دانشجوی گرامی

هدف از تدوین این مجموعه بررسی نکات مهم مربوط به سوالات کنکور درس ساختمان داده در سال‌های اخیر است.

ایده اصلی طراحان سوال در چند سال گذشته استفاده از مسائل کتاب "مقدمه‌ای بر الگوریتم‌ها" نوشته توماس کرمن در مباحث درخت‌ها، مرتبه زمانی، بازگشتی و مرتب‌سازی بوده است.

امید است این مجموعه بتواند کمکی در جهت موفقیت شما در آزمون کارشناسی ارشد باشد.

مولف

در جدول ذیل دروس به سرفصلهای مهم آن طبقه بندی شده و مشخص شده است که در هر سال از هر مبحث چند تست سوال شده است و دانشجوی محترم می تواند زمان باقیمانده تا کنکور را با توجه به اهمیت مباحث مدیریت نماید.

رشته: مهندسی کامپیوتر				درس: ساختمان داده ها				
ردیف	مبحث	۱۳۸۵ تعداد تست	۱۳۸۶ تعداد تست	۱۳۸۷ تعداد تست	۱۳۸۸ تعداد تست	۱۳۸۹ تعداد تست	مجموع ۵ سال	نسبت از کل
1	آنالیز الگوریتم ها و روابط بازگشتی	1	3	1	2	3	10	29%
2	آرایه ها و ماتریس ها	1	0	1	0	0	2	6%
3	پشته و صف	0	0	0	0	0	0	0%
4	لیست پیوندی	0	0	0	0	1	1	3%
5	درخت ها	4	3	6	3	1	17	50%
6	گراف ها	0	0	0	0	0	0	0%
7	Hash	0	0	0	1	0	1	3%
8	روش های مرتب سازی	0	0	0	1	2	3	9%
	جمع	**6**	**6**	**8**	**7**	**7**	**34**	**100%**

روش اصلی برای حل مسائل بازگشتی

این روش تکنیکی برای محاسبه مرتبه زمانی روابط بازگشتی به فرم زیر است:

$$\boxed{T(n)=aT\left(\frac{n}{b}\right)+f(n)}$$

که در آن:

- ✓ $a\geq 1$ و $b\geq 2$
- ✓ $f(n)$ نشانگر زمان اجرای الگوریتمی است که زمان حل یک نمونه مسئله به طول n در آن برابر است با زمان مورد نیاز برای حل a مسئله به اندازه $T\left(\frac{n}{b}\right)$
- ✓ حالات $\frac{n}{b}$، $\left\lceil\frac{n}{b}\right\rceil$ یا $\left\lfloor\frac{n}{b}\right\rfloor$ یکسان هستند.

قضیه اصلی: مرتبه زمانی رابطه بازگشتی با مقایسه $f(n)$ و $n^{\log_b^a}$ به یکی از حالتهای زیر محاسبه می شود:

$$(I):\ \text{if}\quad f(n)=O\left(n^{\log_b a-\varepsilon}\right)\xrightarrow{\varepsilon>0}T(n)=\theta\left(n^{\log_b a}\right)$$

- (اگر $n^{\log_b^a}$ بزرگتر باشد)

$$(II):\ \text{if}\quad f(n)=\theta\left(n^{\log_b a}(\log n)^k\right)\xrightarrow{k\geq 0}T(n)=\theta\left(n^{\log_b a}(\log n)^{k+1}\right)$$

- (اگر $f(n)$ و $n^{\log_b^a}$ صرف نظر از یک ضریب $(\log n)^k$ مساوی باشند)

$$(III):\ \text{if}\quad f(n)=\Omega\left(n^{\log_b a+\varepsilon}\right)\xrightarrow{\varepsilon>0}T(n)=\theta(f(n))$$

- (اگر $f(n)$ بزرگتر باشد)

دقت کنید:

هنگام مقایسه تابع $f(n)$ و $n^{\log_b^a}$ ، هدف از ذکر ثابت ε آن است که نشان دهیم تابع $f(n)$ باید بصورت چند جمله ای کوچکتر (حالت (I)) یا بزرگتر(حالت (III)) از $n^{\log_b^a}$ باشد . به این معنی که مرتبه بزرگی یا کوچکی باید از n^{ε} باشد و نه از مرتبه های $\log n$ یا $\cos n$ یا ... چرا که در غیر اینصورت استفاده از قضیه اصلی صحیح نیست.

مثال ۱: جواب رابطه بازگشتی زیر را محاسبه کنید.

$$T(n)=9T\left(\frac{n}{3}\right)+n$$

حل :

$$\begin{cases}f(n)=n\\ n^{\log_b^a}=n^{\log_3^9}=n^2\end{cases}\xrightarrow[f(n)=O\left(n^{\log_b a-\varepsilon}\right)]{\text{case I}:\ \varepsilon>0}T(n)=\theta(n^{\log_b^a})=\theta(n^2)$$

یادداشت:

....

....

....

....

مثال۲: جواب رابطه بازگشتی زیر را را محاسبه کنید.

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

حل :

$$\begin{cases} f(n) = 1 \\ n^{\log_b^a} = n^{\log_{\frac{3}{2}}^1} = n^0 = 1 \end{cases} \xrightarrow[f(n)=\theta\left(n^{\log_b a}(\log n)^k\right)]{\text{case II : } k=0} T(n) = \theta\left(n^{\log_b a}(\log n)^k\right) = \theta(\log n)$$

مثال ۳: جواب رابطه بازگشتی زیر را را محاسبه کنید.

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log n$$

حل :

$$\begin{cases} f(n) = n\log n \\ n^{\log_b^a} = n^{\log_4^3} \end{cases} \xrightarrow[f(n)=\Omega\left(n^{\log_b a+\varepsilon}\right)]{\text{case III : } \varepsilon>0} T(n) = \theta(f(n)) = \theta(n\log n)$$

مثال: جواب رابطه‌ی بازگشتی $T(n) = 4T\left(\frac{\sqrt{n}}{3}\right) + \log^2 n$ کدام است؟ (سراسری ۸۸)

۱) $\Theta(\sqrt{n})$ ۲) $\Theta(\log^2 n)$ ۳) $\Theta(\log^3 n)$ ۴) $\Theta(\log^2 n \log\log n)$

حل: گزینه ۴ صحیح است.

ابتدا با استفاده از یک تغییر متغیر فرم تابع را به شکل قابل استفاده از روش اصلی در می آوریم :

$$T(n) = 4T\left(\frac{\sqrt{n}}{3}\right) + \log^2 n \xrightarrow{n=2^m} T(2^m) = 4T\left(2^{\frac{m}{2}}\right) + m^2 \xrightarrow{s(m)=T(2^m)} S(m) = 4S\left(\frac{m}{2}\right) + m^2$$

حال می توانیم برای رابطه بدست آمده از حالت II قضیه اصلی به شکل زیر استفاده کنیم :

$$\begin{cases} f(m) = m^2 \\ m^{\log_b^a} = m^{\log_2^4} = m^2 \end{cases} \xrightarrow[f(m)=\theta\left(m^{\log_b a}(\log m)^k\right)]{\text{case II: for } k=0} T(m) = \theta\left(m^2(\log m)^{k+1}\right)$$

$$= \theta\left(m^2 \log m\right)$$

که با تبدیل معکوس جواب رابطه بازگشتی اولیه بصورت زیر بدست می آید :

$$T(m) = \theta\left(m^2\log m\right) \xrightarrow{2^m=n \to m=\log n} T(n) = \theta\left(\log^2 n \log\log n\right)$$

دقت کنید :

هنگام تبدیل $T\left(\frac{\sqrt{n}}{3}\right)$ به $T\left(2^{\frac{m}{2}}\right)$ از آنجائیکه $n = 2^m$ در نظرگرفته بودیم در نتیجه $\sqrt{n} = n^{\frac{1}{2}} = 2^{\frac{m}{2}}$ و از ثابت ۳ صرف نظر کردیم.

یادداشت:

....

....

....

....

درخت:

- ساختمان داده‌ی غیر خطی
- نمایش درخت: لیست پیوندی، آرایه

در یک درخت k تایی دلخواه با n گره و ارتفاع h داریم:

۱ ـ	کل اتصالات	اتصالات پر	اتصالات خالی
	nk	$n-1$	$nk-(n-1)$
درخت دودویی $(k=2)$:	$2n$	$n-1$	$n+1$

۲ ـ	تعداد کل گره‌ها:		$n = n_k + n_{k-1} + \ldots + n_2 + n_1 + n_0$
	درخت دودویی $(k=2)$:		$n = n_2 + n_1 + n_0$

۳- تعداد برگ‌های درخت:

$$n_0 = (k-1)n_k + (k-2)n_{k-1} + \ldots + 2n_3 + n_2 + 1$$

حالت خاص: درخت دودویی $(k=2)$:

$$n_0 = n_2 + 1$$

نتیجه : تعداد برگ‌ها در هر درخت مستقل از تعداد گره‌های تک فرزندی است.

۴ـ کران پائین و بالای تعداد برگها:

الف)با توجه به درجه و تعداد گره های درخت:

$$1 \le n_0 \le \left\lfloor \frac{nk-(n-1)}{k} \right\rfloor$$

حالت خاص: درخت دودویی $(k=2)$:

$$1 \le n_0 \le \left\lfloor \frac{n+1}{2} \right\rfloor$$

ب)با توجه به درجه و عمق درخت:

$1 \le n_0 \le k^{h-1}$ (با فرض سطح ریشهٔ یک)

$1 \le n_0 \le k^{h}$ (با فرض سطح ریشهٔ صفر)

حالت خاص: درخت دودویی $(k=2)$:

$1 \le n_0 \le 2^{h-1}$ (با فرض سطح ریشهٔ یک)

$1 \le n_0 \le 2^{h}$ (با فرض سطح ریشهٔ صفر)

۶ ـ کران پائین و بالای ارتفاع یک درخت k تایی با n گره برابر است:

$\left\lfloor \log_k^{n(k-1)} \right\rfloor + 1 \le h \le n$ (با فرض سطح ریشهٔ یک)

$\left\lfloor \log_k^{n(k-1)} \right\rfloor \le h \le n-1$ (با فرض سطح ریشهٔ صفر)

یادداشت:

....

....

....

....

حالت خاص: درخت دودویی $(k=2)$:

$\left\lfloor \log_2^n \right\rfloor + 1 \le h \le n$ (با فرض سطح ریشهٔ یک)

$\left\lfloor \log_2^n \right\rfloor \le h \le n-1$ (با فرض ریشهٔ صفر)

۷ ـ کران پائین و بالای تعداد گره در یک درخت k تایی با ارتفاع h:

$h \le n \le \frac{1-k^h}{1-k}$ (سطح ریشهٔ یک)

$h+1 \le n \le \frac{1-k^{h+1}}{k-1}$ (سطح ریشهٔ صفر)

حالت خاص: درخت دودویی $(k=2)$:

.

$h \le n \le 2^h - 1$ (سطح ریشهٔ یک)

$h+1 \le n \le 2^{h+1} - 1$ (سطح ریشهٔ صفر)

۸ ـ کران پائین و بالای تعداد گره در سطح i ام: (m تعداد گره)

$1 \le m \le k^{i-1}$ (سطح ریشهٔ یک)

$1 \le m \le k^i$ (سطح ریشهٔ صفر)

حالت خاص: درخت دودویی $(k=2)$:

$1 \le m \le 2^{i-1}$ (سطح ریشهٔ یک)

$1 \le m \le 2^i$ (سطح ریشهٔ صفر)

۹ ـ تعداد درخت‌های دودویی متفاوت با n گره برابر است با:

تعداد درخت‌های دودویی متفاوت با n گره $= \sum b_k b_{n-1-k} = \frac{1}{n+1}\binom{2n}{n} = \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}}\left(1+O\left(\frac{1}{n}\right)\right)$

۱۰ ـ تعداد درخت‌های دودویی با n گره با ارتفاع $n-1$: (حتماً و حتماً با فرض سطح ریشهٔ صفر)

2^{n-1}

۱۱ ـ تعداد درختان دودویی متمایز محض با n گره برابر است با:

$n = 2k+1 \Rightarrow \frac{\binom{2k}{k}}{k+1}$

یادداشت:

....

....

....

....

در یک درخت k تایی کامل و پر با n گره و ارتفاع h:

۱ ـ ارتفاع یک درخت k تایی کامل یا پر با n گره:

$h = \left\lfloor \log_k^{n(k-1)} \right\rfloor + 1$ (با فرض سطح ریشهٔ یک)

$h = \left\lceil \log_k^{n(k-1)} \right\rceil$ (با فرض سطح ریشهٔ صفر)

حالت خاص: درخت دودویی $(k = 2)$:

$h = \left\lfloor \log_2^n \right\rfloor + 1$ (با فرض سطح ریشهٔ یک)

$h = \left\lfloor \log_2^n \right\rfloor$ (با فرض سطح ریشهٔ صفر)

۲ ـ تعداد گره‌های یک درخت k تایی کامل و پر با ارتفاع h:

	درخت پر	درخت کامل
سطح ریشه یک	$n = \frac{1-k^h}{1-k}$	$\frac{1-k^{h-1}}{1-k} + 1 \le n \le \frac{1-k^h}{1-k}$
سطح ریشه صفر	$n = \frac{1-k^{h+1}}{1-k}$	$\frac{1-k^h}{1-k} + 1 \le n \le \frac{1-k^{h+1}}{1-k}$

حالت خاص: درخت دودویی $(k = 2)$:

	درخت پر	درخت کامل
سطح ریشه یک	$n = 2^h - 1$	$2^{h-1} \le n \le 2^h - 1$
سطح ریشه صفر	$n = 2^{h+1} - 1$	$2^h \le n \le 2^{h+1} - 1$

۳ ـ تعداد برگ‌های درخت k تایی کامل یا پر:

$$n_0 = \left\lfloor \frac{nk - (n-1)}{k} \right\rfloor$$

حالت خاص: درخت دودویی $(k = 2)$:

$$n_0 = \left\lfloor \frac{n+1}{2} \right\rfloor = \left\lceil \frac{n}{2} \right\rceil$$

۴ ـ شمارهٔ اندیس برگ‌های درخت k تایی کامل یا پر:

$$\left\lceil \frac{n-1}{k} \right\rceil + 1, \left\lceil \frac{n-1}{k} \right\rceil + 2, .., n$$

حالت خاص: درخت دودویی $(k = 2)$:

$$\left\lfloor \frac{n}{2} \right\rfloor + 1, \left\lfloor \frac{n}{2} \right\rfloor + 2, \dots, n$$

۵ـ اندیس پدر گره‌ای i ام در درخت k تایی کامل یا پر:

$$\text{parent}(i) = \left\lceil \frac{i-1}{k} \right\rceil$$

یادداشت:

....

....

....

....

۶- اندیس فرزندان گره‌ی i ام درخت k تایی کامل یا پر:

$$\underbrace{\ldots, k_{i-1}, k_i, k_{i+1}}_{k \text{ تا}}$$

حالت خاص: درخت دودویی $(k=2)$:

اندیس پدر گره‌ی i ام:

$$\text{Parent}(i) = \left\lfloor \frac{i}{2} \right\rfloor$$

اندیس فرزندان گره‌ی i ام:

$$\text{Leftchild}(i) = 2i$$
$$\text{Rightchild}(i) = 2i+1$$

یادداشت:

....

....

....

....

در یک درخت با n گره:

۱ ـ هر یک از پیمایش‌های inorder، preorder، postorder، زمان $\theta(n)$ صرف می‌کنند.

نکته : از آنجا که در پیمایش درخت، هر گره حداقل یکبار بررسی می‌شود، الگوریتمی وجود ندارد که یک درخت را در زمان کمتر از $\theta(n)$ پیمایش کند.

۲ ـ با هر یک از زوج پیمایش‌های $\left\{ \text{preorder} \right.$، $\left\{ \text{postorder} \right.$، $\left\{ \text{levelorder} \right.$ می‌توان هر درخت دلخواه را به‌صورت یکتا رسم کرد.

که برای این منظور ابتدا با استفاده از هر یک از پیمایش‌های preorder، postorder و levelorder می‌توان ریشه را مشخص کرد. سپس با استفاده از پیمایش inorder زیر درخت‌های چپ و راست ریشه مشخص می‌شوند. این روند مجدداً به‌صورت بازگشتی برای زیر درخت‌های چپ و راست ریشه تکرار می‌شود.

۳ ـ در صورتی‌که درخت کامل باشد، از آنجا که شکل درخت مشخص است، با هر پیمایشی می‌توان آن را به‌صورت یکتا رسم کرد.

۴ ـ با هر پیمایش preorder، postorder و levelorder یک درخت جستجوی دودویی می‌توان آن را به‌صورت یکتا رسم کرد.

۵ ـ پیمایش inorder یک درخت جستجوی دودویی، کلیدها را به‌صورت یکتا برمی‌گرداند.

۶ ـ از آنجاکه Heap یک درخت کامل است، با هر پیمایشی، می‌توان آن را به‌صورت یکتا رسم کرد.

۷ ـ هیچ پیمایشی از Heap الزاماً گره‌ها را به‌صورت مرتب برنمی‌گرداند.

۸ ـ در پیمایش preorder، ریشه اولین و سمت راست‌ترین برگ، آخرین گره‌ای است که ملاقات می‌شوند.

در پیمایش postorder، سمت چپ‌ترین برگ اولین و ریشه آخرین گره‌ای است که ملاقات می‌شوند.

۹ ـ در شرایطی که در هر سطح یک درخت تنها یک گره باشد، یعنی ارتفاع یک درخت با n گره، n باشد، آن‌گاه اولین گره‌ای که در پیمایش Postorder ملاقات می‌شود با آخرین گره‌ای که در پیمایش Preorder ملاقات می‌شود، برابر است

۱۰ ـ ترتیب ملاقات برگ‌ها در هر سه پیمایش Preorder، inorder و Postorder یکی است.

۱۱ ـ تعداد درخت‌های دودویی که پیمایش Preorder و Postorder آن‌ها برابر با رشته‌های مشخص باشد، برابر است با 2^{1} که در آن **1** تعداد گره های تک فرزندی در درخت است.

برای به‌دست آوردن گره‌های تک فرزندی بدین صورت عمل می‌کنیم: از سمت چپ یکی از پیمایش‌ها (برای مثال Preorder) حرکت می‌کنیم و هر ترتیب دوتایی از گره‌ها را در نظر گرفته (ترتیب‌های دوتایی دقیقاً کنار هم) و در پیمایش دیگر بررسی می‌کنیم، اگر همین ترتیب دوتایی را به‌صورت معکوس عیناً مشاهده کردیم، گرهٔ اول ترتیب انتخابی تک فرزندی است.

هم‌چنین برای پیدا کردن برگ‌ها، با دو پیمایش preorder و postorder ابتدا گره‌های تک فرزندی را مشخص می‌کنیم. سپس، از آنجا که ترتیب ملاقات برگ‌ها در دو پیمایش یکی است، با استفاده از دو پیمایش سمت راست‌ترین و سمت چپ‌ترین برگ را مشخص می‌کنیم. گره‌های قبل از سمت چپ‌ترین برگ (در پیمایش Preorder) و گره‌های بعد از سمت راست‌ترین برگ (در پیمایش Postorder) قطعاً برگ نیستند. لذا به راحتی برگ‌ها مشخص می‌شوند.

یادداشت:

....

....

....

....

درخت جستجوی دودویی: (BST)

درخت جستجو دودویی یک درخت دودویی است که در آن کلید هر گره از تمام کلیدهای زیر درخت چپ بزرگ‌تر و از تمام کلیدهای زیر درخت راست کوچک‌تر است.

با هر پیمایش Preorder، Postorder، Levelorder یک درخت جستجوی دودویی، می‌توان آن را به‌صورت یکتا رسم کرد.

پیمایش inorder هر درخت جستجوی دودویی کلیدها را مرتب بر می‌گرداند.

تعداد درخت‌های جستجوی دودویی متفاوت با n گره با تعداد درختان دودویی متفاوت با n گره برابر است (b_n تعداد درختان دودویی متفاوت)

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k} = \frac{1}{n+1}\binom{2n}{n} = \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}}\left(1+O\left(\frac{1}{n}\right)\right)$$

تعداد درختان جستجوی دودویی متفاوت با n گره و با ارتفاعn (سطح ریشه صفر)برابر است با: 2^{n-1}

اعمال مختلف در درخت جستجو: (h ارتفاع درخت جستجوی دودویی)

۱- درج: با پیمایش یک مسیر مشخص از ریشه تا یک برگ، گرهٔ x به عنوان فرزند آن برگ درج می‌شود. درج همواره در برگ است. زمان درج $O(h)$ است.

