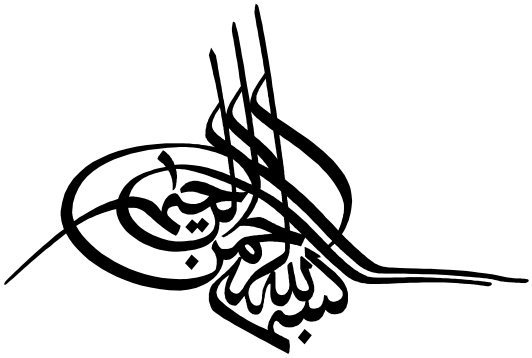


طراحی الگوریتم

سری کتاب‌های کمک آموزشی کارشناسی ارشد

مجموعه مهندسی کامپیوتر و فناوری اطلاعات
مؤلف: سمانه سرورنژاد

سرشناسه	: سرورنژاد، سمانه، ۱۳۶۷
عنوان	: طراحی الگوریتم
مشخصات نشر	: تهران : مشاوران صعود ماهان، ۱۴۰۱
مشخصات ظاهری	: ۲۴۱ ص
فروست	: سری ک تاب‌های کمک آموزشی کارشناسی ارشد
شابک	: ۹۷۸-۶۰۰-۴۵۸-۸۱۲-۶
وضعیت فهرست نویسی	: فیپای مختصر
یادداشت	: این مدرک در آدرس http://opac.nlai.ir قابل دسترسی است.
رده دیویی	: ۳۷۸/۱۶۶۴
شماره کتابشناسی ملی	: ۳۷۶۹۱۵۷

نام کتاب: .....طراحی الگوریتم

مولف: .....سمانه سرورنژاد

مدیر تولید محتوی: ..... سمیه بیگی

ناشر: .....مشاوران صعود ماهان

نوبت و تاریخ چاپ: .....اول/ ۱۴۰۱

تیراژ: ..... ۱۰۰۰ نسخه

قیمت: ۲/۴۹۰/۰۰۰ریال

شابک: ISBN:۹۷۸-۶۰۰-۴۵۸-۸۱۲-۶

انتشارات مشاوران صعود ماهان: خیابان ولیعصر، بالاتر از تقاطع مطهری،

روبروی قنادی هتل بزرگ تهران، 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 حاضر، پس از چندین بار ویرایش، برای ارائه به دوستان متقاضی کنکور کارشناسی ارشد رشته‌های حوزه کامپیوتر آماده گردیده است. این کتاب در ۸ فصل تنظیم و خدمت دانشجویان گرامی ارائه شده است. در برخی فصول از جمله فصل ۱، ۲،۳ و ۶ مطالب مفید و مهمی ارائه شده که توصیه می شود با دقت و درایت بیشتری بررسی و خوانده شود. در پایان هر فصل نیز سعی شده سوالات مربوط به سرفصل‌های همان فصل گنجانده شود. البته در فصل یک سوالات بیشتری وجود دارد که به خاطر اهمیت آنها و ارتباطشان با فصل‌های پیش‌رو، صلاح دیده شده که در این فصل قرار گیرند. امیدوارم این کتاب بتواند شما را در راه رسیدن به اهدافتان یاری کند.

از کلیه عزیزانی که در گردآوری این کتاب زحمت کشیده‌اند سپاسگزارم.

سمانه سرورنژاد

فهرست

عنوان | صفحه

فصل اول

الگوریتم‌ها و مرتبه اجرایی آنها و آشنایی با الگوریتم‌های بازگشتی

- نمادهای مجانبی
- چند جمله‌ای‌ها
- الگوریتم‌های بازگشتی
- برج‌های هانوی
- مراحل فراخوانی یک زیر برنامه
- روش‌های حل روابط بازگشتی

الگوریتم‌ها و مرتبه اجرایی آنها و آشنایی با الگوریتم‌های بازگشتی

نمادهای مجانبی (Asymptotic Notation) O و Ω و θ

این نمادهای مجانبی به‌صورت زیر تعریف می‌شوند:

$$O(g(n)) = \{f(n) \mid \exists\, c > 0,\ n_0 \in N \text{ s.t } 0 \le f(n) \le cg(n) \text{ as } n \ge n_0\}$$

بنابراین هرگاه داشته باشیم $f(n) = O(g(n))$ منظور این است که $f(n) \in O(g(n))$.

$$\Omega(g(n)) = \{f(n) \mid \exists\, c > 0,\ n_0 \in N \text{ s.t } 0 \le cg(n) \le f(n) \text{ as } n \ge n_0\}$$

به‌همین‌ترتیب $f(n) = \Omega((g(n))$ یعنی اینکه $f(n) = \in \Omega((g(n))$.

$$\theta(g(n)) = \{f(n) \mid \exists\, c_1, c_2 > 0,\ n_0 \in N \text{ s.t } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ as } n \ge n_0\}$$

به‌طور شهودی $f(n) = O(g(n))$ یعنی اینکه سرعت رشد f کمتر یا مساوی g است.

به‌طور شهودی $f(n) = \Omega(g(n))$ یعنی اینکه سرعت رشد f بیشتر یا مساوی g است.

به‌طور شهودی $f(n) = \theta(g(n))$ یعنی اینکه سرعت رشد f مساوی g است.

نکته:

$$f(n) = \theta(g(n)) \Leftrightarrow f(n) = O(g(n) \text{ And } f(n) = \Omega(g(n))$$

$$\omega(g(n) = \{f(n) \mid \forall c > 0, \exists\, n_0 \in N \text{ s.t } 0 \le cg(n) < f(n) \ \forall n \ge n_0\}$$

نمادهای مجانبی o ، ω

این نمادها به‌صورت زیر تعریف می‌شوند:

$$o(g(n)) = \{f(n) \mid \forall c > 0, \exists\, n_0 \in N \text{ s.t } \quad 0 \le f(n) < cg(n) \ \ \forall\ n \ge n_0\}$$

یعنی مقدار تابع f برای nها بزرگ در برابر g ناچیز و قابل صرف‌نظر کردن است.

نکته بسیار مهم: استفاده از حد برای تعیین مرتبه

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

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

$$f(n) = \theta(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = a \ne 0$$

با استفاده از حد می‌توان ثابت نمود که:

$$(1) \qquad \text{if } \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Leftrightarrow f(n) = O(g(n))$$

مثال: با توجه به رابطه (۱) داریم:

$$\frac{n^2}{2} \in O(n^3)$$

چون

$$\lim_{n\to\infty} \frac{\frac{n^2}{2}}{n^3} = \lim_{n\to\infty} \frac{1}{2n} = 0$$

$$(2) \qquad \text{if } \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Leftrightarrow f(n) = \omega(g(n))$$

مثال: با توجه به رابطه (۲) داریم:

$$300n^4 + 5n^2 \in 2n^5 + n$$

چون

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2n^5 + n}{300n^4 + 5n^2} = \infty$$

$$(3) \qquad \text{if } \lim_{n\to\infty} \frac{f(n)}{g(n)} = a \neq 0 \Leftrightarrow f(n) = \theta(g(n))$$

مثال: با توجه به رابطه (۳) داریم:

$$5n + 3\log n + 10n\log n + 7n^2 \in \theta(n^2)$$

چون

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{5n + 3\log n + 10n\log n + 7n^2}{n^2} = a \neq 0$$

نکته: قاعده هوپیتال، اگر $f(x)$ و $g(x)$ هر دو دیفرانسیل‌پذیر باشند و مشتقات آنها به‌ترتیب $f'(x)$ و $g'(x)$ باشد و اگر:

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

در آن‌صورت هرگاه حد سمت راست وجود داشته باشد:

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

برای توابعی با متغیرهای حقیقی نکته مذکور برقرار است، حال آنکه توابع پیچیدگی‌ها توابعی از اعداد صحیح هستند. ولی اکثر توابع پیچیدگی‌ها (برای مثال $\log n$ و n و غیره) توابع متغیرهای حقیقی نیز هستند. به علاوه، اگر تابع $f(x)$ تابعی از متغیر حقیقی x باشد، در آن‌صورت:

$$\lim_{n\to\infty} f(n) = \lim_{n\to\infty} f(x)$$

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

مثال:
$$3n^2 - 4 = \omega(2n) \ , \ 2n = o(n^2 + 1) \ , \ 4n^3 - 7n^2 + 10n^2 - 100 = \theta(n^3)$$

نکته: نمادهای مجانبی $\omega, o, \theta, \Omega, O$ برای مقایسه دو تابع به‌طور مجانبی (یعنی رفتار در بی‌نهایت) استفاده می‌شوند.

نکته: به جهت شباهتی که مقایسه مجانبی دو تابع با مقایسه دو مقدار عددی حقیقی وجود دارد می‌توان این شباهت‌ها را به فرم زیر نشان داد:

$$f(n) = O(g(n)) \approx a \leq b$$
$$f(n) = \Omega(g(n)) \approx a \geq b$$
$$f(n) = \theta(g(n)) \approx a = b$$
$$f(n) = o(g(n)) \approx a < b$$
$$f(n) = \omega(g(n)) \approx a > b$$

ویژگی‌های نمادهای مجانبی

۱- تعدّی (Transitivity)

$$f(n)=\theta(g(n)) \text{ and } g(n)=\theta(h(n)) \Rightarrow f(n)=\theta(h(n))$$
$$f(n)=O(g(n)) \text{ and } g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$$
$$f(n)=\Omega(g(n)) \text{ and } g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$$
$$f(n)=o(g(n)) \text{ and } g(n)=o(h(n)) \Rightarrow f(n)=o(h(n))$$
$$f(n)=\omega(g(n)) \text{ and } g(n)=\omega(h(n)) \Rightarrow f(n)=\omega(h(n))$$

بنابراین تمامی عملگرهای مجانبی خاصیت تعدی را دارا می‌باشند.

