تمرین شماره ۱

Time Complexity

and

Recursion

دانشکده مهندسی برق و کامپیوتر

مهلت تحویل:

ساختمان های داده و الگوریتم - پاییز 1401

استاد: دکتر هشام فیلی

طراح تمرین : **علی کرامتی**

---

۱. پیچیدگی زمانی هر یک از قطعه کدهای زیر که در زبان cpp نوشته شده اند را از روش **سیگما** نویسی محاسبه کنید.(راه حل شهودی به تنهایی قابل قبول نیست و تنها می‌تواند تکمیل کننده پاسخ شما باشد) (15 نمره)

الف)

```cpp
if (n % 2 == 0) {
    for (int i = 0; i < n; i++) {
        for (int j = i; j > 0; j--) {
            for (int k = n; k >= 1; k /= 2) {
                count *= 3;
            }
        }
    }
}
```

ب)

```cpp
for (int i = 1; i < n; i++) {
    int j = 1;
    while(j * j <= i){
        j++;
    }
}
```

ج)

```
bool isSymmetric(int mat[][MAX], int N)
{
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++)
            if (mat[i][j] != mat[j][i])
                cout << "Matrix is not symmetric!" << endl;
                return true;
    }
    return false;
}
```

پاسخ.

الف) در بهترین حالت (n فرد) وارد اسکوپ مربوط به if نمی شویم و پیچیدگی زمانی قطعه کد $O(1)$ خواهد بود. اما در بدترین حالت و زمانی که n زوج است، حلقه اول n بار تکرار می شود و حلقه دوم i بار تکرار می شود و در نهایت حلقه سوم $\log n$ بار تکرار می شود. بنابراین داریم:

$$T(n) = \sum_{i=0}^{n}\sum_{j=1}^{i}\sum_{k=1}^{\log n} O(1) = \sum_{i=0}^{n}\sum_{j=1}^{i} \log n = \log n \sum_{i=0}^{n}\sum_{j=1}^{i} 1 = \log n \sum_{i=0}^{n} i$$

$$= \log n \frac{(n+1)(n+0)}{2} = n^2 \log n$$

پس پیچیدگی الگوریتم برابر است با: $O(n^2 \log n)$.

ب) حلقه اول n بار تکرار می شود و حلقه دوم $\sqrt{i}$ بار تکرار می شود. بنابراین داریم:

$$T(n) = \sum_{i=1}^{n}\sum_{j=1}^{\sqrt{i}} O(1) = \sum_{i=1}^{n} \sqrt{i} = \sqrt{1} + \sqrt{2} + \sqrt{3} + \sqrt{4} + ... + \sqrt{n}$$

$$\leq \sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n} + ... + \sqrt{n} = n\sqrt{n}$$

پس پیچیدگی الگوریتم برابر است با: $O(n\sqrt{n})$.

ج) در تابع مربوط به بررسی کردن اینکه ماتریس داده شده متقارن است یا خیر، در بهترین حالت زمانی که $i = 1,\ j = 2$ شرط برقرار شده و اجرای آن تمام می شود که در این حالت پیچیدگی زمانی $O(1)$ خواهد بود.

اما بدترین حالت زمانی ست که ماتریس متقارن باشد و در این حالت تا انتها تمام حلقه ها اجرا خواهند شد. بنابراین در این حالت داریم:

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} O(1) = \frac{n(n+1)}{2} = O(n^2)$$

ولی در قطعه کد داده شده با توجه به اینکه if بدون curly bracket نوشته شده است، عبارت return داخل اسکوپ اولین for میافتد. به عبارتی زمانی که $i = 0$ است، j از 0 تا n پیشمایش کرده و پس از آن دستور return اجرا می شود.

پس در این قطعه کد داده شده پیچیدگی الگوریتم برابر است با: $O(n)$

۲. در هر مورد توابع را برحسب درجه رشدشان مرتب کنید.(از روش حدگیری در بی نهایت تنها برای چک کردن درستی پاسخ نهایی خود می توانید استفاده کنید و صرف نوشتن حد بدون راه حل، نمره ای به شما تعلق نخواهد گرفت)(24 نمره)

الف)

$$n^2, (\frac{5}{4})^n, (\log n)^2, \log n!$$

ب)

$$\log n!, \sqrt{2}^{(log n)^2}, n^{\log\log n}, (\log n)!$$

ج)

$$\sum_{j=1}^{n}\sum_{i=1}^{j} i,\ n^{\frac{1}{\log n}},\ \sum_{k=0}^{n}\frac{n^k}{k!},\ 2^n$$

پاسخ.