۲- حذف: در صورتی‌که گره‌ای می‌خواهد حذف شود، برگ باشد، حذف به راحتی انجام می‌شود. اگر گرهٔ تک فرزندی باشد، زیر درخت آن جایگزین گرهٔ مورد نظر می‌شود و اگر گرهٔ دو فرزندی باشد، عنصر مابعد یا عنصر ماقبل آن در پیمایش inorder جایگزین آن می‌شود. زمان اجرا $O(h)$است.

۳- جستجو: با انتخاب یک مسیر مشخص از ریشه تا حداکثر برگ، جستجو انجام می‌شود. زمان $O(h)$ است.

۴- **جستجوی ماکزیمم یا مینیمم** $(TREE\_MAXIMUM, TREE\_MINIMUM)$: زمان $O(h)$ است.

۵- **جستجوی عنصر بعدی یا قبلی** $(TREE\_SUCCESSOR, TREE\_PREDCCESSOR)$: زمان $O(h)$ است.

تحلیل عملیات مختلف در درخت جستجوی دودویی

از آنجایی که تمام عملیات در درخت جستجوی دودویی وابسته به روال جستجو و جستجو نیز وابسته به عمق **(h)** درخت است ، بنابراین با توجه به آن‌که $\mathbf{(\log_2 n < h < n)}$ می‌باشد و بدترین حالت زمانی رخ می‌دهد که در هر سطح یک گره باشد، مانند درخت‌های اریب به راست و چپ(با ورودی مرتب) که در این صورت هر یک از اعمال گفته شده در زمان $O(n)$ اجرا می‌شوند.

پس هر یک اعمال گفته شده در حالت متوسط و بهترین حالت در زمان $O(\log n)$ و در بدترین حالت در زمان $O(n)$ اجرا می‌شوند.

نتیجه: زمان هرکدام از عملیات فوق حتماً از مرتبه $\mathbf{\Omega(\log_2 n)}$ یا $\mathbf{O(n)}$ می باشد.

ساختن یک درخت جستجوی دودویی

برای ساختن یک درخت جستجوی دودویی، کلیدها پشت سر هم در یک درخت جستجوی دودویی تهی درج می‌شوند. که زمان آن متناسب با درج n کلید در یک BST است. یعنی $O(nh)$ که در حالت متوسط و بهترین حالت $O(n\log n)$ و بدترین حالت زمان $O(n^2)$است

یادداشت:

....

....

....

....

نتیجه: زمان ساختن یک درخت جستجوی دلخواه با n کلید حتماً از مرتبه $\Omega(n\log_2 n)$ یا $O(n^2)$ می باشد.

مرتب سازی درختی:

برای مرتب‌سازی n کلید، ابتدا آنها را در یک BST تهی درج می‌کنیم. سپس با استفاده از پیمایش inorder کلیدها را مرتب می‌کنیم.
بنابراین، کلیدها در زمان $O(nh)$ مرتب می‌شوند.

$$\underbrace{O(nh)}_{\text{زمان درج n کلید}}+\underbrace{\theta(n)}_{\text{زمان پیمایش inorder}}=O(nh)$$

که در حالت متوسط $O(n\log n)$ و بدترین حالت $O(n^2)$ است.

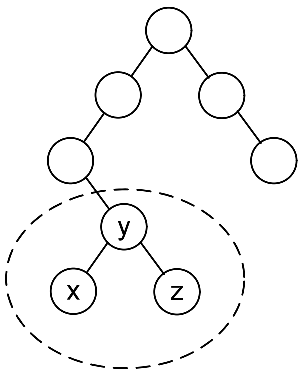
نکته : بدون توجه به اینکه در یک درخت جستجوی دودویی با ارتفاع h از چه گره‌ای شروع کنیم، k فراخوانی موفقیت آمیز TREE – SUCCESSOR زمان $O(k+h)$ را صرف می‌کند.

در یک BST همواره:

$a=key[x]$

$b=key[y]$

$c=key[z]$



$$\begin{cases}\text{SUCCESSOR}(x)=y\\ \text{SUCCESSOR}(y)=z\end{cases}\qquad\begin{cases}\text{PREDECESSOR}(z)=y\\ \text{PREDECESSOR}(y)=x\end{cases}$$

پیمایش inorder : $\dots a\ b\ c\dots$

نکته : پیمایش inorder یک درخت جستجوی دودویی را می‌توان ابتدا با یک فراخوانی TREE – MINIMUM و سپس $n-1$ فراخوانی TREE SUCCESSOR پیاده‌سازی کرد که زمان آن $\theta(n)$ است.

یادداشت:

@RAyanAzmoon

درخت با ارتفاع متوازن

درختی که در آن اختلاف ارتفاع **دو زیر درخت چپ و راست** هر گره دلخواه **حداکثر 1** باشد، **درخت با ارتفاع متوازن** نامیده می‌شود.

حداقل گره برای درخت با ارتفاع متوازن h

اگر $\mathbf{AVL}(\mathbf{h})$ **حداقل تعداد گره** مورد نیاز برای ساختن یک **درخت دودویی با ارتفاع متوازن h** (سطح ریشه صفر فرض می‌شود) باشد، آن‌گاه:

$$\mathbf{AVL}(\mathbf{h}) = \mathbf{AVL}(\mathbf{h-1}) + \mathbf{AVL}(\mathbf{h-2}) + \mathbf{1}$$

$$\mathbf{AVL(0) = 1}$$

$$\mathbf{AVL(1) = 2}$$

درخت جستجوی دودویی با ارتفاع متوازن (AVL)

هر درخت جستجوی دودویی با ارتفاع متوازن، درخت AVL نامیده می‌شود.به عبارت بهتر درختان جستجو با بیشترین عمق $O(\log_2 n)$، درختان جستجوی متعادل نامیده می‌شوند.

از آنجایی‌که عملیات: ۱- جستجو ، ۲- حذف ، ۳- درج ، ۴- جستجوی ماکزیمم و مینیمم ، ۵ ـ جستجوی عنصر بعدی **(succ)** و عنصر قبلی **(pred)** در درخت جستجوی دودویی با زمان **O(h)** انجام می‌شود ، بنابراین کلیه اعمال فوق در درخت جستجوی دودویی متوازن **(AVL)** در زمان $\mathbf{O(\log_2 n)}$ و بهتر از درخت جستجوی دودیی **(BST)** انجام خواهد شد.

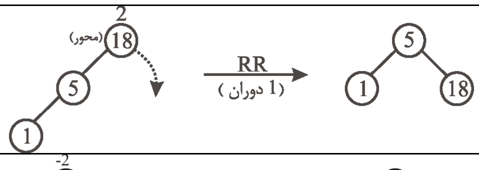
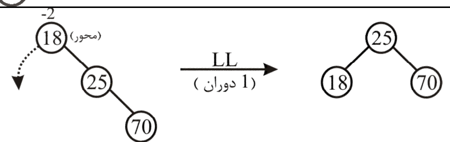
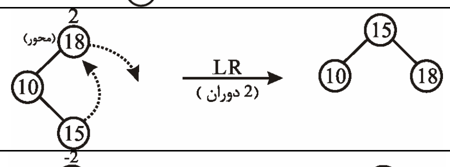
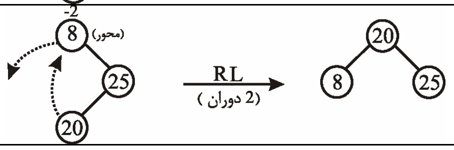
متوازن کردن درخت جستجوی نامتوازن

برای متوازن کردن درخت **جستجوی دودویی نامتوازن** از **دوران (چرخش)** درخت حول **گره محور** استفاده می‌شود.

گره محور: به اولین گره نزدیک به موقعیت درج که اختلاف ارتفاع دو زیر درخت چپ و راست آن بیشتر از **1** می‌باشد.

دوران (چرخش)

به طور کلی چهارنوع دوران با توجه به قرارگرفتن گره جدید در سمت چپ یا راست فرزندان چپ و راست گره محور وجود دارد:

دوران(چرخش)	تعداد دوران	وضعیت دوران
چرخش راست ـ راست (RR)	**۱ دوران**	
چرخش چپ ـ چپ (LL)	**۱ دوران**	
چرخش مضاعف چپ ـ راست (LR)	**۲ دوران**	
چرخش مضاعف راست ـ چپ (RL)	**۲ دوران**	

یادداشت:

....

....

....

....

:*Heap*

$Max-Heap$: یک درخت دودویی کامل که کلید هر گره از فرزندانش بزرگ‌تر است.
$Min-Heap$: یک درخت دودویی کامل که کلید هر گره از فرزندانش کوچک‌تر است.
از آنجا Heap یک درخت کامل است داریم:
۱ ـ ارتفاع Heap

$h=\left\lfloor \log_2^n \right\rfloor+1$ (سطح ریشه یک)

$h=\left\lfloor \log_2^n \right\rfloor$ (سطح ریشهٔ صفر)

۲ ـ با هر پیمایش inorder، preorder، postorder و levelorder می‌توان درخت را به‌صورت یکتا رسم کرد.
۳ ـ برای نمایش Heap از آرایه استفاده می‌شود که یک تناظر یک‌به یک بین خانه‌های آرایه و اندیس گره‌ها Heap وجود دارد.
۴ ـ اندیس پدر گرهٔ i : $\text{parent}(i)=\left\lfloor \frac{i}{2} \right\rfloor$
اندیس فرزند چپ: $\text{Left child}(i)=2i$
اندیس فرزند راست: $\text{Right child}(i)=2i+1$
۵ ـ اندیس برگ‌ها:

$$\left\lfloor \frac{n}{2} \right\rfloor+1 , \left\lfloor \frac{n}{2} \right\rfloor+2 , \ldots , n$$

لذا با $\frac{n}{2}$مقایسه در یک $Max-Heap$ و Min Heap می‌توان به ترتیب عنصر مینیمم و عنصر ماکزیمم را به‌دست آورد.
در یک $Max-Heap$ عنصر ماکزیمم همواره در ریشه و عنصر مینیمم همواره در برگ قرار دارد پس در زمان $O(1)$ عنصر ماکزیمم و در زمان $O(n)$ عنصر مینیمم قابل دسترسی‌اند.
از آنجا که فرزندان یک گره در ترتیب مشخصی قرار نمی‌گیرند، هیچ پیمایشی از Heap، الزاماً گره‌ها را به‌صورت مرتب باز نمی‌گرداند. در یک $Max-Heap$ ، ماکزیمم r ام و در یک $Min-Heap$ ، مینیمم r ام همواره می‌تواند در خانه‌های $\left[2 \ldots 2^r-1\right]$ قرار گیرد.
در یک Heap، حداکثر تعداد گره‌ها با عمق H (سطح ریشهٔ یک) برابر است با $\left\lceil \frac{n}{2^h} \right\rceil$ که اگر سطح ریشه صفر فرض شود آن‌گاه $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ می‌شود.
در صورتی‌که اعداد متمایز $1 , 2 , \ldots , n$ در یک $Min-Heap$ قرار داشته باشند، کوچک‌ترین عددی که می‌تواند در سطح آخر مشاهده شود، $\left\lfloor \log_2^n \right\rfloor+1$ خواهد بود.
در صورتی‌که اعداد متمایز $1 , 2 , \ldots , n$ در یک $Max-Heap$ قرار داشته باشند، بزرگ‌ترین عددی که می‌تواند در سطح آخر مشاهده شود، $n-\left\lfloor \log_2^n \right\rfloor$ خواهد بود.

یادداشت:
....
....
....
....

الگوریتم‌ها:

۱- الگوریتم $\mathbf{MAX\text{-}HEAPIFY(A\,,i)}$

این الگوریتم با فرض اینکه زیر درخت‌های چپ و راست گره i ام در آرایهٔ A، هر کدام یک Max – Heap هستند، عناصر آرایه A را به گونه‌ای جابه‌جا می‌کند که زیر درخت مشتق شده از $A[i]$، یک Max – Heap شود. این الگوریتم درجا است و مرتبهٔ زمانی آن $O(\log n)$است .

(برای این منظور هر گره را با فرزندانش مقایسه کرده و در صورتی‌که کوچک‌تر باشد با فرزند بزرگ‌تر جابه‌جا می‌کند و مجدداً الگوریتم به‌صورت بازگشتی فراخوانی می‌شود.)

(الگوریتم MIN – HEAPIFY، مشابه بالا است.)

۲- الگوریتم BUILD - MAX - HEAP(A)

این الگوریتم به‌صورت درجا یک آرایهٔ ورودی A را به کمک الگوریتم MAX – HEAPIFY به Max – Heap تبدیل می‌کند. که مرتبهٔ زمانی آن $O(n)$ است. بنابراین در زمان $O(n)$ می‌توان مشخص کرد آیا یک آرایه، Max – Heap است یا خیر. (به طور مشابه برای Min – Heap نیز صادق است.)

۳- جستجو: برای جستجو در Heap، باید از جستجوی خطی استفاده کرد که مرتبهٔ زمانی آن $O(n)$است.در عین حال از آنجائیکه عنصر مینیمم در MAXHEAP و ماکزیمم در MINHEAP در برگها قراردارند و تعداد برگها $\left\lfloor \frac{n+1}{2} \right\rfloor$ است زمان جستجوی آنها نیز $O(n)$است.

۴- درج: ابتدا گره در آخرین خانهٔ آرایه اضافه می‌شود سپس تا ریشه، پشت سر هم با اجدادش مقایسه می‌شود و در صورتی‌که از آنها بزرگ‌تر باشد،جابه‌جا می‌شوند که زمان $O(\log n)$ است.

۵- MAXIMUM(A) : عنصر ی از A با بزرگترین کلیدیعنی ریشه را (در MAXHEAP)بر می گرداند که زمان $O(1)$ صرف می‌کند.

۶- MINIMUM(A): عنصر ی از A با کوچکترین کلید یعنی ریشه را (در MINHEAP)بر می گرداند که زمان $O(1)$ صرف می‌کند.

۷- EXTRACT_MAX(A): عنصر ی از A با بزرگترین کلید را (در MAXHEAP)حذف کرده و بر می گرداند که زمان $O(\log n)$صرف می‌کند

۸- EXTRACT_MIN(A): عنصر ی از A باکوچکترین کلید را (در MINHEAP)حذف کرده و بر می گرداند که زمان $O(\log n)$صرف می‌کند

۹-INCREASE(A , x , k): مقدار کلید عنصر x را به مقدار جدید k افزایش می‌دهد که زمان $O(\log n)$صرف می‌کند. (که فرض شده مقدار x حداقل به بزرگی مقدار فعلی کلید عنصر x است)

۱۰- DECREASE(A , x , k): مقدار کلید عنصر x را به مقدار جدید k کاهش می‌دهد که زمان $O(\log n)$صرف می‌کند. (که فرض شده مقدار x حداقل به کوچکی مقدار فعلی کلید عنصر x است)

نکته : Heap یک ساختمان دادهٔ همه منظوره نیست و در مرتب‌سازی آرایه، صف اولولیت و ادغام لیست‌های مرتب (به کمک درخت انتخابی Min – Heap) کاربرد دارد.

۱۱-مرتب سازی Heap: ابتدا با استفاده از روال BUILD - MAX – HEAP آرایه به Heap تبدیل می‌شود (در زمان $O(n)$) سپس عناصر یکی‌یکی از Heap حذف می‌شود که حذف n عنصر زمان $O(n\log n)$ صرف می‌کند. پس زمان کل $O(n\log n)$ است.

۱۲- **ادغام k لیست مرتب:** که اگر مجموع عناصر همهٔ لیست‌ها n باشد، در زمان $O(n\log k)$ انجام می‌شود.

یادداشت:

....

....

....

....

Treap :

یک درخت دودویی است که در آن هر نود یک کلیدو یک اولویت دارد. در این ساختمان داده فرض می‌کنیم همهٔ اولویت‌ها و کلیدها متمایز هستند. یعنی اولویت یا کلید هیچ دو گره‌ای یکسان نیست. پیمایش میان ترتیب بر روی کلیدهای گره‌های یک treap ، کلیدها را به صورت مرتب شده بر می‌گرداند و اولویت هر گره از اولویت فرزندانش بزرگتر است به عبارت دیگر، کلیدهای یک treap از ویژگی درخت جستجوی دودویی و اولویت یک treap از ویژگی Heap پیروی می‌کنند (Max Heap) .

در treap گره‌ها را به صورت زوج مرتب (k,p) نشان داده می‌شوند که K نشان دهندهٔ کلید و P اولویت است. (اگر Max Heap ، Heap باشد.)

$$
\begin{array}{ccc}
 & (k_2,p_2) & \\
\swarrow & & \searrow \\
(k_1,p_1) & & (k_3,p_3)
\end{array}
$$

ترتیب کلیدها: $k_1 < k_2 < k_3$ و ترتیب اولویتها : $\begin{cases} p_2 > p_1 \\ p_2 > p_3 \end{cases}$ است.

از آن جا که فرض کردیم که همهٔکلیدها و اولویت متمایز هستند، شکل treap منحصر به فرد خواهد بود. شکل treap تنها به کلیدها و اولویت بستگی دارد و به‌ترتیب درج و حذف کلید وابسته نیست. بیشتر ساختمان داده‌ها این ویژگی را ندارد و ساختار آن‌ها به ترتیب درج و حذف عناصر بستگی دارد

کاربرد: با استفاده از treap می‌توان مشخص کرد،اگر A و B دو مجموعه باشند، آیا A=B است یا خیر. برای این کار برای عناصر در مجموعهٔ B , A به طور جداگانه treap می‌سازیم. و سپس با هم مقایسه می‌کنیم اگر treap ها یکسان بوند، آن گاه مجموعه‌ها یکی‌اند.

جستجو: (Search)

جستجو در treap مانند جستجو در درخت جستجوی دودویی است. زمان برای جستجوی موفق یک گره با عمق آن گره متناسب است زمان برای یک جستجوی ناموفق با عمق عنصر مابعد یا عنصر ماقبل آن متناسب است.

درج: (INSERT)

برای درج عنصر جدید z ، ابتدا مانند الگوریتم استاندارد درج عنصر در درخت جستجوی دودویی عمل کرده و عنصر را در پایین درخت درج می‌کنیم. حال درخت حاصل ممکن است ویژگی Heap را حفظ نکند، برای این منظور زمانی که گرهٔ z مانند درخت جستجوی دودویی درج شد، اگر اولویت آن گره از پدرش بیشتر باشد،آن گاه یک دوران روی گرهٔ z انجام می‌دهیم

حذف : (DELETE)

حذف یک گره دقیقاً برعکس درج آن است. فرض کنید، می‌خواهیم گرهٔ z را حذف کنیم. از آن جا که z برگ نیست، بر روی فرزند z که اولویت کمتری دارد، یک دوران انجام می‌دهیم. این کار باعث می‌شود z یک واحد به سمت پائین برود و فرزند با اولویت کمترش یک واحد به سمت بالا بیاید. ما باید آن فرزندی از z را برای دوران انتخاب کنیم که خواص Heap را حفظ کند.

سپس زمانی که z برگ شد، از درخت حذف می‌شود.

یادداشت:

....

....

....

....

Hash (در هم‌سازی):

برای ذخیره کرده کلیدها دو راه وجود دارد:

۱) جدول آدرس‌دهی مستقیم ۲) جدول درهم‌سازی

جدول آدرس‌دهی مستقیم: در جدول آدرس‌دهی مستقیم به ازای هر کلید در مجموعهٔ مرجع، متناظراً یک خانه در نظر گرفته می‌شود. همهٔ اعمال درج، حذف و جستجو در زمان $O(1)$ انجام می‌شود.

معایب:۱ ـ اگر تعداد کلیدهایی که قرار است ذخیره شوند کم باشد، حافظه هدر می‌رود.

۲ ـ برای مجموعه‌های مرجع خیلی بزرگ امکان‌پذیر نیست.

جدول درهم‌سازی:

جدول درهم‌سازی یک ساختمان دادهٔ همه منظوره نیست و تنها برای ذخیرهٔ اطلاعات به کار می‌رود و اعمال قابل اجرا روی آن درج، حذف و جستجو است.

هر کلید با استفاده از یک تابع درهم‌سازی به یک مکان درهم‌سازی می‌شود. کلید k به مکان $h(k)$ درهم‌سازی می‌شود.