۲- انعکاس (Reflexivity)

$$f(n)=\theta(f(n)) \ , \ f(n)=O(f(n)) \ , \ f(n)=\Omega(f(n))$$

نمادهای مجانبی Ω,θ,O خاصیت انعکاسی را دارا می‌باشند.

۳- تقارنی (Symmetry)

این خاصیت فقط برای نماد θ به‌صورت زیر برقرار است:

$$f(n)=\theta(g(n)) \Leftrightarrow g(n)=\theta(f(n))$$

نکته:

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

نکته: اگر $f(n)$ و $g(n)$ توابع به‌طور مجانبی مثبتی باشند یعنی مقادیر این توابع به‌ازای nهای به حد کافی بزرگ مثبت باشند آنگاه داریم:

$$\max(f(n),g(n))=\theta(f(n)+g(n))$$

نکته: اگر a و b عضوی از اعداد حقیقی باشند و $b>0$ آنگاه داریم:

$$(n+a)^b=\theta(n^b)$$

نکته:

$$o(g(n))\cap\omega(g(n))=\varnothing$$

نکته: می‌توان نمادهای θ,Ω,O را به توابعی که دومتغیره هستند تعمیم داد. به‌عنوان مثال برای تابع دومتغیره $g(n,m)$ می‌توان $O(g(n,m))$ به‌صورت زیر تعریف می‌شود:

$$O(g(n,m))=\{f(n,m)\mid \exists\, c>0,n_0,m_0\in N,\ \text{s.t. } 0\le f(n,m)\le cg(n,m) \text{ as } n\ge n_0,m\ge m_0\}$$

به‌طور مشابه می‌توان این تعریف را برای θ و Ω نیز ارائه کرد.

تعاریف توابع نزولی صعودی به‌طور خلاصه به‌صورت زیر است:

تابع یکنوای صعودی: $\forall m,n; m\le n \Rightarrow f(m)\le f(n)$

تابع اکیدا صعودی: $\forall m,n; m<n \Rightarrow f(m)<f(n)$

تابع یکنوای نزولی: $\forall m,n; m\le n \Rightarrow f(m)\ge f(n)$

تابع اکیدا نزولی: $\forall m,n; m<n \Rightarrow f(m)>f(n)$

توابع floor و ceiling به‌صورت زیر تعریف می‌شوند:

اگر x عددی حقیقی باشد آنگاه $\lceil x\rceil$ سقف (Ceiling) مقدار x نامیده می‌شود و برابر کوچکترین عدد صحیح بزرگتر از x می‌باشد. به‌طور مثال: $\lceil 2/25\rceil=3$

$\lfloor x\rfloor$ را کف (Floor) مقدار x می‌نامیم و برابر با بزرگترین عدد صحیح کوچکتر از x می‌باشد، مثلاً: $\lfloor 2/25\rfloor=2$

نکته: برای هر عدد حقیقی x داریم: $x-1<\lfloor x\rfloor\le x\le\lceil x\rceil<x+1$

نکته: برای هر عدد صحیح n داریم: $\left\lfloor\frac{n}{2}\right\rfloor+\left\lceil\frac{n}{2}\right\rceil=n$

نکته: برای هر عدد حقیقی و مثبت مانند n و هر دو عدد صحیح و مثبت a و b داریم:

$$\left\lceil \left\lceil \frac{n}{a} \right\rceil / b \right\rceil = \lceil n / ab \rceil$$

$$\left\lfloor \left\lfloor \frac{n}{a} \right\rfloor / b \right\rfloor = \lfloor n / ab \rfloor$$

$$\left\lceil \frac{a}{b} \right\rceil \le (a + (b-1)) / b$$

$$\left\lfloor \frac{a}{b} \right\rfloor \ge ((a - (b-1)) / b$$

نکته: توابع floor ($\lfloor x \rfloor$) و ceiling ($\lceil x \rceil$) هر دو یکنوای صعودی می‌باشند.

محاسبات مانده‌ای (Modular Arithmetic)

برای هر عدد صحیح a و هر عدد صحیح و مثبت n داریم:

$$a \bmod n = a - \left\lfloor \frac{a}{n} \right\rfloor \times n = \text{باقیمانده تقسیم } a \text{ بر } n$$

(نماد $\equiv$) گوییم $a \equiv b \pmod n$ اگر و تنها اگر $a \bmod n = b \bmod n$
در این حالت گوییم a و b به پیمانه n هم‌ارز هستند.

چندجمله‌ای‌ها (Polynomials)

یک چندجمله‌ای برحسب n از درجه d عبارتست از تابعی به فرم $p(n) = \sum_{i=0}^{d} a_i n^i$ که در آن $a_d, \ldots, a_1, a_0$ را ضرایب چندجمله‌ای نامند و $a_d \neq 0$.
ضرایب چندجمله‌ای اعدادی ثابت می‌باشند.

مثال: $p(n) = 10n^4 + 7n - 300$ یک چندجمله‌ای از درجه 4 می‌باشد.

نکته:

$$\text{if} \quad p(n) = \sum_{i=0}^{d} a_i n^i \Rightarrow p(n) = \theta(n^d)$$
$$a_d \neq 0$$

تابع کران‌دار چندجمله‌ای: اگر تابع $f(n)$ به‌ازای برخی اعداد صحیح k به‌گونه‌ای باشد که $f(n) = O(n^k)$ گوینـد f کـران‌دار چندجمله‌ای می‌باشد.

توان‌ها

اگر $n, m, a > 0$ هر سه عدد حقیقی باشند آنگاه داریم:

1) $a^0 = 1$
2) $a^1 = a$
3) $a^{-1} = \frac{1}{a}$
4) $(a^m)^n = a^{mn}$
5) $(a^m)^n = (a^n)^m$
6) $a^m a^n = a^{m+n}$
7) $n^b = o(a^n) \Rightarrow \lim_{n\to\infty} \frac{n^b}{a^n} = 0 \quad ; \quad \forall n, a > 1, b \in R$

نکته: از آنجایی‌که به‌ازای هر $a>1$ داریم $n^b = o(a^n)$ می‌توان نتیجه گرفت که هر تابع نمایی با یک پایه اکیداً بزرگتر از یک دارای رشدی سریع‌تر از هر تابع چندجمله‌ای می‌باشد.

نکته: عدد نپر e پایه‌ای برای لگاریتم نپر: قضایای مربوط به عدد نپر e به‌صورت زیر می‌باشند:

1) $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$

2) $\forall x \in R : e^x \geq 1 + x$

3) if $|x| \leq 1 \Rightarrow 1 + x \leq e^x \leq 1 + x + x^2$

4) if $|x| > 1 \Rightarrow e^x = 1 + x + \theta(x^2)$

5) $\forall x \in R : \lim_{n\to\infty}(1 + \frac{x}{n})^n = e^x$

نکته: اگر a و b و c اعداد حقیقی و مثبت باشند، آنگاه خواص زیر برای لگاریتم‌ها صدق می‌کند:

1) $a = b^{\log_b a}$

2) $\log_c(ab) = \log_c a + \log_c b$

3) $\log_b a = \frac{\log_c a}{\log_c b}$

4) $\log_b(\frac{1}{a}) = -\log_b a$

5) $a^{\log_b c} = c^{\log_b a}$

6) $\log_b a = \frac{1}{\log_a b}$

نکته: برای هر ثابت $a>0$ و هر b دلخواه داریم: $\log^b n = o(n^a)$

تست: اگر T(n)=log(n!) آنگاه:

۱) $T(n)=O(n \log n)$ ۲) $T(n) = \Omega(n \log n)$

۳) $T(n) = \theta(n \log n)$ ۴) $T(n) = \theta(n \log^2 n)$

حل: گزینه «۳»

روش اول: تقریب استرلینگ یک حد قوی برای فاکتوریل به‌صورت زیر می‌باشد:

$$n! = \sqrt{2\pi n}(\frac{n}{e})^n e^{a_n} \qquad \frac{1}{12n+1} < a_n < \frac{1}{12n}$$

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

روش دوم: راه دیگر، یافتن کران‌های پایین و بالا می‌باشد.

$$\log(n!) = \log(n*(n-1)*(n-2)*....2*1) = \log n + \log(n-1) + \log(n-2) + \ldots \log(2) + \log(1)$$

$$= \sum_{i=1}^{n} \log i \leq \sum_{i=1}^{n} \log n = n\log n = O(n\log n)$$

$$\sum_{i=1}^{n} \log i \geq \sum_{i=\frac{n}{2}+1}^{n} \log(\frac{n}{2}+1) = \frac{n}{2}\log(\frac{n}{2}+1) \geq \frac{n}{2}\log(\frac{n}{2}) \geq \frac{n}{2}\log(\sqrt{n})(\text{for } 4 \leq n) = \frac{n}{2}\log(n^{\frac{1}{2}}) = \frac{n}{4}\log(n)$$

$$\Rightarrow \log(n!) = \Omega(n\log n)$$

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

✓ **تست: اگر** $T(n)=\sum_{i=1}^{\log n} i2^i$ **؛ آنگاه:**

۱) $T(n)=O(n\log n)$ ۲) $T(n)=\Omega(n\log n)$

۳) $T(n)=\theta(n\log n)$ ۴) $T(n)=q(n\log^2 n)$

حل: گزینه «۳»