الف) ابتدا $\log n!$ را ساده می کنیم و پیچیدگی زمانی معادل آن را **دو طرفه** محاسبه می کنیم:

طرف اول: $O(\log n!)\ \subseteq\ O(n\log n)$

$$log\, n! = log\,(n\times n-1\times ...\times 1) = log\, n + log\,(n-1) + ... + log\, 1$$

$$\leq log\, n + log\, n + ... + log\, n\ = n\times log\, n$$

بنابراین اثبات کردیم: $O(\log n!)\ \subseteq\ O(n\log n)$

طرف دوم: $O(n\log n) \subseteq O(\log n!)$

$$log\, n! = log\,(n \times n-1 \times ... \times 1) = log\, n + log\,(n-1) + ... + log\, 1$$

حال اگر به جای نیمه بزرگتر عبارت بالا (از $\log n$ تا $\log \lfloor \frac{n}{2} \rfloor$) به جای همه آن ها $\log \frac{n}{2}$ قرار دهیم

$log\, n!$ از عبارت ساخته شده جدید بزرگتر خواهد بود:

$$log\, n! = log\, n + log\,(n-1) + ... + log\, 1$$

$$\geq log\frac{n}{2} + log\frac{n}{2} + ... + log\frac{n}{2} + ... + log\, 2 + log\, 1$$

$$= \lfloor \frac{n}{2} \rfloor \times log\lfloor \frac{n}{2} \rfloor + log\frac{n}{2} + ... + log\, 2 + log\, 1 = O(n\, logn)$$

پس اثبات کردیم که $O(n\log n) \subseteq O(\log n!)$.

در نهایت می توان ادعا کرد: $O(n\log n) = O(\log n!)$.

همچنین از تقریب استرلینگ نیز می توان برای محاسبه پیچیدگی زمانی معادل $\log n!$ استفاده کرد:

$$O(n\log n) = O(\log n!).$$

برای اثبات تقریب استرلینگ به این **لینک** مراجعه شود.

حال می توان گفت که:

1. $\log n! = n\ \log n > (\log n)^2 = \log n \times \log n$
2. $n^2 > \log n! = n\ \log n$

3. $(\frac{5}{4})^n > n^2$ تابع نمایی با پایه بزرگتر از یک از چند جمله ای با توانی ثابت بزرگتر است

ب) ابتدا از چهار تابع $\log n!, \sqrt{2}^{(logn)^2}, n^{\log\log n}, (\log n)!$ لگاریتم می گیریم:

با توجه به اینکه در بخش قبل اثبات کردیم $(\log n)! = n \log n$ موارد 1و2 را با توجه به این

عبارت بازنویسی می کنیم.

1. $\log(\log n!) = \log(n \times \log n) = \log n + \log\log n = O(\log n)$

2. $\log((\log n)!) = (\log n) \times \log(\log n)$ تغییر متغیر $n = \log n$

3. $\log(n^{\log \log n}) = \log\log n \times \log n$

4. $\log(\sqrt{2}^{(\log n)^2}) = (\log n)^2 \times \log\sqrt{2} = O((\log n)^2)$

همانطور که می دانید، اگر لگاریتم گیری دو تابع مشخص کند که یکی از آنها بزرگتر است، حتما خود تابع

اصلی نیز از دیگری بزرگتر است. ولی اگر لگاریتم دو تابع مساوی شود، لزوما دو تابع مساوی نیستند! مثلا به مثال

زیر توجه کنید:

$$O(n^2) \bigcirc^{>=<} O(n) \rightarrow \log(n^2) \bigcirc^{?} \log(n) \rightarrow 2\log n \bigcirc^{?} \log n \Rightarrow O(n^2) = O(n) ※$$

پس برای این مسئله باید راه دیگری یافت. از تغییر متغیر $n = (\log n)^k$ استفاده می کنیم.

$$n = (\log n)^k \rightarrow k = \log^{n}_{\log n} = \frac{\log n}{\log(\log n)}$$