معایب: زمانی که دو(بیشتر) عنصر به یک مکان درهم‌سازی شوند. تصادم رخ می‌دهد که برای حل این مشکل دو راه زیر پیشنهاد می‌شود:

۱ـ حل تصادم با استفاده از زنجیره‌سازی

۲ـ حل تصادم با استفاده از آدرس‌دهی باز

در تحلیل جدول درهم‌سازی از فاکتور لود α استفاده می‌شود که برابر است با:

$$\alpha = \frac{n}{m} = \frac{\text{تعداد عناصر ذخیره شده}}{\text{تعداد کل عناصر جدول درهم‌سازی}}$$

در توزیع یکنواخت کلیدها به m مکان:

$\frac{1}{m}$ = احتمال حضور هر کلید در یک مکان

$\alpha = \frac{n}{m} =$ میانگین تعداد خانه‌های مورد نیاز برای n کلید

درهم‌سازی یکنواخت ساده: هر کلید با احتمال مساوی می‌تواند در یکی از خانه‌های جدول درهم‌سازی شود.

حل تصادم با استفاده از زنجیره‌سازی:

در این روش همهٔ عناصری که به یک مکان درهم‌سازی می‌شوند در یک لیست پیوندی خارج از جدول قرار می‌گیرند که در خانهٔ متناظر آن در جدول درهم‌سازی یک اشارگر به ابتدای لیست قرار دارد.

یادداشت:

....

....

....

....

قضیه: با فرض درهم‌سازی یکنواخت ساده، متوسط زمان جستجوی موفق و ناموفق $\theta(\alpha+1)$ است. یعنی $\theta\left(\frac{n}{m}\right)$.

زمان جستجو:
- ۱) بهترین حالت: هر عنصر به یک مکان درهم‌سازی شود. $\theta(1)$
- ۲ـ حالت متوسط: $\theta\left(\frac{n}{m}\right)$
- ۳ـ بدترین حالت: همهٔ عناصر به یک مکان درهم‌سازی شوند که در این صورت زمان جستجو یک عنصر متناسب با زمان جستجوی یک عنصر در یک لیست پیوندی n عنصری است. $\theta(n)$

اگر لیست پیوندی، یک‌طرفه باشد زمان حذف با جستجو برابر می‌شود. اگر لیست پیوندی دوطرفه باشد، زمان حذف $\theta(1)$ می‌شود.

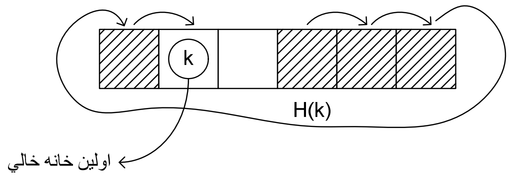
زمان درج، همواره $\theta(1)$ است. (درج ابتدای لیست پیوندی)

در روش زنجیره‌سازی، از آنجاکه هر عنصر خارج از جدول نگه‌داری می‌شود، بنابراین فاکتور لود، α، می‌تواند بیشتر، مساوی یا کمتر از 1 باشد.

حل تصادم با استفاده از آدرس‌دهی باز:

در این روش، همهٔ کلیدهای واقعی در خود جدول ذخیره می‌شوند. بنابراین در جدول درهم‌سازی یا یک کلید واقعی است یا NIL. لذا در این روش فاکتورلود $\alpha=\frac{n}{m}$ می‌تواند حداکثر 1 شود. زمانی‌که $\alpha=1$ جدول پر است.

در آدرس‌دهی باز برای درج یک کلید، با استفاده از وارسی خطی، ابتدا مکانی که به آن درهم‌سازی شده است بررسی می‌گردد و اگر پر باشد آن‌گاه خانه‌های بعدی آن‌قدر مورد بررسی قرار می‌گیرند تا اولین خانهٔ خالی پیدا بشود که کلید در آن قرار بگیرد.



در روش آدرس‌دهی باز با فرض تابع درهم‌سازی یکنواخت ساده، متوسط بررسی‌ها در یک جستجوی ناموفق حداکثر $\frac{1}{1-\alpha}$ است.

در روش آدرس‌دهی باز با فرض تابع درهم‌سازی یکنواخت ساده، درج یک عنصر با فاکتور لود α، حالت میانگین حداکثر به $\frac{1}{1-\alpha}$ بررسی نیاز دارد.

در روش آدرس‌دهی باز با فرض تابع درهم‌سازی یکنواخت ساده و $\alpha<1$، تعداد بررسی‌ها در یک جستجوی موفق حداکثر برابر است با $\frac{1}{\alpha}\mathrm{Ln}\frac{1}{1-\alpha}$.

یادداشت:

....

....

....

....

مرتب‌سازی

روش مرتب‌سازی	بهترین حالت	حالت متوسط	بدترین حالت	تعداد گذر	توضیحات
۱- حبابی	$O(n^2)$ آرایه‌ی ورودی مرتب	$O(n^2)$	$O(n^2)$ آرایه‌ی ورودی مرتب معکوس	$n-1$	متعادل ـ درجا
۲- انتخابی	$O(n^2)$ آرایه‌ی ورودی مرتب	$O(n^2)$	$O(n^2)$ آرایه‌ی ورودی مرتب معکوس	$n-1$	نامتعادل ـ درجا
۳- درجی	$O(n)$ آرایه‌ی ورودی مرتب	$O(n^2)$	$O(n^2)$ آرایه‌ی ورودی مرتب معکوس	$n-1$	متعادل ـ درجا

- برای آرایه‌های کوچک و تقریباً مرتب، مرتب‌سازی درجی بهترین حالت است.

روش مرتب‌سازی	بهترین حالت	حالت متوسط	بدترین حالت	معادله بازگشتی	توضیحات
۴- ادغامی	$O(n\log n)$ تعداد مقایسه: $\frac{n}{2}\log n$	$O(n\log n)$	$O(n\log n)$ تعداد مقایسه: $n\log n-n+1$	$T(n)=2T\left(\frac{n}{2}\right)+O(n)$	غیر درجا ـ متعادل تقسیم و غلبه
۵- سریع	$O(n\log n)$ $T(n)=2T\left(\frac{n}{2}\right)+O(n)$	$O(n\log n)$	$O(n^2)$ $T(n)=T(n-1)+O(n)$ آرایه‌ی ورودی مرتب یا مرتب معکوس		درجا ـ متعادل تقسیم و غلبه

- در حالت متوسط مرتب‌سازی سریع از مرتب‌سازی ادغامی بهتر است.
- از مرتب‌سازی ادغامی می‌توان برای لیست ورودی یکطرفه و دوطرفه استفاده کرد که در این صورت مرتبه‌ی زمانی آن $O(n\log n)$ می‌باشد.
- در الگوریتم مرتب‌سازی سریع از PARTITION و در الگوریتم مرتب‌سازی ادغامی از MERGE استفاده می‌شود که مرتبه‌ی زمانی هر دو $O(n)$ است.

یادداشت:

....

....

....

....

توضیحات	مرتبه زمانی	شرایط ورودی	روش مرتب‌سازی
غیر درجا ـ با برقراری شرایط ورودی در زمان خطی اجرا می‌شود. پایدار	$O(n+k)$	عناصر ورودی اعداد صحیح در بازه‌ی $[0,k]$ که $k=O(n)$	۶- مرتب‌سازی شمارشی
غیر درجا ـ پایدار با برقراری شرایط ورودی و اگر $k=O(n)$ آن‌گاه در زمان خطی اجرا می‌شود.	$O(d(n+k))$	عناصر ورودی اعداد صحیح d رقمی در مبنای k	۷- مرتب‌سازی مبنایی
با فرض شرایط ورودی در زمان خطی اجرا می‌شوند غیر درجا ـ پایدار	$O(n)$	عناصر وردی اعداد در بازه $[0,1]$ که با توزیع یکنواخت در پیمانه‌ها، توزیع می‌شوند	۸- مرتب‌سازی پیمانه‌ای

- در مرتب‌سازی مبنایی، مرتب‌سازی میانی حتماً باید پایدار باشد، لذا مرتب‌سازی شمارشی انتخاب مناسبی هستند.
- مرتب‌سازی‌ها 1 تا 5، مرتب‌سازی مقایسه‌ای هستند، یعنی برای مرتب کردن عناصر ورودی از عملگر مقایسه استفاده می‌شود. ثابت می‌شود که حد پایین تعداد مقایسه، در الگوریتم های مرتب‌سازی مقایسه‌ای در بدترین حالت $\Omega(n\log n)$ است. اما مرتب‌سازی‌های 6، 7، 8 مرتب‌سازی مقایسه‌ای نیستند این مرتب‌سازی‌ها از عملی غیر از مقایسه، استفاده می‌کنند، لذا قابل اجرا در زمان خطی هستند.

یادداشت:

....

....

....

....

تعدادی از تست های سال های گذشته کنکور سراسری

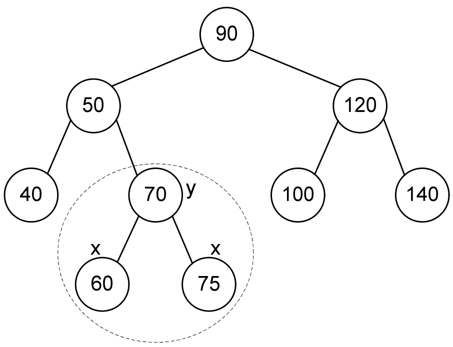
درخت و گراف

۱ـ در درخت جستجوی دودویی T**، اگر** x **گره‌ی برگ و** y **پدر** x **باشد، کدام گزینه در مورد** $a = key[x]$, $b = key[y]$ **صحیح است؟**

۱) b بزرگ‌ترین کلید در T است که کوچک‌تر از a باشد.
۲) a کوچک‌ترین کلید در T است که بزرگ‌تر از b باشد.
۳) b کوچک‌ترین کلید در T است که بزرگ‌تر از a باشد.
۴) هیچ‌کدام از گزینه‌های بالا همواره صحیح نیست.

حل : گزینه ۴ درست است.

درخت جستجوی دودویی زیر را در نظر بگیرید.



اگر گره x فرزند راست باشد آنگاه گزینه های ۱و۲ صحیح خواهند بود و اگر گره x فرزند چپ باشد گزینه ۳ صحیح خواهد بود، حال از آنجائیکه در سوال موقعیت گره x نسبت به پدر مشخص نیست جواب صحیح گزینه ۴ است

۲ ـ گزینهٔ نادرست را انتخاب کنید؟

۱) با داشتن پیمایش inorder یک درخت جستجوی دودویی می‌توان آن درخت را به‌صورت یکتا رسم کرد.

۲) با داشتن هر یک از پیمایش‌های preorder یا postorder یک درخت کامل، می‌توان آن را به‌صورت یکتا رسم کرد.

۳) با داشتن پیمایش‌های inorder و preorder هر درخت دودویی می‌توان آن را به‌صورت یکتا رسم کرد.

۴) با داشتن هر یک از پیمایش‌های Preorder یا Postorder یک درخت جستجوی دودویی، می‌توان آن درخت را به‌صورت یکتا رسم کرد.

حل : گزینه ۱ درست است.

به طور کلی با داشتن هر یک از جفت پیمایش‌های (inorder , preorder) یا (Postorder , inorder) یا (levelorder , inorder) یک درخت دودویی می‌توان آن را به‌صورت یکتا رسم کرد.

در صورتی‌که درخت کامل باشد. از آنجا که شکل درخت مشخص است با هر پیمایشی می‌توان درخت را به‌صورت یکتا رسم کرد.

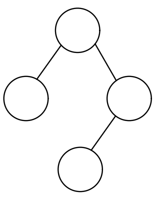
در یک درخت جستجوی دودویی به جز پیمایش inorder، با هر پیمایش دیگر می‌توان درخت را به‌صورت یکتا رسم کرد.

نکته : در هر درخت دودویی تنها با داشتن پیمایش‌های Preorder و Postorder بدون هیچ اطلاع اضافی نمی‌توان درخت را به‌صورت یکتا رسم کرد.

یادداشت:

....
....
....
....

۳ ـ به چند حالت عناصر با کلیدهای $a < b < c < d$ را می‌توان وارد یک درخت جستجوی دودویی تهی کرد تا درختی بــه شــکل زیــر ایجاد شود؟



۱) 1

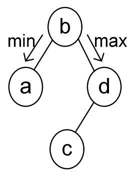
۲) 2

۳) 3

۴) 4

حل : گزینه ۳ درست است.

شکل کلی درخت با چهار کلید $a < b < c < d$ به این صورت است:



در ورودی‌های مجاز برای ساختن هر درخت جستجو ، گره والد باید زودتر از فرزندانش در ورودی قرار گیرد ، به عبارت دیگر در اینجا گره b باید قبل از گره های a,d و گره d باید قبل از گره c در ورودی قرار گیرد.

ورودی‌های مجاز:

b , a , d , c

b , d , a , c

b , d , c , a

۴ ـ کدام گزینه تعداد درخت‌های جستجوی دودویی با n گره را نشان نمی‌دهد؟ (b_n تعداد درختان جستجوی دودویی است.)

۱) $b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k}$

۲) $b_n = \binom{2n}{n}\frac{1}{n+1}$

۳) $b_n = \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}}\left(1 + O\left(\frac{1}{n}\right)\right)$

۴) $2^{(n-1)}$

حل : گزینه ۴ درست است.

اولا : تعداد درختان جستجوی دودویی با n گره با تعداد درختان دودویی با n گره برابر است.

ثانیا : می‌توان ثابت کرد تعداد درختان دودویی با n گره $b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k}$ است.

که با استفاده از تابع مولد و بسط تیلور ثابت می‌شود:

$$b_n = \binom{2n}{n}\frac{1}{n+1}$$

هم‌چنین ثابت می‌شود:

$$b_n = \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}}\left(1 + O\left(\frac{1}{n}\right)\right)$$

ثالثا: $2^{(n-1)}$، تعداد درختان جستجوی دودویی متفاوت به ارتفاع $n-1$ است.

یادداشت:

....

....

....

....

۵ ـ در یک درخت جستجوی دودویی با n گره و ارتفاع h گزینهٔ نادرست کدام است؟

۱) مرتبهٔ زمانی هر یک از پیمایش‌های inorder، Preorder و Postorder، $\theta(n)$ است.

۲) پیمایش inorder را می‌توان ابتدا با یک فراخوانی TREE - MINIMUM و سپس $n-1$ فراخوانی TREE - SUCCESSOR پیاده‌سازی کرد که مرتبهٔ آن $\theta(n \log n)$ است.

۳) مرتبهٔ زمانی هر یک از عملیات جستجو، حذف، درج، عنصر ما بعد و عنصر ما قبل یک گره‌ی x، $O(h)$است.

۴) می‌توان برای مرتب‌سازی n کلید، ابتدا آن‌ها را یکی‌یکی در یک درخت جستجوی تهی درج کرده و سپس از پیمایش inorder استفاده کنیم.

حل : گزینه ۲ درست است.

ابتدا به توضیح گزینه‌های ۱، ۳ و ۴ می‌پردازیم:

گزینهٔ ۱: مرتبهٔ زمانی هر یک از الگوریتم‌های پیمایش inorder، preorder و postorder، $\theta(n)$ است.

گزینهٔ ۳: مرتبهٔ زمانی هر یک از اعمال جستجو، حذف، درج، عنصر مابعد و عنصر ماقبل یک گرهٔ x، $O(h)$ که h ارتفاع درخت است.

(توجه داشته باشید که در هر یک از اعمال فوق، یک مسیر از ریشه تا حداکثر یک برگ طی می‌شود بنابراین مرتبهٔ زمانی آن‌ها $O(h)$ است در حالی‌که در الگوریتم‌های پیمایش درخت، هر گره حداقل یک‌بار بررسی می‌شود پس مرتبهٔ آن $\theta(n)$است)

گزینهٔ ۴: برای مرتب‌سازی n کلید، می‌توان ابتدا آن‌ها را یکی‌یکی در یک درخت تهی درج کرد که مرتبهٔ زمانی آن برابر است با:

$n\,O(h) = O(nh)$

از طرفی چون ارتفاع یک درخت جستجوی دودویی در حالت متوسط $O(\log n)$ است، درج n کلید برابر است با: $O(n \log n)$

در ادامه می‌توان با استفاده از پیمایش inorder، کلیدها را مرتب کرد. که در این صورت مرتبهٔ زمانی آن برابر است با:

$O(n \log n) + \theta(n) = O(n \log n)$

که به این مرتب‌سازی، مرتب‌سازی درختی نیز گفته می‌شود.

گزینهٔ ۲: پیمایش inorder را می‌توان ابتدا با یک فراخوانی TREE – MINIMUM و سپس $n-1$ فراخوانی TREE – SUCCESSOR پیاده‌سازی کرد، که در این صورت مرتبه‌ی زمانی آن $\theta(n)$ خواهد بود نه $\theta(n \log n)$.

۶ ـ تابع زیر چک می‌کند که آیا یک درخت دودویی با عناصر صحیح (int) و متمایز و با ریشهٔ root یک درخت جستجوی دودویی است یا خیر؟

```
Bool IsBST(tree*t)
{
  Return IsBST (t,MININT , MAXINT);
}
Int  IsBST (tree*t , int min , int max)
{
  if (t = = Null) return  TRUE;
  if (t → data < min || t → data > max) return FALSE;
  Return IsBST (t → Left Child , ....A.... , ....B....) & & IsBST (t → Right Child ....C.... , ....D....);
```

یادداشت:

....

....

....

....

کدام گزینه باید در جای خالی A و B و C و D قرار داده شود؟

۱) $D = min$ $\quad C = t \to data + 1$ $\quad B = t \to data$ $\quad A = max$

۲) $D = max$ $\quad C = t \to data + 1$ $\quad B = t \to data$ $\quad A = min$

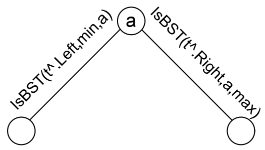
۳) $D = t \to data + 1$ $\quad C = min$ $\quad B = max$ $\quad A = t \to data$

۴) $D = t \to data + 1$ $\quad C = max$ $\quad B = min$ $\quad A = t \to data$

حل : گزینه ۲ درست است.

در یک درخت جستجوی دودویی:

این الگوریتم بازگشتی است. ابتدا بررسی می‌شود که ریشه تهی نباشد. متغیرهای min و max به ترتیب حاوی کوچک‌ترین کلید و بزرگ‌ترین کلید هستند. زمانی که از ریشه به زیر درخت چپ می‌رویم بنا به خاصیت BST، min باید همان minint باقی بماند در حالی‌که max باید ریشه شود و زمانی که از ریشه به زیر درخت راست می‌رویم بنا به خاصیت BST، max باید همان maxint باقی بماند در حالی‌که min باید ریشه شود. سپس به‌صورت بازگشتی الگوریتم برای دو زیر درخت، چپ و راست فراخوانی می‌شود. به گونه‌ای فرزند چپ ریشه به عنوان ریشهٔ زیر درخت چپ و فرزند راست ریشه به عنوان ریشهٔ زیر درخت راست در نظر گرفته می‌شوند.

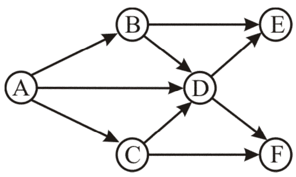


توجه داشته باشید، از آنجا که فرض شده کلیدها متمایز هستند، زمانی‌که به زیر درخت چپ ریشه می‌رویم، max می‌تواند به جای data، $t \to data - 1$ شود یا زمانی که به زیر درخت راست می‌رویم. Min می‌تواند $t \to min+1$ شود.

۷ ـ کدام گزینه یک ترتیب درست از رأس‌های گراف زیر را که در جستجوی اول ـ عمق (Depth-First Search) علامت ملاقات شدن (Visited) می‌خورند، نشان نمی‌دهد؟

۱) ABCDEF $\qquad$ ۲) ABEDFC

۳) ABDFEC $\qquad$ ۴) ADEFBC



حل : گزینه ۱ درست است.

با توجه به گزینه‌ها از رأس A شروع می‌کنیم.

گزینه ۲:

پشته	رئوس ملاقات شده	مرحله
B, C, D	A	1
E, D, C	A , B	2

یادداشت:

....

....

....

....

