۲) + ۱ \Rightarrow T(n) = ۲T(n-۲) + ۱$$

$$T(n) = O\left(۲^{\frac{n}{۲}}\right) \Rightarrow T(n) = \Omega\left(۲^{\frac{n}{۲}}\right)$$

مسائل بازگشتی کلاسیک

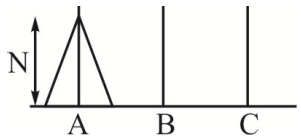
مساله برج هانوی

سه میله و n حلقه داریم که همگی روی یک میله قرار دارند. هدف انتقال حلقه‌ها به میله دیگر با رعایت ۳ شرط زیر است:

۱- تک به تک منتقل شود.

۲- در روند انتقال هیچ‌گاه نباید حلقه بزرگ‌تر روی حلقه کوچک‌تر قرار بگیرد.

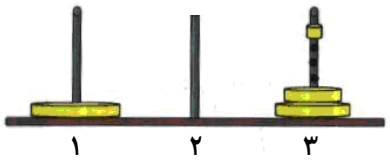
۳- در روند انتقال حلقه‌ها همواره روی میله‌ها قرار دارند.



برای حل این مساله به‌صورت بازگشتی از ایده زیر استفاده می‌کنیم. می‌خواهیم n مهره را از میله ۱ به میله ۲ منتقل کنیم برای این کار مراحل زیر را طی می‌کنیم:

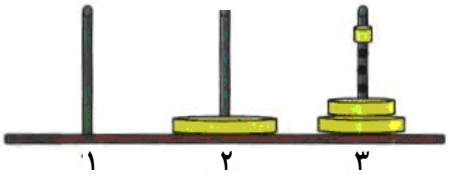
۱) انتقال $n-1$ دیسک بالایی از میله مبدا به میله کمکی

مرحله (۱)



۲) انتقال بزرگ‌ترین دیسک از میله مبدا به میله مقصد

مرحله (۲)



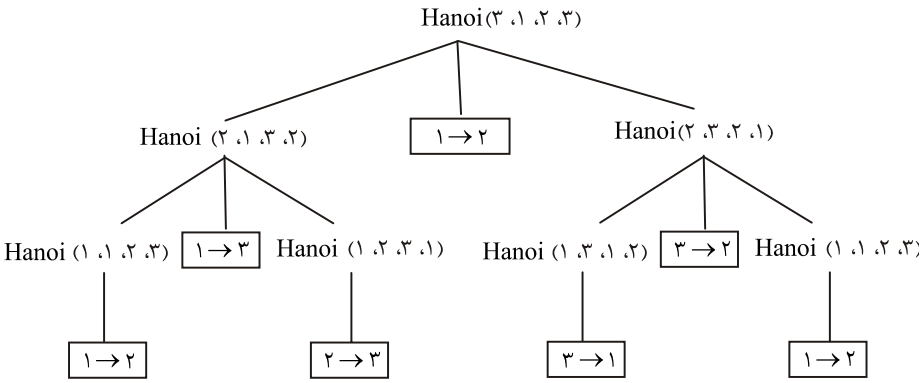
۳) انتقال $n-1$ دیسک باقیمانده از میله کمکی به میله مقصد

مرحله (۳)

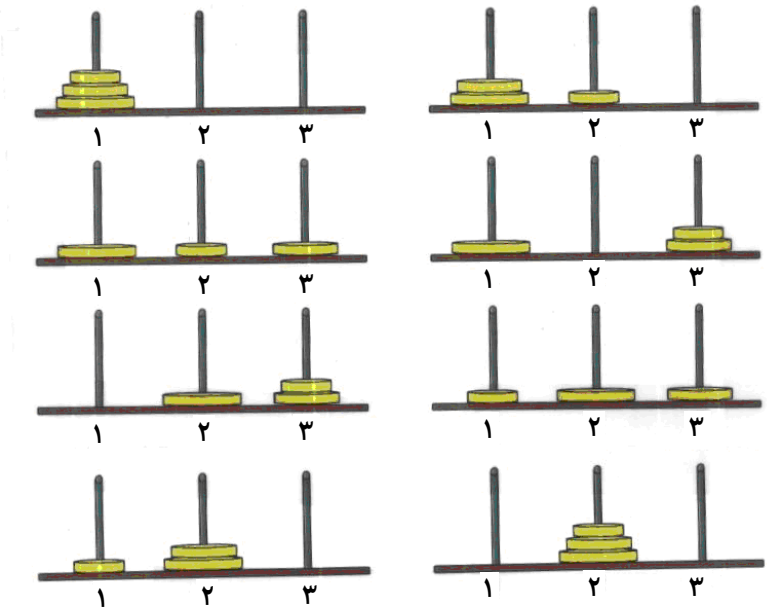
الگوریتم مساله برج هانوی به‌صورت زیر می‌باشد:

```
procedure Hanoi(n,f,t,h :integer);
begin
  if (n = ۱) then
        writeln(f,"→",t)
  else  begin
         Hanoi(n - ۱,f,h,t);
        write ln(f,"→",t);
         Hanoi(n - ۱,h,t,f)
   end
end;
```