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

از طرفی طبق تقریب استرلینگ داریم:

$$n! \sim \sqrt{2\pi n}\,(\frac{n}{e})^{n}$$

اگر به جای n عبارت $\log n$ را قرار دهیم داریم:

$$(\log n)! \sim \sqrt{2\pi(\log n)}\,(\frac{(\log n)}{e})^{(\log n)} \sim \frac{(\log n)^{\log n+0.5}}{e^{\log n}} = (\log n)^{\log n} \times \frac{\sqrt{\log n}}{e^{\log n}} \quad (*)$$

حال باید $e^{\log n}$ را ساده کنیم.

$$e^{\log n} = z \rightarrow \log(e^{\log n}) = \log(z) \rightarrow \log n \times \log e = \log z$$

از آنجایی که لگاریتم طبیعی عدد e برابر 1 یک می باشد. می توان نتیجه گرفت:

$$\log n = \log z \Rightarrow n = z \Rightarrow e^{\log n} = n$$

حال با جایگذاری $e^{\log n} = n$ در عبارت $(*)$ داریم:

$$(\log n)! \sim (\log n)^{\log n} \times \frac{\sqrt{\log n}}{n}$$

و در نهایت می توان گفت: $O((\log n)!) < O(n^{\log(\log n)})$

بنابراین: $\sqrt{2}^{(\log n)^2} > n^{\log \log n} > (\log n)! > \log n!$

ج) ابتدا دو تابع سیگمایی را ساده تر می کنیم:

1. $\sum_{j=1}^{n}\sum_{i=1}^{j} i = n^{3}$

2. $\sum_{k=0}^{n} \frac{n^k}{k!} \sim e^n$ این تابع در بی نهایت، تعریف تابع e^n است

تا اینجا می توان گفت که $\sum_{k=0}^{n} \frac{n^k}{k!}$ از 2^n بزرگتر است، همچنین می دانیم هر تابع نمایی با پایه بزرگتر از یک از هر چندجمله ای با توان ثابت بزرگتر است. پس $\sum_{j=1}^{n} \sum_{i=1}^{j} i$ از هر دو تابع نمایی کوچکتر است.

حال برای مقایسه $\sum_{j=1}^{n} \sum_{i=1}^{j} i = n^3,\ n^{\frac{1}{\log n}}$ از دو طرف لگاریتم می گیریم.

$$\log(n^3) = 3 \log n \qquad\qquad \log(n^{\frac{1}{\log n}}) = \frac{1}{\log n} \times \log n = 1$$

بنابراین:

$$\sum_{k=0}^{n} \frac{n^k}{k!} > 2^n > \sum_{j=1}^{n} \sum_{i=1}^{j} i > n^{\frac{1}{\log n}}$$

۳.هر یک از موارد زیر را اثبات یا رد کنید.(24نمره)

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

ب) $f(n) \in O(g(n)) \Rightarrow O(\log_2 f(n)) \in O(\log_2 g(n))$

ج) $f(n) = O(g(n)),\ g(n) = O(h(n)) \rightarrow f(n) = O(h(n))$

د) $\log_k n = \theta(\log_2 n)$

پاسخ.

الف) غلط است. مثال نقض:

دو تابع دو ضابطه ای زیر را تعریف می کنیم.

$$f(n) = \{if\ n\ is\ odd: 0,\ if\ n\ is\ even: 1\},\ g(n) = \{if\ n\ is\ odd: 1,\ if\ n\ is\ even: 0\}$$

بررسی $f(n) = O(g(n))$:

در این حالت، باید نشان دهیم که $c, n_0 > 0$ وجود دارد، به طوری که به ازای هر $n > n_0$ داشته باشیم:

$$f(n) \le c.g(n).$$

حال فرض کنید $n = z.n_0 + 1$. بنابراین طبق تعریف بالا: $f(n) = 0,\ g(n) = 1$. که چنین c ای وجود ندارد

که رابطه بالا را برقرار کند.

حال به صورت مشابه نشان می دهیم که برای دو تابع مثال زده شده، رابطه $f(n) = \Omega(g(n))$ نیز برقرار نخواهد بود.

مثال نقض دیگر: $f(n) = x.\sin(x), g(n) = x.\cos(x)$

ب) غلط است. مثال نقض:

$$f(n) = 2,\ g(n) = 1$$

مشخص است که رابطه $f(n) = O(g(n))$ برقرار است.