3	A , B , E	D C D
4	A , B , E , D	F C D
5	A , B , E , D , F	C D
6	A , B , E , D , F , C	D
7	A , B , E , D , F , C	

(چون D قبلاً ملاقات شده است،
آن را مجدداً نمی‌نویسیم)

گزینه ۳:

1	A	B C D
2	A , B	D E C D
3	A , B , D	F E C D
4	A , B , D , F	E C D

یادداشت:

....

....

....

....

5	A , B , D , F , E	C D
6	A , B , D , F , E , C	D
7	A , B , D , F , E , C	

(چون D قبلاً ملاقات شده، مجدداً آن را نمی‌نویسیم)

گزینه ۴:

1	A	D B C
2	A , D	E F B C
3	A , D , E	F B C
4	A , D , E , F	B C
5	A , D , E , F , B	C
6	A , D , E , F , B , C	

یادداشت:

....

....

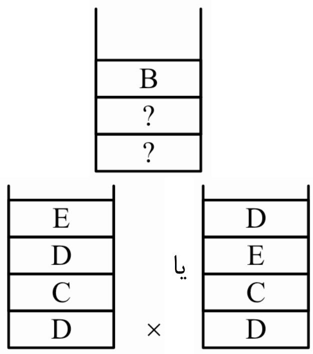
....

....

گزینه ۱:

1   A

2   A , B



پس از ملاقات گرمای B، باید یکی از گره E یا D ملاقات شوند.

۸ ـ اگر پیمایش Preorder **یک درخت جستجوی دودویی به‌صورت زیر باشد، آن‌گاه پیمایش پس ترتیب آن کدام است؟**

Preorden : $\overline{40}$, 30 , 10 , 20 , 80 , 70 , 50 , 60

۱) 40 , 10 , 20 , 30 , 50 , 70 , 60 , 80

۲) 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80

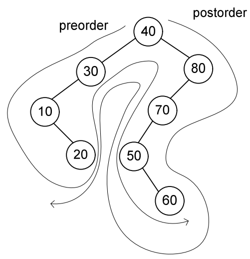
۳) 20 , 10 , 30 , 60 , 50 , 70 , 80 , 40

۴) 40 , 30 , 10 , 20 , 80 , 70 , 50 , 60

حل : گزینه ۳ درست است.

با هر یک از پیمایش‌های Postorder , Preorder و Levdorder یک درخت جستجوی دودویی می‌توان آن را به‌صورت یکتا رسم کرد.

با پیمایش Preorder داده شده، شکل درخت به‌صورت زیر خواهد بود:



Preorder : 40 , 30 , 10 , 20 , 80 , 70 , 50 , 60

Postorder : 20 , 10 , 30 , 60 , 50 , 70 , 80 , 40

راه حل تستی: در روش پس‌ترتیب ریشه آخرین گره‌ای است که ملاقات می‌شود، از طرفی در پیمایش پیش‌ترتیب اولین گره‌ای است که در ابتدا ملاقات می‌شود بنابراین 40 ریشه است، پس باید در گزینه‌ها، آن گزینه‌ای که 40 در آخر دیده ملاقات شده است، انتخاب شود.

۹ ـ ساختمان داده‌ای را بر روی مجموعهٔ A **از اعداد صحیح در نظر بگیرید. که اعمال درج** (insert)**، حذف** (delete) **و پیدا کردن نزدیک‌ترین** (Find closest) **را فراهم می‌آورد. منظور از** Find closest **پیدا کردن** $y \in A$ **است که** $(x-y)$ **نسبت به بقیه** y**ها کمینه باشد. فرض کنید** T **بیشترین زمان اجرای اعمال فوق در بدترین حالت باشد در این صورت کدام ساختمان داده کم‌ترین مقدار** T **خواهد داشت؟**

۱) درخت جستجوی دودویی متوازن

۲) لیست نامرتب

۳) لیست مرتب

۴) heap

یادداشت:

....

....

....

....

حل : گزینه ۱ درست است.

به بررسی تک‌تک ساختمان داده‌ها می‌پردازیم:

درخت جستجوی دودویی متوازن:

اعمال حذف و درج در یک درخت جستجوی دودویی با ارتفاع h، $O(h)$ است. از آنجا که ارتفاع یک درخت جستجوی دودویی متوازن همواره $\theta(\log n)$ است، لذا هر یک از اعمال حذف و درج در زمان $\theta(\log n)$ اجرا می‌شوند. از طرفی برای گره‌ی x عمل Find closest، y یا عنصر مابعد یا عنصر ماقبل است، بنابراین پیدا کردن عنصر y در زمان $\theta(\log n)$ انجام می‌شود. لذا هر سه‌ی این اعمال در این ساختمان داده در زمان $\theta(\log n)$ قابل اجرا است.

لیست نامرتب:

درج در یک لیست نامرتب در زمان $\theta(1)$انجام پذیر است. در حالی‌که برای حذف زمان $\theta(n)$ صرف می‌کند.(n تعداد عناصر لیست) از طرفی برای پیدا کردن گره‌ی y برای عمل Find closest، نیاز به جستجوی خطی در لیست است و زمان $\theta(n)$ نیاز دارد.

لیست مرتب:

حذف در یک لیست مرتب در زمان $\theta(1)$ انجام پذیر است. در حالی که برای درج زمان $O(n)$ صرف می‌کند. (n تعداد عناصر لیست) از طرفی برای پیدا کردن گره‌ی y برای عمل Find closest (x)، گره بعدی در زمان $\theta(1)$ و گرهٔ قبلی با پیمایش از ابتدا در زمان $\theta(n)$ به‌دست می‌آید.

Heap:

درج و حذف یک گره در heap در زمان $\theta(\log n)$ انجام می‌شود ولی از آنجا که فرزندان یک گره ترتیب مشخصی ندارند برای پیدا کردن y برای Find closest (x) نیاز به جستجوی خطی در آرایه است که زمان $\theta(n)$ صرف می‌کند.

۱۰ ــ یک Max – Heap با n عنصر به‌صورت آرایه پیاده‌سازی شده است. (عنصر ماکزیمم در ریشه است). مناسب‌ترین گزینه را برای پیدا کردن عنصر مینیمم در این ساختمان داده کدام است؟

۱) این کار را همواره می‌توان با $O(\log n)$ مقایسهٔ بین عناصر heap انجام داد.

۲) این کار حداکثر به 2 مقایسه بین عناصر heap نیاز دارد.

۳) این کار ممکن است به $n-1$ مقایسه بین عناصر heap نیاز داشته باشد.

۴) تنها در صورتی‌که heap عناصر تکراری نداشته باشد، می‌توان این کار را با $O(\log n)$ مقایسه بین عناصر heap انجام داد.

حل : گزینه ۲ درست است.

در Max – Heap عنصر مینیمم قطعاً در برگ است. چرا که هیچ گره با کلید کمتری وجود ندارد که فرزند آن باشد، از طرفی اندیس برگ‌ها در یک heap با n عنصر $\left\lfloor \frac{n}{2} \right\rfloor+1, \left\lfloor \frac{n}{2} \right\rfloor+2, \ldots, n$ است. لذا تنها باید بین خانه‌های آرایه با این اندیس‌ها مقایسه انجام شود

بنابراین تنها در 2 مقایسه عنصر مینمم پیدا می‌شود.

۱۱ ــ در یک Min – Heap با N عنصر متمایز، که با یک آرایه پیاده‌سازی شده است، سومین کوچک‌ترین عنصر در کدام یک از درایه‌های زیر نمی‌تواند قرار بگیرد؟

۱) $6, 7, 8$　　۲) $4, 5, 7$　　۳) $2, 3, 7$　　۴) $3, 5, 7$

حل : گزینه ۱ درست است.

یادداشت:

....

....

....

....

به طور کلی در یک Min – Heap با ارتفاع h (یا یک Max – Heap با ارتفاع h) مینمم rام (یا ماکزیمم rام) می‌تواند در خانه‌های $\left[2 \ldots 2^r - 1\right]$ قرار بگیرد.

بنابراین طبق سؤال، سومین کوچک‌ترین عنصر می‌تواند در خانه‌های $\left[2 \ldots 2^3 - 1\right]$ یعنی $[2 \ldots 7]$ قرار بگیرد.

۱۲ ـ گزینهٔ نادرست را انتخاب کنید؟

۱) در هر heap با n گره، هر یک از اعمال درج، حذف و افزایش کلید یک گره، زمانی $O(\log n)$ صرف می‌کند.

۲) در هر heap با n گره، حداکثر $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ گره با ارتفاع h وجود دارد.

۳) از heap، در مرتب‌سازی آرایه، صف اولویت و ادغام لیست‌های مرتب (به کمک درخت انتخابی Min – Heap) استفاده می‌شود.

۴) ادغام k لیست مرتب با استفاده از Min – Heap که مجموع کل عناصر k لیست n باشد، در زمان $O(nk)$ انجام می‌شود.

حل : گزینه ۴ درست است.

در یک heap، هر یک از اعمال زیر را در زمان $O(\log n)$ انجام می‌شود:

INCREASE(x): مقدار کلید گرهٔ x را افزایش می‌دهد.

INSERT(x): درج گره با کلید x در یک heap.

DELETE(x):حذف گرهٔ ریشه از یک heap.

در هر heap با n گره، حداکثر $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ گره با ارتفاع h وجود دارد.

کاربرد heap در صف اولویت، ادغام لیست‌های مرتب (به کمک درخت انتخابی Min – Heap) و مرتب‌سازی آرایه است.

ادغام k لیست مرتب که مجموع کل عناصر n باشد، زمان $O(n\log k)$ صرف می‌کند.

۱۳ ـ در یک Max – Heap خالی به ترتیب گره‌هایی با کلیدهای (از راست به چپ) $\overleftarrow{40}$, 60 , 30 , 50 , 70 , 20 , 10 , 80 درج می‌شوند، سپس عمل حذف 3 بار انجام می‌شود. پیمایش پس ترتیب درخت حاصل مطابق با کدام گزینه است؟

۱) 50 , 40 , 20 , 10 , 30

۲) 20 , 40 , 10 , 50 , 30

۳) 20 , 10 , 40 , 30 , 50

۴) 50 , 40 , 30 , 20 ,10

یادداشت:

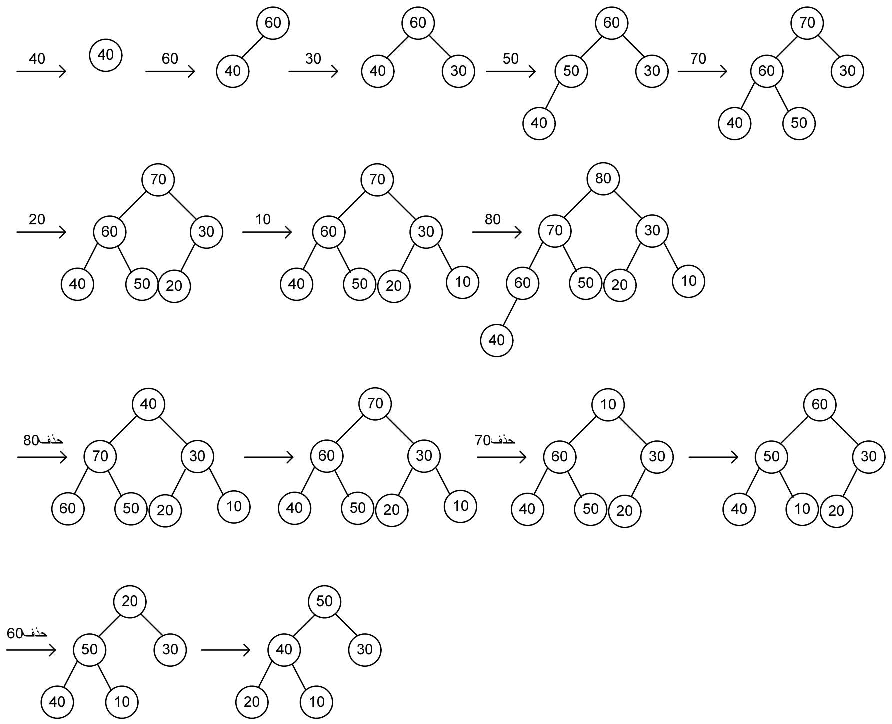
....

....

....

....

حل : گزینه ۳ درست است.



$Postorder : 20 , 10 , 40 , 30 , 50$: پیمایش

راه‌حل تستی: پس از درج کل کلیدها، درخت حاصل Max-Heap خواهد بود. حذف هر کلید از درخت، ریشه (بزرگ‌ترین کلید) را حذف می‌کند در نتیجه پس از عمل حذف، 3 تا عنصر بزرگ‌ترین به‌ترتیب حذف می‌شود و چهارمین بزرگ‌ترین عنصر در ریشه قرار می‌گیرد. از آنجا که در پیمایش پس‌ترتیب ریشه آخرین گره‌ای است که ملاقات می‌شود پس چهارمین بزرگ‌ترین عنصر، باید آخرین گره‌ای باشد که ملاقات می‌شود. یعنی باید در گزینه‌ها، آنجا که 50 (چهارمین بزرگ‌ترین عنصر)، در آخر آمده (گزینهٔ ۳) انتخاب می‌کنیم.

۱۴ ـ گزینهٔ نادرست را انتخاب کنید؟

۱) با داشتن هر پیمایشی از یک heap می‌توان آن را به‌صورت یکتا رسم کرد.

۲) اگر در یک درخت جستجوی دودویی تهی، عناصر به ترتیب صعودی یا نزولی درج شوند، ارتفاع درخت برابر با تعداد گره‌های درخت می‌شود.

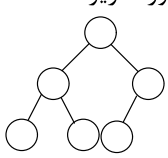
۳) اعمال درج و حذف در یک درخت جستجوی دودویی می‌تواند در زمان $O(1)$ انجام شود.

۴) در یک heap، خروجی یکی از پیمایش‌های میان ترتیب، پس‌ترتیب، سطح ترتیب یا پیش‌ترتیب حتماً به‌صورت مرتب است.

حل : گزینه ۴ درست است.

از آنجا که heap یک درخت کامل است. با هر پیمایشی می‌توان درخت آن را به‌صورت یکتا رسم کرد. به‌عنوان مثال: اگر پیمایش پیش‌ترتیب زیر (از چپ به راست) مربوط به یک Max – Heap باشد، آن‌گاه می‌دانیم شکل یک درخت کامل با 6 گره به‌صورت زیر است:

$Preorder : 80 , 60 , 30 , 50 , 70 , 20$



یادداشت:

....

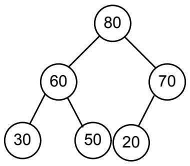
....

....

....

در نتیجه درخت به‌صورت زیر خواهد بود:

که سایر پیمایش‌ها:

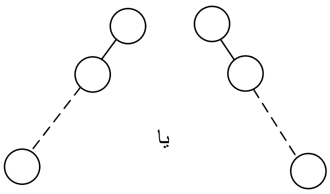


Postorder : 30 , 50 , 60 , 20 , 70 , 80

inorder : 30 , 60 , 50 , 80 , 20 , 70

Levelorder : 80 , 60 , 70 , 30 , 50 , 20

همان‌طور که مشخص است خروجی هیچ‌کدام از 4 پیمایش، کلیدها را به‌صورت مرتب تحویل نمی‌دهد. در نتیجه گزینهٔ 4 نادرست است.



اگر در یک درخت جستجوی دودویی تهی عناصر در ترتیب صعودی یا نزولی درج شوند، از آنجا هر کلید از کلید و کلیدهای قبلی همواره بزرگ‌تر یا کوچک‌تر است، پس هر کلید همیشه یا در زیر درخت راست یا در زیر درخت چپ درج می‌شود، پس شکل درخت به‌صورت زیر خواهد بود که در این‌صورت ارتفاع درخت $\theta(n)$، n تعداد گره‌های درخت خواهد شد.

در درخت‌های اریب حذف ریشه $O(1)$ است. همچنین درج یک گرهٔ بزرگ‌تر از ریشه در BST اریب به چپ، و درج یک گرهٔ کوچک‌تر از ریشه در BST اریب به راست $O(1)$ است.

۱۵ ـ هیپ زیر داده شده است:

$A[1...18] = 20 \quad 15 \quad 18 \quad 7 \quad 9 \quad 14 \quad 16 \quad 3 \quad 6 \quad 8 \quad 4 \quad 13 \quad 10 \quad 12 \quad 11 \quad 1 \quad 2 \quad 5$

عمل Change(i,k) کلید $A[i]$ را به k تغییر می‌دهد و با انجام تعداد جابه‌جایی کاری می‌کند که آرایه مجدداً به‌صورت هیپ درآید. ما این 2 عمل را به‌ترتیب انجام می‌دهیم:

$\text{Change}(11,16)\ \text{Change}(2,4)$

مجموع تعداد جابه‌جایی (swap)ها چند تاست؟

۱) 3   ۲) 4   ۳) 5   ۴) 6

حل : گزینه ۲ درست است

در صورتیکه درخت را رسم کنیم جابجایی ها به شکل زیر انجام خواهد شد:

$\text{change}(11,16) \xrightarrow{A[11]=4 \text{ change to } 16} \text{swap} : (16,9) \text{ and } (16,15)$

$\text{change}(2,\ 4) \xrightarrow{A[2]=16 \text{ change to } 4} \text{swap} : (4,15) \text{ and } (4,9)$

درحقیقت با تغییر اول کلید ۱۶ به خاطر افزایش مقدار با مقایسه با اجدادش به سمت بالا می رود و با تغییر دوم کلید ۴ به خاطر کاهش مقدار با مقایسه با فرزندانش به سمت پائین می رود

۱۶ ـ در یک درخت دودویی کامل با ارتفاع h، چند گره وجود دارد؟

۱) 2^h گره   ۲) 2^{h+1} گره   ۳) بین 2^{h-2} و 2^{h-1} گره   ۴) بین 2^{h-1} و 2^h گره

حل : گزینه ۴ درست است.

یادداشت:

....

....

....

....

یک درخت کامل به ارتفاع h، حتماً تا ارتفاع $h-1$ پر است. از طرفی تعداد گره‌ها در یک درخت پر با ارتفاع h برابر است با:

$$n = 2^h - 1$$

بنابراین، تعداد گره‌های درخت تا ارتفاع $h-1$، $2^{h-1}-1$ است. لذا با اضافه شدن یک گره به درخت، ارتفاع درخت h می‌شود. پس کران پایین تعداد گره‌های یک درخت کامل با ارتفاع h برابر است با:

$$2^{h-1} - 1 + 1 = 2^{h-1}$$

از طرفی بیشترین تعداد گره‌های یک درخت کامل با ارتفاع h برابر با تعداد گره‌های یک درخت پر با ارتفاع h است. پس کران بالای تعداد گره‌های یک درخت کامل ارتفاع h برابر است با:

$$n = 2^h - 1$$

$2^{h-1} \leq$ تعداد گره‌های یک درخت دودویی کامل با ارتفاع h $\leq 2^h - 1$

بنابراین گزینهٔ ۴ درست است.

۱۷ ـ کدام‌یک از گزینه‌های زیر در مورد یک درخت 5 تایی کامل با 95 گره صحیح است؟ (سطح ریشه را یک فرض کنید و در هر سطح گره‌ها از چپ به راست پر می‌شود.)

۱) این درخت 76 برگ دارد و گرهٔ پانزدهم آن پدر گرهٔ هفتاد و دوم آن است.

۲) این درخت 76 برگ دارد و گرهٔ چهاردهم آن پدر گرهٔ هفتاد و دوم آن است.

۳) این درخت 77 برگ دارد و گرهٔ چهاردهم آن پدر گرهٔ هفتاد و دوم آن است.

۴) این درخت 77 برگ دارد و گرهٔ پانزدهم آن پدر گرهٔ هفتاد و دوم آن است.

حل : گزینه ۱ درست است.

تعداد برگ‌ها در یک درخت k تایی با n گره برابر است با:

$$n_0 = \left\lfloor \frac{n(k-1)+1}{k} \right\rfloor$$

با توجه به سؤال:

$$n_0 = \left\lfloor \frac{95(5-1)+1}{5} \right\rfloor = 76$$

از طرفی در یک درخت k تایی کامل یا پر:

اندیس پدر گرهٔ i برابر است با:

$$\text{parent}(i) = \left\lceil \frac{i-1}{k} \right\rceil$$

اندیس فرزندان گرهٔ i برابر است با: (در صورتی که از n کوچک‌تر باشند)

$$\underbrace{\ldots\ldots, ki-2, ki-1, ki, ki+1}_{\text{تا } k}$$

بنابراین پدر گرهٔ 72 ام برابر است با:

$$\text{parent}(72) = \left\lceil \frac{72-1}{5} \right\rceil = 15$$

بنابراین گزینهٔ ۱ درست است.