روش اول: می‌دانیم که:

$$\sum_{i=1}^{n} i2^i=(n-1)2^{n+1}+2$$

بنابراین:

$$\sum_{i=1}^{\log n} i2^i=(\log n-1)2^{\log n+1}+2=(\log n-1)2^{\log n}*2^1+2=2(\log n-1)n+2=2(n\log n-n+1)$$

$$\Rightarrow \sum_{i=1}^{\log n} i2^i=\theta(n\log n)$$

روش دوم: راه دیگر، یافتن کران‌های پایین و بالا می‌باشد.

$$\sum_{i=1}^{\log n} i2^i\le\sum_{i=1}^{\log n}\log n2^i\le\log n\sum_{i=1}^{\log n}2^i=\log n(2^{\log n+1}-2)=\log n(2*2^{\log n}-2)=\log n(2n-2)\Rightarrow\sum_{i=1}^{\log n} i2^i=O(n\log n)$$

$$\sum_{i=1}^{\log n} i2^i\ge\sum_{i=\frac{\log n}{2}+1}^{\log n}(\frac{\log n}{2}+1)2^i\ge\sum_{i=\frac{\log n}{2}+1}^{\log n}(\frac{\log n}{2})2^i=(\frac{\log n}{2})\sum_{i=\frac{\log n}{2}+1}^{\log n}2^i=(\frac{\log n}{2})(\sum_{i=0}^{\log n}2^i-\sum_{i=0}^{\frac{\log n}{2}}2^i)$$

$$=(\frac{\log n}{2})(2^{\log n+1}-1-\left[2^{\frac{\log}{2}+1}-1\right])$$

$$(\frac{\log n}{2})(2^{\log n+1}-2^{\frac{\log n}{2}+1})=(\frac{\log n}{2})(2*2^{\log n}-2*2^{\frac{\log n}{2}})=\log n(2^{\log n}-2^{\frac{\log n}{2}})=\log n(n-\sqrt{n})=\Omega(n\log n)$$

$$\Rightarrow\sum_{i=1}^{\log n} i2^i=\theta(n\log n)$$

✓ **تست: اگر** $T(n)=\sum_{i=1}^{n}\frac{1}{i}$ **؛ آنگاه:**

۱) $T(n)=O(\log n)$ ۲) $T(n)=\Omega(\log n)$

۳) $T(n)=\theta(\log n)$ ۴) $T(n)=\theta(\log^2 n)$

حل: گزینه «۳»

روش اول: می‌دانیم که:

$$1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}\approx\ln n$$

بنابراین داریم:

$$T(n)=\sum_{i=1}^{n}\frac{1}{i}\approx\ln n\Rightarrow T(n)=\theta(\ln n)=\theta(\log n)$$

روش دوم: برای تابع نزولی یکنواخت $F(i)$ به‌صورت زیر محدود می‌شود:

$$\int_{m}^{n+1} f(x)dx\le\sum_{i=m}^{n} f(k)\le\int_{m-1}^{n} f(x)dx$$

از طرفی:

$$\sum_{i=1}^{n}\frac{1}{i}\geq\int_{1}^{n+1}\frac{dx}{x}=\ln(n+1)-\ln 1=\ln(n+1)\quad,\quad\sum_{i=1}^{n}\frac{1}{i}=1+\sum_{i=2}^{n}\frac{1}{i}\leq 1+\int_{1}^{n}\frac{dx}{x}=1+\ln(n)$$

$$\sum_{i=1}^{n}\frac{1}{i}=\theta(\ln n)=\theta(\log n)$$

✓ **تست: اگر** $T(n)=\sum_{i=1}^{n}\log^{k} i$ **آنگاه:**

۱) $T(n)=\theta(n\log^{k} n)$

۲) $T(n)=\theta(n\log\ (k+n))$

۳) $T(n)=\theta(\log^{k} n)$

۴) $T(n)=\theta(n^{k})$

حل: گزینه «۱»

می‌دانیم که:

$$\int_{1}^{m}\log^{k}(x)dx=\sum_{i=0}^{k}m\log^{k-i}(m)\frac{k!}{(k-i)!}(-1)^{i}$$

از آنجا که $\log^{k}(i)$ تابع صعودی و یکنواخت است؛ پس بنابر عکس مطلب تست قبل داریم:

$$\sum_{i=1}^{n}\log^{k} i\leq\int_{1}^{n+1}\log^{k}(x)dx=\sum_{i=0}^{k}(n+1)\log^{k-i}(n+1)\frac{k!}{(k-i)!}(-1)^{i}=O(n\log^{k}(n))$$

$$\sum_{i=1}^{n}\log^{k} i=\log^{k} 1+\sum_{i=2}^{n}\log^{k} i\geq\int_{1}^{n}\log^{k} i=\sum_{i=0}^{k}(n)\log^{k-i}(n)\frac{k!}{(k-i)!}(-1)^{i}=\Omega(n\log^{k}(n))$$

بنابراین داریم:

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

مثال: اگر $f_1(n)=O(g_1(n))$ **و** $f_2(n)=O(g_2(n))$ **آنگاه درستی یا نادرستی رابطه زیر را بررسی نمایید.**

$$f_1(n)^{f_2(n)}=O(g_1(n)^{g_2(n)})$$

حل: رابطه فوق نادرست است. به‌عنوان مثال نقض داریم:

$$g_1(n)=0.5\ ,\ g_2(n)=n\ ,\ f_1(n)=2\ ,\ f_2(n)=n$$

در این‌صورت داریم:

$$2^{n}\neq O((0.5)^{n})$$

مثال: درستی یا نادرستی گزاره زیر را بررسی نمایید.

" به‌ازای هر دو تابع دلخواه f و g یکی از روابط $f(n)=\Omega(g(n))\ or\ f(n)=O(g(n))\ or\ f(n)=\theta(g(n))$ برقرار است."

حل: گزاره فوق نادرست است. به‌عنوان مثال نقض داریم:

$$f(n)=n\quad و\quad g(n)=n^{1+\sin n}$$

مثال: توابع زیر را برحسب رشد مرتب نمایید.

$(\sqrt{2})^{\log n}$	n^{2}	$n!$	e^{n}	2^{n}
n^{3}	$\log(n!)$	$n^{\frac{1}{\log n}}$	n	$n^{\log\log n}$
$\ln n$	$\ln\ln n$	$3^{\log n}$	$(\log n)^{\log n}$	$n\log n$
$(\log n)!$	$(2^{2})^{n}$	$n2^{n}$	$2^{\log n}$	$2^{\sqrt{2\log n}}$
$(2^{2})^{n+1}$	$\sqrt{\log n}$	$(n+1)!$	$4^{\log n}$	$(\frac{3}{2})^{n}$

حل:

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

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

$4^{\log n} = n^2 \quad , \quad 2^{\log n} = n$

$n^{\frac{1}{\log n}} = n^{\frac{\log 2}{\log n}} = (n^{\log 2})^{\frac{1}{\log n}} = (2^{\log n})^{\frac{1}{\log n}} = 2$

$(\sqrt{2})^{\log n} = n^{\log\sqrt{2}} = n^{\log 2^{\frac{1}{2}}} = n^{\frac{1}{2}\log 2} = n^{\frac{1}{2}} = \sqrt{n}$

$e^n = (2.71)^n$

$\log(\sqrt{2})^{\log n} = \log(2^{\frac{1}{2}\log n}) = \frac{1}{2}\log n * \log 2 = \frac{1}{2}\log n$

$\log 2^{\sqrt{2\log n}} = \sqrt{2\log n}$

$\frac{1}{2}\log n = \omega\sqrt{2\log n}$

$\Rightarrow (\sqrt{2})^{\log n} = \omega(2^{\sqrt{2\log n}})$

$\log\log^2 n = \log(\log n)^2 = 2\log(\log n)$

$\log 2^{\sqrt{2\log n}} = \sqrt{2\log n}$

$2^{\sqrt{2\log n}} \geq \log^2 n \Rightarrow \log 2^{\sqrt{2\log n}} \geq \log\log^2 n$

$\sqrt{2\log n} = \omega(2\log(\log n))$

$\Rightarrow 2^{\sqrt{2\log n}} = \omega(2^{2\log(\log n)}) = \omega((\log n)^{2\log 2}) = \omega(\log^2 n)$

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

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

با توجه به توضیحات فوق ترتیب توابع داده‌شده به‌صورت زیر می‌باشد:

$$(2^2)^{n+1} > (2^2)^n > (n+1)! > n! > e^n > n2^n > 2^n > (\frac{3}{2})^n > n^{\log\log n} = (\log n)^{\log n} > \log(n!) > n^3 > 4^{\log n}$$

$$> \log(n!) = n\log n > 3^{\log n} > 2^{\log n} = n > (\sqrt{2})^{\log n} = \sqrt{n} > 2^{\sqrt{2\log n}} > \ln n > \sqrt{\log n} > \ln\ln n > n^{\frac{1}{\log n}}$$

نکته: نمادهای مجانبی تابع $n!$ ($n! = 1\times 2\times \ldots \times n$) به‌صورت زیر می‌باشند:

1) $n! = o(n^n)$

2) $n! = \omega(2^n)$

3) $\lg n! = \theta(n\lg n)$

4) $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\theta(\frac{1}{n}))$

فرمول استرلینگ:

5) $n! = \sqrt{2\pi n}(\frac{n}{e})^n e^{\alpha_n}, \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$

نکته: اگر $f(n)$ و $g(n)$ توابع یکنوای صعودی باشند؛ آنگاه توابع $f(n)+g(n)$ و $f(g(n))$ نیز چنین هستند.

نکته: اگر $f(n)$ و $g(n)$ توابع یکنوای صعودی و غیرمنفی باشند؛ آنگاه تابع $f(n).g(n)$ نیز یکنوای صعودی می‌باشد.

