تحلیل الگوریتم :

۱_ آرایه T شامل n عنصر متمایز را درنظر بگیرید. بین یک جفت اندیس مانند (i,j) از این آرایه یک نابه‌جایی وجود دارد اگر:

$j > i \ , \ T[j] < T[i]$

الف) رابطه‌ی بین زمان اجرای insertion sort و تعداد نابه‌جایی‌ها در آرایه ورودی چیست؟

می‌دانیم الگوریتم insertion sort این است:

1. for j = 2 to n	$c_1 \cdot n$
2. key ← A[j]	$c_2 \cdot (n-1)$
3. i ← j-1	$c_3 \cdot (n-1)$
4. while (i>0 & key<A[i])	$c_4 \cdot (t_j+1)$
5. A[i+1] ← A[i]	$c_5 \cdot t_j$
6. i ← i-1	$c_6 \cdot t_j$
7. A[i+1] ← key	$c_7 \cdot (n-1)$

t_j زمانی است که برنده‌ی حلقه‌ی while وقتی j ام حلقه‌ی for تکرار می‌شود

t_j متغیر است چه تعداد نابجایی ها بیشتر باشد از آن جایی که ما می‌خواهیم صعودی مرتب کنیم، پس هر تعداد نابجایی ها بیشتر باشد، خانه های بیشتری از آرایه را باید جابه جا کنیم و مسلماً زمان بیشتری هم طول می‌کشد.

best worst

$0 \leq t_j \leq j-1$:

best case → آرایه صعودی مرتب شده باشد. → $0 =$ تعداد نابجایی → $T(n) = a \cdot n + b$

worst case → آرایه نزولی مرتب شده باشد. → در مرحله‌ی j ام باید j-1 مقایسه صورت گیرد

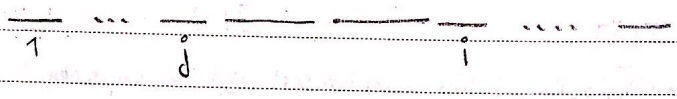
$T(n) = a' \cdot n^2 + b' \cdot n + c'$

⇐ به طور کلی می‌توان گفت زمان insertion sort و تعداد نابجایی با هم رابطه‌ی عکس دارند.

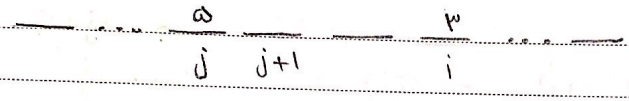
PAPCO

Scanned with CamScanner

ب) اگر در آرایه T یک نابه‌جایی بین دو اندیس i و j وجود داشته باشد، نشان دهید این آرایه دارای حداقل j-i نابه‌جایی است.



آرایه‌ی بالا را در نظر بگیرید. فرض بر این است که یک نابه‌جایی بین i و j وجود دارد. پس خانه‌ی iام کوچکتر از j است. برای مثال مقادیر فرضی در آن‌ها قرار می‌دهیم.



برای هر خانه بین i و j دو حالت داریم

مقدار آن از d (خانه‌ی jام بیشتر باشد) ← نابه‌جایی برای خانه‌ی iام (مقدار ۳ در اینجا)

مقدار آن از d (خانه‌ی jام کمتر باشد) ← نابه‌جایی برای خانه‌ی jام (اینجا مقدار ۵)

پس برای همه‌ی خانه‌های بین i و j نابه‌جایی برقرار است. ⟸ i-j-1

خود خانه‌ی i و j هم نابه‌جایی دارند ⟸ j-i=i-j-1

و این j-i تعداد حداقل نابه‌جایی است زیرا اگر در خانه‌های قبل خانه‌ی jام، خانه‌ای مقدار آن بیشتر از خانه‌ی jام باشد، نابه‌جایی داریم و اگر در خانه‌های بعد خانه‌ی iام، خانه‌ای با مقدار کمتر از خانه‌ی iام وجود باشد، باز هم نابه‌جایی داریم. ⟸ حداقل j-i نابه‌جایی داریم.

Scanned with CamScanner

۲- زمان اجرای شبه کد زیر را (خط به خط) تحلیل کنید.