اما در حالت $O(\log_2 f(n)) \in O(\log_2 g(n))$ باید نشان دهیم که $c, n_0 > 0$ وجود دارد، به

طوری که به ازای هر $n > n_0$ داشته باشیم:

$$f(n) \le c.g(n) \to 1 \le c.0.$$

مشخص است در این حالت هیچ c ای وجود ندارد که رابطه $1 \le c.0$ برقرار باشد.

پ.ن: این رابطه در صورتی برقرار است که دو تابع $f(n),\ g(n)$ *صعودی و نامنفی باشند.*

ج) درست است. اثبات:

$$f(n) = O(g(n)) \Rightarrow \exists_{c_1, n_1} > 0 \quad such\ that \ \ \forall_{n>n_1} 0 \le f(n) \le c_1 . g(n)$$

$$g(n) = O(h(n)) \Rightarrow \exists_{c_2, n_2} > 0 \quad such\ that \ \ \forall_{n>n_2} 0 \le g(n) \le c_2 . h(n)$$

اگر $n_3 = \max(n_1, n_2)$ و $c_3 = c_1 . c_2$باشد، داریم:

$$\forall_{n>n_3} 0 \le f(n) \le c_1 . g(n) \le c_1 . c_2 h(n) \rightarrow \forall_{n>n_3} 0 \le f(n) \le c_1 . g(n) \le c_3 . h(n)$$

$$\forall_{n>n_3} 0 \le f(n) \le c_3 h(n) \Rightarrow f(n) = O(h(n))$$

د) درست است. اثبات:

برای برقراری این رابطه، باید اثبات کنیم:

$$\log_k n = O(\log_2 n), \ \log_k n = \Omega(\log_2 n)$$

برای اینکه رابطه $\log_k n = O(\log_2 n)$ برقرار باشد، باید اثبات کنیم:

$$\exists_{c\ ,n_0} > 0 \quad such\ that \ \ \forall_{n>n_0} 0 \le \log_k n \le c_0 . \log_2 n \qquad (*)$$

می توان گفت رابطه (*)، در صورتی همواره برقرار است که: $c_0 \ge \frac{\log_k n}{\log_2 n} = \log_k 2$.

از آنجایی که $\log_k 2$ یک عدد ثابت است، پس می توان هر عدد ثابت بزرگتر از $\log_k 2$ را انتخاب کرد، به

طوری که رابطه $(*)$ برقرار باشد. پس به ازای $c_0 = \log_k 2,\ n_0 = 1$ رابطه $(*)$ برقرار خواهد بود و در

نتیجه اثبات کردیم: $\log_k n = O(\log_2 n)$.

به صورت مشابه و تنها با جاگذاری علامت کوچکتر مساوی به جای علامت های بزرگتر مساوی، برای رابطه

$\log_k n = \Omega(\log_2 n)$ نیز اثبات می شود.

۴.پیچیدگی زمانی روابط بازگشتی زیر را با روش قضیه اصلی محاسبه کنید.(28نمره)

الف) $T(n) = 9T(\frac{n}{3}) + n^2\log^3 n$

ب) $T(n) = 2^n T(\frac{n}{2}) + n^n$

ج) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

د) $T(n) = T(\frac{n}{2}) + n(5 - cos(n))$

پاسخ.

الف) اگر طبق قضیه اصلی پیش برویم،

$$a = 9,\ b = 3 \Rightarrow c = 2\ ,\ f(n) = n^2(\log n)^3$$

به نظر می رسد که بند سوم برقرار است اما $n^2(\log n)^3 = \Omega(n^{2+\varepsilon})$ به ازای هیچ $\varepsilon > 0$ برقرار

نیست. چون رشد هر توانی از n از رشد هر توانی از log n بزرگتر است.

راه حل از طریق جایگذاری:

$$T(n) = 9^k T(\frac{n}{3^k}) + n^2(\log^3(\frac{n}{3^{k-1}})) +\ldots+ \log^3 n$$

اگر $k = \log_3 n \rightarrow 3^k = n$، داریم:

$$T(n) = (n)^2 T(1) + n^2(\log^3(\frac{n}{3^k.3^{-1}})) + n^2(\log^3(\frac{n}{3^k.3^{-2}})) +\ldots+ \log^3 n$$

$$= n^2 + n^2(\log^3 3 + \log^3 9 +\ldots+ \log^3(3^{\log_3 n}))$$

$$= n^2 + n^2(1 + 2^3 + 3^3 +\ldots+ \log^3 n) = n^2 + n^2 \frac{(\log^2 n(\log n+1)^2)}{4}$$

بنابراین می توان گفت: $T(n) = O(n^2 \log^4 n)$

همچنین این مسئله از طریق قضیه اصلی تعمیم یافته نیز قابل حل است:

$$T(n) = aT(\frac{n}{b}) + n^{\log_b a} \log^k n,\ k \geq 0 \Rightarrow T(n) = \theta(n^{\log_b a}.\log^{k+1} n)$$

ب) با قضیه اصلی نمی توان حل کرد چون a عدد ثابت نیست.