ویژگی‌های نماد جمع‌بندی سیگما

1) $\sum(ca_k + b_k) = c\sum a_k + \sum b_k$

2) $\sum\theta(f(k)) = \theta(\sum f(k))$

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

4) $\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} = \theta(n^3)$

5) $\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4} = \theta(n^4)$

6) $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$ (Geometric Series)

7) if $|x|<1 \Rightarrow \sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$

8) $H_n = 1+\frac{1}{2}+\ldots+\frac{1}{n} = \sum_{k=1}^{n}\frac{1}{k} = \ln n + o(1)$ (Harmonic Series)

9) if $|x|<1 \Rightarrow \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$ (Integration & Differentiating Series)

10) $\sum_{k=1}^{n}(a_k - a_{k-1}) = a_n - a_0$ (Telescoping Series)

11) $\sum_{k=0}^{n-1}(a_k - a_{k+1}) = a_0 - a_n$ (Telescoping Series)

12) $\sum_{k=1}^{n-1}\frac{1}{k(k+1)} = \sum_{k=1}^{n-1}(\frac{1}{k}-\frac{1}{k+1}) = 1-\frac{1}{n}$

✓ **تست: کدام‌یک ازگزینه‌های زیر صحیح است؟**

۱) $f(n)+g(n) = O(\min(f(n),g(n)))$ ۲) $f(n) = O((f(n))^2)$

۳) $f(n)+O(f(n)) = O((f(n)))$ ۴) هیچ‌کدام

حل: گزینه «۳»

گزینه ۱ نادرست است. به‌عنوان مثال نقض $f(n)=n$ و $g(n)=n^2$ را درنظر بگیرید.

گزینه ۲ نادرست است. به‌عنوان مثال نقض $f(n)=1/n$ را درنظر بگیرید.

گزینه ۳ صحیح است. زیرا:

$$g(n) = O(f(n)) \Rightarrow 0 \le g(n) < cf(n)$$
$$\Rightarrow f(n) \le f(n)+g(n) < (c+1)f(n)$$
$$\Rightarrow f(n) \le f(n)+O(f(n)) < (c+1)f(n)$$
$$\Rightarrow f(n)+O(f(n)) \in f(n)$$

مثال: فرض کنید $f_i(n)$ دنباله‌ای از توابع باشد به‌طوری‌که برای هر i داشته باشیم: $f_i(n)=O(n)$. اگر $g(n)=\sum_{i=1}^{n} f_i(n)$، **آیا** $g(n)=O(n^2)$ **است یا خیر؟**

حل: خیر. به‌عنوان مثال نقض فرض کنید $f_i(n) = i*n$، در این صورت بدیهی است که برای هر i داریم: $f_i(n)=O(n)$ اما:

$$g(n) = \sum_{i=1}^{n} i*n = n\sum_{i=1}^{n} i = n(\frac{n(n+1)}{2}) = O(n^3)$$

نکته: معادلات زمانی الگوریتم‌های بازگشتی، توابعی بازگشتی می‌باشند. به‌عنوان مثال تابع بازگشتی زیر را درنظر بگیرید:

```
int P(int *A, int n){
  if (n<0)
     return 0
  else
     return (A[n]+P(A,n -1));
}
```

مرتبه زمانی $T(n)$ این رویه برحسب n به‌صورت زیر می‌باشد:

$$T(n)=\begin{cases} o(1) & n<0 \\ T(n-1)+o(1) \end{cases}$$

بنابراین $T(n)=\theta(n)$.

نکته بسیار مهم:

1) $\lim\limits_{n\to\infty}\frac{f(n)}{g(n)}=a\in R^{+}\Rightarrow f(n)=\theta(g(n))$

2) $\lim\limits_{n\to\infty}\frac{f(n)}{g(n)}=0\Rightarrow f(n)=O(g(n))$

3) $\lim\limits_{n\to\infty}\frac{f(n)}{g(n)}=\infty\Rightarrow f(n)=\Omega(g(n))$

اما توجه داشته باشید که عکس این قضیه صحیح نمی‌باشد، یعنی ممکن است $f(n)=O(g(n))$ باشد بدون اینکه حد موردنظر $(\lim\limits_{n\to\infty} f(n)/g(n))$ موجود باشد.

برای به‌دست آوردن حد توابع می‌توان از قاعده هوپیتال استفاده کرد.

مثال: $\log n\in O(\sqrt{n})$ ، **زیرا داریم:**

$$\lim_{n\to\infty}\frac{\log_2 n}{\sqrt{n}}\overset{hop}{=}\lim_{n\to\infty}\frac{\frac{1}{n}}{\frac{1}{2}n^{-\frac{1}{2}}}$$

اما $\sqrt{n}\notin O(\log n)$ ، زیرا عکس این حد وجود ندارد.

تحلیل الگوریتم مرتب‌سازی حبابی

ورودی: آرایه نامرتب $A[1...n]$.

خروجی: این آرایه مرتب‌شده به‌صورت صعودی.

```
Algorithm Bubble Sort (A,n)
1 for   i=1  to  n-1  do
2       for   j=n  down to i+1 do
3        if  A[j-1]>A[j] then
         {
4             temp=A[j-1]
5             A[j-1]=A[j]
6             A[j]=temp
          }
       }
}
```

بررسی بدترین حالت: بدترین حالت زمانی اتفاق می‌افتد که فرمان خط ۳ همواره درست باشد. این مساله عملا زمانی اتفاق می‌افتد که خود آرایه $A[1...n]$ به‌صورت نزولی مرتب شده باشد، دراین‌صورت داریم:

شماره دستورالعمل	تعداد دفعات اجرا
1	$n-1$
2	$(n-1)+(n-2)+...+2+1$
3	$(n-1)+(n-2)+...+2+1$
4	$(n-1)+(n-2)+...+2+1$
5	$(n-1)+(n-2)+...+2+1$
6	$(n-1)+(n-2)+...+2+1$
$t(n)=n-1+5\frac{n(n-1)}{2}$	⇐ جمع

بنابراین $t(n)=O(n^2)$.

بررسی بهترین حالت: بهترین حالت زمانی اتفاق می‌افتد که دستور خط ۳ هیچ‌وقت اجرا نشود و این زمانی اتفاق می‌افتد که خود آرایه به‌صورت صعودی مرتب شده باشد، در این حالت داریم:

شماره دستورالعمل	تعداد دفعات اجرا
1	$n-1$
2	$(n-1)+(n-2)+...+2+1$
3	$(n-1)+(n-2)+...+2+1$
4	0
5	0
6	0
$t(n)=n-1+2\frac{n(n-1)}{2}=\Omega(n^2)$	جمع

لذا نتیجه می‌شود که $t(n)=\Omega(n^2)$.

تحلیل الگوریتم مرتب‌سازی درجی

ورودی: آرایه نامرتب $A[1...n]$.

خروجی: این آرایه مرتب‌شده به‌صورت صعودی.

```
Algorithm Insertion sort (A,n)
    A[0] = -∞   / برای سهولت /
    for   j = 2   to   n  do
    {
        Item = A[j]  ,  i = j - 1
        While  item < A[i]*  do
        {
            A[i+1] = A[i]    ,   i = i - 1
        }
        A[i+1] = item
    }
```

برای سهولت در تحلیل این الگوریتم، فقط تعداد دفعات اجرای دستورالعمل مقایسه‌ای را حساب می‌کنیم.

بررسی بدترین حالت: بدترین حالت زمانی اتفاق می‌افتد که دستور شرطی همواره درست باشد و این مساله عملا زمانی اتفاق می‌افتد که آرایه $A[1...n]$ به‌صورت نزولی مرتب شده باشد.

تعـداد دفعـات اجـرای دسـتورالعمل	مقـدار پـارامتر j
2	1
3	2
⋮	⋮
n	n

جمع $t(n) = 2+3+...+n = \frac{n(n+7)}{2} - 1 = O(n^2)$

بررسی بهترین حالت: بهترین حالت زمانی اتفاق می‌افتد که حلقه داخلی هیچ وقت اجرا نشود (آرایه به‌صورت صعودی مرتب شده باشد)

تعـداد دفعـات اجـرای دسـتورالعمل	مقـدار پـارامتر j
1	2
1	3
1	⋮
n	n

جمع $t(n) = 1+1+...+1 = n-1 = \Omega(n)$

با توجه به مطالب فوق نتیجه می‌شود که میانگین زمان اجرای الگوریتم مرتب‌سازی درجی $O(n)$، $O(n\log n)$ یا $O(n^2)$ است که اثبات می‌شود برابر $\theta(n^2)$ است.

الگوریتم‌های بازگشتی

به‌طورکلی اگر یک الگوریتم در بدنه خود، خود را فراخوانی نماید، الگوریتم را بازگشتی می‌نامند.

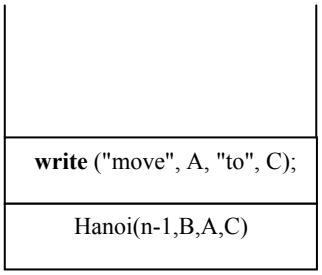
برج‌های هانوی

برنامه بازگشتی برج هانوی به‌صورت زیر می‌باشد:

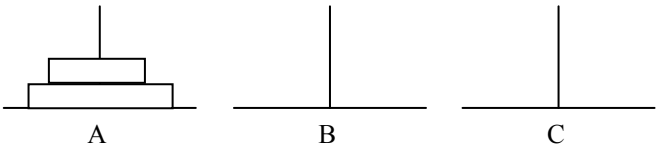
```
Procedure Hanoi (n,A,B,C)
Begin
        If n = 1 then write ("move", A, "to", C);
        Else Begin
                        Hanoi(n – 1,A,C,B);
                        write ("move", A, "to", C);
                        Hanoi(n – 1,B,A,C)
                End;
End;
```

در مسأله برج هانوی هدف این است که n دیسک را از میله A به کمک میله B به میله C منتقل کنیم. با این شرط که در هر بار تنها می‌توان یک دیسک را منتقل نمود و دیسک بزرگتر نمی‌تواند روی دیسک کوچک‌تر قرار گیرد.
به‌عنوان مثال با فراخوانی hanoi(2,A,B,C) می‌توان دو مهره را از میله A به میله C به‌صورت زیر منتقل کرد.