1. x = 0

initialization ⟶ از دستورات پایه ⇒ $\boxed{c_1}$

2. for (i=1; i<=n; i++) {

هزینه‌ی یک بار اجرای header در for ثابت است (c_2) و تکرار header از بدنه یک بیشتر است. ⇐ $\boxed{c_2 \cdot (n+1)}$

3. $\underbrace{\text{for (j=1; j<=n; j++)}}_{1}$ $\underbrace{\text{x++;}}_{2}$

1: برای این قسمت مثل خط دوم عمل می‌کنیم؛ خودش $c_3 \cdot (n+1)$ هزینه دارد و به خاطر for بالایی n بار تکرار می‌شود:

⇒ $\boxed{c_3 \cdot n(n+1)}$

2: خودش هزینه‌ی ثابت دارد و چون در بدنه‌ی for دوم است، n بار و for دوم هم خود n بار تکرار می‌شود:

⟹ $\boxed{c_4 \cdot n^2}$ ⟹ $\boxed{c_3 \cdot n(n+1) + c_4 \cdot n^2}$

4. j=1; خودش از دستورات پایه است و هزینه‌ی ثابت c_5 را دارد چون داخل بدنه‌ی for اصلی است، n بار تکرار می‌شود ⇐ $\boxed{c_5 \cdot n}$

5. while (j<n) {

بستگی به مقدار n دارد چون شروع j از 1 است و به موازات با توجه به بدنه‌ی while ۲ برابر می‌شود؛ و چون تعداد تکرار header از بدنه یک بیشتر است.

$c_6 \cdot (n+1) \underbrace{\left(\left\lceil \frac{n}{2} \right\rceil + 1\right)}_{\text{تکرار while (header)}}$

* باید توجه شود که خود while روی for است پس تکرار آن هم حساب می‌شود

6. $\underbrace{\text{x++;}}_{1}$ $\underbrace{\text{j=j*2;}}_{2}$

1: از دستورات پایه است و هزینه‌ی ثابت دارد

$c_7 \cdot \underbrace{\left(\left\lceil \frac{n}{2} \right\rceil\right)}_{\text{تکرار while}} \cdot \underbrace{(n+1)}_{\text{تکرار for}}$

Scanned with CamScanner

هزینه ثابت ← 2 : از دستورات پایه

$c_8 \cdot \left(\left\lceil \frac{n}{2} \right\rceil\right) \cdot (n+1)$

$$\Longrightarrow \boxed{c_7 + c_8 \cdot \left(\left\lceil \frac{n}{2} \right\rceil \cdot (n+1)\right)}$$

7. }

8. }

$$\Rightarrow T(n) = c_1 + c_2(n+1) + c_3 \cdot n(n+1) + c_4 \cdot n^2 + c_5 \cdot n + c_6(n+1)\left(\left\lceil \frac{n}{2} \right\rceil + 1\right) + (c_7 + c_8) \cdot \left(\left\lceil \frac{n}{2} \right\rceil \cdot (n+1)\right)$$

$$= \boxed{a \cdot n^2 + b \cdot n + c}$$

۳- با استفاده از شبه کد زیر می‌خواهیم index عدد a در آرایه‌ی A[1...n] بیابیم. اگر بدانیم احتمال حضور a در تمام درایه‌های $\frac{1}{3}$ ابتدایی آرایه برابر x باشد، در تمام درایه‌های $\frac{1}{3}$ میانی برابر 3x و در تمام درایه‌های $\frac{1}{3}$ پایانی برابر 4x باشد؛ شبه کد زیر را در بهترین، بدترین و حالت متوسط بررسی کنید. (محاسبات دقیق)

```
for (i=1 ; i<=n ; i++)
        if (A[i] ==a)
            return i;
```

بسته به اینکه عدد a در کجای آرایه قرار بگیرد، تعداد تکرار متفاوت است؛ پس آن را t_i می‌نامیم چون به i وابسته است. هر چه i در خانه‌های اولیه قرار گرفته باشد، الگوریتم زودتر به پایان می‌رسد و تکرار آن کمتر است؛ در این مسئله بحث احتمال نیز دخالت دارد و روی t_i اثر می‌گذارد.

Scanned with CamScanner

احتمال

if $1 \leq i < \left\lfloor \frac{n}{3} \right\rfloor$ $\quad \frac{1}{10}$

if $\left\lfloor \frac{n}{3} \right\rfloor \leq i < \left\lfloor \frac{2n}{3} \right\rfloor$ $\quad \frac{3}{10}$

if $\left\lfloor \frac{2n}{3} \right\rfloor \leq i \leq n$ $\quad \frac{6}{10}$

* از آن جایی که داخل if، return دایم دیدنه پایان ش یا به تعداد تکرار if و for با هم برابر است و تفاوت ی اینکه head or tail تکرارش یکی بیشتر است، اینجا صدق نمی کند.

best case:

زمانی که a در خانه ی اول آرایه قرار گرفته باشد.