درخت بازگشتی اجرای این الگوریتم به‌ازای Hanoi(۳,۱,۲,۳) به‌صورت زیر می‌باشد:



اگر بخواهیم سه مهره را از میله ۱ به کمک میله ۳ به میله ۲ منتقل کنیم باید دستور Honoi(۳,۱,۲,۳) را اجرا کنیم که حاصل این اجرا به‌صورت زیر خواهد بود:



تعداد جابه‌جایی‌ها در این روش از رابطه بازگشتی زیر به‌دست می‌آید:

$$T(n) = T(n-۱) + ۱ + T(n-۱)$$
$$T(۱) = ۱$$

با استفاده از روش جایگذاری می‌توان این معادله بازگشتی را به‌صورت زیر حل کرد:

$$T_{(n)} = ۲\left(۲\left(T(n-۲)+۱\right)\right)+۱ = ۲^۲T(n-۲)+۲+۱$$
$$= ۲^۳T(n-۳)+۲^۲+۲+۱$$
$$\vdots$$

حال با قراردادن $k = n-۱$ داریم:

$$= ۲^k T(n-k)+۲^{k-۱}+۲^{k-۲}+\ldots+۱$$
$$= ۲^{n-۱}T(۱)+۲^{n-۲}+۲^{n-۳}+\ldots+۱ \Rightarrow \frac{t_۱\left(۱-q^n\right)}{۱-q} = \frac{۱\left(۱-۲^n\right)}{۱-۲} = ۲^n - ۱ \Rightarrow T(n) = O(۲^n)$$

✓ **تست: مساله برج‌های هانوی را با ۳ میله A، B و C درنظر می‌گیریم. در اینجا هیچ حلقه‌ای را نمی‌توان مستقیماً از A به B یا از B به A منتقل کرد. یعنی چنین انتقال‌هایی تنها به کمک میله C انجام‌پذیرند. اگر در ابتدا، n حلقه در میله A داشته باشیم و $T(n)$ حداقل تعداد عملیات لازم برای انتقال n حلقه از A به B باشد، $T(n)$ با کدام رابطه مشخص می‌شود؟ فرض کنید شرایط مساله برج‌های هانوی تماماً برقرار است.**

۱) $T(n)=۶T(n-۱)+۳$ ۲) $T(n)=۳T(n-۱)+۲$

۳) $T(n)=T(n-۱)+T(n-۲)+۱$ ۴) $T(n)=T(n-۱)+T(n-۲)+۲$

حل: گزینه «۲»

این رویه بازگشتی در زیر آمده است:

```
Procedure nhanoi(n, A, B, C)
 {if n = ۱ then
   {move(A → C);
    move(C → B);
   }
  else
   {nhanoi(n - ۱, A, B, C);
    move(A → C);
    nhanoi(n - ۱, B, A, C);
    move(C → B) ;
    nhanoi(n - ۱, A, B, C);
   }
 }
```

تعیین مقدار یک تابع بازگشتی به‌ازای یک ورودی خاص

✓ **تست: تابع بازگشتی زیر را درنظر بگیرید.**

```
function Recursive (n)
{   if (n = ۱) then
    Recursive := ۱
    else
    Recursive := Recursive(n - ۱) + Recursive(n - ۱)
}
```

مقدار Recursive (۵) در کدام گزینه آمده است؟

۱) ۱۶ ۲) ۸ ۳) ۲۳ ۴) ۱۴

حل: گزینه «۱»

برای حل چنین مسائلی، بهترین راه استفاده از درخت بازگشت است. برای رسم این درخت به‌صورت بهینه موارد زیر را رعایت می‌کنیم.