با توجه به آنکه مقدار n برابر ۲ است اجرای برنامه وارد قسمت Else می‌شود و بایستی سه دستور آن قسمت را اجرا کنیم. از آنجایی‌که دستور Hanoi(1,A,C,B) به‌صورت بازگشتی رویه را فراخوانی می‌کند، بنابراین دو دستور بعد از آن به‌صورت زیر وارد پشته سیستم می‌شود.

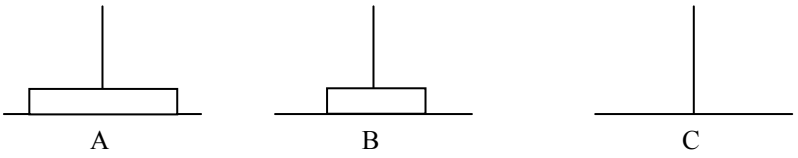


با اجرای دستور Hanoi(1,A,C,B) از آنجایی‌که n برابر یک می‌باشد عبارت move A to B چاپ می‌شود. حال به سراغ پشته سیستم می‌رویم و دستور **write** ("move", A, "to", C); اجرا می‌شود که حاصل آن نوشته شدن عبارت move A to C می‌باشد. حال مجددا به سراغ پشته سیستم می‌رویم و دستور Hanoi(n-1,B,A,C) را اجرا می‌کنیم. از آنجایی‌که مقدار n برابر یک می‌باشد با اجرا شدن این دستور عبارت move B to C چاپ می‌شود.

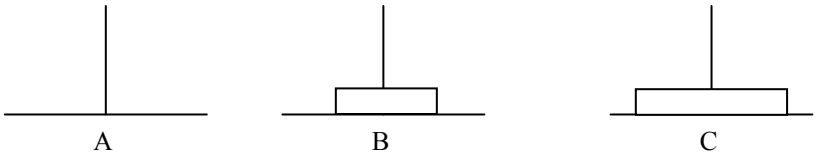


بنابراین به‌ترتیب عبارت‌های زیر چاپ می‌شوند:

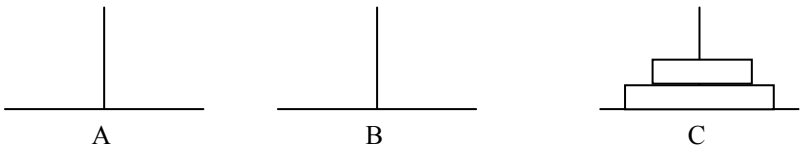
move A to B



move A to C

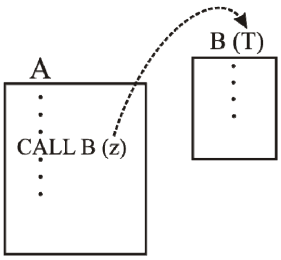


move B to C



مراحل فراخوانی یک زیربرنامه (CALL)

در هر فراخوانی یک زیربرنامه، مراحل زیر انجام می‌شود:
۱) انتقال متغیرهای محلی، پارامترها و آدرس بازگشت داخل حافظه پشته (stack).
منظور از آدرس بازگشت (return address) آدرس اولین دستور بعد از CALL می‌باشد.
متغیرهای محلی (local variables) متغیرهایی هستند که داخل یک بلوک خاص تعریف شده‌اند و هنگامی‌که کنترل اجرای یک برنامه وارد یک بلوک گردید، حافظه به آنها تخصیص می‌یابد.
حافظه stack از محل سگمنت پشته (stack segment) به یک برنامه یا رویه بازگشتی اختصاص می‌یابد. پارامترها همان مقادیری است که در آرگمان‌های رویه قرار می‌گیرد.
در هر CALL علاوه بر مقادیر ذکرشده در بالا برخی مقادیر سیستمی مانند ثبات Flags و... داخل پشته قرار می‌گیرند.
در هر CALL یک قاب پشته‌ای یا stack-frame حاوی مقادیر ذکرشده در بالا در stack قرار می‌گیرد.
۲) جانشینی پارامترها (parameter substituation)، انتقال نظیر به نظیر مقادیر پارامترها به داخل آرگومان‌ها، به این مرحله اصطلاحاً parameter passing گویند.
۳) رجوع به آدرس رویه فراخوانده‌شده



عملکرد دستور Return

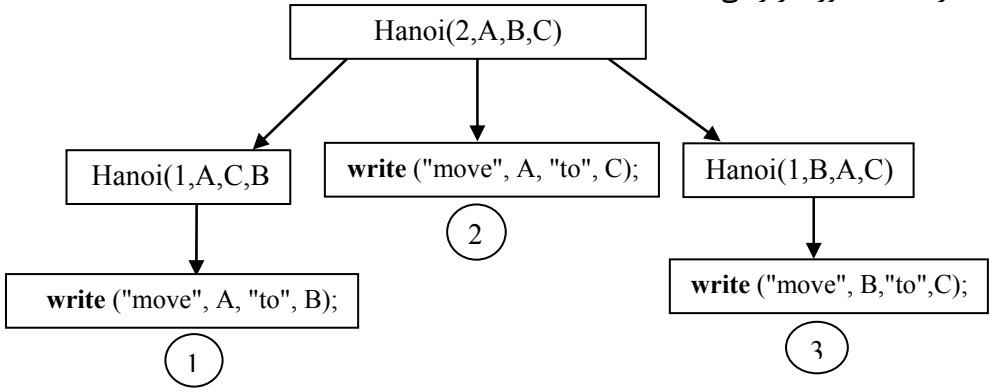
در هر رویه باید یک یا چند دستور return به منظور بازگشت از رویه به برنامه فراخواننده (caller) وجود داشته باشد که این دستور باعث می‌شود که:
۱) مقادیر پارامترها و متغیرهای محلی و غیره از stack بازیابی شوند.
۲) رجوع به آدرس بازگشت (return address) انجام شود.

درخت فراخوانی‌های بازگشتی (Recursive Call Tree)

توسط این درخت که هر رأس آن نشان‌دهنده یک CALL می‌باشد می‌توان عملکرد یک رویه بازگشتی را بررسی کرد.
تعداد فرزندان هر گره می‌تواند نشان‌دهنده تعداد آدرس‌های بازگشت در هر CALL باشد، یعنی اینکه مثلاً اگر در یک رویه تعداد فراخوانی‌های بازگشتی در هر مرحله دو تا باشد، آنگاه درخت فراخوانی‌های بازگشتی یک درخت دودویی خواهد بود.

مثال: درخت فراخوانی را برای Hanoi(2,A,B,C) رسم کنید.

حل: درخت بازگشت به‌صورت زیر می‌باشد:



تحلیل زمانی الگوریتم هانوی

از آنجایی‌که در الگوریتم هانوی به‌ازای اندازه ورودی n دو بار خود را با مقدار $n-1$ به‌صورت بازگشتی فراخوانی می‌کند و یک جابجایی انجام می‌دهد، می‌توان پیچیدگی زمانی آن را به‌صورت تابع بازگشتی زیر نوشت:

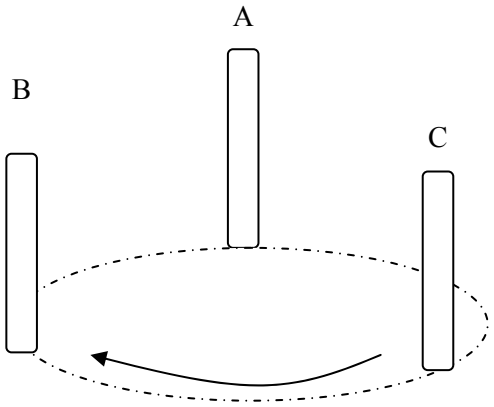
$$T(n)=\begin{cases}\theta(1) & n=1\\ 2T(n-1)+\theta(1) & n>1\end{cases}$$

اگر $n=1$ آنگاه زمان الگوریتم مقداری ثابت و از مرتبه $\theta(1)$ است اما اگر $n>1$ آنگاه الگوریتم دارای دو فراخوانی بازگشتی با $n-1$ می‌باشد، که زمان هر کدام $T(n-1)$ می‌باشند و زمان ثابت $\theta(1)$ برای مقایسه n با 1 و عمل Move و عمل return صرف می‌شود.

الگوریتم غیربازگشتی برج هانوی

اگر n (تعداد دیسک) زوج باشد جهت حرکت را ساعتگرد و در غیراین‌صورت پادساعتگرد در نظر می‌گیریم. حال مراحل زیر را تا هنگامی‌که همه دیسک‌ها در میله مقصد قرار بگیرد تکرار می‌کنیم.

- کوچکترین دیسک را به میله بعدی در جهت حرکت منتقل کنید.
- در صورت امکان تنها حرکتی را که شامل کوچکترین دیسک نباشد انجام بدهید.



نکته: تعداد جابجایی این الگوریتم برابر تعداد جابجایی‌های روش بازگشتی می‌باشد. برای اثبات داریم:

پایه استقرا: $n=1$: روشن است و این کار با یک حرکت انجام می‌شود.

فرض استقرا: برای $n-1$ دیسک درست کار می‌کند. یعنی $n-1$ دیسک به‌صورت مستقل و با $2^{n-1}-1$ حرکت از میله 1 به میله بعدی در جهت ساعتگرد (برای $n-1$ زوج) یا پادساعتگرد (برای $n-1$ فرد) به‌درستی و مطابق مقررات مسأله منتقل می‌کند.