1. $1 \cdot (c_1)$
2. $1 \cdot (c_2)$
3. c_3

$$T = a \times \overset{\text{احتمال}}{\frac{1}{10}} \in O(1)$$

worst case:

زمانی که a در خانه ی آخر آرایه قرار گرفته باشد.

1. $n \cdot c_1$
2. $n \cdot c_2$
3. c_3

$$T(n) = (an) \times \frac{3}{10} \in O(n)$$

average:

زمانی که a در خانه ی وسط قرار گیرد.

1. $\frac{n}{2} \cdot c_1$
2. $\frac{n}{2} \cdot c_2$
3. c_3

$$T(n) = \frac{n}{2} \cdot c_1 + \frac{n}{2} \cdot c_2 + c_3$$

$$= (an) \times \frac{6}{10} \in O(n)$$

PAPCO

Scanned with CamScanner

مرتب سازی

۴- در یک آرایه می‌خواهیم بررسی کنیم که آیا دو عدد مانند a و b وجود هستند بطوریکه (k متغیر) : $a+b=k$. یک بار برای آرایه‌ی مرتب شده و بار دیگر برای آرایه نامرتب الگوریتم هایی پیشنهاد دهید و پیچیدگی زمان آن‌ها را مقایسه کنید.
(در حالت حل مسئله با آرایه‌ی مرتب شده پیچیدگی زمانی مرتب کردن آرایه را در نظر نگیرید.)

برای آرایه‌ی مرتب شده :

۱) Binary Search : در هر مرحله یک خانه از آرایه را در نظر می‌گیریم (A[i]) و به با استفاده از Binary Search به دنبال $k-A[i]$ می‌گردیم.

آرایه : A ← size: n

```
1. for ( i =0   to   n-1 ) {
2.      int first = 0 ;
3.      int last = n-1;
4.      int middle = (first + last) / 2;
5.      while ( first <= last){
6.          if ( array [middle] < (k-A[i]))
7.              first = middle +1;
8.          else if ( arry [middle] == (k-A[i])
9.              return 0;
10.         else   last =middle -1;
11.         middle = (first + last) /2 } }
12.         if (first > last)
13.             return -1;
```


Scanned with CamScanner

Time Complexity :

size of array in iteration 1 : n

" " 2 : $n/2$

" " 3 : $n/2^2$

⋮

" " k : $n/2^k$

فرض می‌کنیم بعد از k مرحله، به اندازه‌ی یک می‌رسد ← $\frac{n}{2^k} = 1 \longrightarrow n = 2^k$

$\xrightarrow{\log} \log_2 n = \log_2 2^k = k \cancel{\log_2 2} \longrightarrow k = \log_2 n$

$\longrightarrow T(n) = n \cdot \lg n = n \lg n$ $\boxed{T(n) \in O(n \lg n)}$

برای for اصلی ← (for worst case)

2. با استفاده از pointer :

یک پوینتر در ابتدا و یکی در انتها می‌گذاریم. در هر مرحله sum ۲ عدد را مقایسه می‌کنیم با k. اگر بزرگتر بود، پوینتر انتها را یک عقب می‌آوریم و اگر کوچکتر بود، پوینتر ابتدا را یک جلوتر می‌بریم

```
int i=0
int j=n-1;
while(i<j){
    if ( A[j] + A[i] == k )
        return 0;
    else if ( A[j]+ A[i] >k )
        j--;
    else if ( A[j]+ A[i] <k)
        i++; }
return -1;
```


Scanned with CamScanner

Time complexity: while تا زمانی تکرار می‌شود که i و j به هم برسند.