راه حل از طریق حدس. ادعا: $T(n) \leq 2n^n$

$$T(n) \leq 2^n.2(\frac{n}{2})^{\frac{n}{2}} + n^n = 2^{\frac{n}{2}}.n^{\frac{n}{2}}.2 + n^n = (2n)^{\frac{n}{2}}.2 + n^n$$

پس می توان گفت با شرط $n \geq 4$ داریم:

$$T(n) \leq 2n^n \Rightarrow T(n) = O(n^n)$$

ج) با قضیه اصلی نمی توان حل کرد چون a عدد ثابت نیست.

راه حل از طریق تغییر متغیر: $m = \log n \rightarrow n = 2^m$

$$T(2^m) = 2^{\frac{m}{2}}.T(2^{\frac{m}{2}}) + 2^m \to \frac{T(2^m)}{2^m} = \frac{T(2^{\frac{m}{2}})}{2^{\frac{m}{2}}} + 1$$

حال از تغییر تابع $S(m) = \frac{T(2^m)}{2^m}$ استفاده می کنیم:

$$\to S(m) = S(\frac{m}{2}) + 1$$

حال می توانیم از قضیه اصلی استفاده کنیم:

$$a = 1, b = 2 \to c = \log_2 1 = 0 \Rightarrow S(m) = O(\log m) \to \frac{T(2^m)}{2^m} = O(\log m)$$

$$T(2^m) = 2^m.O(\log m) \Rightarrow T(n) = O(n \times \log(\log n))$$

د) کیس سوم برقرار است ولی شرط regularity نقض شده است. مثلا فرض کنید $n = 2\pi k$ و k عددی فرد

و بسیار بزرگ باشد، به ازای هر n می توانیم نشان دهیم $c \geq \frac{3}{2}$ که این شرط regularity را نقض می کند،

پس از قضیه اصلی نمی توان استفاده کرد. بنابراین باید رابطه بازگشتی را به شکل دیگری بازنویسی کنیم. از

آنجایی که $|\cos n| \geq 1$، داریم:

$$|\cos n| \geq 1 \to -1 \leq \cos n \leq 1 \to 4 \leq 5 - \cos n \leq 6$$

پس از آنجایی که $5 - \cos n$ همواره بین دو عدد ثابت است می توان رابطه بازگشتی را به شکل روبه رو

بازنویسی کنیم: $T(n) = T(\frac{n}{2}) + \theta(n)$

حال می توانیم از قضیه اصلی استفاده کنیم:

$$a = 1,\ b = 2 \to c = \log_2 1 = 0 \Rightarrow T(n) = \theta(n)$$

۵) فرض کنید در مساله برج های هانوی، تنها بتوان حلقه ها را بین میله های مجاور حرکت داد. به عبارت دیگر، حلقه ها نمی توانند مستقیما بین میله های مبدا و مقصد جا به جا شوند. یک تابع بازگشتی برای این مسئله نوشته و پیچیدگی زمانی تابع خود را تحلیل کنید.(10نمره)

پاسخ.

در قطعه کد زیر n تعداد دیسک ها، from و with و to به ترتیب شماره دیسک های اول، دوم و سوم هستند.

```
void LimitedHanoi(int n, int from, int with, int to)
{
    if(n == 1)
    {
        cout << " : " << form << " ----> " << with << endl;
        cout << " : " << with << " ----> " << to << endl;
    }
    else
    {
        LimitedHanoi(n-1, from, with, to);
        cout << " : " << form << " ----> " << with << endl;
        LimitedHanoi(n-1, to, with, from);
        cout << " : " << with << " ----> " << to << endl;
        LimitedHanoi(n-1, from, with, to);
    }
}
```

رابطه بازگشتی تابع: $T(n) = 3T(n-1) + 2$ که $T(1) = 2$.