یادداشت:

....

....

....

....

۱۸ ـ تعداد درخت‌های دودویی جستجویی که می‌توان با 36 کلید متمایز داده شده به‌صورت مجزا ساخت به‌طوری‌که اختلاف عمق برگ‌های آن درخت حداکثر 1 باشد، چند تاست؟

۱) $\binom{16}{5}$ ۲) $\binom{32}{5}$ ۳) $\binom{3}{5}$ ۴) $51\binom{36}{32}$

حل : گزینه ۲ درست است.

می‌دانیم تعداد درخت‌های جستجوی دودویی متمایز با n گره با تعداد درخت‌های دودویی با n گره برابر است.

برای آن که اختلاف عمق برگ‌های این درخت حداکثر 1 باشد، شکل درخت باید شبیه درخت کامل باشد، یعنی اگر ارتفاع را h فرض کنیم، درخت باید تا ارتفاع $h-1$ پر بوده و در سطح آخر گره‌های باقی‌مانده در هر مکان دلخواه قرار بگیرند. بنابراین ارتفاع این درخت با ارتفاع یک درخت کامل با 36 گره برابر است. پس:

$$h=\left\lfloor \log_2^n \right\rfloor+1=\left\lfloor \log_2^{36} \right\rfloor+1=6$$

پس درخت باید تا ارتفاع 5 پر باشد. که تعداد این گره‌ها برابر است با: $n=2^5-1=32-1=31$

گره‌های باقیمانده برابر است با:

$$\underset{\text{کل گره‌ها}}{36}-\underset{\substack{\text{تعداد گره تا ارتفاع}\\ h-1}}{31}=\underset{\text{تعداد گره‌ها در سطح h ام}}{5}$$

از طرفی در سطح i، تعداد مکان‌های موجود برای گره‌ها برابر است با: 2^{i-1}

لذا در سطح ششم (h ام) $2^{6-1}=32$ مکان وجود دارد که باید از بین آن‌ها 5 تا برای گره‌های باقی‌مانده انتخاب شود. پس تعداد کل درخت‌های متمایز با توجه به شرایط سؤال برابر است با: $\binom{32}{5}$

۱۹ ـ در یک درخت $n-ary$، هر گره حداکثر n فرزند می‌تواند داشته باشد، در درخت $n-ary$ با k گره و ارتفاع h، کدام‌یک از روابط زیر حد بالایی برای تعداد برگ‌های درخت می‌باشد؟

۱) h^n ۲) n^h ۳) $\log_h^k$ ۴) $\dfrac{k}{\log_n^k}$

حل : گزینه ۲ درست است.

حداکثر تعداد گره‌ها در سطح i ام برابر است با:

با فرض سطح ریشه یک $=n^{i-1}$

با فرض سطح ریشه صفر $=n^{i}$

بنابراین حداکثر تعداد برگ‌ها در یک درخت $n-ary$ با ارتفاع h برابر است با:

با فرض سطح ریشه یک $=n^{h-1}$

با فرض سطح ریشه صفر $=n^{h}$

یادداشت:

....

....

....

....

۲۰ ـ یک درخت دودویی به نام T با n گره داریم. اگر اولین گره در پیمایش Postorder درخت با آخرین گره از پیمایش Preorder درخت یکسان باشد، کدام گزینه را می‌توان نتیجه گرفت؟

۱) زیر درخت چپ T خالی است. ۲) درخت T حداکثر 3 گره دارد.

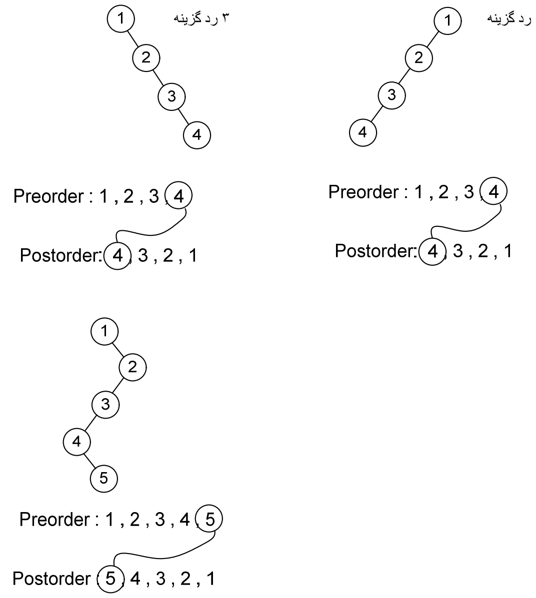
۳) زیر درخت راست T خالی است. ۴) ارتفاع درخت T برابر n است.

حل : گزینه ۴ درست است.

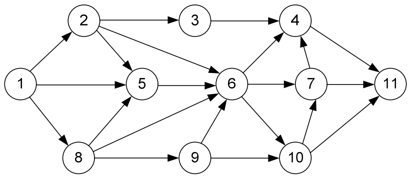
در پیمایش Preorder اولین گره‌ای که ملاقات می‌شود ریشه و آخرین گره، سمت راست‌ترین برگ است.

در پیمایش Postorder اولین گره‌ای که ملاقات می‌شود سمت چپ‌ترین برگ و آخرین گره، ریشه است.

بنابراین زمانی آخرین گره در پیمایش Preorder با اولین گره در پیمایش Postorder برابر می‌شود که تنها یک برگ وجود داشته باشد و این در صورتی است که در هر سطح تنها یک گره باشد. که در این صورت ارتفاع درخت n می‌شود.



۲۱ ـ در گراف زیر جستجوی اول عمق (Depth – first – Search) انجام می‌دهیم. جستجو را از گره‌ی 1 شروع شده و گره‌ها بـه ترتیـب عددی ملاقات می‌شوند. در این صورت، کدام گزینه این جستجو را نشان می‌دهد؟



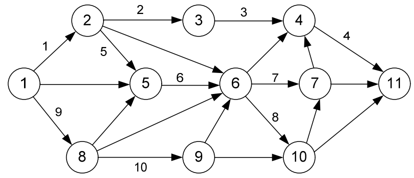
۱) $1,2,5,8,3,9,6,4,10,7,11$

۲) $1,2,8,5,3,9,6,4,11,10,7$

۳) $1,2,3,4,11,5,6,7,10,8,9$

۴) $1,2,3,4,5,6,7,8,9,10,11$

حل : گزینه ۳ درست است.



یادداشت:

....

....

....

....

ترتیب پیمایش یال‌ها روی آن نوشته شده است.

خروجی: $1,2,3,4,11,5,6,7,10,8,9$

۲۲ ـ نمی‌توان ساختمان داده‌ای برای n عنصر طراحی کرد که ......

۱) ساخت آن $O(n\log n)$ و حذف بزرگترین عنصر آن، درج، حذف کلید یک عنصر دلخواه در آن $O(\log n)$ باشد.
۲) ساخت آن $O(n)$ و حذف بزرگترین عنصر، درج، حذف و افزایش و کاهش کلید یک عنصر دلخواه در آن $O(\log n)$ باشد.
۳) ساخت آن $O(n)$ و حذف بزرگترین عنصر آن ، درج، حذف و افزایش کلید یک عنصر دلخواه در آن $O(\log n)$ باشد.
۴) ساخت آن $O(n)$ و حذف بزرگترین عنصر آن $O(1)$ و درج و حذف یک عنصر دلخواه در آن $O(\log n)$ باشد.

حل : گزینه ۴ درست است.

گزینهٔ ۱:

درخت	ساخت	درج	حذف بزرگترین عنصر یا هر کلید دلخواه
درخت جستجوی دودویی متوازن	$O(n\log n)$	$O(\log n)$	$O(\log n)$

گزینهٔ ۲ :

درخت	ساخت	درج	حذف بزرگترین عنصر	افزایش و کاهش کلید یک عنصر
heap	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

گزینهٔ ۳ : (مانند گزینهٔ ۲)

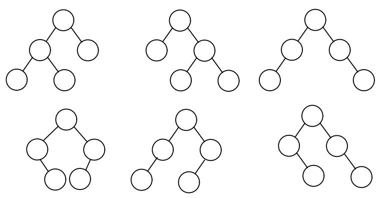
گزینهٔ ۴ : چنین ساختمان داده‌ای وجود ندارد.

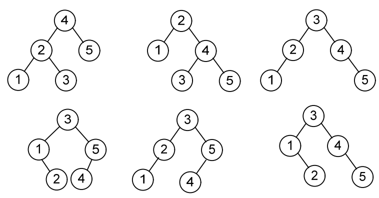
۲۳ ـ با عناصر $5,4,3,2,1$ حداکثر چند درخت AVL می‌توان ساخت؟

۱) 5 ۲) 6 ۳) 7 ۴) 8

حل : گزینه ۲ درست است.

شکل‌های درخت به صورت زیر خواهد بود.





یادداشت:

....
....
....
....

۲۴ ـ می‌دانیم که در یک درخت دودویی، سطح (یا عمق) یک گره برابر طول مسیر از آن گره تا ریشه است. ارتفاع درخت هم بزرگ‌ترین سطح گره‌ها در آن درخت است. «پهنای» یک درخت دودویی T را برابر بیش‌ترین تعداد گره‌های هم‌سطح در T تعریف می‌کنیم. آیا درخت دودویی با n گره و ارتفاع و پهنای زیر وجود دارد؟

I. ارتفاع $\Theta(n)$ و پهنای 1

II. ارتفاع $\Theta(\log n)$ و پهنای $\Theta(n)$

III. ارتفاع $\Theta(n)$ و پهنای $\Theta(n)$

IV. ارتفاع $\Theta(\log n)$ و پهنای $\Theta(\sqrt{n})$

جواب چند تا از موارد فوق درست است؟

۱) 2 ۲) 1 ۳) 3 ۴) 4

حل : گزینه ۳ درست است.

I. **اگر درختی در هر سطح یک گره داشته باشد، دارای ارتفاع $\theta(n)$ خواهد بود. مانند درخت اریب به چپ یا اریب به راست.**

II. **درختان کامل و یا پر نمونه بسیار عالی برای این مورد است. ارتفاع درخت کامل و یا پر برابر است با:**

$\lfloor \log n \rfloor + 1 = \theta(\log n)$

از طرفی تعداد گره‌ها در سطح آخر (برگ‌ها) درخت پر برابر است با $\frac{n+1}{2} = \theta(n)$. در درخت کامل نیز بیشترین پهنا یا مربوط به سطح آخر یا مربوط به سطح یکی مانده به آخر است که در هر دو حالت پهنا $\theta(n)$ است.

III. به نظر می‌آید چنین درختی وجود نداشته باشد. زیرا برای رسیدن به پهنای $\theta(n)$ درخت باید حداقل هم مرتبه با تعداد گره‌های یک درخت پر، گره داشته باشد. بنابراین با پهنای $\theta(n)$ ارتفاع درخت $\theta(\log n)$ باید باشد.

ما اینگونه نیست. زیرا می‌توان به یکی از گره‌های سطح آخر درخت فوق یک درخت با تعداد گره $\theta(n)$ و ارتفاع $\theta(n)$ (مانند درخت اریب به چپ یا به راست) اضافه کرد.

IV. برای اثبات اینکه چنین درختی وجود ندارد، ابتدا بهترین درخت را با پهنای $\sqrt{n}$ در نظر می‌گیریم و سپس ثابت می‌کنیم ارتفاع این درخت نمی‌تواند $\theta(\log n)$ باشد.

پر گره‌ترین درخت با پهنای $\sqrt{n}$، درخت پری با $\sqrt{n}$ برگ است. ارتفاع این درخت پر $\theta(\log\sqrt{n}) = \theta(\log n)$ است.

به نظر مشکلی وجود ندارد ولی با شمارش گره‌های درخت می‌توان دریافت که این درخت حداکثر $2\sqrt{n} - 1 = \theta(\sqrt{n})$ گره دارد و نه $n = \theta(n)$ گره. بنابراین ارتفاع این درخت $\theta(\log n)$ نیست، چرا که تعداد $n - (2\sqrt{n} - 1)$ گره دیگر باید در برگ‌های $\sqrt{n}$ گره پهنا قرار گیرند. (البته به گونه‌ای که پهنا از $\theta(\sqrt{n})$ بزرگ‌تر نشود)

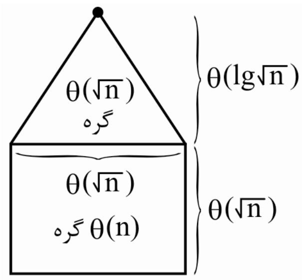
یادداشت:

....

....

....

....



از آنجا که $n-\left(2\sqrt{n}-1\right)=\theta(n)$؛ می‌توان درختی را با پهنای $\sqrt{n}$ در نظر گرفت ولی در بهترین حالت ارتفاع این درخت $\theta\left(\log\sqrt{n}\right)+\theta\left(\sqrt{n}\right)=\theta\left(\sqrt{n}\right)$ خواهد بود نه $\theta(\log n)$.

۲۵ ـ پیمایش preorder **درخت دودویی** T **(از چپ به راست) به صورت** acbdhigef **می‌باشد. همچنین می‌دانیم رأس‌های** b، d، e، f و i **در درخت** T **برگ هستند. پیمایش** postorder **درخت** T **کدام است؟IT**

۱) bcihefgda ۲) bcdaihegf ۳) bdcefgiha ۴) bdciefgha

حل : گزینه ۴ درست است.

در پیمایش preorder، ترتیب قرار گرفتن گره‌ها به صورت زیر است:

R	L	V
زیر درخت راست	**زیر درخت چپ**	**ریشه**

یادآوری:

۱ـ ترتیب دیدن برگ‌ها در هر سه پیمایش عمقی یکسان بوده و همواره از چپ به راست است.

۲ـ در پیمایش preorder اولین گره ملاقات شده ریشه است. در پیمایش postorder آخرین گره ملاقات شده ریشه است.

با توجه به پیمایش preorder، a گره ریشه است. بنابراین در پیمایش postorder ، a باید در آخر ملاقات شود.

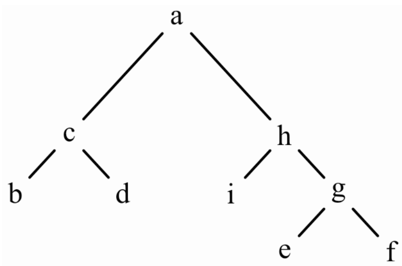
پس گزینه ۲ رد می‌شود.

در بین گزینه‌های ۱، ۳ و ۴، تنها در گزینه ۴ ترتیب ملاقات برگ‌ها رعایت شده است.

رد گزینه ۱: گره i قبل از d ملاقات شده است.

رد گزینه ۳: گره e قبل از I ملاقات شده است.

پیمایش پس ترتیب درخت فوق به صورت زیر خواهد بود: postorder : b d c i e f g h a



۲۶ ـ هفتمین کلید در پیمایش preorder **یک درخت جستجوی دودویی** (BST) **که پیمایش** postorder **آن به شکل زیر است (از چپ به راست)، کدام گزینه است؟**

Postorder :5,6,15,10,23,24,22,26,20

۱) 24 ۲) 26 ۳) 22 ۴) 15

حل : گزینه ۳ درست است.

یادداشت:

....

....

....

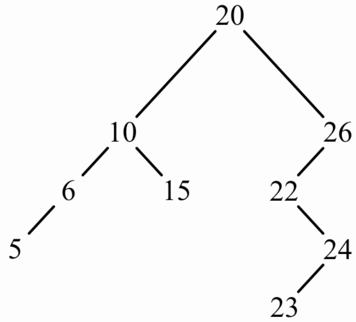
....

با در اختیار داشتن پیمایش postorder (یا preorder) یک درخت جستجوی دو دویی می‌توان آن را به صورت یکتا رسم کرد.

postorder : 5 , 6 , 15 , 10 , 23 , 24 , 22 , 26 , 20

preorder : 20 , 10 , 6 , 5 , 15 , 26 , 22 , 24 , 23

هفتمین کلید، 22 است.



۲۷ ـ فرض کنید صد کلید یک تا صد را به یک درخت جستجوی دودویی اضافه کرده‌ایم. ترتیب اضافه شدن کلیدها مشخص نیست اما احتمال تمام حالات ترتیب‌های اضافه شدن کلیدها (تمام Permutationهای ممکن یک تا صد) با هم برابر است. با چه احتمالی در روند اضافه شدن کلیدها به درخت کلید 4 و کلید 9 با هم مقایسه می‌شوند، در صورتی که اولین کلید اضافه شده کلید 43 باشد؟

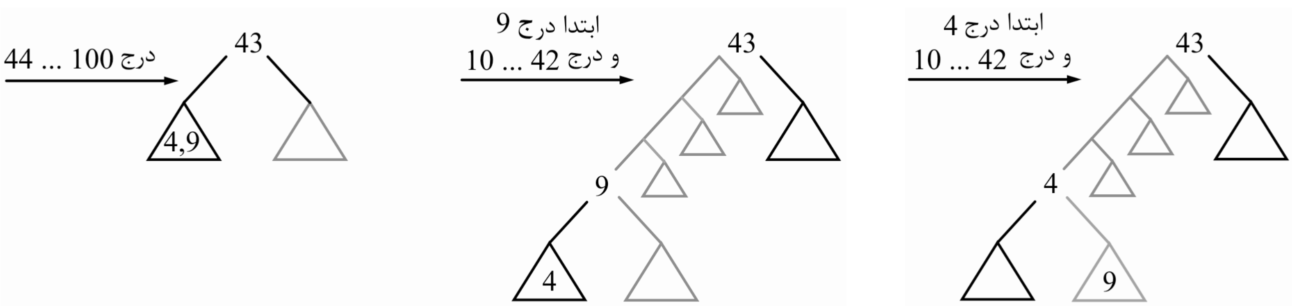
۱) $\frac{1}{3}$ ۲) $\frac{1}{2}$ ۳) $\frac{1}{4}$ ۴) $\frac{43}{100}\times\frac{1}{4}$

حل : گزینه ۱ درست است.

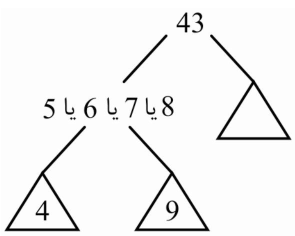
باید احتمال مقایسه شدن کلیدهای 4 , 9،یعنی احتمال اینکه یا 4 در زیر درخت 9 و یا 9 در زیر درخت 4 باشد را، محاسبه کنیم.

این احتمال دو حالت دارد: یا اینکه 4 در درخت باشد و 9 بخواهد درج شود و با 4 مقایسه شود و یا اینکه 9 در درخت باشد و 4 بخواهد درج شود و مسیر پیموده‌شده تا برگ توسط 4، 9 را نیز دربرگیرد.

کلیدهای 100 ... 44 , 42 ... 10 , 3 , 2 , 1 در هر فرم از درخت و در هر زمانی می‌توانند درج شوند بدون اینکه در وقوع مقایسه 4 , 9 تأثیری داشته باشند. برای مثال:



اما کلیدهای 8 , 7 , 6 , 5 تأثیرگذارند. در واقع درج هر کدام از این چهار کلید قبل از دو کلید 4 , 9 موجب می‌شود تا درخت دچار یک انشعاب مهم شود. این انشعاب، ریشه‌ی زیردرختی است که کلید 4 را به زیر درخت چپ و کلید 9 را به زیر درخت راست خود می‌فرستد و در نتیجه مانع مقایسه کلیدهای 4 , 9 با یکدیگر می‌شود.



یادداشت:

....

....

....

....

بنابراین باید این احتمال را محاسبه کرد که چهار کلید 8 , 7 , 6 , 5 قبل از 4 یا 9 نیاید. این احتمال برابر است با اینکه 4 یا 9 قبل از 8 یا 7 یا 6 یا 5 بیاید.

به عبارت دیگر احتمال اینکه از بین 6 عدد یا 4 اول بیاید یا 9. **پیشامد این‌که 4 اول بیاید** : A

پیشامد این‌که 9 اول بیاید : B

این دو پیشامد با هم ناسازگارند زیرا A و B نمی‌توانند هر دو با هم رخ دهند، بنابراین:

$$P(A \cup B) = P(A) + P(B) = \frac{1}{6} + \frac{1}{6} = \frac{1}{3}$$

۲۸ ـ درخت‌های دودویی که Preorder **و** Postorder **آن‌ها در زیر ذکر شده است، چه تعداد است؟**

Pre:a b d e f g c h i j
Post:d g f e b i j h c a

۱) 4
۲) 8
۳) 1
۴) هیچ درختی را نمی‌توان پیدا کرد.

