# تمرین اول ساختمان داده

نیما بهرنگ ۹۶۱۰۰۱۱۴

۶ آبان ۱۳۹۷

استاد فروغمند

## سوال ۱

۱. طبق گفته کتاب $a > 0 \mid lim(\frac{log(n)^b}{n^a}) = 0$ پس $\sqrt{n} = \Omega(log(n))$

۲. $nlog(n) = \Theta(log(n!))$

$$log(n!) = log(1) + log(2) + ... + log(n) \leq nlog(n)$$

$$log(n!) \geq c_1 \times \frac{n}{2} \times log(\frac{n}{2}) \geq 2 \times \frac{n}{2} log(n) \geq nlog(n) \mid n \geq 10$$

۳. $(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{n}) = \Theta log(n)$

$$A = (1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{n})$$

$$A \leq (1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + ... + \frac{1}{log(n)} \leq log(n) \times 1 = log(n))$$

$$A \geq (\frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + ... \frac{1}{log(n)} \geq log(n)$$

۴. $\sqrt[10000]{n} = \Omega(log(n))$

به طور مشابه برای الف و طبق گفته کتاب

# سوال ۲

۱. n بار مغیر ما زیاد می شود پس کلا $\Theta(n)$

۲. به اندازه log(n) بار حلقه اجرا می شود زیرا هر بار نصف می شود و هر بار دو عملیات درونش انجام می شود پس $\Theta(n)$

۳. همان دنباله فیبوناچی است که کوچکتر از حالتی است که دوبار جمله قبلی صدا شود یعنی $f(n-1)+f(n-2) \leq 2 * f(n-1) \Rightarrow O(2^n)$

۴. همان تابع
$$T(n) = T(\frac{n}{2}) + n$$
که حاصل آن مجموع :
$$n + \frac{n}{2} + \frac{n}{4} + ...$$
$\Theta(n)$
طبق مسأله حل شده در کتاب

۵. همان تابع
$$T(n) = T(\frac{n}{3}) + n$$
که حاصل آن مجموع :
$$n + \frac{n}{3} + \frac{n}{9} + ...$$
$\Theta(n)$
چون از جمله دوم به بعد مجموعا برابر n نمی شوند پس در کل از دوبرابر آن کمتر و از خود n بیشترند

۶. دو تا حلقه که هر کدام به اندازه ورودی انجام می شوند و متغیر را زیاد می کنند و مجموعا $n^2$ عملیات انجام می شود
$\Theta(n^2)$

۷. حلقه اول $\frac{n}{2}$
حلقه دوم log(n)
حلقه سوم log(n)
و مجموعا
$$\frac{n}{2} \times log(n) \times log(n) = \Theta(nlog^2(n))$$

۸. t به صورت زیر زیاد می شود
$$1 + 1 + 2 + 3 + 4 + ...x = \frac{x \times (x-1)}{2} + 1 = n \Rightarrow x = \sqrt{2 \times n}$$
پس مجموعا به اندازه رادیکال ورودی عملیات انجام می شود
$\Theta(\sqrt[2]{n})$

۹. تابع به اندازه ورودی عملیات انجام می دهد و متن اصلی برنامه آن را به ازای مقادیر زیر صدا می کند پس کلا به تعداد $\frac{n}{2} + \frac{n}{3} + ... \frac{n}{n}$ عملیات انجام می شود
که طبق اثبات کتاب مجموع $\frac{1}{i}$ همان log(n) است به صورت انتگرالی پس مجموعا می شود
$\Theta(nlog(n))$

# multiplying

۱. به طور مشابه هر چند جمله ای را به دو بخش می شکنیم
یکی با درجات بیش از $\frac{n}{2}$ یکی با درجات کمتر
و حاصل مورد نظر برابر ضرب هر یک از قسمت های چند جمله ای اول در چند جمله ای دوم است
$$P_1 = A1 + A2, P_2 = B1 + B2$$
$$P_1 \times P_2 = A1 \times B1 + A1 \times B2 + A2 \times B1 + A2 \times B2$$
$$T(n) = 4 \times T(\frac{n}{2}) + n$$
که طبق قضیه مستر
$$n = O(n^{2-\epsilon}) \Rightarrow T(n) = \Theta(n^2)$$

