پاسخنامه کوییز شماره ۱
Time Complexity and Recursion

دانشکده مهندسی برق و کامپیوتر
استاد: دکتر هشام فیلی
ساختمان های داده و الگوریتم - پاییز ۱٤۰۰

---

۱)

0.

$$T(n) = 7T\left(\frac{n}{7}\right) + \frac{n}{\log(n)} = 7^2T\left(\frac{n}{7^2}\right) + 7\frac{\frac{n}{7}}{\log\left(\frac{n}{7}\right)} + \frac{n}{\log(n)}$$

$$= 7^2T\left(\frac{n}{7^2}\right) + \frac{n}{\log(n) - \log 7} + \frac{n}{\log(n)} = nT(1) + \sum_{i=0}^{logn-1} \frac{n}{logn - i}$$

$$= nT(1) + n\sum_{i=1}^{logn} \frac{1}{i} = \theta(n) + \theta\big(nlog(\log(n))\big) = \theta(nlog(\log(n))$$

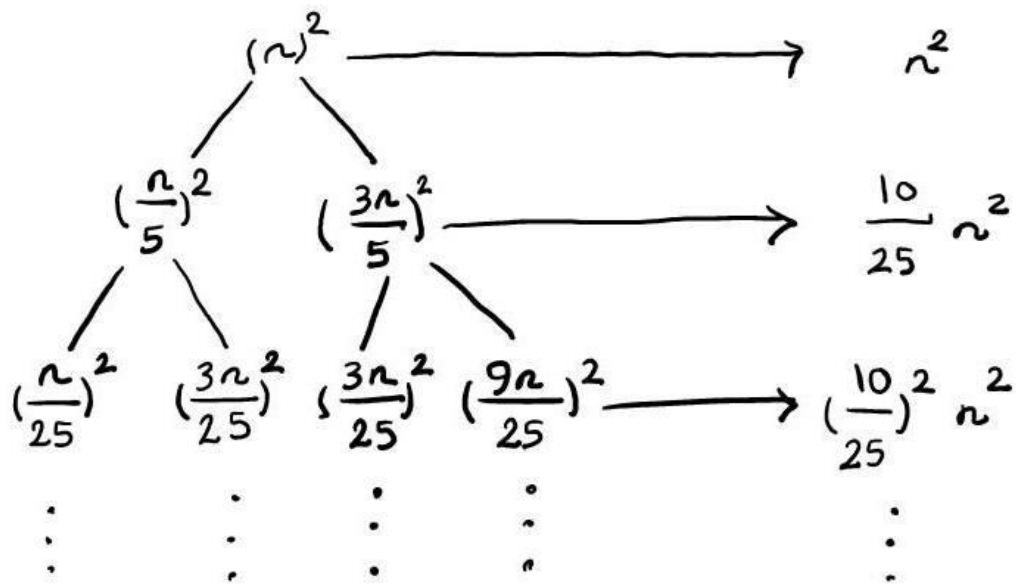
*این رابطه با استفاده از قضیه اصلی قابل حل نمی‌باشد زیرا $\frac{1}{logn} = \frac{\frac{n}{logn}}{n^{\log_7 7}}$ یک عبارت چندجمله‌ای نمی‌باشد.

1.

ابتدا تابع جدید $F(n) = nT(n)$ را در نظر میگیریم و پیچیدگی زمانی آن را محاسبه میکنیم :

$$F(n) = F\left(\frac{n}{5}\right) + F\left(\frac{3n}{5}\right) + n^2$$

$F(n)$ را با استفاده از درخت بازگشت حل میکنیم:



واضح است که درخت نامتوازن می‌باشد و ارتفاع درخت از مرتبه $\log(n)$ می‌باشد، پس داریم:

$$F(n) = n^2\left(1+\frac{10}{25}+\left(\frac{10}{25}\right)^2+\cdots\right)$$

$$= n^2 \sum_{i=0}^{\log(n)-1}\left(\frac{10}{25}\right)^i \leq n^2\sum_{i=0}^{\infty}\left(\frac{10}{25}\right)^i = n^2\frac{1}{1-\frac{10}{25}} = \frac{15}{25}n^2 \rightarrow F(n) = O(n^2)$$

$$F(n) = nT(n) = O(n^2) \rightarrow T(n) = O(n)$$

2.

