س... فتحعلی زاده تکلیف اول _ ۹۸۲۳۰۳۳

الف) $i = 1$

۱۳ , ۱ , ۶ , ۱۰ , ۲ , ۳ , ۱۱ , ۵ , ۱۴ , ۹ , ۸ , ۷ , ۴

(1) ۱ , ۱۳ , ۶ , ۱۰ , ۲ , ۳ , ۱۱ , ۵ , ۱۴ , ۹ , ۸ , ۷ , ۴ — $i = ۲$

(2) ۱ , ۶ , ۱۳ , ۱۰ , ۲ , ۳ , ۱۱ , ۵ , ۱۲ , ۹ , ۸ , ۷ , ۴ — $i = ۳$

(3) ۱ , ۶ , ۱۰ , ۱۳ , ۲ , ۳ , ۱۱ , ۵ , ۱۲ , ۹ , ۸ , ۷ , ۴ — $i = ۴$

(4) ۱ , ۲ , ۶ , ۱۰ , ۱۳ , ۳ , ۱۱ , ۵ , ۱۲ , ۹ , ۸ , ۷ , ۴ — $i = ۵$

(5) ۱ , ۲ , ۳ , ۶ , ۱۰ , ۱۳ , ۱۱ , ۵ , ۱۲ , ۹ , ۸ , ۷ , ۴ — $i = ۶$

۶ مقایسه

(6) ۱ , ۲ , ۳ , ۶ , ۱۰ , ۱۱ , ۱۳ , ۵ , ۱۲ , ۹ , ۸ , ۷ , ۴ — $i = ۷$

۷ مقایسه

(7) ۱ , ۲ , ۳ , ۵ , ۶ , ۱۰ , ۱۱ , ۱۳ , ۱۲ , ۹ , ۸ , ۷ , ۴ — $i = ۸$

۸ مقایسه

(8) ۱ , ۲ , ۳ , ۵ , ۶ , ۱۰ , ۱۱ , ۱۲ , ۱۳ , ۹ , ۸ , ۷ , ۴ — $i = ۹$

۹ مقایسه

(9) ۱ , ۲ , ۳ , ۵ , ۶ , ۹ , ۱۰ , ۱۱ , ۱۲ , ۱۳ , ۸ , ۷ , ۴ — $i = ۱۰$

۱۰ مقایسه

(10) ۱ , ۲ , ۳ , ۵ , ۶ , ۸ , ۹ , ۱۰ , ۱۱ , ۱۲ , ۱۳ , ۷ , ۴ — $i = ۱۱$

۱۱ مقایسه

(11) ۱۲ = ن ۴ ، ۱۳ ، ۱۲ ، ۱۱ ، ۱۰ ، ۹ ، ۸ ، ۷ ، ۶ ، ۵ ، ۳ ، ۲ ، ۱

۱۲ مقایسه

(12) ۱۳ ، ۱۲ ، ۱۱ ، ۱۰ ، ۹ ، ۸ ، ۷ ، ۶ ، ۵ ، ۴ ، ۳ ، ۲ ، ۱

در مرحله اول ۱ = ن قرار می دهیم که همان عدد ۱ است و با عدد قبلی اش مقایسه نمود که کوچکتر از ۱۳ است پس جای آن را عوض می شود. (۱ مقایسه)

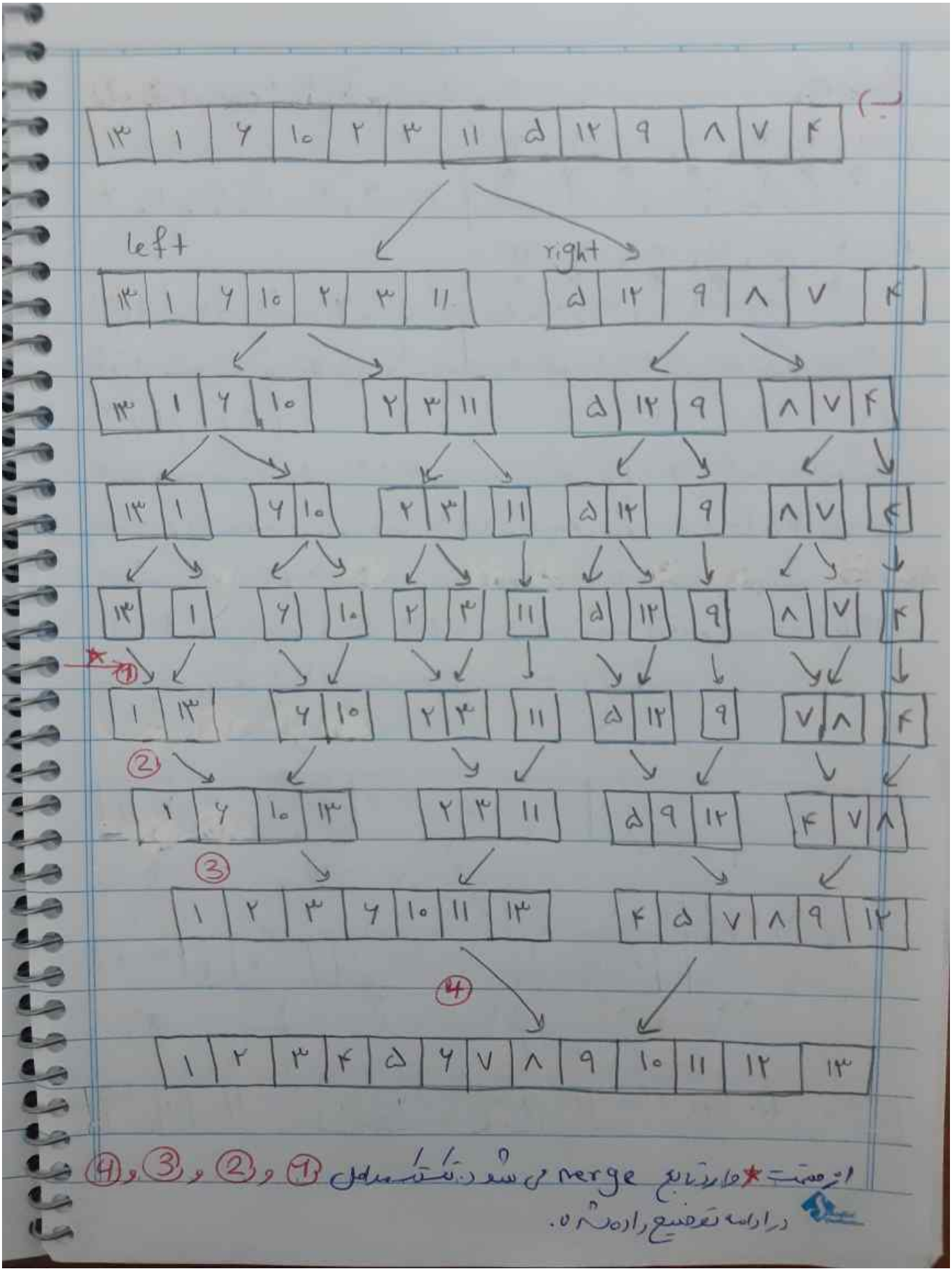
در مرحله دوم ۲ = ن قرار می دهیم که همان عدد ۶ است و با دو عدد قبلی اش یعنی ۱۳ و ۱ مقایسه می شود و جای ۶ با ۱۳ عوض می شود. (۲ مقایسه)