$\Longrightarrow \boxed{T(n) \in O(n)} \longrightarrow$ (in worst case)

برای آرایه‌ی مرتب نشده:

① . استفاده از «حلقه‌ی for» (nested):

```
1. for(int i=0 to n-1 ){
2.     for (int j=i+1 to n-1){
3.         if( A[j] + A[i] == k)
4.             return 1; }
5.     return -1; }
```

Time complexity: in worst case → تمام جفت‌ها بررسی می‌شوند → تعداد تکرار: $\frac{n(n-1)}{2}$

$\longrightarrow \boxed{T(n) \in O(n^2)}$

رشد توابع:

② Hash Table: یک hashtable را به اندازه‌ی آرایه initialize می‌کنیم.
برای هر کدام از خانه‌های A، در یک حلقه می‌گردیم که آیا K-A[i] در hashtable موجود است یا نه؟
اگر نبود آن A[i] را به hashtable اضافه می‌کنیم.

```
1. Initialize Hash Table B of size O(n)
2. for (int i=0 to n-1) {
3.         if ( B.search (K-A[i]) is true)
4.                 return 0;
5.         B. insert (A[i]); }
6. return -1;                    → if it is not found
```

Time Complexity:

worst case → زمان طی کردن کل آرایه + زمان درج کردن خانه‌ها در hashtable

$\rightarrow T(n) = n \cdot O(1) + n \cdot O(1) = O(n)$

$\rightarrow \boxed{T(n) \in O(n)}$

زمان‌ها به ترتیب کم به زیاد:

این روش درست ① pointer بدون مرتب کردن (sort) از درجه‌ی n برند و زمان کمتری نسبت به binary search ② در حلقه می‌برند؛ در حلقه نیز از همه زمان بیشتر کسی برد ③

$T(n) \in O(n^2)$ $T(n) \in O(n \lg n)$

Scanned with CamScanner

$\longrightarrow \boxed{T(n) \in O(n^2)}$

رشد توابع :

۵- موارد زیر را ثابت کنید. (با توجه به نکات گفته شده در صورت سؤال.)

ادامه در ص آخر

a) $\lg(n!) = \Theta(\lg(n^n))$

10 $\lg(n!) = O(\lg(n^n))$: $\lg(n^n) = n\lg(n)$

$n! = \overbrace{(n-0)}^{n}(n-1)\cdots(n-(n-1))$

فاکتورگیری از n $\Longrightarrow = n\left(1-\frac{0}{n}\right)\cdot n\left(1-\frac{1}{n}\right)\cdot n\left(1-\frac{2}{n}\right)\cdots\cdots n\left(1-\frac{n-1}{n}\right)$

$= n^n\cdot\left(1-\frac{0}{n}\right)\left(1-\frac{1}{n}\right)\cdots\cdots\left(1-\frac{n-1}{n}\right)$

$= n^n\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)$

$\lg(n!) = \lg\left(n^n\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)\right)$

$= \lg(n^n) + \lg\left(\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)\right)$ $\qquad \log_b^a + \log_b^c = \log_b^{ac}$

$= n\lg(n) + \lg\left(\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)\right)$

Scanned with CamScanner

رشد توابع :

۵- موارد زیر را ثابت کنید. (با توجه به نکات گفته شده در صورت سؤال.)

a) $\lg(n!) = \Theta(\lg(n^n))$

10 $\lg(n!) = O(\lg(n^n))$: $\lg(n^n) = n\lg(n)$

$n! = \overbrace{(n-0)}^{n}(n-1)\cdots(n-(n-1))$

فاکتورگیری از n $\Longrightarrow = n\left(1-\frac{0}{n}\right)\cdot n\left(1-\frac{1}{n}\right)\cdot n\left(1-\frac{2}{n}\right)\cdots\cdots n\left(1-\frac{n-1}{n}\right)$

$= n^n\cdot\left(1-\frac{0}{n}\right)\left(1-\frac{1}{n}\right)\cdots\cdots\left(1-\frac{n-1}{n}\right)$

$= n^n\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)$

$\lg(n!) = \lg\left(n^n\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)\right)$

$= \lg(n^n) + \lg\left(\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)\right)$ $\qquad \log_b^a + \log_b^c = \log_b^{ac}$

$= n\lg(n) + \lg\left(\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)\right)$

Scanned with CamScanner

$$\longrightarrow \lg(n!) = n\lg(n) + \lg\left(\prod_{i=0}^{n-1}\left(1-\frac{i}{n}\right)\right)$$

$$\Rightarrow \boxed{\lg(n!) = O(n\lg(n))} \quad \mathrm{I}$$

② . $\lg(n!) = \Omega(\lg(n^n))$

$$\lg(n!) = \lg(n \times n-1 \times \cdots \times 2 \times 1) = \lg n + \lg(n-1) + \cdots + \lg 2 + \lg 1$$