حل : گزینه ۲ درست است.

تعداد درخت‌های دو دویی که پیمایش‌های preorder و postorder آن‌ها برابر با رشته‌های مشخص زیر باشد، برابر است با 2^{n_1} که در آن n_1 تعداد گره‌های تک‌فرزندی در درخت است.

برای به دست آوردن گره‌های تک فرزندی به این صورت عمل می‌کنیم: از سمت چپ یکی از پیمایش‌ها (برای مثال preorder) حرکت می‌کنیم و هر ترتیب دوتایی از گره‌ها (ترتیب‌های دو تایی دقیقاً کنار هم) در نظر گرفته و در پیمایش دیگر بررسی می‌کنیم. اگر دقیقاً همین ترتیب دوتایی را به صورت معکوس عیناً مشاهده کردیم، گره اول ترتیب تک‌فرزندی است.

طبق سوال داریم:

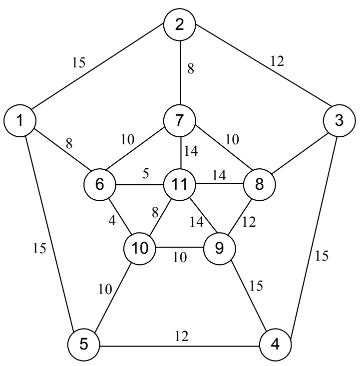
preorder : a b d e f g c h i j

ترتیب‌های دوتایی

postorder : d g f e b i j h c a

معکوس سه ترتیب دوتایی ef، fg و ch در پیمایش preorder، در پیمایش postorder وجود دارد پس 3 گره تک فرزندی داریم ($n_1 = 3$)، بنابراین تعداد درخت‌های دو دویی که پیمایش‌های preorder و postorder آن‌ها برابر با رشته‌های بالا باشد، $2^3 = 8$ تا است.

۲۹ ـ مجموعه وزن یال‌های درخت پوشای کمینه (MST) گراف مقابل چیست؟



۱) 91
۲) 97
3) 89
4) 85

یادداشت:

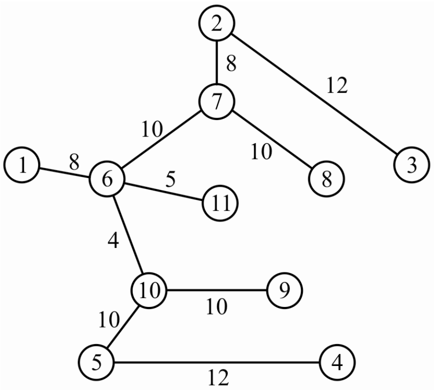
....

....

....

....

حل : گزینه ۳ درست است.



طبق الگوریتم کراسکال ابتدا یال‌ها را به صورت صعودی مرتب می‌کنیم و سپس تا جایی که کلیه گره‌ها به درخت پوشای مینیمم اضافه نشده‌اند، کوچک‌ترین یال را در صورتی که در درخت ایجاد دور نکند اضافه می‌کنیم. درخت حاصل به صورت روبروست.

مجموعه وزن یال‌های درخت برابر است با:

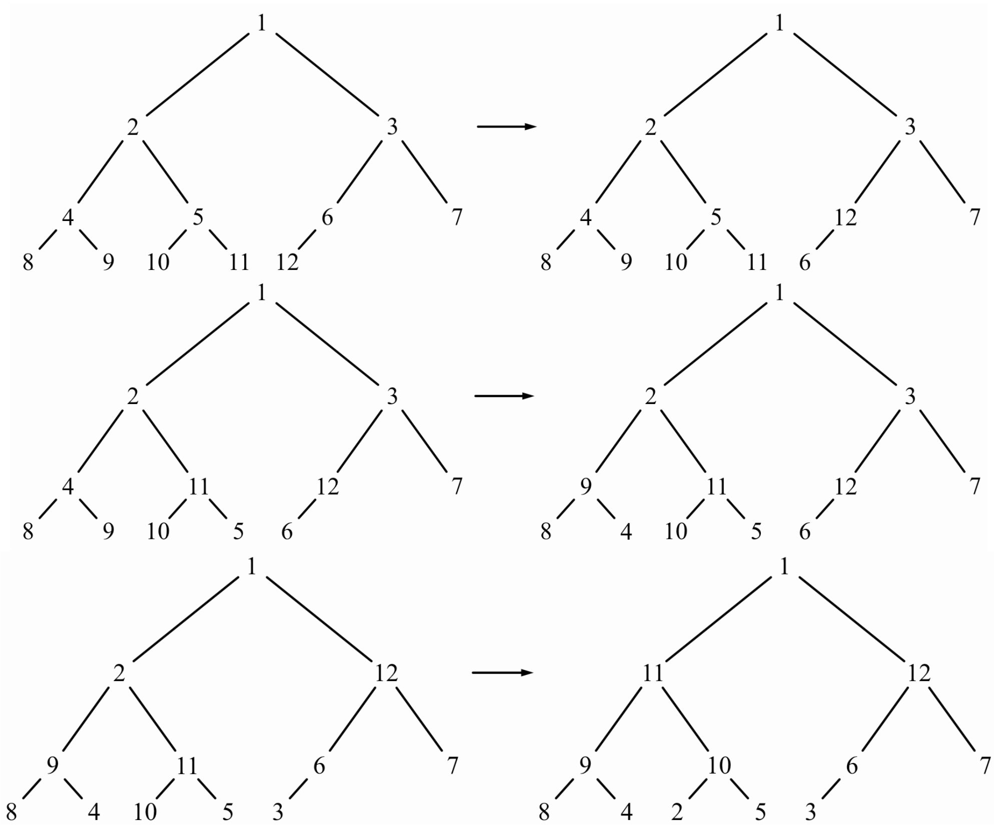
$$2\times12+4\times10+2\times8+5+4=89$$

۳۰ ـ یک لیست 12 **عنصری حاوی کلیدهای یک تا دوازده به صورت صعودی مرتب است. اگر این لیست به صورت درجا تبدیل بـه یـک** Max Heap **شود، عنصر پنجم لیست کدام گزینه است؟**

۱) 10 ۲) 11 ۳) 9 ۴) 8

حل : گزینه ۱ درست است.

کافی است اعداد فوق را در یک درخت دو دویی کامل قرار داده و پس از اجرای الگوریتم BUILD − MAX − HEAP، گره پنجم درخت را بازخوانی کنیم.



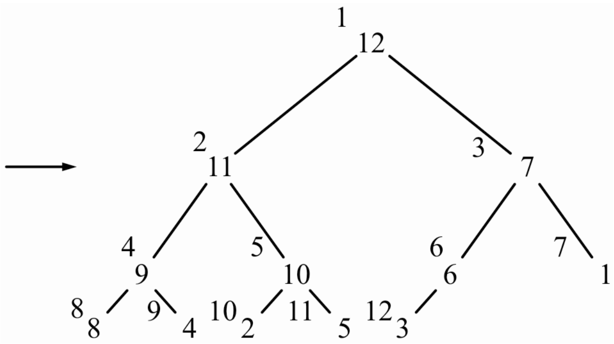
یادداشت:

....

....

....

....



عضو پنجم کلید 10 است.

۳۱ ـ فرض کنید که در Max − heap **استفاده شده در ساختمان داده‌ی** X**، هر عنصر یک زوج مرتب به صورت** < Key , Value > **است که معیار مقایسه‌ی عناصر مقدار** key **آن‌ها می‌باشد. ساختمان داده** X **یک پیاده‌سازی از کدام گزینه است؟**

```
X::A(x){
   Count++;
   Max-heap-insert(H,<count,x>);
}
X::B(x){
P = Heap-Extract-max(H)
Return-value(p);
}
```

۱) Queue　　۲) Stack

۳) Max Heap　　۴) Min Heap

حل : گزینه ۲ درست است.

در هر بار فراخوانی A با ورودی x، x در Max − Heap درج می‌شود. توجه داشته باشید در هر بار فراخوانی A، مقدار count افزایش می‌یابد و نیز از طرفی این مقدار به عنوان کلید عنصر در نظر گرفته می‌شود. بنابراین در هر بار درج x در heap، x به ریشه می‌آید. (زیرا مقدار کلید آن از سایر عناصر حتماً بیشتر است.)

در هر بار فراخوانی B با ورودی x، بزرگ‌ترین عنصر از heap حذف می‌شود. (یعنی ریشه حذف می‌شود.) درواقع آخرین عنصری که درج شده (و چون بزرگ‌ترین عنصر است در ریشه قرار می‌گیرد) حذف می‌شود. بنابراین ساختمان داده x پیاده‌سازی پشته است.

بازگشتی

۳۲ ـ جواب رابطهٔ بازگشتی $T(n)=8T\left(\frac{n}{9}\right)+n\log n$ **کدام‌یک از گزینه‌های زیر است؟**

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

حل : گزینه ۴ درست است.

$$T(n)=8T\left(\frac{n}{9}\right)+n\log n$$

یادداشت:

....

....

....

....

بنا به قضیهٔ اساسی داریم:

$b=9 \quad , \quad a=8 \qquad f(n)=n\log n$

که حالت 3 اتفاق می‌افتد:

$f(n)=\Omega\left(n^{\log_9^{8+\varepsilon}}\right)$

در نتیجه:

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

۳۳ ـ مرتبه زمانی الگوریتمی با تابع زمانی زیر، کدام گزینه است؟

$$T(n)=\begin{cases}0 & n=0\\ 1 & n=1\\ 3T(n-1)+4T(n-2) & \text{otherwise}\end{cases}$$

۱) n^2 ۲) 4^n ۳) $2^n \log n$ ۴) $n^4\log n$

حل : گزینه ۲ درست است.

با استفاده از معادلهٔ مشخصه حل می‌کنیم:

$$T(n)-3T(n-1)-4T(n-2)=0$$

$$r^2-3r-4=0 \Rightarrow (r-4)(r+1)=0 \Rightarrow \begin{array}{l} r_1=4\\ r_2=-1\end{array}$$

$$\Rightarrow T(n)=C_1(4)^n+C_2(-1)^2$$

$$\begin{array}{l}T(0)=0\Rightarrow \ C_1+C_2=0\\ T(1)=1\Rightarrow \ C_1-C_2=1\end{array} \Rightarrow \ C_1=\frac{1}{5}\ ,\ C_2=-\frac{1}{5}\ ,\ \Rightarrow T(n)=\frac{1}{5}\left[4^n-(-1)^n\right]$$

$$\Rightarrow T(n)=\theta\left(4^n\right)$$

۳۴ ـ جواب رابطهٔ بازگشتی زیر کدام است؟

$T(n)=4T\left(\sqrt{n}\right)+1 \qquad T(2)=1$

۱) $T(n)=\frac{1}{3}(\log n)^2-\frac{4}{3}$ ۲) $T(n)=\frac{4}{3}(4)^n-\frac{1}{3}$

۳) $T(n)=\frac{4}{3}(\log n)^2-\frac{1}{3}$ ۴) $T(n)=\frac{4}{3}\left(n^2\right)-\frac{1}{3}$

حل : گزینه ۳ درست است.

ابتدا با استفاده از یک تغییر متغیر فرم تابع را به شکل قابل استفاده از روش اصلی در می آوریم :

$$T(n)=4T\left(\sqrt{n}\right)+1 \xrightarrow{n=2^m} T\left(2^m\right)=4T\left(2^{\frac{m}{2}}\right)+1 \xrightarrow{s(m)=T(2^m)} S(m)=4S\left(\frac{m}{2}\right)+1$$

حال می‌توانیم برای رابطه بدست آمده از حالت I قضیه اصلی به شکل زیر استفاده کنیم :

$$\begin{cases} f(m)=1\\ m^{\log_b^a}=m^{\log_2^4}=m^2\end{cases} \xrightarrow[f(m)=O\left(m^{\log_b a-\varepsilon}\right)]{\text{case I: for } \varepsilon>0} T(m)=\theta\left(m^2\right) \xrightarrow{n=2^m} T(n)=\theta\left(\log^2 n\right)$$

یادداشت:

....

....

....

....

با توجه به گزینه‌ها، رابطهٔ $T(n)$ باید شکلی به فرم زیر داشته باشد:

$$T(n) = C_1(\log n)^2 + C_2$$
$$T(2) = 1 \Rightarrow C_1 + C_2 = 1$$

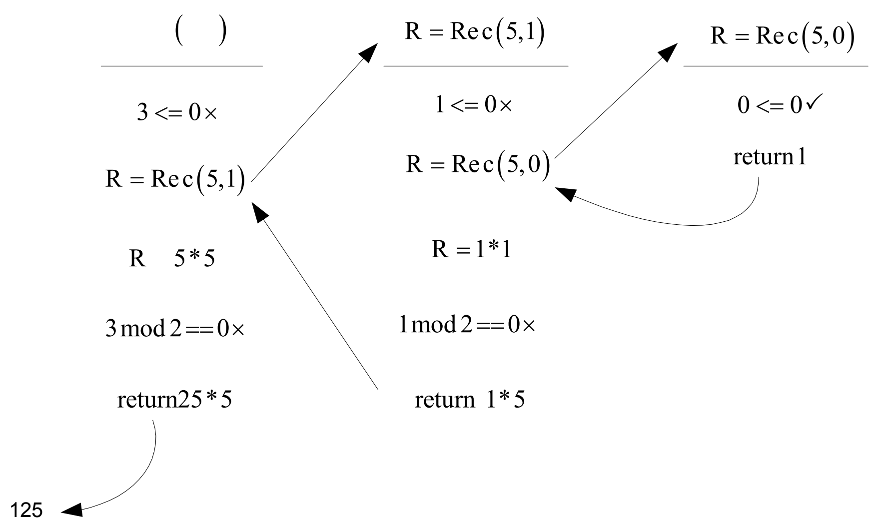
که با توجه به گزینه‌ها، تنها گزینهٔ ۳ در معادلهٔ بالا صدق می‌کند.

۳۵ ـ در روال بازگشتی زیر مقدار $Rec(5,3)$ کدام است؟

```
int Rec(int P, int q)
{
  int R;
  if (q <=   )
  R = Rec(p, q/2);
  R = R * R;
  if (q mod  2 == 0)
     return R;
  else
     return R * P;
}
```

۱) 15
۲) 25
۳) 75
۴) 125

حل : گزینه ۴ درست است.



مشاهده می‌شود که با $Rec(5,3)$ مقدار 125 به عنوان جواب بازگشتی دریافت می‌شود.

-رابطه‌ی بازگشتی زیر را در نظر بگیرید:

$$F(x,0) = F(x+1,0) + F(x+1,1), \quad \text{if } x < n$$
$$F(x,1) = 2F(x+1,0) + F(x+1,1), \quad \text{if } x < n$$
$$F(n,0) = 1$$
$$F(n,1) = 0$$

یادداشت:

....

....

....

....

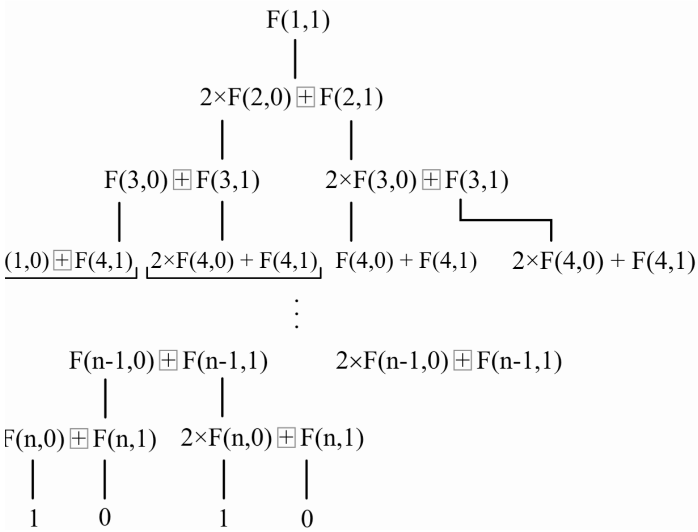
۳۶ ـ اگر از این رابطه بخواهیم مقدار $F(1,1)$ را به صورت کارا حساب کنیم، چند بار عمل «جمع» (همان $+$ در رابطه‌های فوق) را باید انجام دهیم؟

۱) $O(2^n)$ ۲) $O(n^2)$ ۳) $O(n)$ ۴) $O(2^{n-1})$

حل : گزینه ۳ درست است.

روابط بازگشتی معمولاً بدین صورت است که هر جمله بزرگ‌تر از جملات کوچک‌تر به دست می‌آید تا اینکه به شرط پایانی برسیم. اما تعریف بازگشتی فوق بدین صورت است که برای یک جمله کوچک به جملات بزرگ‌تر نیاز است و شرط پایانی روی بزرگ‌ترین جمله است. در واقع این تعریف متناظر با برنامه‌سازی پویاست.

x	تعداد +	تعداد + کارا
1	0	0
2	1	1
3	2	2
4	4	2
$\vdots$	$\vdots$	$\vdots$
$n-1$	2^{n-3}	2
n	2^{n-2}	2
	$2^{n-1}-1=O(2^{n-1})$	$2n-3=O(n)$



بسیاری از محاسبات در سطوح درخت تکراری هستند. بنابراین یک محاسبه کارا به گونه‌ای است که از این محاسبات تکراری صرف‌نظر شود. همان طور که مشاهده می‌شود در هر سطح، محاسبه حداکثر 2 عبارت کافی است.

۳۷ ـ در یک زمستان سرد، خرس قطبی n قطعه گوشت دقیقاً به اندازه‌های 1، 2، تا n را در غاری ذخیره کرده است. او هر روز یکی از این قطعه گوشت‌ها را به صورت تصادفی انتخاب می‌کند. اگر اندازه‌ی گوشت عدد فردی بود، آن را کاملاً می‌خورد. اگر زوج بود، آن را دقیقاً نصف می‌کند، یک نصف آن را می‌خورد و نصف دیگر را مجدداً در غار قرار می‌دهد. اگر گوشتی موجود نباشد، خرس می‌میرد. با این الگوریتم، برای nهای خیلی بزرگ روزهای باقیمانده از عمر خرس ما تابع کدام‌یک از گزینه‌ها خواهد بود؟

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

حل : گزینه ۱ درست است.

عمر خرس به تعداد قطعات گوشته وابسته است نه مجموع آن‌ها. از آنجا که عمر خرس زمانی به پایان می‌رسد که تمام قطعات خورده شوند باید مرتبه زمانی خورده شدن همه قطعات را حساب کنیم.

یادداشت:

....

....

....

....

$n =$ تعداد قطعه گوشت　　　　　$1,2,\cdots,n$: ابتدا (مرحله اول)

در مرحله بعدی فرض می‌کنیم. از هر قطعه یک بار استفاده شده است.

مرحله دوم : $\cancel{1},2,\cancel{3},4,\cancel{5},6,\cdots,n$ (فردها حذف شده و زوجها نصف می‌شوند)

$1,2,3,\cdots,\frac{n}{2}$　　　تعداد قطعه گوشت $= \frac{n}{2}$

مرحله سوم : $\cancel{1},2,\cancel{3},4,\cdots,\frac{n}{2}$ (فردها حذف شده و زوجها نصف می‌شوند)

$1,2,3,\cdots,\frac{n}{4}$　　　تعداد قطعه گوشت $= \frac{n}{4}$

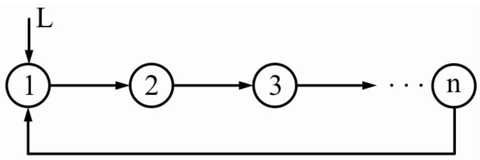
چون عمر خرس زمانی به پایان می‌رسد که تمام قطعات خورده شوند، این روند باید تا به پایان رسیدن همه قطعات ادامه پیدا کند.

تعداد قطعات گوشت : $n+\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\cdots = n\left(1+\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{8}\right) = n\left(\frac{n}{n-\frac{1}{2}}\right) = 2n = \theta(n)$

نکته : در این روش زمان خورده شدن یک قطعه مهم نیست بلکه مهم، خورده شدن یک قطعه است.