۱) هر گره از درخت بازگشت را به ۲ قسمت تقسیم می‌کنیم. قسمت بالایی، ورودی‌ها و قسمت پایینی، خروجی‌های گره را نشان می‌دهد.

نکته: ریشه درخت بازگشت، به‌عنوان ورودی، ورودی مساله را می‌پذیرد و خروجی آن حل مساله خواهد بود.

۲) تعداد فرزندان هر گره با توجه به تعداد فراخوانی‌های بازگشتی موجود در برنامه تعیین می‌شود.

۳) ارتباط بین خروجی‌های فرزندان هر گره و خروجی گره را مشخص می‌کنیم.

۴) تمامی‌گره‌های موجود را با تکنیک فوق گسترش می‌دهیم، به‌شرطی‌که گرهی مشابه گرهی که می‌خواهیم گسترش دهیم، موجود نباشد. گره‌های مشابه، ورودی‌های مساوی دارند.
۵) گره‌هایی که ورودی‌شان با پایه بازگشت منطبق است گسترش داده نمی‌شوند.
۶) ساختار مطرح شده برای درخت بازگشت، به‌ازای تمامی‌خاصیت‌های مورد نظر از یک تابع بازگشتی، یکسان است و آنچه در خواص درخت متفاوت خواهد بود، ارتباط بین خروجی‌های فرزندان یک گره و مقادیر طرح شده به‌عنوان پایه بازگشت است.
۷) اگر اندازه ورودی بسیار بزرگ باشد، استفاده از درخت بازگشت توصیه نمی‌شود. در این حالت بهتر است یک رابطه بازگشتی دقیق به‌دست آورده و به‌دقت آن را حل کنیم.

این سوال را نیز می‌توان از طریق بازگشتی نیز حل نمود:

برای حل مساله بازگشتی از پشته کمک می‌گیریم:

۴) $R(۲)=R(۱)+R(۱)$
۳) $R(۳)=R(۲)+R(۲)$
۲) $R(۴)=R(۳)+R(۳)$
۱) $R(۵)=R(۴)+R(۴)$

حال از بالای پشته شروع می‌کنیم چون $R(۱)=۱$ است پس $R(۲)=۱+۱=۲$ می‌شود $R(۲)$ را از بالای پشته برداشته و مقدار آن را در محاسبه $R(۳)$ به‌کار می‌بریم و به‌همین ترتیب ..... تا محاسبه $R(۵)$ پیش می‌رویم و آن را محاسبه می‌کنیم که برابر است با $R(۵)=۸+۸=۱۶$

✓ **تست: در برنامه مقابل مقدار $F(۳,۶)$ چند است؟**

```
int  F(int m,int n)
{   if(m==۱||n==۱|m==n)
      return ۱
else
return   F(m-۱,n)+F(m-۱,n-۱)
 }
```

۱) ۲
۲) ۳
۳) ۸
۴) ۴

حل: گزینه «۴»

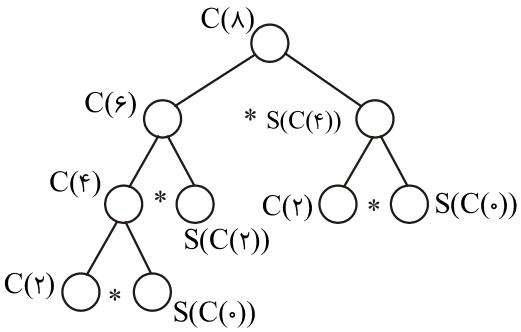
✓ **تست: در فراخوانی زیر برای $n=۸$، چند ضرب انجام می‌شود؟ (فرض کنید هر عمل Square یک ضرب نیاز دارد.)**

```
Function  count(n)
  if  n≤۰  then  return۱
  if  n=۱  then  return۲
  if  n=۲  then  return۳
  return(Count(n-۲)*square(count(n-۴)));
```

۱) ۴ ۲) ۸ ۳) ۱۰ ۴) ۹

حل: گزینه «۲»

درخت بازگشت به‌صورت زیر می‌باشد:



که نیاز به ۴ حزب می‌باشد. از طرفی به‌ازای هر $S(c(n))$ نیز یک ضرب نیاز است. پس: $۴+۴=۸$