گام استقرا: فرض کنید n زوج است. این الگوریتم مطابق فرض استقرا پس از $2^{n-1}-1$ حرکت $n-1$ دیسک بالای میله را به میله بعدی در جهت ساعتگرد (یعنی میله ۲) منتقل می‌کند. در این حرکت‌ها بزرگ‌ترین دیسک نقشی ندارد و در انتها این دیسک در میله ۱ باقی می‌ماند. سپس تنها حرکت ممکن را انجام می‌دهیم که بزرگترین دیسک را از میله 1 به میله 3 می‌بریم سپس در جهت حرکت که ساعتگرد است کاری که الگوریتم به‌صورت استقرایی انجام می‌دهد. حرکت $n-1$ دیسک موجود در میله ۲ به میله بعدی (میله ۳) است و این کار با فرض استقرا پس از $2^{n-1}-1$ حرکت دیگر به‌درستی انجام می‌شود. مشابه این استدلال برای اعداد فرد هم برقرار است. جمع کل $1+2(2^{n-1}-1) = 2^{n-1}-1$ است. اگر طبق استقرا تعداد حرکت‌ها زیر مسأله‌ها بهینه است. پس تعداد کل حرکت‌های این الگوریتم بهینه است.

روش‌های حل روابط بازگشتی

۱) روش درخت بازگشت

مراحل درخت بازگشت عبارتند از:

۱- ساخت درخت بازگشت

۲- تعیین ارتفاع درخت بازگشت برحسب اندازه ورودی

۳- تعیین هزینه هر سطح از درخت بازگشت

۴- محاسبه مجموع هزینه‌های تمامی سطوح

۵- اضافه کردن تعداد برگ‌ها (هر کدام با هزینه $\underline{O(1)}$) به هزینه به‌دست آمده در گام قبل.

نکته: در اکثر مواقع مقدار محاسباتی که به کمک درخت بازگشتی حاصل می‌شود دقیق نمی‌باشد و برای محاسبه دقیق آن از استقرای قوی ریاضی یا روش جایگزینی استفاده می‌شود.

نکته: اگر درخت بازگشتی کامل باشد، می‌توان متوسط زمان اجرا را به‌دست آورد ولی اگر درخت بازگشتی کامل نباشد فقط می‌توان big-O یا امگا را محاسبه نمود.

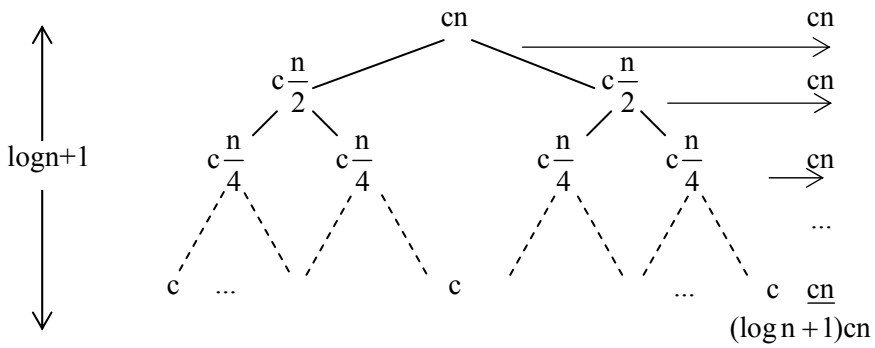
✓ **تست: مرتبه زمانی تابع زیر کدام است؟**

$$T(n)=\begin{cases} c & n=1 \\ 2T(\frac{n}{2})+cn & n>1 \end{cases}$$

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

حل: گزینه «۱»

درخت فراخوانی بازگشتی به‌صورت زیر می‌باشد:



بنابراین داریم:

$$T(n)=(\log n+1)cn=cn\log n+cn=O(n\ \log n)$$

✓ **تست: مرتبه زمانی رابطه بازگشتی زیر کدام است؟**

$$T(n)=\begin{cases} 1 & n=1 \\ 3T(\left\lfloor\frac{n}{4}\right\rfloor)+n^2 & n>1 \end{cases}$$

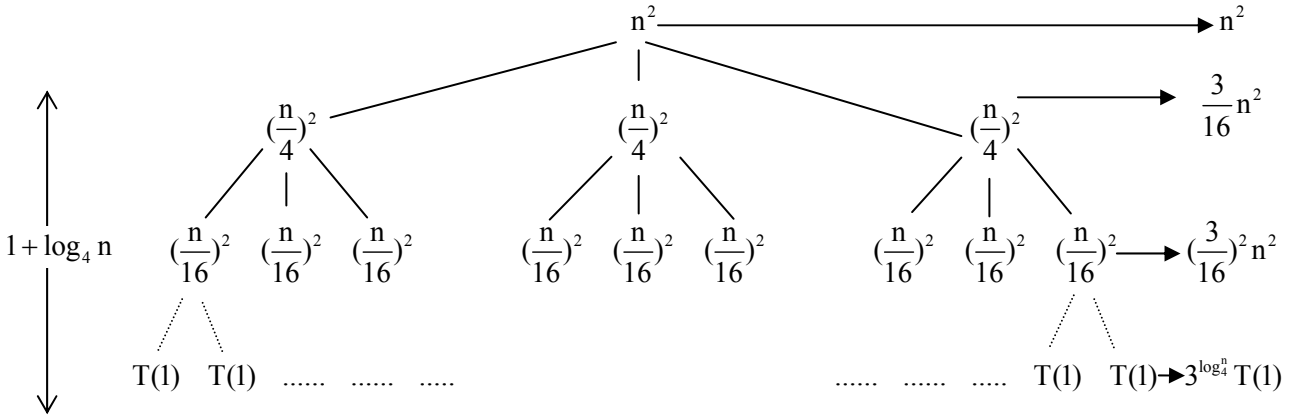
۱) $T(n)=\theta(n\log n)$ ۲) $T(n)=\theta(n^2)$ ۳) $T(n)=\theta(\log n)$ ۴) $T(n)=\theta(n)$

حل: گزینه «۲»

معمولاً کران بالا و پایین در حل معادلات بازگشتی قابل توجه نیستند، بنابراین رابطه بازگشتی را به‌صورت زیر می‌نویسیم:

$$T(n)=\begin{cases}1 & n=1\\ 3T(\frac{n}{4})+n^2 & n>1\end{cases}$$

درخت فراخوانی بازگشتی به‌صورت زیر می‌باشد:



با توجه به درخت فراخوانی فوق داریم:

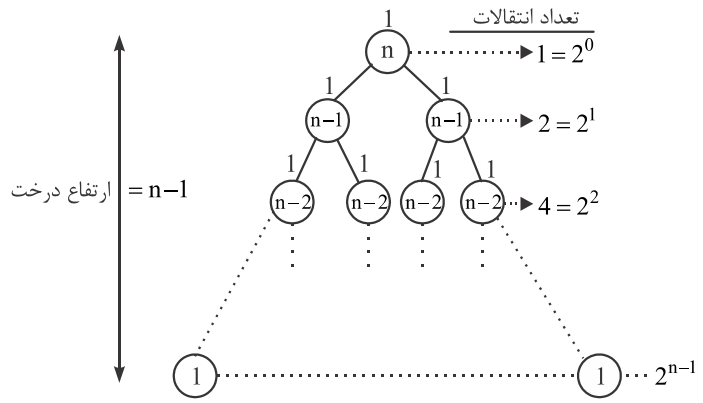
$$T(n)=n^2+\sum_{i=1}^{\log_4 n-1}(\frac{3}{16})^i n^2+3^{\log_4 n}T(1)=n^2+\left(\frac{(\frac{3}{16})^{\log_4 n}-1}{\frac{3}{16}-1}-1\right)n^2+n^{\log_4 3}$$

هنگامی‌که مقدار n به‌سمت بی‌نهایت میل می‌کند، داریم $(\frac{3}{16})^{\log_4^n}\simeq 0$ بنابراین:

$$T(n)=n^2+\left(\frac{-1}{\frac{3}{16}-1}-1\right)n^2+n^{\log_4 3}=\left(\frac{1}{1-\frac{3}{16}}\right)n^2+n^{\log_4 3}=\frac{13}{16}n^2+n^{0.79}=O(n^2)$$

مثال: برای الگوریتم Hanoi بازگشتی، با استفاده از درخت فراخوانی مرتبه زمانی را به‌دست آورید.

حل: ابتدا درخت فراخوانی بازگشتی را به‌صورت زیر رسم می‌کنیم:



با توجه به این درخت داریم: تعداد انتقالات $= T(n) = \sum_{i=0}^{n-1} 2^i = 2^n - 1 \Rightarrow T(n) = 2^n - 1$

در درخت فراخوانی‌های بازگشتی بالا، در سطح i ام، 2^i گره داریم. هر گره در این درخت، یک فراخوانی بازگشتی را مشخص می‌کند.
بدیهی است که زمان اجرای الگوریتم $\theta(2^n)$ خواهد بود و عمق پشته فراخوانی بازگشتی نیز $\theta(n)$ می‌باشد.