۳۸ ــ با توجه به تابع روبه‌رو و لیست حلقوی مذکور به ازای مقادیر n برابر 729 و 2200 خروجی به ترتیب برابر چند خواهد بود؟

```
int SO(LIST*L){
while(L->next != L){
   L→next = L->next->next
   L = L→next;
}
return L→data;
}
```



۱) 1 و 40

۲) 1 و 1

۳) 729 و 2200

۴) هیچ‌کدام

حل : گزینه ۴ درست است.

عددی که به عنوان خروجی توسط تابع SO بازگردانده می‌شود برابر است با:

خروجی : $2\times\left(n-2^{\lfloor \log n\rfloor}\right)+1$

بنابراین:

اگر $n = 729 \Rightarrow$ خروجی $= 2\times\left(729-2^{\lfloor \log 729\rfloor}\right)+1$

$$= 2\times\underbrace{\left(729-\overset{512}{2^{9}}\right)}_{217}+1 = 435$$

اگر $n = 2200 \Rightarrow$ خروجی $= 2\times\left(2200-2^{\lfloor \log 2200\rfloor}\right)+1$

یادداشت:

....

....

....

....

$$= 2\times\underbrace{\left(2200 - \cancel{2^{11}}^{2048}\right)}_{152} + 1 = 305$$

۳۹ ـ کم‌رشدترین حد بالای تابع بازگشتی $f(n) = 4f(n/2) + n^2 \log n, f(1) = 1$ کدام است؟

۱) $O\left(n^2 \log n\right)$ ۲) $O\left(n^3\right)$ ۳) $O\left(n^2 (\log n)^2\right)$ ۴) $O\left(n^2 \log(\log n)\right)$

حل : گزینه ۳ درست است.

$$f(n) = 4f\left(\frac{n}{2}\right) + 2^2 \log n$$

طبق قضیه اصلی برای $a = 4$، $b = 2$ و $f(n) = n^2 \log n$ داریم:

$$\log_b^a = 2$$

بنابراین مرتبه زمانی این تابع بازگشتی برابر است با:

$$O\left(n^2 \log^2 n\right)$$

هرگاه $f(n) = \theta\left(n^{\log_b^a} (\log n)^k\right)$ برای $k \ge 0$ آن‌گاه $T(n) = \left(n^{\log_b^a} (\log n)^{k+1}\right)$

درهم سازی

۴۰ ـ در جدول درهم‌سازی (hashing) با وارسی خطی (linear probing) اگر تابع درهم‌سازی برای 7 عضو ورودی به صورت زیر باشد.

key	A	B	C	D	E	F	G
hash	3	5	3	4	5	6	3

کدام‌یک از گزینه‌های زیر نمی‌تواند حاصل درج این عناصر با هر ترتیب دلخواه در آرایه‌ی 7 تایی (که در ابتدا تهی است.) باشد؟

۱) $H[0...6] = [E\ F\ G\ A\ C\ B\ D]$ ۲) $H[0...6] = [C\ E\ B\ G\ F\ D\ A]$

۳) $H[0...6] = [B\ D\ F\ A\ C\ E\ G]$ ۴) $H[0...6] = [C\ G\ B\ A\ D\ E\ F]$

حل: گزینه ۲ درست است.

گزینه ۱: در صورتی که ترتیب درج عناصر (از چپ به راست) A B C D E F G باشد آن‌گاه:

	0	1	2	3	4	5	6
H:	E	F	G	A	C	B	D

گزینه ۳: در صورتی که ترتیب درج عناصر (از چپ به راست) A E C G B D F باشد آن‌گاه:

	0	1	2	3	4	5	6
H:	B	D	F	A	C	E	G

گزینه ۴: در صورتی که ترتیب درج عناصر (از چپ به راست) A D E F C G B باشد آن‌گاه:

	0	1	2	3	4	5	6
H:	C	G	B	A	D	E	F

یادداشت:

....

....

....

....

گزینه ۲: ابتدا G درج شده است (زیرا G در مکان در هم‌سازی خود قرار دارد.) اگر بعد از آن F درج شود، باید در خانه 6 قرار بگیرد و اگر بعد از G، A یا C درج شوند، طبق linear probing باید در خانه 4 قرار بگیرند. در نتیجه گزینه 2 غلط است.
توجه داشته باشید ترتیب درجی که برای گزینه‌های ۱ و ۳ و ۴ مثال زده شد، تنها حالت ممکن نیست و به ازای جایگشت‌های دیگری از عناصر نیز می‌توان به چنین آرایه‌هایی رسید. به عنوان مثال در گزینه ۱، اگر ترتیب درج عناصر (از چپ به راست) B A C D E F G باشد، نتیجه آرایه گزینه ۱ می‌شود.

۴۱ ـ فرض کنید که یک جدول درهم سازی به اندازه 80 (هشتاد) از روش آدرس‌دهی خطی باز استفاده می‌کند. ابتدا جدول خالی بوده است و تنها عملیات اضافه کردن و جستجو روی جدول انجام شده است. در حال حاضر وضعیت درایه‌های 45 تا 56 جدول به صورت زیر است. اعداد بالای آرایه، اندیس درایه‌ها و اعداد پایین آرایه خروجی تابع درهم سازی است. اگر یکبار دیگر از جدول درهم سازی خالی شروع کنیم و همان عملیات را به غیر از اضافه کردن کلید e دوباره انجام دهیم. در درایه‌ی 50 چه کلیدی قرار می‌گیرد؟

45	46	47	48	49	50	51	52	53	54	55	56
	a	b	c	d	e	f	g	h	i	j	
	46	46	46	47	46	51	47	46	48	49	

۱) h
۲) i
۱) f
۲) g

حل : گزینه ۴ درست است.
پس از قرار گرفتن کلید a در خانه 46، با استفاده از روش آدرس‌دهی خطی باز کلیدهای d,c,b به ترتیب در خانه‌های 47 , 48 , 49 قرار می‌گیرند. سپس کلید f در خانه 51 که خالی است می‌نشیند. خروجی تابع درهم‌سازی به ازای g خانه 47 است؛ اما ازآنجاکه قبلاً یک عنصر به این خانه در هم‌سازی شده است، g با روش آدرس‌دهی خطی باز در خانه 50 قرار می‌گیرد. جدول درهم‌سازی بدون اضافه کردن کلید e به شکل زیر خواهد بود.

45	46	47	48	49	50	51	52	53	54	55	56
	a	b	c	d	g	f	h	i	j		
	46	46	46	47	47	51	46	48	49		

۴۲ ـ فرض کنید که در ساخت جدول درهم سازی زیر از روش آدرس‌دهی خطی باز استفاده شده است. اگر تابع Hash به صورت باقیمانده تقسیم بر 5 تعریف شده باشد، کدام گزینه نمی‌تواند ترتیب صحیح درج عناصر در جدول باشد؟ ترتیب از چپ به راست است.

0	10
1	11
2	6
3	7
4	15

۱) 15، 10، 7، 6، 11
۲) 15، 7، 10، 6، 11
۳) 15، 7، 6، 11، 10
۴) 7، 15، 6، 10، 11

حل : گزینه ۴ درست است.
در صورت سؤال توضیحی درباره چگونگی ترتیب درج عناصر داده شده است. اما از آنجا که جدول عناصر به صورت مرتب هستند، احتمالاً منظور طراح ترتیب درج عناصر به صورت مرتب‌سازی است.

یادداشت:
....
....
....
....

در جدول زیر عناصر هر یک از گزینه‌ها را در یک جدول درهم‌سازی درج کرده‌ایم.

باقی‌مانده	عناصر			گزینه	جدول درهم‌سازی				
					4	3	2	1	0
0	10,15								
1	6,11			1	15	7	6	11	10
2	7			2	15	7	6	11	10
3				3	15	7	6	11	10
4				4	7	15	6	11	10

در صورت بروز تصادم از روش آدرس‌دهی باز استفاده شده است.

اگر تابع درهم‌سازی برای یک کلید، آدرسی را تولید کند که این آدرس قبلاً اشغال شده باشد، قرار دادن این کلید را در اولین خانه خالی بعد از آدرس تولید شده، آدرس‌دهی باز گویند

حل : گزینه ۲ درست است.

گزینهٔ ۱ :

ترتیب درج کلیدها

$$\xrightarrow{\qquad}$$

$$\begin{cases} A,C,B,D,E,F,G \\ A,B,C,D,E,F,G \\ B,A,C,D,E,F,G \end{cases} \Rightarrow H:\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \atop \begin{bmatrix} E & F & G & A & C & B & D \end{bmatrix}$$

گزینهٔ ۳:

ترتیب درج کلیدها

$$\xrightarrow{\qquad}$$

$$\begin{cases} A,C,,G,B,D,F \\ A,E,C,G,B,D,F \\ E,A,C,G,B,D,F \end{cases} \Rightarrow H:\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \atop \begin{bmatrix} B & D & F & A & C & E & G \end{bmatrix}$$

گزینهٔ ۴:

ترتیب درج کلیدها

$$\xrightarrow{\qquad}$$

$$\begin{Bmatrix} A \\ D \\ E \\ F \end{Bmatrix},C,G,B \Rightarrow H:\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \atop \begin{bmatrix} C & G & B & A & D & E & F \end{bmatrix}$$

$4!$

$$H:\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \atop \begin{bmatrix} \ & \ & \ & G & \ & \ & \ \end{bmatrix}$$

گزینهٔ ۲ :

اگر بعد از G، F بیاید باید در 6 بنشیند که با توجه به گزینهٔ 2 غلط است. هر عنصر A یا C اگر بعد از G بیاید، حتماً باید در ۴ قرار بگیرند.

یادداشت:

....

....

....

....

۴۳ ـ در یک تابع درهم‌سازی فاکتورلود عبارت است از درهمسازی n کلید به m مکان که به صورت $\alpha = \frac{n}{m}$ است با توجه به دو روش زنجیره‌سازی و آدرس‌دهی باز، مقادیری که α می‌تواند در این دو روش داشته باشد چیست؟

۱) زنجیره‌سازی: بیشتر، مساوی یا کمتر از 1
آدرس‌دهی باز: حداکثر 1

۲) در هر دو روش α حداکثر می‌تواند 1 باشد.

۳) زنجیره‌سازی: حداکثر 1
آدرس‌دهی باز: همیشه برابر با 1 است.

۴) زنجیره‌سازی: همیشه برابر 1 است.
آدرس‌دهی باز: حداکثر 1

حل : گزینه ۱ درست است.

از آنجا که در روش زنجیره‌سازی عناصر خارج از جدول درهم‌سازی ذخیره می‌شوند، α می‌تواند بیشتر، مساوی یا کمتر از 1 باشد. در روش آدرس‌دهی باز چون همه‌ی عناصر در جدول ذخیره می‌شوند، α می‌تواند حداکثر 1 باشد.

۴۴ ـ در یک جدول درهم‌سازی با روش زنجیره‌سازی زمان جستجوی یک عنصر در بدترین حالت چقدر است؟($\alpha = \frac{n}{m}$، فاکتورلود)

۱) $\Theta(n^2)$ ۲) $\Theta(1)$ ۳) $\Theta(n)$ ۴) $\Theta(\log n)$

حل : گزینه ۳ درست است.

بدترین حالت زمانی رخ می‌دهد که هر n عنصر به یک مکان درهم‌سازی می‌شوند که در این صورت زمان جستجو برابر با زمان جستجو در یک لیست پیوندی نامرتب با n عنصر یعنی $\theta(n)$ است.

مرتب سازی

۴۵ ـ بر اساس قطعه کد زیر، کدام‌یک از گزینه‌های زیر در محل A درست است؟

```
for(item = n;item > 1;item--)
{
  int Large = a[1];
  int index =1;
for(i = 2;i <= item;i++)
{
  Large = a[i];
  index = i;
}
a[index] = a[item];
a[item] = Large;
//A
}
```

۱) برای تمام jهایی که $1 < j <= item$، $a[j] < a[j-1]$

۲) برای تمام jهایی که $item < j <= n$، $a[j] < a[j+1]$

۳) برای تمام jهایی که $item <= j < n$، $a[j] <= a[j+1]$

۴) برای تمام jهایی که $item <= j < n$، $a[j] < a[j+1]$

یادداشت:

....

....

....

....

حل : گزینه ۳ درست است.

شبه کد داده شده مربوط به مرتب‌سازی انتخابی است. در حلقه‌ی for داخلی (با شمارنده i) عناصر در زیرآرایه‌ی نامرتب با Large مقایسه شوند تا بزرگترین عنصر زیرآرایه‌ی نامرتب در Large قرار بگیرد. در انتهای حلقه‌ی for با شمارنده i، بزرگترین عنصر (Large) در آخرین خانه قرار می‌گیرد. در نتیجه این مرتب‌سازی، عناصر آرایه را از انتها به ابتدا مرتب می کند.

۴۶ ـ پیچیدگی کدام‌یک از الگوریتم‌های مرتب‌کننده‌ی زیر (بر حسب تابعی از ورودی) در حالت متوسط (Average Case) و بدترین حالت (Worst Case) با هم متفاوت است؟

۱) Quick Sort
۲) Binary Insertion Sort
۳) Heap Sort
۴) Merge Sort

حل : گزینه ۱ درست است.

مرتب‌سازی	بهترین حالت	بدترین حالت	حالت متوسط
Quick Sort	$O(n\log n)$	$O(n^2)$	$O(n\log n)$
Binary Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

۴۷ ـ یک آرایه با 8 عنصر را در نظر بگیرید. مرتب‌سازی ادغامی برای مرتب‌سازی این آرایه حداکثر چند مقایسه انجام خواهد داد؟

۱) 31
۲) 13
۳) 17
۴) 33

حل : گزینه ۳ درست است.

مرتب‌سازی ادغامی در یک آرایه ورودی به طول n، حداکثر $n\log n-n+1$ مقایسه و حداقل $\frac{n}{2}\log n$ مقایسه انجام می‌دهد.

بنابراین مرتب‌سازی ادغامی در یک آرایه 8 عنصری، حداکثر $n\log n-n+1=8\times 3-8+1=24-7=17$ مقایسه انجام می‌دهد.

۴۸ ـ می‌دانیم که هزینه‌ی الگوریتم مرتب‌سازی درجی (Insertion Sort) برای مرتب‌سازی یک آرایه‌ی A با n عنصر متناسب با تعداد «وارونگی» (inversion)های عناصر آن آرایه است. زوج $(i\ ,\ j)$ را یک عدد وارونگی می‌گوییم اگر $i<j$ و $A[i]>A[j]$ با فرض احتمال این‌که یک زوج اندیس دلخواه از A یک وارونگی باشد برابر $1/2$ است، میانگین تعداد وارونگی‌های یک آرایه‌ی A با عناصر متمایز چگونه است؟

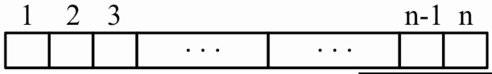
۱) $\frac{n^2-n}{2}$
۲) $\frac{n^2}{2}$
۳) $\frac{n^2}{4}$
۴) $\frac{n^2-n}{4}$

یادداشت:

....
....
....
....

حل : گزینه ۴ درست است.

تعداد کل زوج‌ها در یک آرایه n عنصری برابر است با: $\frac{n(n-1)}{2}$



i	j	**تعداد زوج** (i, j)
1	$2 \cdots n$	$n-1$
2	$3 \cdots n$	$n-2$
3	$4 \cdots n$	$n-3$
$\vdots$	$\vdots$	$\vdots$
$n-1$	n	1
n	—	0
		$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$

از طرفی احتمال اینکه یک زوج وارونه باشد $\frac{1}{2}$ است. $\left(f(x) = \frac{1}{2}\right)$

بنابراین میانگین وارونگی کل زوج‌ها برابر است با: $\frac{1}{2} \times \frac{n(n-1)}{2}$

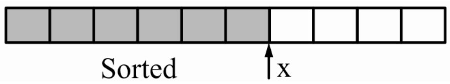
به عبارت دیگر:

۴۹ ـ در الگوریتم مرتب‌سازی آرایه‌ی A با n عنصر فرض کنید $b > 1$ یک عدد ثابت است. همچنین فرض کنید که هزینه‌ی مقایسه‌ی دو عنصر $A[i]$ و $A[j]$، یا تعویض آن‌ها، اگر $|j-i| \le b$ برابر صفر (خیلی کم) و در غیر این صورت برابر 1 (خیلی زیاد) است. توجه کنید که با این فرض، هزینه‌ی مرتب‌سازی درجی، حبابی (Bubble sort) برابر $O(1)$ می‌شود. چون فقط عناصر مجاور را مقایسه و تعویض می‌کنند. با این فرض هزینه‌ی مرتب‌سازی ادغامی (Merge sort) A در بدترین حالت چقدر است؟ (بهترین جواب را انتخاب کنید.) (بدیهی است که اگر $T(n)$ زمان اجرا باشد داریم: $n < b$ و $T(n) = 1$.)

۱) $O(n\lg(n/b))$ ۲) $O(n/b\lg(n/b))$ ۳) $O(n\lg n)$ ۴) $O(n\lg n)$

حل : گزینه ۱ درست است.

برای درک بهتر تعریف فوق، مرتب‌سازی درجی را بررسی می‌کنیم. در هر گذر مرتب‌سازی درجی، عنصری که باید در ناحیه مرتب شده درج شود فقط با عناصر مجاور خود مقایسه می‌شود و مقایسات و تعویض هزینه‌ای برابر $O(1)$ دارد.



بنابراین کل مرتب‌سازی نیز دارای هزینه خیلی کم $O(1)$ است.

یادداشت:

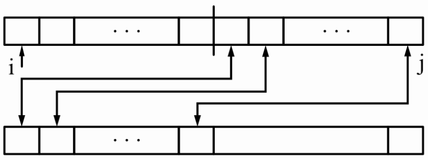
....

....

....

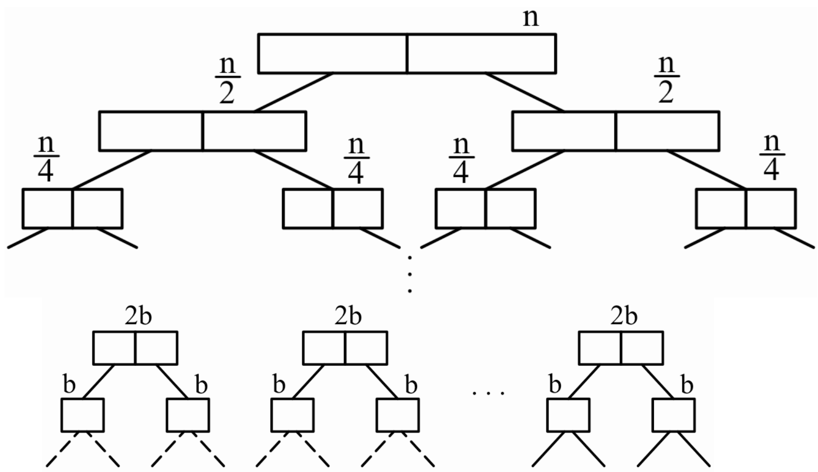
....

$O(1)$ بودن مرتب‌سازی حبابی را نیز بررسی کنید.



طبق تعریف فوق در مورد هزینه مقایسه و تعویض دو عنصر، بدترین حالت برای ادغام در زیر آرایه مرتب متوالی زمانی رخ می‌دهد که اولین خانه از زیر آرایه سمت چپ از عناصر زیر آرایه سمت راست بزرگ‌تر باشد. زیرا تنها در این حالت است که حداکثر فاصله بین دو اندیس i, j از یکدیگر رخ می‌دهد. حال اگر دو زیر آرایه ادغامی دارای اندازه $\frac{b}{2}$ باشند، آن‌گاه $|j-i| \le b$ خواهد بود و بنابراین آرایه مرتب شده حاصل هزینه‌ای ندارد.