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

$$F(n) = \frac{T(n)}{n} \rightarrow F(n) = F(\sqrt{n}) + \log(\log(\mathrm{n}))$$

$$n = 2^{2^m} \rightarrow F\left(2^{2^m}\right) = F\left(2^{2^{m-1}}\right) + m$$

$$G(m) = F\left(2^{2^m}\right) \rightarrow G(m) = G(m-1) + m = G(m-2) + G(m-1) + m$$

$$G(m) = G(1) + (1+2+\cdots+(m-1)+m) \rightarrow G(m) = O(m^2)$$

$$G(m) = O(m^2) = F\left(2^{2^m}\right) = F(n) = O((\log(\log(n))^2) = \frac{T(n)}{n}$$

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

۲)

$$* \sum_{i=0}^{n} \frac{n!}{i!} = e^n$$

$$* \sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = O(n^4)$$

$$\mathbf{0}.\ logn! < n^2 < n^{loglogn} < 2^n < n2^n < \sum_{i=0}^{n} \frac{n^i}{i!}$$

$$\mathbf{1}.\ log(log * n) < \ log * logn < loglogn < \sum_{i=0}^{n} \frac{n^i}{i!} < n! < n^n$$

$$\mathbf{2}.\ log * n < nloglogn < \sum_{i=0}^{n} i^3 < logn^{logn} < n^n < n^{2^n}$$

۳)

.۰

زمان اجرای حلقه بیرونی $O(log * n)$ می‌باشد و زمانی که حلقه داخلی زام اجرا می‌شود، داریم: $k = \frac{j(j+1)}{2}$. اگر فرض کنیم که حلقه داخلی m بار اجرا شود، داریم:

$$\frac{m(m+1)}{2} < n \rightarrow m^2 + m - 2n < 0$$

$$m^2 + m - 2n = 0 \rightarrow m = \frac{\sqrt{1+8n} - 1}{2} \rightarrow m = O(\sqrt{n})$$

پس حلقه داخلی $O(\sqrt{n})$ بار اجرا می‌شود. بنابراین کل زمان اجرا $O(\sqrt{n}log * n)$ می‌باشد.

.۱

حلقه سوم n بار اجرا می‌شود و پس از هر دفعه اجرا شدن (جز دفعه آخر) j دو برابر می‌شود (n-1 بار).

بنابراین دفعه دومی که شرط j > n چک می‌شود $n \ll 2^{n-1}$ خواهد بود و برنامه خاتمه میابد.

اگر فرض کنیم که حلقه بیرونی m بار اجرا شود، داریم:

$$n^{1/5^m} > 2 \rightarrow 1/5^m \log_2 n \geq 1 \rightarrow 5^m \leq \log_2 n \rightarrow m \leq O(loglogn)$$

بنابراین زمان کل اجرا $O(nloglogn)$ می‌باشد.

.۲

هربار که حلقه بیرونی iام اجرا می‌شود، حلقه داخلی به اندازه $\log_5 i$ بار اجرا می‌شود. از آنجایی که مبنای لگاریتم در محاسبه زمان اجرا مهم نمی‌باشد، مبنای تمام لگاریتم‌ها را ۲ در نظر می‌گیریم و کل زمان اجرا به صورت زیر محاسبه می‌شود:

$$\log\frac{n}{2} + \log\frac{n}{4} + \log\frac{n}{8} + \cdots = (logn)^2 - (1 + 2 + \cdots + logn) = (logn)^2 - \frac{(logn)(logn+1)}{2}$$
$$= O((logn)^2)$$

(٤

f(n) = 4^{logn} = 2^{2logn} = $(2^{logn})^2$ = n^2

هنگامیکه دقیق نمیتوانیم بین توابع پیچیدگی ارزیابی کنیم، میتوانیم از تبدیل استفاده کنیم. بطور مثال میتوانید لگاریتم همه توابع را محاسبه کنید

lg(f(n)) = lgn

lg(g(n)) = $(\lg n)^2$

lg(h(n)) = lg(lg n)

واضح است که صریحا میتوان گفت که درجه رشد بصورت زیر است:

Lg(h(n))<lg(f(n))<lg(g(n))

اگر درجه رشد برای lg یک سری توابع اکیدا بزرگتر (یا کوچکتر) باشد، انگاه صریحا درجه رشد همان توابع بزرگتر(یا کوچکتر است) پس اکیدا

$h(n)<f(n)<g(n)$

باتوجه به این گزاره ، گزینه های یک و ۵ درست هستند ولی گزینه های ۲و ۳و ۴ نادرست است.