۲) روش معادلات بازگشتی کلاسیک

برای مثال می‌خواهیم با این روش معادله بازگشتی $T(n) = 2T(n-1) + 1$ با شرایط اولیه $T(1) = 1$ و $T(2) = 3$، ابتدا معادله مشخصه را به‌صورت زیر به‌دست آورده و سپس ریشه‌های آن را به‌دست می‌آوریم.

$$\begin{matrix}(x-2)(x-1) = 0 \\ T(n) = c_1 2^n + c_2 1^n\end{matrix} \Rightarrow \begin{cases}1 = T(1) = 2c_1 + c_2 \\ 3 = T(2) = 4c_1 + c_2\end{cases} \Rightarrow \begin{cases}c_1 = 1 \\ c_2 = -1\end{cases}$$

بنابراین: $T(n) = 2^n - 1$

✓ **تست: مسأله برج هانوی را با سه 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 انجام می‌شوند.**
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، آنگاه $T(n)$ با کدام رابطه مشخص می‌شود؟ $(T(1) = 2)$ **(مهندسی کامپیوتر سراسری ۸۱)**

۱) $T(n) = 6T(n-1) + 3$
۲) $T(n) = 3T(n-1) + 2$
۳) $T(n) = T(n-1) + T(n-2) + 1$
۴) $T(n) = T(n-1) + T(n-2) + 2$

حل: گزینه «۲»

برای انتقال n مهره از A به B طبق شرایط گفته شده، باید مراحل زیر را طی کنیم:

- $n-1$ مهره را به‌صورت بازگشتی از A به B منتقل می‌کنیم.
- مهره n ام را از A به C منتقل می‌کنیم.
- $n-1$ مهره را به‌صورت بازگشتی از B به A منتقل می‌کنیم.
- مهره n ام را از C به B منتقل می‌کنیم.
- $n-1$ مهره را به‌صورت بازگشتی از A به B منتقل می‌کنیم.

بنابراین داریم:

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

نکته: فرض کنید دو الگوریتم A و B عمل یکسانی را انجام می‌دهند و معادله زمانی آنها با یکدیگر برابر است اما یکی بازگشتی و دیگری غیربازگشتی است، در این‌صورت الگوریتم غیر بازگشتی سریع‌تر از الگوریتم بازگشتی اجرا می‌شود زیرا الگوریتم بازگشتی در هر فراخوانی بازگشتی یک سربار زمانی ناشی از عملیات روی stack انجام می‌پذیرد.

نکته: با توجه به نکته بالا معمولاً بهتر است که برای اجرای الگوریتم روی ماشین، آن را به غیربازگشتی تبدیل نماییم.

✓ **تست: اعداد فیبوناچی (Fibonacci numbers) توسط دنباله بازگشتی زیر تعریف می‌شود و تابع بازگشتی زیر نیز جمله nام دنباله فیبوناچی را ارائه می‌دهد.** $(F_0 = 0, F_1 = 1, F_i = F_{i-1} + F_{i-2}, i \geq 2)$

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

معادله زمانی این الگوریتم عبارتست از:

۱) $T(n) = 2T(n-1) + \theta(1)$
۲) $T(n) = T(n-1) + T(n-2) + \theta(1)$
۳) $T(n) = 2T(n-2) + \theta(1)$
۴) هیچ‌کدام

حل: گزینه «۲»

این الگوریتم به روش بالا به پایین نوشته شده است و مرتبه زمانی آن به‌صورت $T(n) = O(2^n)$ می‌باشد.

نکته: با توجه به اینکه جمله n ام دنباله فیبوناچی از رابطه $F_n = \frac{\varphi^n + \hat{\varphi}^n}{\sqrt{5}}$ به‌دست می‌آید و همچنین $\left|\hat{\varphi}^n\right| / \sqrt{5} < \frac{1}{\sqrt{5}} < \frac{1}{2}$ بنابراین F_n با روند کردن $\varphi^n / \sqrt{5}$ به نزدیکترین عدد صحیح به‌دست می‌آید.

✓ **تست: برنامه زیر برای محاسبه سری فیبوناچی ارائه شده است. گزینه صحیح را انتخاب کنید.**

(مهندسی کامپیوتر سراسری ۸۰)

```
Function fibo (n:integer):integer;
Var f,f1,f2,I :integer;
begin
  f1:=1;
  f2:=1;
  for i:=1 to n do
       begin
       f:= f1+ f2;
       f1:= f2;
       f2:=f;
     end;
  fibo:=f;
end;
```

۱) الگوریتم این برنامه از رده برنامه‌ریزی پویا و مرتبه آن خطی است.
۲) الگوریتم این برنامه از رده تقسیم و حل و مرتبه آن خطی است.
۳) الگوریتم این برنامه از رده برنامه‌ریزی پویا و مرتبه آن بیش از خطی است.
۴) الگوریتم این برنامه از رده تقسیم و حل و مرتبه آن بیش از خطی است.

حل: گزینه «۱»

همان‌طورکه ملاحظه می‌کنید این الگوریتم به‌صورت پایین به بالا طراحی شده است و این کار سبب بهبود بسیار زیاد در زمان الگوریتم شده است.

✓ **تست: مرتبه زمانی یک الگوریتم با رابطه بازگشتی** $T(n) = T(\frac{n}{2}) + \frac{T(\frac{n}{2})}{T(\frac{n}{2}) - 1}$ **بیان شده است. مرتبه زمانی الگوریتم کدام است؟**

(مهندسی کامپیوتر سراسری ۸۱)

۱) $O(2^n)$ ۲) $O(3^n)$ ۳) $O(n^{\log_2 3})$ ۴) $O(n^{\frac{3}{2}})$

حل: گزینه «۳»

با توجه به رابطه زمانی بازگشتی داده‌شده معلوم است که الگوریتم با هر مقدار n خود را سه بار به‌صورت بازگشتی با مقدار n/2 فراخوانی می‌کند، بنابراین در درخت فراخوانی، فاکتور شاخه برابر 3 و عمق درخت برابر $\log_2 n$ می‌باشد. بنابراین داریم:

$T(n) = 3^{\log_2 n} = n^{\log_2 3}$

۳) روش جایگذاری برای حل معادلات بازگشتی

این روش دارای دو مرحله است:

۱) حدس و گمان در مورد فرم جواب

۲) استفاده از استقرای ریاضی جهت پیدا کردن ثابت‌ها و نشان‌دادن اینکه جواب صحیح است.

مثال: معادله بازگشتی $T(n) = 2T(\left\lfloor \frac{n}{2} \right\rfloor + n)$ **را در نظر بگیرید، نشان دهید:** $T(n) = O(n \log n)$.

حل: برای این کار باید نشان دهیم که رابطه $T(n) \le cn \log n (*)$ به‌ازای برخی مقادیر مناسب $c > 0$ برقرار است. ابتدا با فرض اینکه برای تمام مقادیر کوچکتر از n (از جمله $\left\lfloor \frac{n}{2} \right\rfloor$)، در رابطه * صدق می‌کنند داریم:

$T(\left\lfloor \frac{n}{2} \right\rfloor) \le c\left\lfloor \frac{n}{2} \right\rfloor \log(\left\lfloor \frac{n}{2} \right\rfloor)$

حال برای $T(n)$ رابطه را به‌صورت زیر به‌دست می‌آوریم:

$$T(n)=2T(\left\lfloor\frac{n}{2}\right\rfloor)+n\le 2c\left\lfloor\frac{n}{2}\right\rfloor\log(\left\lfloor\frac{n}{2}\right\rfloor)+n$$

$$\le cn\log(\frac{n}{2})+n=cn\log n-cn\log 2+n$$

$$=cn\log n-cn+n=c\log n-n(c-1)$$

بدیهی است که اگر $c\ge 1$ باشد آنگاه $n(c-1)$ جمله‌ای مثبت است بنابراین بدیهی است

$$T(n)=cn\log n-n(c-1)<cn\log n \quad \text{as} \quad c\ge 1$$

$$\Rightarrow T(n)<cn\log n \quad \text{as} \quad c\ge 1$$

نکته: پیدا کردن یک حدس اولیه خوب چندان ساده نیست بلکه نیاز به تجربه و خلاقیت دارد و روش درخت بازگشتی می‌تواند برای ایجاد یک حدس اولیه خوب استفاده شود.

نکته: برخی اوقات به‌دلیل وجود برخی ثابت‌ها با وجود اینکه حدس و گمان برای جواب صحیح است اما نمی‌توان با استقرا کار را ادامه داد زیرا استقرا به حد کافی برای اثبات جزئیات کران‌های قوی نیست. در این‌صورت می‌توان یک جمله از مرتبه پایین را از حدس جواب کسر نمود.

مثال: معادله $T(n)=T(\left\lfloor\frac{n}{2}\right\rfloor)+T(\left\lceil\frac{n}{2}\right\rceil)+1$ **را با حدس** $O(n)$ **در نظر بگیرید. در این‌صورت:**

$$T(n)=T(\left\lfloor\frac{n}{2}\right\rfloor)+T(\left\lceil\frac{n}{2}\right\rceil)+1\le c\left\lfloor\frac{n}{2}\right\rfloor+c\left\lceil\frac{n}{2}\right\rceil+1=cn+1$$

بنابراین شرط $T(n)\le cn$ برقرار نیست. در این مواقع بهتر است حدس را به‌صورت $T(n)\le cn-b(b\ge 0)$ درنظر بگیریم. پس با به‌کار بردن استقرا داریم:

$$T(n)\le(c\left\lfloor\frac{n}{2}\right\rfloor-b)+(c\left\lceil\frac{n}{2}\right\rceil-b)+1=cn-2b+1$$

واضح است که اگر $b\ge 1$ انتخاب شود؛ آنگاه جمله $-b+1$ منفی بوده و درنتیجه داریم:

$$T(n)\le cn-b+(-b+1)\le cn-b\Rightarrow T(n)\le cn-b \quad \text{as} \quad b\ge 1$$