$S = \lg n + \cdots + \lg 1$ این جمع را S نام گذاری می‌کنیم:

به ۲ بخش آن را تقسیم می‌کنیم:

$$S = T + U \begin{cases} \longrightarrow T = \lg 1 + \lg 2 + \cdots + \lg \frac{n}{2} \\ \longrightarrow U = \lg(\frac{n}{2}+1) + \cdots + \lg n \end{cases}$$

$$T \geqslant \lg 1 + \lg 1 + \cdots + \lg 1 = 0 + 0 + \cdots + 0 = 0 \longrightarrow T \geqslant 0$$

$$U \geqslant \lg\frac{n}{2} + \lg\frac{n}{2} + \cdots + \lg\frac{n}{2} = \frac{n}{2} \times \lg\frac{n}{2}$$

$$\longrightarrow S = T + U \geqslant 0 + \frac{n}{2} \times \lg\frac{n}{2} \longrightarrow \lg(n!) \geqslant \frac{n}{2} \times \lg\frac{n}{2}$$

$$\frac{n}{2}\lg\left(\frac{n}{2}\right) \in \Omega\, n\lg(n) \Longrightarrow \boxed{\lg(n!)\ \Omega\ n\lg(n)} \quad \mathrm{II}$$

$$\xrightarrow{\mathrm{II},\ \mathrm{I}} \lg(n!) = \Theta\, n\lg(n)$$

b) $n^{\frac{1}{\lg n}} = \Theta(1)$

$$n^{\frac{1}{\lg n}} \underset{\frac{1}{\log_b a} = \log_a b}{=\!=\!=} n^{\log_n 2} \underset{a^{\log_b c} = b^{\log_a c}}{=\!=\!=} 2^{\log_n n}$$

$$\underset{\log_n n = 1}{=\!=\!=} 2 \xrightarrow[\text{تابع ثابت}]{} \begin{matrix} n^{\frac{1}{\lg n}} \in O(1) \\ n^{\frac{1}{\lg n}} \in \Omega(1) \end{matrix} \Longrightarrow n^{\frac{1}{\lg n}} \in \Theta(1)$$

Scanned with CamScanner

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

طبق تقریب استرلینگ داریم : $n! = \sqrt{2\pi n}\left(1+O\left(\frac{1}{n}\right)\right)\left(\frac{n}{e}\right)^n$

$$\lim_{n\to\infty}\frac{n!}{2^n} = \lim_{n\to\infty}\frac{\sqrt{2\pi n}\times\left(1+O\left(\frac{1}{n}\right)\right)\times\left(\frac{n}{e}\right)^n}{2^n}$$

$$= \lim_{n\to\infty}\sqrt{2\pi n}\times\left(\frac{n}{2e}\right)^n\times\cancel{\left(1+O\left(\frac{1}{n}\right)\right)}^{1}$$

$\hookrightarrow n\to\infty : \cancel{(1+0)}$

$$\lim_{n\to\infty}\left(\frac{n}{2e}\right)^n \longrightarrow \text{for } n\geq 4,\ \frac{n}{2e}\geq 1 \Longrightarrow \lim_{n\to\infty}\left(\frac{n}{2e}\right)^n = \infty$$

$$\Longrightarrow \lim_{n\to\infty}\frac{n!}{2^n} = \infty \Longrightarrow \boxed{n! \in \omega(2^n)}$$

Scanned with CamScanner

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

طبق تقریب استرلینگ داریم : $n! = \sqrt{2\pi n}\left(1+\Theta\left(\frac{1}{n}\right)\right)\left(\frac{n}{e}\right)^n$

$$\lim_{n\to\infty} \frac{n!}{n^n} = \lim_{n\to\infty} \frac{\sqrt{2\pi n}\left(1+\Theta\left(\frac{1}{n}\right)\right)\left(\frac{n}{e}\right)^n}{n^n}$$

$$= \lim_{n\to\infty} \sqrt{2\pi n} \times \left(1+\Theta\left(\frac{1}{n}\right)\right)\overbrace{\left(\frac{\cancel{n}}{\cancel{n}e}\right)^n}^{\left(\frac{1}{e}\right)^n}$$

$$\lim_{n\to\infty} \left(\frac{1}{e}\right)^n = \lim_{n\to\infty} \frac{1}{e^{n\to\infty}} = 0$$

$$\Longrightarrow \lim_{n\to\infty} \frac{n!}{n^n} = \lim_{n\to\infty} \sqrt{2\pi n} \times 0 = 0 \longrightarrow n! \in O(n^n)$$