در مرحله سوم ۳ = ن قرار می دهیم که همان عدد ۱۰ است و با سه عدد قبلی اش مقایسه می شود که جای ۱۰ با ۱۳ عوض می شود. (۳ مقایسه)

در مرحله چهارم ۴ = ن قرار می دهیم که همان عدد ۲ است و با ۴ عدد قبلی اش مقایسه می شود. (۴ مقایسه)

در مرحله پنجم ۵ = ن قرار می دهیم که همان عدد ۳ است و با ۵ عدد قبلی اش مقایسه می شود. (۵ مقایسه)

بدین ترتیب در هر مرحله نفری اضافه شده و هر بار با همه اعداد قبلی اش مقایسه می شود.


left
right
merge

① ۱ با ۱۳ مقایسه می‌شود و چون ۱ کوچک‌تر است ابتدا نوشته می‌شود و بعد عدد ۱۳.
۶ با ۱۰ مقایسه شده و چون ۶ کوچک‌تر است ابتدا نوشته شده و بعد عدد ۱۰ قرار می‌گیرد.
۲ با ۳ مقایسه می‌شود ~ ۲ ~ ~ ~ قرار می‌گیرد ~ ~ ۳ ~ قرار می‌گیرد.
همچنان ۵ با ۱۲ مقایسه شده و ۷ با ۸ نیز مقایسه می‌شود.

② در این مرحله [۱۳, ۱] با [۱۰, ۶] مقایسه می‌شود.
ابتدا به ۱ و ۶ اشاره می‌کنند. ۱ با ۶ مقایسه می‌شود و چون ۱ کوچک‌تر است ابتدا قرار گرفته و بعد روی آرایه L در خانه صفر می‌رویم و به ۱۳ اشاره می‌کنیم. ۱۳ با ۶ مقایسه می‌شود و چون ۶ کوچک‌تر است ۶ ابتدا قرار گرفته و بعد یکی روی آرایه R صفر می‌رویم. حالا به اعداد ۱۳ و ۱۰ اشاره می‌شود که ۱۰ از ۱۳ کوچک‌تر بوده پس ابتدا ۱۰ قرار می‌گیرد و بعد عددی که باقی مانده ۱۳ است که نوشته می‌شود ← [۱۳, ۱۰, ۶, ۱] پس ۳ تا مقایسه داریم.

L: ۱, ۱۳
R: ۶, ۱۰

در این مرحله همچنین [۳, ۲] با [۱۱] مقایسه می‌شود.
ابتدا به ۲ و ۱۱ اشاره می‌کند. ۲ کوچک‌تر است پس ابتدا ۲ قرار می‌گیرد و یکی روی آرایه L صفر می‌رویم و به ۳ اشاره می‌کند. عدد ۳ با ۱۱ مقایسه شده و چون ۳ کوچک‌تر است بنابراین ۳ قرار می‌گیرد و بعد از آن که باقی مانده نوشته می‌شود. (۲ تا مقایسه داریم)

L: ۲, ۳
R: ۱۱

در این مرحله همچنین [۱۲, ۵] با [۹] مقایسه می‌شود.
ابتدا به ۵ و ۹ اشاره می‌شود چون ۵ کوچک‌تر از ۹ است ابتدا ۵ قرار می‌گیرد و یکی روی L صفر می‌رویم (به ۱۲ اشاره می‌کند). ۱۲ با ۹ مقایسه شده و ۹ کوچک‌تر است بنابراین ۹ قرار می‌گیرد و بعد از آن باقی‌مانده قرار می‌گیرد. (۲ تا مقایسه داریم)

L: ۵, ۱۲
R: ۹

L: ۷ , ۸

R = ۴

در این مرحله [۸ و ۷] با [۴] مقایسه می‌شوند.
ابتدا ۷ با ۴ مقایسه می‌شود و ۴ کوچکتر است پس ۴ قرار می‌گیرد.
در R عنصر دیگری نیست که بخواهیم به آن اشاره کنیم پس مابقی آرایه
L به ترتیب کپی می‌شود. (۱ مقایسه انجام شد)