برای حل رابطه بالا، به دو طرف رابطه بازگشتی عدد 1 را اضافه می کنیم.

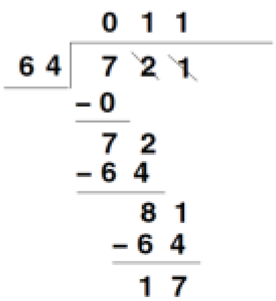
$$T(n) + 1 = 3T(n-1) + 3 \rightarrow T(n) + 1 = 3(T(n-1) + 1)$$

$$\Rightarrow T(n) + 1 = 3^{n-1}(T(1) + 1) \Rightarrow T(n) = 2.3^{n-1} - 1$$

پس پیچیدگی زمانی تابع LimitedHanoi برابر است با: $T(n) = O(3^n)$

۶) الگوریتم یا شبه کدی ارائه دهید که تقسیم بلند را در پایه ای ثابت انجام می دهد و باقی مانده و خارج قسمت نهایی را برگرداند. (پیچیدگی زمانی الگوریتم شما باید در اردر خطی باشد.)

برای مثال تقسیم بلند 721 بر 64 مشابه زیر است (اگر عادت دارید مقسوم علیه را در سمت راست بنویسید به میل خود جای آن را عوض کنید).



ابتدا الگوریتم چک می کند که چند بار 64 عدد 7 را میشمارد، زیرا عدد 7 بیش مرتبه ترین رقم عدد 721 است. از آنجا که 7 < 64، پس جواب 0 است. سپس الگوریتم بیش ترین مرتبه رقم باقیمانده یعنی 2 را به سمت راست باقیمانده تقسیم قبلی یعنی 7 اضافه میکند و مشابه کار قبلی را انجام میدهد. در اینجا 64 یک بار عدد 72 را میشمارد و باقیمانده عدد 8 می شود. باقیمانده پایانی باقیمانده کل تقسیم است.

پاسخ.

$Div(a, b, n) =$

PRE: $a > 0, b \geq 0$, a and b are stored with n bits.

POST: $qa + r = b, 0 \leq r < a$

$r \leftarrow 0$ O(1)

for $i \leftarrow n-1$ **to** 0 **do**

INV: $(q_{n-1}...q_{i+1}) \cdot a + r = (b_{n-1}...b_{i+1}), 0 \leq r < a$

$r \leftarrow (r \ll 1) + b_i$ /* Switch to next digit */ O(1)

$q' \leftarrow 0$ O(1)

$a' \leftarrow 0$ O(1)

O(10) **while** $a' + a \leq r$ **do** /* Find the maximum q' so that $q'a \leq r$ */

INV: $a' = q'a \leq r$

$a' \leftarrow a' + a$

$q' \leftarrow q' + 1$

$q_i \leftarrow q'$ O(1)

$r \leftarrow r - a'$ O(1)

return $\langle q, r \rangle$ /* quota and remainder */

برای محاسبه پیچیدگی الگوریتم، از حلقه for شروع می کنیم.

سه خط داخلی و قبل از while پیچیدگی زمانی $O(1)$ دارند. از آنجایی که اسکوپ مربوط به while تکه کدی ست که خارج قسمت را رقم به رقم محاسبه می کند، پس q' حداکثر می تواند عدد 9 را اختیار کند، بنابراین پیچیدگی زمانی while حداکثر می تواند $O(10)$ باشد. دو خط بعد while نیز پیچیدگی زمانی $O(1)$ دارند. بنابراین اجزای داخلی حلقه for در بدترین حالت پیچیدگی زمانی $O(10)$ دارند. پس پیچیدگی زمانی تابع div برابر $O(10n)$ خواهد بود که در اردر خطی می باشد.

نکات تکمیلی

- پاسخ های خود را تا زمان معین شده در سایت آپلود نمایید.
- هدف این تمرین یادگیری شماست. لطفا تمرین را خودتان انجام دهید. در صورت کشف تقلب مطابق با قوانین درس با آن برخورد خواهد شد.

- دقت فرمایید که پاسخ سوال‌ها یکتا نیست و به دیگر پاسخ‌های صحیح نیز نمره تعلق می‌گیرد.
- در صورت وجود ابهام در مورد سوالات می‌توانید از طریق ایمیل با من در ارتباط باشید.

شاد باشید.