۶- صحیح یا غلط بودن گزاره‌های زیر را ثابت کنید : (درصورتی که گزاره‌ها غلط هستند، مثال نقض کافی است.)

a) $f(n) = O(g(n)) \Rightarrow g(n) = O(f(n))$ غ

صحیح اش: $f(n) \in O(g(n)) \Longrightarrow g(n) \in \Omega(f(n))$

مثال نقض: $f(n) = n^2$, $g(n) = n^3$

$$n^2 \in O(n^3) \not\Longrightarrow n^3 \in O(n^2)$$

b) $f(n) + g(n) = \Theta(\min(f(n), g(n)))$ غ

مثال نقض: $f(n) = n$, $g(n) = 1$

$$\underbrace{n+1}_{\Theta(n)} \neq \Theta(\underbrace{\min(n, 1)}_{1}) = \Theta(1) \quad \times$$

Scanned with CamScanner

ص

c) $f(n) = O(g(n)) \Rightarrow \lg(f(n)) = O(\lg(g(n)))$,

$\forall n : \lg(g(n)) \geq 1$ and $f(n) \geq 1$

$\exists c, n_0 > 0 \;,\; \forall n \geq n_0 \longrightarrow f(n) \leq c \cdot g(n)$

$1 \leq f(n)$

$\xrightarrow{\lg} \lg f(n) \leq c \cdot \lg(g(n)) \Rightarrow \lg(f(n)) = O(\lg(g(n)))$

← یا می‌توانیم حد بگیریم:

$$\lim_{n \to \infty} \frac{\lg(f(n))}{\lg(g(n))} = \lim_{n \to \infty} \lg\left(\frac{f(n)}{g(n)}\right)$$

0

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

درجه‌ی f کمتر از g است.

d جا موند! در ص بعد است

ص

e) $f(n) = O(f(n^2))$

$\exists c, n_0 > 0, \quad \forall n \geq n_0 : \quad \underline{f(n) \leq c \cdot f(n^2)}$

برقرار است. (برای $f(n) \geq 1$)

$\Downarrow$

$f(n) \in O(f(n^2))$

ص

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

$f(n) = O(g(n)) \longrightarrow \forall n \geq n_0 : \quad f(n) \leq c \cdot g(n)$

$\exists c, n_0 > 0$

$\div c$

$\longrightarrow \exists k, n_0 > 0 \;,\; \forall n \geq n_0 : \underbrace{\left(\frac{1}{c}\right)}_{k} f(n) \leq g(n)$?

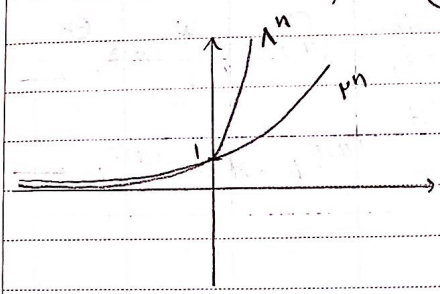
$\longrightarrow k = \frac{1}{c}$ ✓

Scanned with CamScanner

d) $f(n) = O(g(n)) \Rightarrow 2^{f(n)} = O(2^{g(n)})$

مثال نقض :

$n \in O(3n) \rightarrow 2^n \notin O(\underbrace{2^{3n}}_{8^n})$

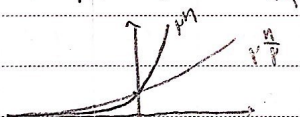


* g) $f(n) = \Theta\left(f\left(\frac{n}{2}\right)\right)$

مثال نقض :

$f(n) = 2^n \longrightarrow \exists c_1, c_2, n_0 > 0 \ni \forall n \geq n_0 \rightarrow c_2 \cdot f\left(\frac{n}{2}\right) \overset{?}{\leq} f(n) \leq c_1 \cdot f\left(\frac{n}{2}\right)$

$\longrightarrow c_2 \cdot 2^{\frac{n}{2}} \leq 2^n \leq c_1 \cdot 2^{\frac{n}{2}}$ ✗



دقت نمودار هم داریم.

h) $f(n) + \underbrace{o(f(n))}_{g(n)} = \Theta(f(n))$

حکم : $\exists c_1, c_2, n_0 > 0 \ni \forall n \geq n_0 : c_2 \cdot f(n) \leq f(n) + g(n) \leq c_1 \cdot f(n)$

طبق فرض $\forall c > 0, \exists n_0 > 0, \forall n \geq n_0 :$ $g(n) \leq c \cdot f(n)$ *

کافی است یک c_1 و c_2 وجود داشته باشد تا برقرار باشد :

ارجع * با $f(n)$ بدست می‌آید.