هزینه ادغام برای تولید سطوح درخت:



سطح	تعداد زیرآرایه × هزینه	
1	$2\times\frac{n}{2}=$	$\theta(n)$
2	$4\times\frac{n}{4}=$	$\theta(n)$
3	$8\times\frac{n}{8}=$	$\theta(n)$
	$\vdots$	
$\log\frac{n}{b}$	$\frac{n}{b}\times b=$	$\theta(n)$

$$\theta\left(n\log\frac{n}{b}\right)$$

ادغام در سطوح پایین‌تر از b یعنی هنگامی که اندازه زیر آرایه‌ها از b کوچک‌تر می‌شود، هزینه‌ای ندارد. بنابراین تعداد گره‌هایی که در آخرین سطح هزینه‌برند $\frac{n}{b}$ است و ارتفاع درخت تا سطحی که هزینه‌بر می‌باشد برابر است با $\theta\left(\log\frac{n}{b}\right)$. از طرفی هزینه ادغام در هر سطح برابر است با $\theta(n)$. در نتیجه هزینه کل مرتب‌سازی ادغامی برابر است با:

$$\theta(n)\times\theta\left(\log\frac{n}{b}\right)=\theta\left(n\log\frac{n}{b}\right)$$

۵۰ ــ آرایه‌ی نامرتب n عنصری A شامل n عدد صحیح مجزا بین 0 تا n است. به جز یک عدد x، می‌خواهیم با یک الگوریتم کارا x را پیدا کنیم. کدام‌یک از گزینه‌های زیر الگوریتم درست و از بقیه سریع‌تر و مناسب‌تر برای این کار است؟

۱) A را مرتب کنیم و سپس با پیمایش آرایه‌ی مرتب، x را پیدا کنیم.

۲) یک heap بر روی A بسازیم. از ریشه حرکت می‌کنیم و به یکی از زیردرخت‌های آن می‌رویم. کار جستجو را دنبال می‌کنیم.

۳) میانه‌ی A را پیدا کنیم و بر اساس آن به عنوان عنصرمحور (pivot)، Partition را انجام می‌دهیم و بر این اساس کار جستجو را ادامه می‌دهیم.

۴) از یک آرایه کمی $n+1$ عنصر استفاده می‌کنیم. ابتدا تمام خانه‌های آن را صفر می‌کنیم. سپس در طول آرایه‌ی اصلی حرکت می‌کنیم و به ازای هر عددی که در A دیدیم، در آرایه‌ی کمکی، خانه با اندیس آن عدد را 1 می‌کنیم. این کار را برای تمام عناصر در A انجام می‌دهیم، در انتها از آرایه کمکی x را پیدا می‌کنیم.

یادداشت:

....

....

....

....

حل : گزینه ۳ درست است.

ابتدا گزینه‌های ۱ و ۲ و ۴ را بررسی می‌کنیم.

گزینهٔ ۱ : مرتب کردن آرایه‌ی A در زمان $O(n\log n)$ انجام می‌شود سپس می‌توان در زمان $O(\log n)$، عدد x را پیدا کرد.

زمان کل: $O(n\log n)+O(\log n)=O(n\log n)$

گزینهٔ ۲ : این روش صحیح نیست. از آنجا که در یک heap، فرزندان یک پدر ترتیب مشخصی ندارد نمی‌توان از ریشه با انتخاب یک مسیر مشخص عدد x را پیدا کرد.

گزینهٔ ۴ : این روش مشابه مرتب‌سازی شمارشی است. در زمان $O(n)$ عناصر در A یکی‌یکی بررسی شده و در خانه‌ی متناظر آن در آرایه کمکی 1 می‌شوند. در نتیجه با یک جستجوی خطی در طول آرایه کمکی، می‌توان عدد x را پیدا کرد. (اولین خانه‌ی صفر) که این جستجو زمان $O(n)$ نیاز دارد. پس زمان کل $O(n)+O(n)=O(n)$ می‌شود.

گزینهٔ ۳ : از آنجا که عناصر A اعداد صحیح بین 0 تا n هستند، با فرض وجود عدد x، می‌توان میانه‌ی این n+1 عنصر را در زمان $O(1)$ $\left(\text{میانه}=\frac{n+1}{2}\right)$ محاسبه کرد. سپس در زمان $O(n)$ عنصر میانه را در A جستجو کرد. اگر میانه پیدا نشود، x میانه است، در غیر اینصورت میانه را به عنوان عنصر محور Partition در نظر گرفته و Partition را اجرا کرد. سپس اگر میانه با اندیس خانه برابر بود، یعنی x در زیر آرایه‌ی سمت راست عنصر میانه قرار دارد. در غیر اینصورت x در سمت چپ عنصر میانه قرار دارد. و به صورت بازگشتی، الگوریتم مجدداً برای زیرآرایه‌ی مناسب فراخوانی کرد. که زمان این الگوریتم $O(n)$ است.

زمان هر دو الگوریتم گزینهٔ ۳ و ۴، $O(n)$ است اما از آنجا که گزینهٔ ۴ درجا نیست، گزینهٔ ۳ انتخاب بهتری می‌باشد.

۵۱ ــ فرض کنید می‌خواهیم یک لیست حاوی عناصر (از چپ به راست) 2 و 4 و 5 و 3 و 1 را به لیست عناصر (از چپ به راست) 1 و 2 و 3 و 4 و 5 تبدیل کنیم. تنها عملیات مجاز جابجا کردن درایه‌های متوالی (پشت سر هم) با یکدیگر است. حداقل چند جابجایی لازم است؟

۱) 4 ۲) 5 ۳) 6 ۴) 7

حل : گزینه ۳ درست است.

برای تبدیل لیست 1,3,5,4,2 به لیست نزولی، یک راه حل خوب رساندن بزرگ‌ترین یا کوچک‌ترین عدد به مکان مناسب خود است.

در واقع حداقل تعداد جابه‌جایی‌های خانه‌های مجاور برابر است با تعداد جابه‌جایی‌ها در Insertion Sort، چرا که در هر مرحله دو عدد متوالی جابه‌جا می‌شوند.

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

مرتبه زمانی و تعداد دفعات تکرار

۵۲ ــ در کد زیر ++x، چند بار تکرار می‌شود؟ (فرض کنید n >= 3 است)

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

۱) $\left\lfloor \log\frac{n}{3} \right\rfloor$ ۲) $\left\lfloor \frac{(\log n)}{3} \right\rfloor$ ۳) $\left\lfloor \log\frac{n}{3}+1 \right\rfloor$ ۴) $\left\lfloor \frac{(\log n+1)}{3} \right\rfloor$

یادداشت:

....

....

....

....

حل : گزینه ۳ درست است.

روش اول: از تغییر متغیر استفاده می‌کنیم. $j=\frac{i}{3}$

$$\text{for}\left(j=1;j<=\frac{n}{3};j=j*2\right)$$

تعداد تکرار: $\left\lfloor \log_2^{\frac{n}{3}} \right\rfloor + 1$

روش دوم:

به ازای یک مقدار دلخواه، مثلاً $n=6$، تعداد تکرار x++ را محاسبه کرده و سپس در گزینه‌ها چک می کنیم.

تکرار	n	i
1	6	$3\le 6$
1	6	$6\le 6$
خروج	x	$12\le 6$

$$\left\lfloor \log\frac{6}{3} \right\rfloor = 1$$

$$\left\lfloor \frac{(\log 6)}{3} \right\rfloor = 1$$

$$\left\lfloor \log\frac{6}{3}+1 \right\rfloor = 2$$

$$\left\lfloor \frac{(\log 6+1)}{3} \right\rfloor = 1$$

سپس به ازای n=6، 2 تکرار داریم، در نتیجه گزینهٔ ۳ درست است.

۵۳ ــ در برنامه زیر تعداد دفعات تکرار دستورالعمل شماره 3 برابر است با:

```
1        for(k = 0;k <= n-1;k++)
2        for(i = 1;i <= n-k;i++)
3            a[i][i+k] = k;
```

۱) n^2 ۲) $\frac{(\ +\ )}{2}$ ۳) $\frac{\quad}{2}$ ۴) $\frac{(\ -\ )}{2}$

حل : گزینه ۲ درست است.

k		i		تکرار	
0		$1...n-0$		n	
1		$1...n-1$		$n-1$	
2		$1...n-2$		$n-2$	تعداد تکرار $=n+(n-1)+(n-2)+...+2+1$
$\vdots$		$\vdots$		$\vdots$	تعداد تکرار $=\frac{(\ +\ )}{2}$
$n-1$		$1...1$		1	

۵۴ ــ توابع $f(n)=4^{\lg n}$، $g(n)=\lg^{\lg n} n$ و $h(n)=\lg^2 n$ را در نظر بگیرید. کدام یک از گزاره‌های زیر صحیح است؟

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

۲) $f(n)\in\Theta(h(n))\,,\,g(n)\in\Omega(f(n))$

۳) $g(n)\in\Omega(h(n))\,,\,h(n)\in\Omega(f(n))$

۴) $h(n)\in O(g(n))\,,\,f(n)\in\Theta(g(n))$

یادداشت:

....

....

....

....

حل : گزینه ۱ درست است.

برای مقایسه رشد توابع می‌توان رشد لگاریتم آن‌ها را با یکدیگر مقایسه کرد.

$$f(n) = 4^{\log n} \rightarrow \log f(n) = \log n \ \log 4 = 2\log n$$

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

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

یادآوری: $X(n) = A(n)^{B(n)} \rightarrow \log X(n) = B(n)\log A(n)$

بنابراین:

$$\begin{array}{l} 2\log\log n \le 2\log n \le \log n \log\log n \\ h(n) \le f(n) \le g(n) \end{array} \Rightarrow \begin{array}{l} h(n) = O(f(n)) \\ h(n) = O(g(n)) \\ f(n) = O(g(n)) \\ f(n) = \Omega(h(n)) \\ g(n) = \Omega(h(n)) \\ g(n) = \Omega(f(n)) \end{array}$$

۵۵ ـ کدام‌یک از گزاره‌های زیر غلط است؟

۱) $\Omega(f(n)) + O(f(n)) = \Theta(f(n))$
۲) $f(n) + O(f(n)) = \Theta(f(n))$
۳) $g(n) = \Omega(f(n)) \Rightarrow f(n) = O(g(n))$
۴) $g(n) = \Omega(f(n)) \Rightarrow g(n) = \Omega(O(f(n)))$

حل : گزینه ۱ درست است.

گزینه ۱: به طور کلی $\Omega(f(n)) \ne \theta(f(n))$

گزینه ۲: $O(f(n))$ حداکثر دارای مرتبه $f(n)$ است. بنابراین $\theta(f(n)) + O(f(n)) = \theta(f(n))$

گزینه ۳: واضح است که اگر $g \ge f$ آن‌گاه $f \le g$.

گزینه ۴: $h(n) = O(f(n)) \Rightarrow h \le f$. $g(n) = \Omega(f(n)) \rightarrow g \ge f$ **بنابراین** $g \ge f \ge h \rightarrow \Omega(O(f(n)))$

متفرقه

۵۶ ـ اگر رشتهٔ اعداد (از راست به چپ) $\overleftarrow{5, 4, 3, 2, 1}$ را به‌ترتیب در یک STACK وارد کنیم، کدام‌یک از خروجی‌های زیر از این STACK امکان‌پذیر خواهد بود؟ (خروجی‌های STACK را از چپ به راست بخوانید.)

۱) $2-3-5-1-4$
۲) $5-4-3-1-2$
۳) $1-3-5-4-2$
۴) $5-1-3-2-4$

حل : گزینه ۳ درست است.

ورودی: $\overrightarrow{1}, 2, 3, 4, 5$

یادداشت:

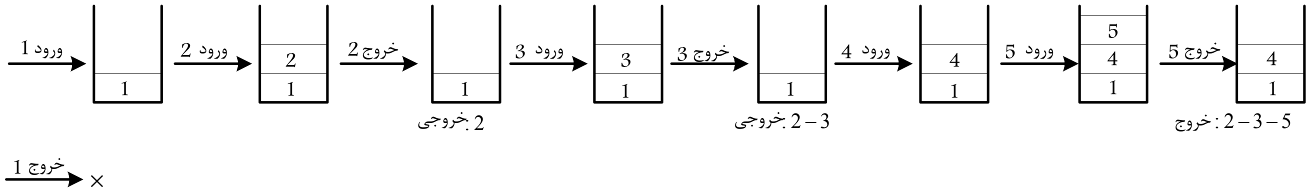
....

....

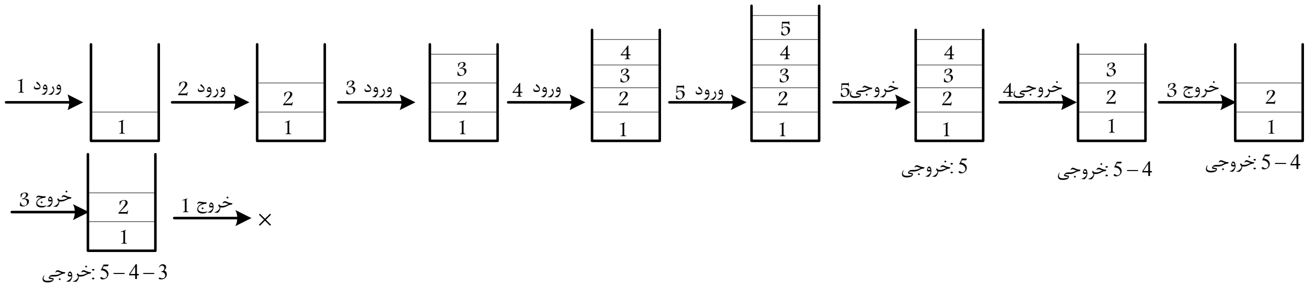
....

....

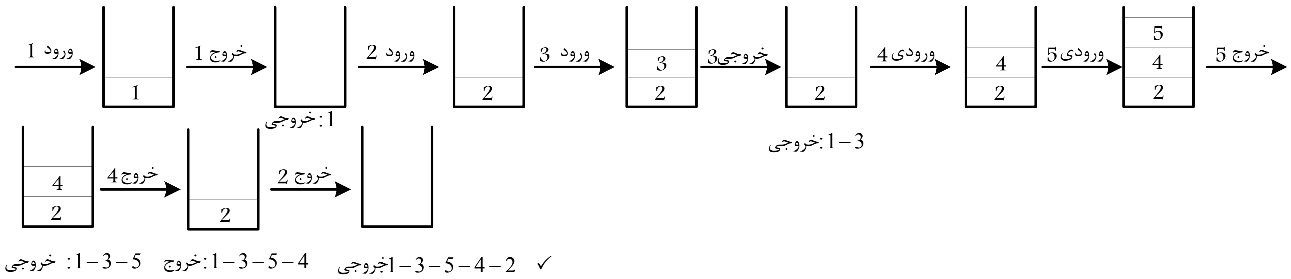
گزینهٔ ۱:



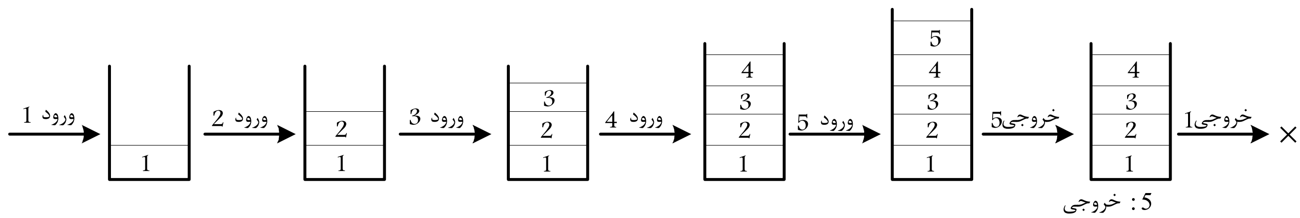
گزینهٔ ۲:



گزینهٔ ۳:



گزینهٔ ۴:



۵۷ ـ عبارت prefix **روبه‌رو داده شده است.** $+-*\uparrow ABCD/E/F+GH$

کدام‌یک از عبارات زیر معادل infix برای این عبارت است؟

۱) $A^{B*(C-D)}+E\left(F/G+H\right)$

۲) $A^{B}*\left(C-D\right)+\frac{EF}{G}+H$

۳) $A^{B}*C-D+E/F/G+H$

۴) $A^{B}*C-D+E/\left(F/\left(G+H\right)\right)$

یادداشت:

....

....

....

....

حل : گزینه ۴ درست است.

از روش پرانتر گذاری استفاده می‌کنیم.

$$+-*\uparrow ABCD/E/F+GH$$
$$(((A\uparrow B)*C)-D)+E/(F/(G+H))$$
$$A^B*C-D+E/(F/(G+H))$$

۵۸ ـ اگر تخصیص حافظه برای آرایه‌ها سطری باشد، در آن صورت برای آرایهٔ سه بعدی $A[1..n_1,1..n_2,1..n_3]$ مؤلفهٔ $A[i$ و j و $k]$ در چه محلی از حافظه قرار می‌گیرد؟ (a آدرس اولین خانه حافظه آرایه می‌باشد.)

۱) $a+(i-1)n_1n_2+(k-1)$

۲) $a+(i-1)n_2n_3+(j-1)n_3+(k-1)$

۳) $a+(k-1)n_1n_2+(j-1)n_2+(i-1)$

۴) $a+(k-1)n_2n_3+(j-1)n_3+(i-1)$

حل : گزینه ۲ درست است.

اگر آرایهٔ $A[L_1..U_1, L_2..U_2, L_3..U_3]$ به‌صورت سطری در حافظه ذخیره شود و a آدرس شروع آرایه باشد آن‌گاه:

$$A[i,j,k]=a+[(i-L_1)(U_2-L_2+1)(U_3-L_3+1)+(j-L_2)(U_3-L_3+1)+(K-L_3)]*\beta$$

که β اندازهٔ هر خانه حافظه است. بنا به فرضیات بالا:

$$\begin{cases}L_1=1\\U_1=n_1\end{cases}\quad\begin{cases}L_2=1\\U_2=n_2\end{cases}\quad\begin{cases}L_3=1\\U_3=n_3\end{cases}$$

$$A[i,j,k]=a+[(i-1)(n_2-1+1)(n_3-1+1)+(j-1)(n_3-1+1)+(k-1)]\times\beta$$

که $\beta=1$:

$$A[i,j,k]=a+[(i-1)n_2n_3+(j-1)(n_3)+(k-1)]$$
$$A[i,j,k]=a+(i-1)n_2n_3+(j-1)n_3+(k-1)$$

۵۹ ـ -کدام‌یک از ترتیب های زیر ساختمان داده‌های همه منظوره برای ذخیره مرتب اشیا (بر اساس مقدار کلید) را به ترتیب از کندترین به سریع‌ترین لیست کرده است؟

۱) جدول درهم‌سازی، آرایه‌های مرتب، لیست‌های زنجیره‌ای

۲) آرایه‌های مرتب، درخت های جستجوی دودویی، لیست‌های زنجیره‌ای، heap

۳) آرایه‌های مرتب، لیست‌های زنجیره‌ای مرتب، درخت‌های جستجوی دودویی

۴) آرایه‌های نامرتب، لیست‌های زنجیره‌ای نامرتب، درخت‌های جستجوی دودویی

حل : گزینه ۳ درست است.

جدول‌های درهم‌سازی یک ساختمان داده‌ی همه منظوره نیست و تنها در ذخیره‌سازی داده و اعمال درج، حذف و جستجو کاربرد دارد.
heap نیز یک ساختمان داده‌ی همه منظوره نیست و کاربردهای آن در صف‌های اولویت، مرتب‌سازی، ادغام لیست‌های مرتب است.
آرایه‌های مرتب و نامرتب، لیست‌های پیوندی مرتب و نامرتب و درخت‌های جستجوی دودویی، ساختمان داده‌های همه منظور است.
که از نظر سرعت، ابتدا درخت‌های جستجوی دودویی سپس لیست‌های مرتب (نیاز به مرتب‌سازی ندارد.) و در آخر آرایه‌های مرتب است.

یادداشت:

....

....

....

....

۶۰ـ برای معکوس کردن یک لیست پیوندی با n گره، به چه تعداد اشاره‌گر نیاز است؟ و مرتبه‌ی زمانی آن چیست؟

۱) 3 اشاره‌گر و $O(n)$

۲) n اشاره‌گر و $O(n)$

۳) 3 اشاره‌گر و $O(n^2)$

۴) 3 اشاره‌گر و $O(n\log n)$

حل : گزینه ۱ درست است.

برای معکوس کردن یک لیست پیوندی با هر تعداد گره تنها به سه اشاره‌گر (p،q،r) نیاز است.

الگوریتم آن به شکل زیر است:

```
p = start;
q = null;
while p ≠ null do
begin
1)r = q;
2)q = p;
3)p = p.link;
4)q.link = r;
end;
start = q;
```

که مرتبه‌ی زمانی آن $O(n)$، (n تعداد گره‌های لیست پیوندی) است.

یادداشت:

....

....

....

....