L: ۱ , ۶ , ۱۰ , ۱۳

R: ۲ , ۳ , ۱۱

③ در این مرحله داریم:
ابتدا به ۱ و ۲ اشاره می‌کنند که با هم مقایسه می‌شوند و چون ۱
کوچکتر است ابتدا قرار می‌گیرد و نشانگر روی آرایه L جلو می‌رود و به ۶
اشاره می‌کند. ۶ با ۲ مقایسه می‌شود و بعد ۲ قرار می‌گیرد و نشانگر روی آرایه R جلو می‌رود و به
۳ اشاره می‌کند. ۳ با ۶ مقایسه می‌شود و ۳ قرار می‌گیرد و نشانگر روی R جلو می‌رود و به ۱۱ اشاره
می‌کند. ۱۱ با ۶ مقایسه می‌شود و ۶ بعد از ۳ قرار می‌گیرد و نشانگر روی L جلو می‌رود و به
۱۰ اشاره می‌کند. ۱۱ با ۱۰ مقایسه می‌شود و ۱۰ قرار می‌گیرد و نشانگر روی L جلو می‌رود و به
۱۳ اشاره می‌کند. ۱۳ با ۱۱ مقایسه می‌شود و ۱۱ بعد از ۱۰ قرار می‌گیرد. در نهایت ۱۳
باقی می‌ماند که بعد از ۱۱ نوشته می‌شود. (۶ تا مقایسه انجام می‌شود)

L: ۵ , ۹ , ۱۲

R: ۴ , ۷ , ۸

در این مرحله داریم:
ابتدا ۵ با ۴ مقایسه می‌شود. طبق استدلال‌هایی که قبلاً
مطرح شد دوباره ۷ با ۵ مقایسه می‌شود. بعد ۷ با ۹
مقایسه می‌شود. بعد ۸ با ۹ مقایسه می‌شود. در نهایت ۹ و ۱۲ که باقی مانده‌اند کپی می‌شوند
(۴ تا مقایسه انجام می‌شود.)

مرحله (4): در این مرحله داریم:

L: 1, 2, 3, 6, 10, 11, 13

R: 4, 5, 7, 8, 9, 12

طبق استدلال هایی که قبلا بیان شد در این مرحله ابتدا 1 با 4 مقایسه می‌شود. عدد 2 با 4 مقایسه می‌شود. دوباره 3 با 4 مقایسه می‌شود. دوباره 6 با 4 مقایسه می‌شود. عدد 6 با 5 مقایسه می‌شود و عدد 6 با 7 مقایسه می‌شود. عدد 10 با 7 مقایسه می‌شود. عدد 10 با 8 مقایسه می‌شود. دوباره 10 با 9 مقایسه می‌شود. دوباره 10 با 12 مقایسه می‌شود. عدد 12 با 11 مقایسه می‌شود. در آخرین مرحله 12 با 13 مقایسه می‌شود. (12 تا مقایسه در این مرحله انجام می‌شود.)

سوال ۱ :

۲) در روش sleeping sort به ترتیب در آرایه پیمایش کرده و در حد اصل زمان مساوی

هر کدام از عناصر آرایه به خواب می روند. عناصر آرایه بسته به مقدارشان در خواب هستند

و پس از چند ثانیه بعد (که حاصل مقدار هر عضو است) از خواب بیدار می شوند که همان موقع

در خروجی چاپ شده و در آرایه به صورت شده قرار می گیرند.

در این مثال فرض می کنیم برای راحتی شد که ۵ میلی ثانیه بخوابد و هر ۱ ثانیه یک بار

عملیات بعدی را روی هر کدام از عناصر آرایه انجام می دهد:

[۱۳, ۱, ۶, ۱۰, ۲, ۳, ۱۱, ۵, ۱۲, ۹, ۸, ۷, ۴]

مثلا:
۳s: sleep ۱۴ ۳ + ۱۴ = ۱۷ → زمان خواب می‌رود

۷s: sleep ۱ ۷+۱ = ۸

۸s: "۱" (۱ wakes up so print it)

۹s: sleep ۷ ۹+۷ = ۱۵

۱۲s: sleep ۱۰

۱۵s: sleep ۲

۱۵s: "۷" (۷ wakes up so print it)

۱۷s: "۱۴" (۱۴ wakes up so print it)

۱۸s: "۳" (۳ wakes up so print it)

۱۸s: sleep ۳

۲۱s: sleep ۱۱

۲۱s: "۳" (۳ wakes up so print it)

۲۲s: "۱۰" (۱۰ wakes up so print it)

۲۴s: sleep ۵

۲۷s: sleep ۱۲

۲۹s: "۵" (۵ wakes up so print it)

۳۰s : sleep ۹

۳۲s : "۱۱" (۱۱ wakes up so print it)

۳۳s : sleep ۸

۳۶s : sleep ۷

۳۹s : sleep ۴

۳۹s : "۱۲" (۱۲ wakes up so print it)

۳۹s : "۹" (۹ wakes up so print it)

۴۱s : "۸" (۸ wakes up so print it)

۴۲s : ___

۴۳s : "۷" (۷ wakes up so print it)

۴۳s : "۴" (۴ wakes up so print it)

output : ۱, ۶, ۱۳, ۲, ۳, ۱۰, ۵, ۱۱, ۱۲, ۹, ۸, ۷, ۴

می‌دانیم که خروجی sleeping sort همیشه درست نیست. خروجی زمانی اشتباه می‌شود که یک مقدار خیلی کوچک در سمت چپ یک مقدار خیلی بزرگ در آرایه ورودی باشد.

برای حل مشکل باید مجدداً یکبار روی خروجی مجدد سورت انجام دهیم تا خروجی درست بدست آید.

روی خروجی عبارت را انجام می دهیم:

۴s: sleep 1 ۳+1=۴

۴s: "1" (1 wakes up so print it)

۷s: sleep ۷ ۷+۷=۱۴

۹s: sleep ۱۳ ۹+۱۳=۲۲

۱۲s: sleep ۲

۱۲s: "۶" (۶ wakes up so print it)

۱۴s: "۲" (۲ wakes up so print it)

۱۵s: sleep ۳

۱۸s: "۳" (۳ wakes up so print it)

۲۱s: sleep ۱۰

۲۳s: "۱۳" (۱۳ wakes up so print it)

۲۶s: sleep ۵

۲۶s: "10" (10 wakes up so print it)

۲۷s: sleep 11

۲۹s: "۵" (۵ wakes up so print it)

۳۰s: sleep ۱۲

۳۲s: sleep ۹

۳۶s: sleep ۸

۳۸s: "۱۱" (۱۱ wakes up so print it)

۳۹s: sleep ۷

۴۲s: "۱۲" (۱۲ wakes up so print it)

۴۲s: "۹" (۹ wakes up so print it)

۴۲s: sleep ۴

۴۴s: "۸" (۸ wakes up so print it)

۴۶s: "۷" (۷ wakes up so print it)

۴۶: "۴" (۴ wakes up so print it)

output: ۱, ۶, ۲, ۳, ۱۳, ۱۰, ۵, ۱۱, ۱۲, ۹, ۸, ۷, ۴

آنقدر باید سورت را انجام دهیم تا در نهایت آرایه مرتب شده برگرداند.