$f(n) \leq f(n) + g(n) \leq f(n)\ (c+1)$

if $c_2 = 1$, $c_1 = c+1$ ✓

Scanned with CamScanner

۷- مشخص کنید که به ازای هر جفت (A,B) آیا A از Θ ، ω ، Ω ، o و O تابع B هست یا خیر. (مانند ردیف اول)

	A	B	O	o	Ω	ω	Θ
	n^2	n^3	yes	yes	no	no	no
1.	$(\lg n)^k$	n^e	yes	yes	no	no	no
2.	n^k	c^n	yes	yes	no	no	no
	2^n	$2^{\frac{n}{2}}$	no	no	yes	yes	no
4.	$n^{\lg c}$	$c^{\log n}$	yes	no	yes	no	yes
5.	$2^{\lg n}$	n^2	yes	no	yes	no	yes
* 6.	$n!$	$n.2^n$	yes	yes	no	no	no
7.	$\sqrt{2}^{\lg n}$	$2^{\sqrt{2.\lg(n)}}$	yes	yes	no	no	no
8.	$(\lg(n))!$	2^{2^n}	yes	yes	no	no	no
* 9.	$n^{\lg(\lg(n))}$	$(\lg(n))^{\lg(n)}$	yes	no	yes	no	yes

1. $\lim_{n\to\infty} \frac{\lg^k n}{n^e} = 0$ $\quad k, e > 0$ $\qquad \lg^k n \in o(n^e)$

$\lg^k n \in O(n^e)$

$(\lg n)^k = \lg^k n$

2. $n^k \longrightarrow$ چندجمله‌ای $\qquad \lim_{n\to\infty} \frac{n^k}{c^n} = 0$

$c^n \longrightarrow$ نمایی

Scanned with CamScanner

4. $\begin{cases} n^{\log_2 c} \\ c^{\log_2 n} = n^{\log_2 c} \end{cases}$

5. $\begin{cases} 4^{\log_2 n} = n^{\cancel{\log_2 4}\,2} = n^2 \\ n^2 \end{cases}$

6.

$$\lim_{n\to\infty} \frac{n!}{n.2^n} = \frac{\sqrt{2\pi n}\left(1+O\left(\frac{1}{n}\right)\right)\left(\frac{n}{e}\right)^n}{n.2^n}$$

$$= \lim_{n\to\infty} \frac{\sqrt{2\pi n} \times \overbrace{\left(1+O\left(\frac{1}{n}\right)\right)}^{1}}{n} \times \left(\frac{n}{2e}\right)^n = \lim_{n\to\infty} \sqrt{2\pi} \times \underbrace{\frac{n^{-\frac{1}{2}}}{\sqrt{n}}}_{0} \times \infty = \boxed{0}$$

$$\lim_{n\to\infty} \left(\frac{n}{2e}\right)^n = \infty$$

7. $\begin{cases} 2^{\lg n} = (2^{\frac{1}{2}})^{\lg n} = (\underbrace{2^{\log_2 n}}_{=n})^{\frac{1}{2}} = \sqrt{n} = n^{\frac{1}{2}} \\ 2^{\sqrt{2\lg n}} = 2^{\sqrt{2}\times\sqrt{\lg n}} = (2^{\sqrt{\lg n}})^{\sqrt{2}} = \left(\left(\underbrace{2^{\lg n}}_{n}\right)^{\frac{1}{2}}\right)^{\sqrt{2}} \end{cases}$

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

8. $(\log n)! = \log n . \log(n-1) \ldots \log 1$

$\xrightarrow{\log n \leq 2^n} (\log n)! \leq 2^n . 2^n . \ldots . 2^n = (2^n)^2 \leq 2^{2n}$

$n^2 \leq 2^{2n}$

9. $\begin{cases} n^{\lg(\lg(n))} = n^{\log_2(\lg(n))} = \lg n^{\lg n} \\ (\lg n)^{\lg n} \end{cases}$

Scanned with CamScanner