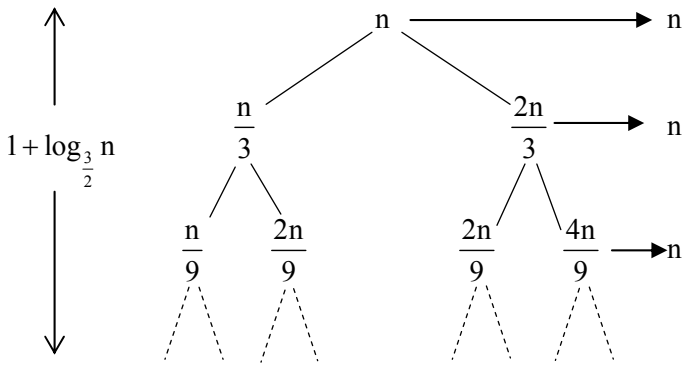
تست: مرتبه زمانی رابطه بازگشتی زیر کدام است؟

$$T(n)=\begin{cases}1 & n=1\\ T(\frac{n}{3})+T(\frac{2n}{3})+O(n) & n>1\end{cases}$$

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

حل: گزینه «۱»

روش اول: درخت فراخوانی بازگشتی را به‌صورت زیر رسم می‌کنیم:



برای به‌دست آوردن حداکثر ارتفاع، شاخه‌ای که تغییرات کمتری دارد را به‌صورت زیر درنظر می‌گیریم:

$$n\to\frac{2}{3}n\to\frac{4}{9}n=(\frac{2}{3})^2n\to(\frac{2}{3})^3n\to\ldots\to1$$

با توجه به رابطه فوق:

$$(\frac{2}{3})^k n=1\Rightarrow n=(\frac{3}{2})^k\Rightarrow k=\log_{\frac{3}{2}} n$$

و ارتفاع درخت برابر $k+1$ می‌باشد.

در این گونه مسائل که درخت کامل نیست حد بالا را در نظر می‌گیرید و فرض می‌کنید درخت کامل است تا بتوان حد بالا را به‌دست آورد.

بنابراین داریم: $T(n)=(\log_{\frac{3}{2}}^n+1)n=O(n\log_{\frac{3}{2}}^n)=O(n\log n)$

روش دوم: می‌توان از روش جایگزینی نیز استفاده نمود. در ابتدا حدس می‌زنیم $T(n)=O(n\log n)$ می‌باشد سپس باید حدس خود را اثبات کنیم:

$$T(n)\le dn\log n$$

$$T(n)=T(\frac{n}{3})+T(\frac{2n}{3})+n$$

$$T(n)\le d\frac{n}{3}\log\frac{n}{3}+d\frac{2n}{3}\log\frac{2n}{3}+n=d\frac{n}{3}\log n-d\frac{n}{3}\log3+d\frac{2n}{3}\log n+d\frac{2n}{3}\log2-d\frac{2n}{3}\log3+n$$

$$=dn\log n-dn\log3+d\frac{2n}{3}+n=dn\log n+n(1+0.66d-1.58d)$$

با فرض $d>2$ عبارت $1+0.66d-1.58d<0$. بنابراین ضریب n منفی می‌شود و می‌توان آن را از نامساوی حذف نمود.

$$T(n)\le dn\log n$$

برای به‌دست آوردن مقدار دقیق d باید از تعریف big-O استفاده نمود.

۴) روش تغییر متغیر برای حل معادلات بازگشتی

در این روش سعی می‌شود با یک تغییر متغیر مناسب معادله را به فرم مناسب تغییر داد.

به مثال زیر توجه نمایید:

$$T(n)=2T(\sqrt{n})+\log n$$

با قرار دادن $m=\log n$ داریم $n=2^m$ و با قرار دادن در معادله بالا داریم:

$$T(2^m)=2T(\sqrt{2^m})+m\Rightarrow T(2^m)=2T(2^{\frac{m}{2}})+m$$

حال اگر قرار دهید $S(m)=T(2^m)$ در آن‌صورت فرم نهایی معادله بالا به فرم زیر خواهد بود:

$$S(m)=2S(\frac{m}{2})+m$$

به‌راحتی می‌توان دید که $S(m)=O(m\log m)$ می‌باشد (طبق مثال‌های قبلی)، در نتیجه:

$$T(n)=O(\log n\ \log\log n)$$

نکته مهم: اگر $a\ge1$ و $c>0$ آنگاه معادله بازگشتی

$$T(n)=T(n-a)+T(a)+cn$$

دارای جواب $T(n)=O(n^2)$ می‌باشد.

(برای اثبات می‌توان از درخت فراخوانی‌های بازگشتی استفاده نمود)

نکته مهم: معادله بازگشتی $T(n)=T(\alpha n)+T((1-\alpha)n)+cn$ با شرایط $0<\alpha<1$ و $c>0$ دارای جوابی به فرم $T(n)=O(n\log n)$ می‌باشد که مبنای لگاریتم عبارتست از $\min\left\{\frac{1}{\alpha},\frac{1}{1-\alpha}\right\}$.

به‌عنوان مثال با به‌کار بردن روش درخت بازگشتی نشان دهید که معادله بازگشتی $T(n)=T(\frac{n}{3})+T(2\frac{n}{3})+cn$ دارای حد پایین مجانبی $T(n)=\Omega(n\log n)$ می‌باشد.

۵) قضیه اصلی (Master theorem)

رابطه بازگشتی زیر را در نظر 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 این‌صورت داریم:

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

$$2)\text{if } f(n)=\theta\left(n^{\log_b a}\times\log_2^k n\right)\Rightarrow T(n)=\theta\left(n^{\log_b a}\times\log_2^{k+1} n\right)$$

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

نکته: در به‌کار بردن قضیه master باید دقت کرد که شرایط به‌طورکامل برقرار باشد مخصوصا در مورد سوم که ممکن است برخی اوقات همه شرایط به‌طور کامل برقرار نباشد که در سؤال‌های چهارگزینه‌ای نمونه به آن اشاره خواهیم داشت.

نکته مهم: در معادله $T(n)=aT(\frac{n}{b})+f(n)$ ، $a\geq 1$ و $b>1$ چنانچه داشته باشیم:

$$f(n)=\theta(n^{\log_b^a}\log_n^k)$$
$$\forall k\geq 0$$

آنگاه داریم:

$$T(n)=\theta(n^{\log_b^a}\log_n^{k+1})$$

مثال: تابع آکرمن را به‌صورت زیر تعریف می‌کنیم، بهترین زمان اجرای آن را به‌دست آورد.

$$A(i,j)=\begin{cases}A(1,j)=2^j & j\geq 1\\ A(i,1)=A(i-1,2) & i\geq 2\\ A(i,j)=A(i-1,A(i,j-1)) & i,j\geq 2\end{cases}$$

حل: تابع آکرمن را برای مقادیر کوچک i و j به‌صورت زیر محاسبه می‌کنیم:

	j=1	j=2	j=3	j=4
i=1	2^1	2^2	2^3	2^4
i=2	2^2	2^{2^2}	$2^{2^{2^2}}$	$2^{2^{2^{2^2}}}$
i=3	2^{2^2}	$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\}16$	$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\}16$	$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\}16$

$A(2,2)=A(1,A(2,1))=A(1,\ 2^2)=2^{2^2}$

$A(2,3)=A(1,A(3,1))=A(1,\ 2^{2^2})=2^{2^{2^2}}$

$A(2,4)=A(1,A(4,1))=A(1,\ 2^{2^{2^2}})=2^{2^{2^{2^2}}}$

$A(3,1)=A(2,2)= 2^{2^2}$

$A(3,2)=A(2,A(3,1))=A(2,\ 2^{2^2})=A(2,16)= 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}16}$

$A(3,3)=A(2,A(3,2))=A(2,\ 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}16})= 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}16}}$

$A(3,4)=A(2,A(3,3))=A(2,\ 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}16}})= 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}16}}}$

$A(4,1)=A(3,2)= 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}\}16} > 10^{80}$

این تابع از تمام توابع که تاکنون مشاهده کرده‌اید بدتر است و داریم: $A(n,n)=\Omega(n^n)$ یا $A(n,n) << (n^n)$

✓ **تست: مرتبه زمانی تابع بازگشتی زیر کدام است؟**

$$T(n)=\begin{cases}1 & n=0\\ \dfrac{1+n(n-1)T(n-1)}{n^2+1} & n>0\end{cases}$$

۱) $T(n)=\theta(\dfrac{\log n}{n})$

۲) $T(n)=\theta(n\log n)$

۳) $T(n)=\theta(\log n)$

۴) $T(n)=\theta(\dfrac{\log n}{n!})$

✍ **حل: گزینه «۱»**

$$n\,T(n)=[\frac{1}{n}+(n-1)T(n-1)]\frac{n^2}{n^2+1}$$
$$n\,T(n)=g(n)$$
$$g(n)=\frac{n^2}{n^2+1}[\frac{1}{n}+g(n-1)]$$
$$g(n)<\frac{1}{n}+g(n-1)$$
$$g(n)=1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}=\log n$$
$$g(n)\in O(\frac{\log n}{n})$$

$$T(n)=\frac{1+n(n-1)T(n-1)}{n^2+1}\Rightarrow T(n)=\frac{n^2-1}{n^2-1}\times\frac{1+n(n-1)T(n-1)}{n^2+1}$$
$$T(n)=\frac{n^2-1}{n^4-1}\times[1+n(n-1)T(n-1)]\Rightarrow\frac{T(n)}{n+1}=\frac{n-1}{n^4-1}\times[1+n(n-1)T(n-1)]$$
$$\frac{T(n)}{n+1}=\frac{1}{n^4-1}\times[\frac{n-1}{1}+n(n-1)^2T(n-1)]$$
$$\frac{T(n)}{n+1}=\frac{n^2}{n^4-1}\times[\frac{n-1}{n^2}+\frac{(n-1)^2}{n}T(n-1)]$$