۶) از آنجایی که تابع sleep() ، کیو صف اصلی ایجاد شده و هر کدام از عناصر آرایه را به صف اصلی اضافه می کند که پیچیدگی این قسمت از $O(N \log N)$ است.

و از آنجایی که زمانی خروجی آماده می شود که همه عناصر آرایه از صفر بیدار شده باشند و می دانیم که برای هر عنصر نام آرایه $O(arr[i])$ زمان نیاز است تا wake up شود.

بنابراین برای بیشترین عنصر آرایه $O(\max(input))$ زمان نیاز است تا wakeup شود.

پس پیچیدگی زمانی کل الگوریتم $O(N \log N + \max(input))$ است که N تعداد عناصر آرایه می باشد.

پیچیدگی مسئله ج از $O(13 \log^{13} + 13)$ می باشد.

۲

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

ب) اگر $i = n$ باشد حلقه داخلی n بار اجرا می‌شود. اگر $i = \frac{n}{2}$ باشد حلقه داخلی $\frac{n}{2}$ بار اجرا می‌شود و اگر حلقه داخلی $i = \frac{n}{4}$ باشد حلقه داخلی $\frac{n}{4}$ بار اجرا می‌شود.

$$T(n) = n + \frac{n}{2} + \frac{n}{4} + \dots + 1 = 2 \times n \Rightarrow T(n) = O(n)$$

ج)

$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{\log i} 1 = \sum_{i=1}^{n} \log i = \log^1 + \log^2 + \dots$$

$$+ \log^n = \log^2 + \log^3 + \dots + \log^n = \log^{2 \times 3 \times \dots \times n} =$$

$$\log^{n!} \qquad T(n) = \log n!$$

د) می‌دانیم که n عددی یا زوج است یا فرد پس یا if اجرا می‌شود یا $else$. اگر هرکدام از if یا $else$ اجرا شوند n و k یا هر بار تقسیم بر ۲ یا ضرب در ۲ می‌شوند پس هرکدام از اردر $\log n$ ای باشند. حلقه بیرونی نیز n بار اجرا می‌شود پس:

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

ادامه سوال ۲ :

هـ)

$$T(n) = \underbrace{T(\tfrac{n}{2})}_{①} + \underbrace{T(\tfrac{n}{3})}_{②} + \underbrace{2n}_{③}$$

① در این الگوریتم یک بار تابع بازگشتی کال می‌شود که n تقسیم بر ۲ می‌شود.

② 〃 〃 〃 〃 〃 n 〃 ۳ می‌شود.

③ دو تا متغیر داریم که هر کدام n بار اجرا می‌شوند و از اردر n هستند.

برای اینکه T(n) به فرم قضیه اصلی تبدیل شود و اصطلاحاً قابل حل باشد یک بار $T(\frac{n}{3})$ را همان $T(\frac{n}{2})$ با فرض در نظر می‌گیریم و پیچیدگی را محاسبه می‌کنیم و یکبار دیگر $T(\frac{n}{2})$ را $T(\frac{n}{3})$ در نظر می‌گیریم و T(n) را محاسبه می‌کنیم و هر جوابی که از اردر بزرگتر بود به عنوان جواب نهایی پذیرفته می‌شود.

① $T(n) = 2T(\frac{n}{2}) + 2n$

قضیه اصلی : $a = 2 , b = 2 , f(n) = 2n$

$$2n = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n) \Rightarrow$$

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

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

مقادیر: $a = 2,\ b = 3,\ f(n) = 2n$

$f(n) = \Omega(n^{\log_3 2 + \epsilon})$

$\log_3 2$ مقدار کمتر از 1 دارد اگر اپسیلون کوچک را در نظر بگیریم که باز هم $\log_3 2 + \epsilon$

کمتر از 1 شود رشد $f(n)$ بیشتر مقادیر است از آن

$$a f(\frac{n}{b}) = 2 f(\frac{n}{3}) = 2 \times \frac{2n}{3} = \frac{4n}{3} \leq c \times 2n$$

اگر c را مثلا $\frac{2}{3}$ فرض کنیم این رابطه برقرار است پس

$$T(n) = \Theta(f(n)) = \Theta(2n)$$

(i) و (iii) را هم مقایسه می‌کنیم رشد $n \log n$ بیشتر از رشد خطی $2n$ است پس

حاصل جمع از $O(n \log n)$ می‌باشد.

۳_

a) طبق قضیه اصلی داریم: $a=4\ ,\ b=5\ ,\ f(n)=(\log n)^2$

$n^{\log_b^a}=n^{\log_5^4}$

می توانیم عددی را بدست آوریم که بزرگتر از صفر شود: $\log_5^4>0 \quad \log_5^4-\varepsilon>0$

می دانیم $f(n)$ یک تابع polylogarithmic است اگر $f(n)=O(\log^k n)$

باشد. همچنین می دانیم $f(n)$ یک تابع polynomial است اگر $f(n)=O(n^k)$

(به ازای هر مقدار ثابت و بزرگتر از صفر k) باشد. از طرفی می دانیم هر تابع polynomial

مجانباً سریع تر از یک تابع polylogarithmic رشد می کند.