۲. اگر چند جمله ای اول به A1, A2 و دومی به B1, B2 تقسیم شوند چون درجات $B_2 \times A_1 and A_2 \times$ $B_1$ با هم برابر است می توان
$$C_1 = A_1 \times B_1$$
$$C_2 = A_2 \times B_2$$
$$C_3 = (A_1 + A_2) \times (B_1 + B_2)$$
$$A_1 \times B_2 + A_2 \times B_1 = C_3 - C_1 - C_2$$
و به این صورت ضرایت تک تک درجات بدست می آیند ولی ۳ بار ضرب n/2 تایی در هم انجام شده
$$T(n) = 3 * T(\frac{n}{2}) + O(n)$$
که طبق قضیه مستر
$$f(n) = O(n^{\log_2 3}) \Rightarrow T(n) = \Theta(n^{\log_2 3})$$

# fibonacci

۱. $T(n) \leq 2 \times T(n-2) \Rightarrow T(n) = \Omega(2^{\frac{n}{2}}) = \Omega\sqrt{2}^{n}$
$T(n) \geq 2 \times T(n-1) \Rightarrow T(n) = O(2^{n})$
پس رشد تابع گفته شده بین دو تابع نمایی می باشد پس خود آن نیز نمایی است

۲. خانه های بالا راست و پایین چپ با هم برابر و برابر عدد فیبو ناچی توان منهای یک اند. عدد چپ بالا برابر عدد فیبو ناچی برابر توان است و عدد راست پایین عدد فیبوناچی توان منهای دو است.

با هر بار ضرب این ماتریس می توان عدد فیبوناچی بعدی را پیدا کرد و هزینه آن برابر عملیات است پس با 8 * n عملیات می توان عدد فیبوناچی n ام را پیدا کرد که نسبت به ورودی خطی است و از $\Theta(n)$ است

# inversion

$$E[inv] = E[\Sigma_{i=0}^{i=n}\Sigma_{j=i+1}^{j=n} inv(i,j)]$$
$$E[inv] = \Sigma_{i=0}^{i=n}\Sigma_{j=i+1}^{j=n} E[inv(i,j)]$$
$$E[inv] = \Sigma_{i=0}^{i=n}\Sigma_{j=i+1}^{j=n} 1 \times PR(i,j)$$
$$PR(i,j) = \frac{1}{2}$$

چون به ازای هر حالت آن می توان با جابجا کردن جای آن دو به حالتی رسید که اینورژن نباشند و به دلیل این تقارن و برابر بودن شانس آن ها به دلیل تصادفی بودن هر جایگشت تعداد جواب های خوب با بد برابر و احتمال این رویداد یک دوم است

$$E[inv] = \Sigma_{i=0}^{i=n}\Sigma_{j=i+1}^{j=n} \frac{1}{2}$$
$$E[inv] = \frac{n \times (n-1)}{2} \times \frac{1}{2} = \frac{n \times (n-1)}{4}$$

# divide

برای این کار ابتدا به ازای هر خانه مشخص می کنیم موقعیت عدد آن خانه کجاست یعنی به ازای خانه شماره i عدد i در کدام خانه است

۱. لم۱ چون با مرتب کردن هر زیر آرایه باید همه مرتب شوند پس هم خانه i و هم خانه ای که عدد i در آن است باید در یک زیر بازه باشند

حال شروع می کنیم خانه اول را نگاه می کنیم و اولین زیر بازه ما خانه اول تا خانه ای است که عدد یک در آن است و به تمام خانه هایی که در این میان است نیز نگاه می کنیم و اگر اعداد آنها در بیرون از زیر آرایه ما بود. زیر آرایه ما تبدیل می شود به زیر آرایه ای از خانه اول تا دورترین خانه ای که عددی از خانه های این بین در آن است یعنی

```
B[i] : number of cell that number i is in
for i = 1 to max1
max1 = max(max1, b[i])
```