پس می توانیم ثابت کنیم حتی یک مقدار خیلی کوچک ε وجود دارد که $\log_5^4-\varepsilon$ باز هم یک

مقدار بزرگتر از صفر شود پس $f(n)=O(n^{\log_5^4-\varepsilon})$

$$\Downarrow$$

$$T(n)=\Theta(n^{\log_5^4})$$

$$\left.\begin{aligned}(\log n)^2&=O(\log^k n)\\ n^{\log_5^4-\varepsilon}&=O(n^k)\end{aligned}\right\}\Longrightarrow (\log n)^2=O(n^{\log_5^4-\varepsilon})$$

b) $a = 9,\ b = 3,\ f(n) = \log n!$ صورت تمرین داده شده است:

$n^{\log_b^a} = n^{\log_3^9} = n^2$ تمرین

$\log n! = O(n^2)$ صورت تمرین

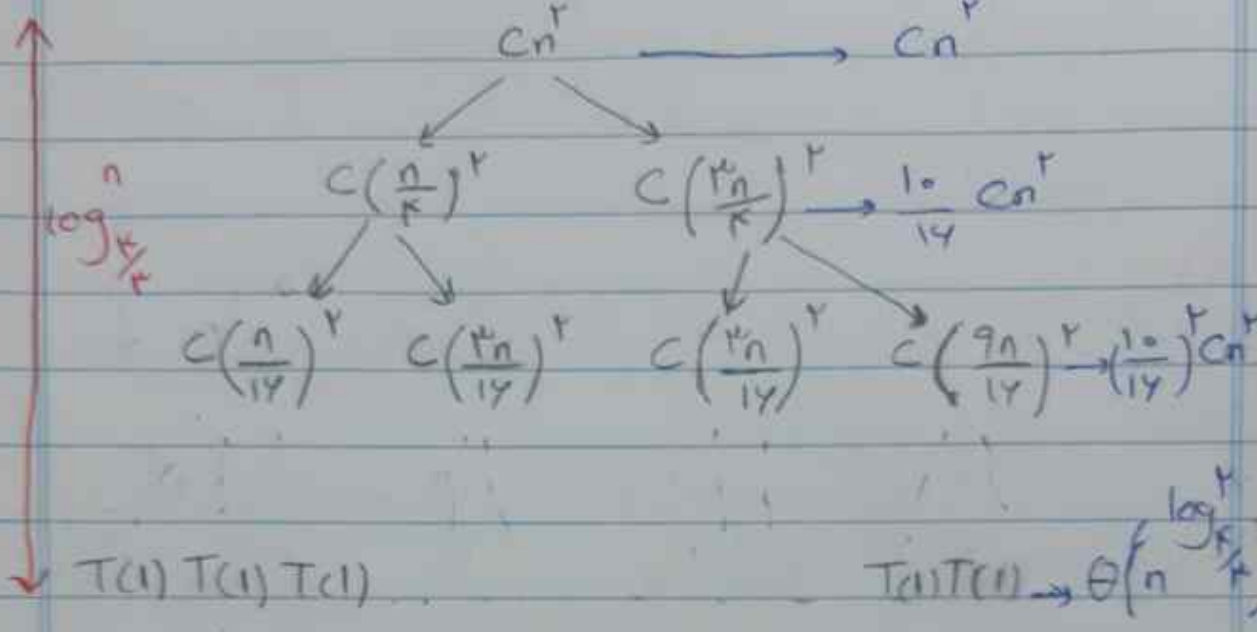
$\log n! = \Theta(n \log n)$ می‌دانیم $\xrightarrow{\text{3.1 صورت تمرین}}$ $\log n! = O(n \log n)$

$n \log n = O(n^2)$ از طرفی می‌دانیم

یک مقدار ضعیف‌تر پیدا می‌شود که باز هم رشد بیشتری داشته باشد.

① صورت $\Rightarrow f(n) = O(n^{2-\varepsilon}) \Rightarrow T(n) = \Theta(n^{\log_b^9}) = \Theta(n^2)$

c) $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + cn^2$



تعداد نودها در هر مرحله از راست با 2^c می‌شود تعداد نودها در مرحله آخر برابر

است با $2^{\log_{\frac{4}{3}} n}$ هزینه $T(1)$ برابر ۱ است پس هزینه مرحله آخر برابر است با:

$$T(1) \times 2^{\log_{\frac{4}{3}} n} = 2^{\log_{\frac{4}{3}} n} = n^{\log_{\frac{4}{3}} 2}$$

$$T(n) = cn^2 + \frac{10}{16}cn^2 + \left(\frac{10}{16}\right)^2 cn^2 + \dots + \left(\frac{10}{16}\right)^{\log_{\frac{4}{3}} n - 1} cn^2 + \Theta\left(n^{\log_{\frac{4}{3}} 2}\right)$$

$$T(n) = \sum_{i=0}^{\log_{4/3} n - 1} \left(\frac{10}{16}\right)^i cn^2 + \Theta\left(n^{\log_{4/3}^2}\right)$$

می دانیم که جمله آخر وقتی است که n به ∞ می رود.

$$\sum_{i=0}^{\infty} \left(\frac{10}{16}\right)^i cn^2 + \Theta\left(n^{\log_{4/3}^2}\right) =$$

$$\frac{1}{1-\frac{10}{16}} cn^2 + \Theta\left(n^{\log_{4/3}^2}\right) = \frac{16}{6} cn^2 + \Theta\left(n^{\log_{4/3}^2}\right)$$

$\log_{4/3}^2$ مقدار کمتری نسبت به 2 دارد پس جمله بار مهمتری نسبت به $\frac{16}{6} cn^2$ است پس این جمله حذف می شود پس:

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

d) $T(n) = T(\sqrt[3]{n}) + \log n = \log n + \log n^{\frac{1}{3}} + T(\sqrt[9]{n^{3}})$

$= \log n + \log n^{\frac{1}{3}} + \log n^{\frac{1}{9}} + \cdots + \log n^{0} =$

$\log(n \times n^{\frac{1}{3}} \times n^{\frac{1}{9}} \times \cdots) = \log(n^{1+\frac{1}{3}+\frac{1}{9}+\cdots})$

$= (1 + \frac{1}{3} + \frac{1}{9} + \cdots)\log n = \left(\frac{1}{1-\frac{1}{3}}\right)\log n \Rightarrow$

← مقدار ثابت است که نشون میدیم

$\Rightarrow T(n) = \Theta(\log n)$

e) $T(n) = 2T(n-2) + 1 = 2(2T(n-4)+1) + 1 =$

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

$8T(n-3) + 4 + 2 + 1 = 8(2T(n-4)+1) + 4 + 2 + 1 =$

$16(n-4) + 8 + 4 + 2 + 1 =$

$= 2^k T(n-2k) + 2^{k-1} + 2^{k-2} + 2^{k-3} + 2^{k-4}$

متغیر ثابت

$n - 2k = c \rightarrow k = \frac{n-c}{2}$

ادامه e)

$$T(n) = 2^k + 2^{k-1} + 2^{k-2} + \dots + 1 = \sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

$$T(n) = O(2^{k+1}) \xrightarrow{k=\frac{n-c}{2}} O\left(2^{\frac{n-c}{2}+1}\right) = O(2^n)$$

f) مقدار 2^k را ابتدا n فرض می‌کنیم پس داریم:

$$T(2^{k-2}) = T(2^k \times 2^{-2}) = T(2^k \times \tfrac{1}{4}) = T(2^k/4)$$

$$T(n) = 7T(n/4) + n$$

$a = 7,\ b = 4,\ f(n) = n$ طبق قضیه اصلی داریم:

$$\log_b^a = \log_4^7 > 1 \quad n^{\log_4^7} > n^1 \rightarrow n^{\log_4^7 - \varepsilon} > n^1 \quad (\log_4^7 = 1.4\dots)$$

اگر ε را مقدار خیلی کمی انتخاب کنیم باز هم فرض بیشتر بودن را رعایت می‌کند.

$$\Longrightarrow f(n) = O\left(n^{\log_b^a - \varepsilon}\right) \xrightarrow[\text{اصلی داریم}]{\text{پس طبق Case اول قضیه}} T(n) = O\left(n^{\log_b^a}\right)$$

$$\Longrightarrow T(n) = O\left(n^{\log_4^7}\right)$$

$$\xrightarrow[\text{طبق فرض}]{n=2^k} T(2^k) = T\left((2^k)^{\log_4^7}\right)$$

۲

الف) قانون جمع:

$$\underbrace{O(f+g)}_{g(x)} = \underbrace{O(f)}_{h(x)} + \underbrace{O(g)}_{m(x)}$$

$g(x) = O(f+g) \rightarrow \exists c_1, n_0 \quad \forall n \geq n_0 \quad g(x) \leq c_1(f+g)$

$h(x) = O(f) \rightarrow \exists c_2, n_0 \quad \forall n \geq n_0 \quad ① \; h(x) \leq c_2 f$

$m(x) = O(g) \rightarrow \exists c_3, n_0 \quad \forall n \geq n_0 \quad ② \; m(x) \leq c_3 g$

$h(x) + m(x) \leq c_2 f + c_3 g$

$g(x) \leq c_1 f + c_1 g$

$h(x) + m(x) \leq c_2 f + c_3 g$

$\hookrightarrow g(x) - \big(h(x) + m(x)\big) \leq (c_1 - c_2) f + (c_1 - c_3) g$

اگر $c_2 = c_1$ و $c_3 = c_1$ باشد فوق است تا این صفر شود که $g(x) - h(x) - m(x)$

نیز صفر است که در این حالت می‌شود این رابطه درست است.

ب) قانون ضرب

$$\overbrace{O(f.g)}^{g(x)} = \overbrace{O(f)}^{h(x)} \cdot \overbrace{O(g)}^{m(x)}$$

$g(x) = O(f.g) \rightarrow \exists c_1, n_0 \quad \forall n \geq n_0 \quad g(x) \leq C_1(f.g)$

$h(x) = O(f) \rightarrow \exists C_2, n_0 \quad \forall n \geq n_0 \quad h(x) \leq C_2 f$

$m(x) = O(g) \rightarrow \exists C_3, n_0 \quad \forall n \geq n_0 \quad m(x) \leq C_3 g$

$g(x) \leq C_1(f.g)$

$h(x).m(x) \leq C_2 C_3 (f.g) \rightarrow \frac{g(x)}{h(x).m(x)} \leq \underbrace{\frac{C_1(f.g)}{C_2 C_3 (f.g)}}_{\frac{C_1}{C_2 C_3}}$

اگر $C_1 = C_2 C_3$ باشد مشخص است نسبت ما ۱ می‌شود و حد را میگه ۱ $\frac{g(x)}{h(x).m(x)}$

بدیهی است که این نسبت می‌تواند متغیر باشد. پس این رابطه نیز درست و برقرار است.

ج) تراگذری: if $g = o(f)$ and $h = O(f) \Rightarrow g = O(h)$

راه حل:

ابتدا خاصیت تقارن را داریم:

if $g = O(f)$ / $f = O(h)$ } آنگاه $\Rightarrow g = O(h)$ ①

پس حال سوال با این فرض که $g = O(f)$ و $h = O(f)$ است می‌توان به نتیجه

① دست یافت. سوال درست یا غلط به چنین نتیجه‌ای باید $f = O(h)$ باشد یعنی

فرض $h = O(f)$ با سوال این سوال درست نبوده و به نتیجه اشتباه ختم می‌شود.

پس این رابطه نادرست است و شکل صحیح آن این گونه باید باشد:

if $g = O(f)$ and $f = O(h)$ then $g = O(h)$

د) $5n^2 + 100n^3 + 4n = O(n^4)$

این رابطه صحیح است چون بزرگترین درجه سمت چپ n^3 است در حالی که n^4 درجه بیشتری دارد پس رشد آن بیشتر است.

ه) $5n^2 + 100n^3 + 4n = O(n^2 \log n)$

این رابطه برقرار نیست. بزرگترین جمله سمت چپ $100n^3$ است که چون از درجه 3 است رشد بیشتری نسبت به $n^2 \log n$ دارد پس رشد سمت چپ بیشتر است.