حال این زیر آرایه شامل اعداد متوالی است زیرا حتما شامل اعداد یک تا k است و همچنین k خانه دارد پس فقط اعداد یک تا k در آن است و همچنین با مرتب کردن انها حتما آن تیکه از آرایه مرتب می شود

حال به سراغ خانه k + 1 می رویم و همین الگوریتم را پیاده می کنیم

در اصل ما به دنبال کوچک ترین مولفه ها می رویم و می دانیم که هر جواب دیگری برای این مسئله باید طبق لم گفته شده, زیر آرایه گفته شده توسط ما را نیز داشته باشد

```
get A
C : ansers of problem
for(int i = 0; i < n; i++)
B[A[i]] = i
B[i] : number of cell that number i is in
for k = 0 to n-1 : {
max1 = B[k]
for i = k to max1:{
max1 = max(max1, b[i])
}
C[cnt++] = max1
k = max1+1
}
```

# couple

از روش استفاده از دو پوینتر استفاده می کنیم
ابتدا با مرج سورت آرایه را مرتب می کنیم که nlogn است
حال اولین خانه ای(سمت چپ ترین) را پیدا می کنیم که جمعش با خانه اول آرایه بیشتر از k شود فرض کنیم خانه i باشد حال اگر جمع اولی با k i - 1 , بود که جواب است در غیر این صورت خانه i را با خانه دوم مقایسه می کنیم و آنقدر زیادش می کنیم که مثل خانه اول بزرگتر مساوی k شود و به همین ترتیب با باقی خانه ها می دانیم که اگر جمع دو عدد k شود در الگوریتم ما عدد کوچکتر را با آن مقایسه کرده ایم تا به نقطه مساوی یا بزرگتر برسیم
پس در الگوریتم ما نیز ظاهر می شود

```
sort
int j = 0;
for(int i = 0; i < n; i++)
while(a[i] + a[j] < k){
k++
}
if(j < n && a[i] + a[j] == k){
return i, j
}
```

همان طور که مشخص است حلقه اول به اندازه ورودی و حلقه دورونی آن مجموعا به اندازه ورودی بار عملیات انجام می دهند پس

$$f(n) = O(nlog(n)) + O(n) = O(nlog(n))$$

# minesweeper

از اولین خانه شروع و به طول ابتدا ۱ سپس ۲ سپس ۴ سپس ۸ و .... جلو می رویم و چک می کنیم یعنی خانه ۱ سپس ۲ تا ۴ سپس ۴ تا ۸ و ....
حال اگر بمب را بیابیم تا اینجای کار به اندازه $log(n)$ عملیات انجام داده ایم حال باید در یک بازه به طول حداکثر $n / 2$ به دنبال بمب بگردیم که با روش حذف کردن نصف در $log(n)$ ممکن است
به این صورت که در بازه ای که بمب بود ابتدا نصف سمت چپ را چک می کنیم و می فهمیم بمب در کدام نصفه است
حال نصف آن نصفه و به همین ترتیب تا بلاخره به یک خانه برسیم
پس در مجموع $2*log(n)$ عملیات نیاز است که از $o(n)$ است

# triangle

همانند سوال قبل ابتدا مرتب می کنیم و سپس دو ضلع کوچک تر را فیکس و ضلع سوم را در اوردری کم میابیم

```
sort a[]
for(int i = 0; i < n - 2; i++){
l = i + 2;
for(int j = i + 1; j < n - 1; j++){
while(a[i] + a[j] > a[l]){
l++ }
ans += l - j
}
}
```

اوردر این برنامه یک مرتب سازی ان لاگ ان است بعلاده دو عدد حلقه از صفر تا n که به ازای هر بار که کوچکترین ضلع تغییر کند یک حلقه به اندازه حداکثر n روی l انجام می گیرد و در مجموع

$$nlogn + n^2 + n^2 = O(n^2)$$

عملیات داریم

چون l نشان دهنده خانی است که دیگر نمی توان تشکیل مثلث بدهد

پس با تمام قبلیهایش می توان مثلث ساخت