	درستی یا نادرستی	علت صحیح در صورت نادرست بودن
قانون جمع	✓	
قانون ضرب	✓	
تراگذری	×	if $g=O(f)$ and $f=O(h)$ then $g=O(h)$ خاصیت تعدی است.
$5n^2+100n^3+4n=O(n^4)$	✓	
$5n^2+100n^3+4n=O(n^2\log n)$	×	$5n^2+100n^3+4n=\Omega(n^2\log n)$ $5n^2+100n^3+4n=\omega(n^2\log n)$

$n^n, 2^n, n^2, n!, \pi^n, \pi^n, \sqrt{2^{\sqrt{n}}}, n^4\binom{n}{4}, 2^{\log^4 n}, (\log n)^n, n\log n, (\log n)^{\log n}$

سوال:

جواب:

$$n^n \geq n! \geq (\log n)^n \geq \pi^n \geq 2^n \geq (\log n)^{\log n} \geq n^4\binom{n}{4} \geq \ldots$$

$$\ldots \geq n^\pi \geq n^2 \geq 2^{\log^4 n} \geq \sqrt{2^{\sqrt{n}}} \geq n\log n$$

اثبات: $n > \log^4 n > \frac{\sqrt{n}}{2} \Rightarrow 2^n > 2^{\log^4 n} > 2^{\frac{\sqrt{n}}{2}}$

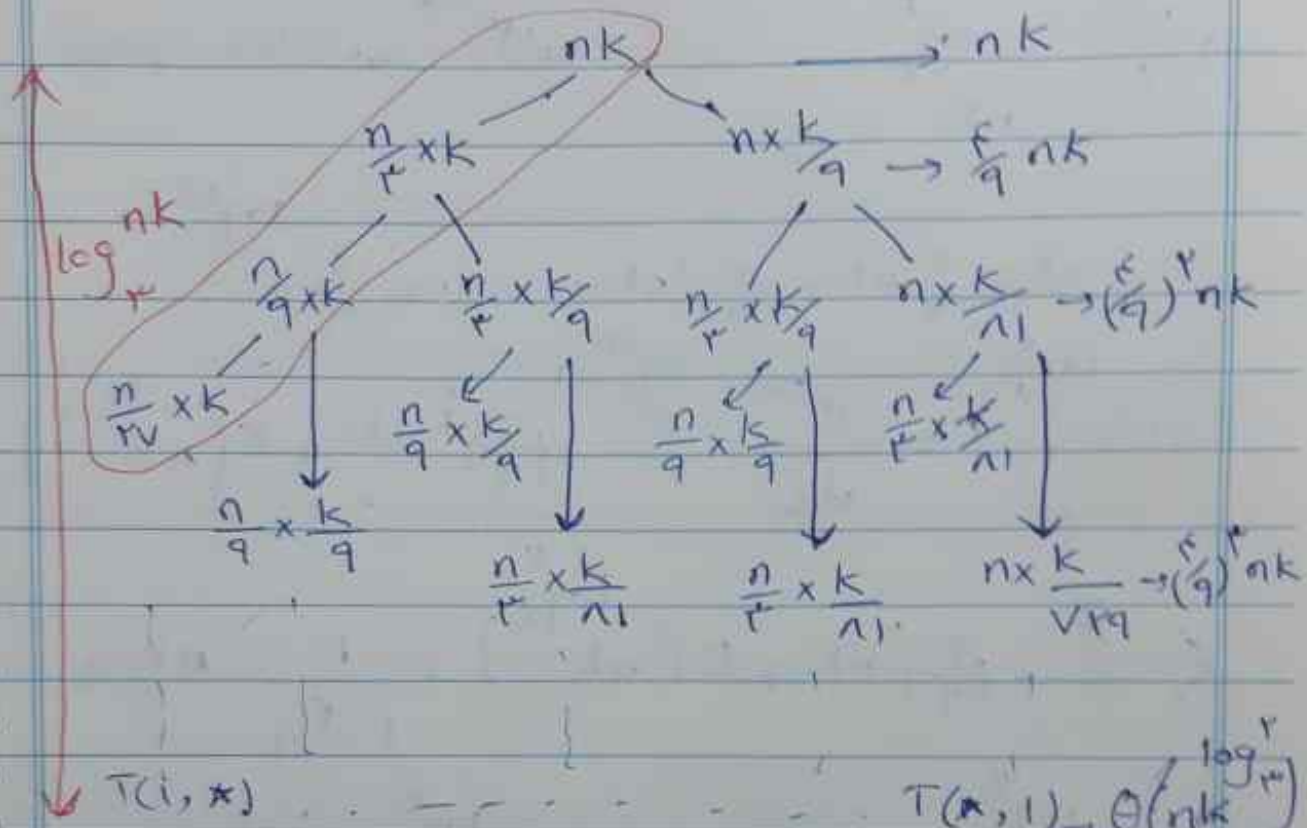
$$\sqrt{2^{\sqrt{n}}} = 2^{\frac{\sqrt{n}}{2}}$$

$$n^4\binom{n}{4} = n^4 \times \frac{n!}{(n-4)! \times 4!} = n^4 \times \frac{n(n-1)(n-2)(n-3)}{4!}$$

$$= \frac{n^8}{4!}$$

سوال ۲:

$$T(n,k) = T(\tfrac{n}{2}, k) + T(n, \tfrac{k}{9}) + kn$$



(الف) کوتاهترین مسیر وقتی است که همیشه تقسیم بر ۳ و سمت

$$nk \to (\tfrac{1}{3})nk \to (\tfrac{1}{9})nk \to \dots \to 1$$

$$(\tfrac{1}{3})^h nk = 1 \to h = \log_3^{nk}$$ ارتفاع درخت

تعداد برگ‌ها در عمق i برابر است با 2^i. پس تعداد برگ‌ها در عمق $\log_3^{nk}$ برابر $2^{\log_3^{nk}}$ است.

هر کدام از برگ‌ها 1 است.

$$2^{\log_3^{nk}} = nk$$

$$T(n) = nk + \frac{4}{9}nk + \left(\frac{4}{9}\right)^2 nk + \dots + \left(\frac{4}{9}\right)^{\log_3^{nk}-1} nk + \Theta\left(nk^{\log_3^2}\right)$$

$$\Theta\left(nk^{\log_3^2}\right) = \sum_{i=0}^{\log_3^{nk}-1} \left(\frac{4}{9}\right)^i nk + \Theta\left(nk^{\log_3^2}\right)$$

$$< \sum_{i=0}^{\infty} \left(\frac{4}{9}\right)^i nk + \Theta\left(nk^{\log_3^2}\right) = \frac{1}{1-4/9}\, nk + \Theta\left(nk^{\log_3^2}\right)$$

$$\Theta\left(nk^{\log_3^2}\right) = \frac{9}{5}nk + \Theta\left(nk^{\log_3^2}\right) \Rightarrow O(nk)$$

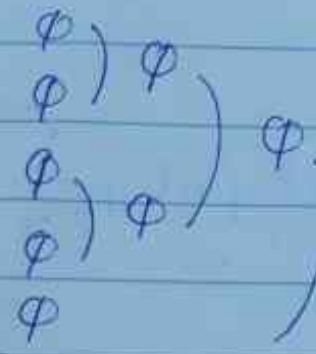
مقدارش از nk کمتر است پس nk درجه بزرگی نسبت به $nk^{\log_3^2}$ دارد و قسن گشته است.

۷

الف) بدترین حالت زمانی است که همه بیت‌ها صفر باشند

که در این حالت هزینه $n-1$ است.

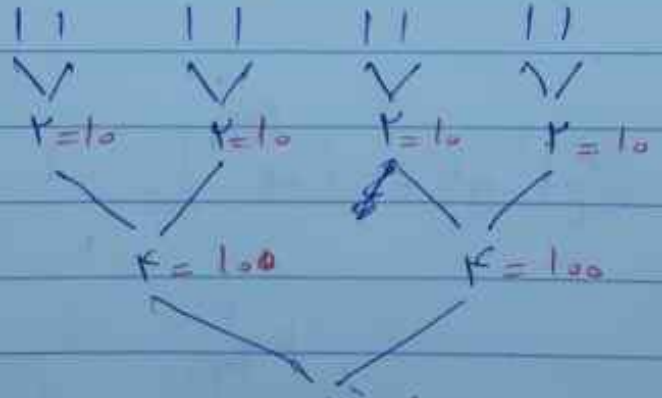
$n=5$



$1+1+1+1=4$

$order = \Theta(n)$

بهترین حالت: زمانی است که همه بیت‌ها ۱ باشند.



$$Cost = \frac{n}{2} \times 1 + \frac{n}{4} \times 2 + \frac{n}{8} \times 3 + \cdots + \frac{n}{2^i} \times i + \cdots$$

$$= n \times \sum_{i=1}^{\log n} \frac{i}{2^i} = n \times C = \Theta(n)$$

سوال ۷ :

ب) می خواهیم عدد x را به توان n برسانیم. اگر عدد x را x فرض کنیم و n را y فرض کنیم داریم :

```
power (x, y)
{
  if (y = 0)
     return 1
  else if (y%2 = 0)
     return Power(x, y/2) * power (x, y/2)
  else
     return x* Power (x, y/2) * power (x, y/2)
}
```

در این الگوریتم از روش تقسیم و غلبه استفاده کرده و هر بار y را به دو تا substanse با اندازه های نصف ($\frac{y}{2}$) می شکنیم و بازگشتی را کامل می کنیم.

پیچیدگی این الگوریتم از O(n) خواهد بود اما این الگوریتم مشکلی که دارد این است که به ازای هر مقدار دو بار فراخوانی و تابع بازگشتی دوبار کامل شده و محاسبه می شود. برای بهبود الگوریتم می توانیم یک بار این مقدار را محاسبه کرده و ذخیره کنیم تا هر بار از مقدار ذخیره شده استفاده کنیم و نیازی به محاسبه مجدد نباشد.

بعد الگوریتم قبلی:

```
power(x, y)
{   int temp
    if(y==0)
      return 1
    temp = power(x, y/2)
    if (y%2 == 0)
        return temp * temp
    else
      return x * temp * temp
}
```

مقدار بازگشت را یک بار محاسبه کرده و در temp ذخیره کردیم تا هر بار آن را حساب نکنیم.

پیچیدگی این الگوریتم در بقیه موارد از $O(n)$ است و فقط در حالتی که تابع بازگشتی صدا زده میشود از پیچیدگی $O(1)$ است که هر بار y نصف میشود پس پیچیدگی از $O(\log n)$ است.

۸

الف) $H_n = \sum_{i=1}^{n} \frac{1}{i}$ $\lim_{n\to\infty} H_n = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{i}$ ①

از طرفی: $\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{i} \simeq \int_1^n \frac{1}{x}\,dx$ ②

① ، ② $\Rightarrow \lim_{n\to\infty} H_n \simeq \int_1^n \frac{1}{x}\,dx \Rightarrow \lim_{n\to\infty} H_n \simeq \ln(n)$

می‌دانیم: $\ln n \leq 1 + \ln n$

$\xrightarrow[\text{نامساوی}]{\text{جمع راست}}$ $H_n \leq 1 + \int_1^n \frac{1}{x}\,dx$

ب) $n! = 1 \times 2 \times 3 \times \dots \times (n-1)(n)$

$\ln(n!) = \ln(1 \times 2 \times 3 \times \dots \times (n-1) \times n) = \ln(1) + \ln(2) +$

$\ln(3) + \dots + \ln(n-1) + \ln(n) = \sum_{i=1}^{n} \ln(i)$

$\sum_{i=1}^{n} \ln(i) \simeq \int_1^n \ln x\,dx$

از طرفی می‌دانیم که همواره $\ln(n) < \ln(n+1)$ است پس داریم:

$\int_1^n \ln x\,dx \leq \ln(n!) \leq \int_1^{n+1} \ln x\,dx$